Randomized algorithms: Quick reference

vishvAs vAsuki

March 15, 2012

Based on [1]. Also see the probabilistic analysis quick reference.

Contents

1 Notation

whp: With high probability. PPT: Probabilistic polynomial time. Monte carlo algorithm: uses repeated random sampling for computation.

2 Generate random bits

Random bit generator is usually a physical device. Usually, $Pr(X = 1) = p$; from this, easily get random generator with $Pr(X = 1) = 1/2$: flip every alternate bit. Flippling every t-th bit, get Pr(X = 1) = 1/t.

2.1 2-wise independent bits generation

$2^b - 1$ 2-ind bits Y_i from b ind bits X_i: For each subset, $Y_i = \oplus X_i$. In GF(p), p 2-ind elements from 2 ind elements: $Y_i = (X_1 + iX_2) \mod p$ for every i in GF(p). 2-universal ($Pr(h(x) = h(x')) \leq n^{-1}$) hash function family: H has $h : U \to V; |V| = n; a, b \in GF(p), a \neq 0$; $h_{a,b}(x) = ((b + xa) \mod p) \mod n$. If a can be 0, strongly 2-universal ($Pr(h(x) = y, h(x') = y') = n^{-2}$).

2.2 Pseudorandom generator G

Based on operational meaning of randomness: Better than early definitions, which were based on statistical properties of bit string.
$G : \{0,1\}^l \to \{0,1\}^n$, computable in time $2^{O(l)}$, is an $(\epsilon, s(n))$ pseudorandom generator if $\forall$ ckts c of size s(n) [strength parameter], Indistinguishability / unpredictability property holds: $Pr_{y\in\{0,1\}^n}[c(y) = 1] - Pr_{x\in\{0,1\}^l}[c(G(x)) = 1] \leq \epsilon$ [error parameter].
Distinguishers c usually try to learn from string and guess if G generated it. For example of distinguisher, see colt ref.
Notion similar to PFF: If ye pick y, you've picked random functions f; if ye pick x, you've picked a subset of pseudorandom functions.

2.2.1 Hard function f

f is (a(n), s(n)) hard if $\forall$ ckts of size $\leq s(n)$,
$Pr_{y\in\{0,1\}^n}[c(y) = f(y)] \leq 2^{-1} + a(n)$. (a(n), s(n), D) hardness: a generalized notion: take Pr wrt D.
Worst case hardness: $a(n) = 2^{-1}+2^{-n}$: otherwise, c can memorize +ve inputs.
Any pseudorandom generator is also hard. [**Find proof**]
Any hard function is also a pseudorandom generator. [**Find proof**]

2.2.2 Hard core function f

f is (a(n), s(n), M) hard core wrt
measure M (defining distribution D_M) if f is $(a(n), s(n), D_M)$ hard.
If f is hard wrt distribution D which is uniform over set S, it is (a(n), s(n), S) hard core. Then S is a (a(n), s(n), f) hard core set. If (a(n), s(n), M) hard, then (a(n), s(n), S) hard for S of a certain size.
(Impagliazzo): If f is (a(n), s(n), U) hard, it is $(b(n), b(n)^2a(n)^2, M)$ hard core where D_M is close to U: if not hard core, could smoothly boost the good guessing ckt to get a ckt which violates hardness.

2.2.3 BBS pseudorandom generator

(Blum, Blum, Shub). Input: N=pq: p, q are primes = 2 mod 4; Initial seed s_0 of n bits. Output: A stream of poly(n) bits which look random. $s_i = s_{i-1}^2 \mod N = s_0^{2^i} \mod N$; ith bit output: $b_i := LSB(s_i)$.
If factoring is hard, you cannot distinguish between a truly random m bit string and an m bit string obtained by choosing s_0 at random and running BBS gen. [**Find proof**]

2.3 Pseudo random function family (PFF)

2.3.1 Function family F

Set of polynomial sized boolean circuits with input length n, of size $O(e^n)$ with samplable index set I; $\exists$ alg A to input $i \in I$, return $f_i \in F$, simulate its input/ output behavior by accepting x and returning $f_i(x)$ in poly time.

2.3.2 Distinguisher D

Any Poly time alg; inputs function f: black box access to it; outputs 1 if it thinks f is random.

2.3.2.1 Indistinguishability

Let f be truly random function. F is PFF if for every D, $Pr_{f \in rand}(D^f = 1) - Pr_{i \in_U I}(D^{f_i} = 1) < O(e^{-n})$.

2.3.3 Existance

Goldwasser, Goldreich, Micali (GGM): If one way functions exist (if factoring is hard), then PFF's exist.
[**Incomplete**]

3 Sampling from distribution

3.1 Sampling a random variable X

3.1.1 Problem

The pseudo random number generator yields a sequence of almost independent random bits: see randomized alg ref. How do you use them to sample from a given distribution?

3.1.1.1 Visualization

Want to 'cover' the entire range(X) by the sampling: visualize as throwing darts in a oval: dart density corresponds to probability; the shape formed by the darts corresponds to the Pr(X) contour.

3.1.1.2 Challenges

Can't sample from $\Re$: computer can't even store all possible $\Re$. So, must sample from Q.
Or, Pr(X=x) or $Pr(X = x|Y = y)$ may be difficult to calculate.

3.1.2 Sampling some distributions

3.1.2.1 Uniform distribution

Sampling from U[a,b] is easy: sample 1 bit at a time form the range.

3.1.2.2 Discrete probability distribution

Take a line segment [0, 1], partition it into ranges $\{R_i\}$ corresponding to probabilities of various values; Sample x from U[0, 1]; identify its range j; output value j corresponding to R_j.

3.1.2.3 Normal distribution

Use rejection sampling: accept x drawn from uniform distribution with probability proportional to $e^{(x-\mu)^T \Sigma^{-1} (x-\mu)}$. Can similarly sample from any distribution.
Or, sample from discrete probability distribution which approximates N: but this is costly as the number of rational numbers in the range is large.

Multidimensional Normal distribution Sampling from a spherical normal distribution is easy: just apply a linear transformation to the inputs before sampling.

3.2 Problems solved by sampling X

3.2.1 The integration/ summation problem

Maybe you want to $E[f(X)|T]$ where T is an event (maybe 'true'), but ye find it difficult to do this analytically. The expectation could involve a combinatorial sum: the RV f(X) can take on many different values with different probabilities.

3.2.1.1 Find normalization const Z of Pr(X)

Given prob distribution $Pr(x) = Z^{-1} f(x)$ specified only by f(x) to find Z.

3.2.1.2 Finding Pr(T) for hard to analyze T

Same as finding the expectation of an indicator random variable.
Visualize sample space as an oval, and sampling as putting dots in the oval; you estimate the size of a smaller oval where the event is true.
Aka counting if every sample point is equally probable.

3.2.2 Optimization

Sampling with special attention being paid towards finding an optimum is considered elsewhere.

Maybe you want to find $\min_X f(X)$. As seen in optimization ref, in general it is hard to find a global minimum; so repeated sampling / random restarts with gradient descent can be a useful strategy.

3.2.2.1 Optimization after summation

Maybe want $\min_X E[f(X)|T]$.

3.2.3 Simulation

Maybe you want to simulate roll of dice, or want AI's which behave realistically in games.

3.3 Sampling Random vectors

3.3.1 Arbitrary randomg vectors

Suppose random vector X is 3-dimensional, and suppose that we are given the joint potential ϕ_X. One can first deduce and sample x_1 from f_{X_1}, then x_2 from $f_{X_2|X_1=x_1}$, and finally x_3 from $f_{X_3|X_1=x_1,X_2=x_2}$. The cost mostly comes from having to sum over $|ran(X_2)||ran(X_3)|$ terms in deducing the distributions.

3.3.2 From graphical models

3.3.2.1 Forward sampling for DAG's

Consider a directed graphical model with fully specified conditional probability distributions for non-root nodes and distributions for root nodes.
One simply samples successive each level of the DAG, starting at level 0, utilizing the conditional probability distributions given samples drawn from the previous level at each step.
This is an exact sampling technique.

3.3.2.2 Undirected trees

Sampling from undirected tree-structured graphical models can be accomplished by converting them efficiently to tree structured directed graphical models. Computing marginal probabilities by summing over pairwise potentials is efficient when $range(X_i)$ is finite, which then makes computation of conditional probabilities for the DAG efficient.

3.3.2.3 Factor trees

One can forward-sample tree structured factor graphs by conversion to equivalent directed trees whose nodes correspond to factor-nodes.

3.3.3 Arbitrary undirected graphs

Arbitrary undirected graphs may be sampled by conversion to corresponding factor trees.

3.4 Repeated Sampling

Aka Monte Carlo sampling. Repeated sampling is done for several purposes, like inference. It is usually much simpler when Pr(X) is modelled by a graphical model.
Here we consider various techniques for repeated sampling themselves, rather than their special applications to problems such as inference (as in rejection sampling).

3.4.1 Markov chain Stationary distribution

Aka Markov Chain Monte Carlo (MCMC).
Consider a stochastic process whose state space is $ran(X)$. For any distribution f_X, we can usually construct markov chain whose stationary distribution is f_X. So, doing appropriate random walks on the state transition graph of this Markov chain (see probabilistic models for details), one can sample various states from f_X.
But before each sample is picked, one must 'throw away some samples' in order to let the state distribution approximate the stationary distribution.
The art is in finding a fast mixing one. If it is time reversible, can get very fast mixing Markov chain.

3.4.1.1 Sampling from discrete distribution

Aka Metropolis alg.
Want to sample from distribution over
$range(X) = \{v_i\}$. Make ergodic Markov chain with right stationary distribution: Pick $M \geq maxDegree$, set $Pr(v_i \rightarrow v_j) = (1/M)min(1, \pi_{v_j}/\pi_{v_i})$ otherwise self-loop. Note that finding π_{v_j}/π_{v_i} usually easy : ye don't need to know normalization constant Z.

3.4.2 Repeated conditional sampling

Aka Gibbs sampling.

3.4.2.1 Algorithm

Start with random assignment to (X_i). For each i, sample $X_i \sim Pr(X_i|X_{j\neq i})$; repeat.
The conditional probability distribution is computed using
$f_{X_i|X_{-i}} = \phi_X/\phi_{X_{-i}} = \phi_X/\sum_{x_i}\phi_X$, where the joint potential ϕ_X is not necessarily normalized.

3.4.2.2 Correctness

Why is this same as sampling from Pr(X)? [**Find proof**]

3.4.2.3 Efficiency

Gibbs sampling is efficient as long as the ϕ_X can be efficiently computed. This is because of the fact that the normalization constant cancels out, and need not be computed; and becuase $range(X_i)$ is limited.

3.4.2.4 Modifications

This is akin to a 'random walk on a graph' behavior, so often takes long to cover the entire sample space well. Consider (X_1, X_2) with them being very correlated: visualize sample space as elongated ellipse around some line; so going L units away on X_i requires $O(L^2)$.
So, maybe use random restarts.

3.4.3 Block conditional sampling

Aka block-Gibbs sampling.

3.4.3.1 Algorithm

Here, we sample whole blocks of variables at a time. Consider a partition of variables X into A, B. Starting with a random assignment, we sample from $f_{A|B}$ and $f_{B|A}$ in turn. $f_{A|B} = \phi_X/\phi_B$, where ϕ is the potential function which is not necessarily normalized.
Finding $\phi_B(b) = \sum_a \phi_X(a, b)$ requires summing over $\prod_i ran(B_i)$ items in general, which is not efficient.

3.4.3.2 Advantages

It is better than Gibbs sampling when the subgraphs can be more efficiently or accurately (perhaps even exactly) sampled from, and when summing over them is easy. This is true for example when the partitions correspond to tree-structured subgraphs of directed graphical models.

3.5 Physics-inspired sampling

3.5.1 Energy temperature analogy

Can write $Pr(x) = Z^{-1}e^{-\frac{E(x)}{T}}$, where E is energy function, T is the temperature. [**Incomplete**]

4 Inference

Here we only consider inference by sampling. Inference techniques particular to various model classes are considered elsewhere.

4.1 Inference problems

Find Pr(X=x) or $Pr(X = x|Y = y)$. Inference algorithms may be exact or approximate.
Or find E[X] or $E[X|Y = y]$.
Or find the mode of $Pr(X = x|Y = y)$: as done in case of Maximum likelihood estimation.

4.1.1 Difficulty in calculating Pr(X=x)

Pr(X=x) can be difficult to calculate analytically. One could specify $Pr(X = x) = Z^{-1}f(x)$ by leaving the normalization constant Z unspecified, and describing only $f(x)$.
Similarly, $Pr(X = x|Y = y)$ can involve solving the 'summation problem'.
Can be difficult due to not knowing the normalization constant.
Often probability of an event T cannot be estimated analytically, even when probability of an atomic event is known. Eg: Consider uniform density over a bounded space in R^d, consider an irregular shape in it representing event T; now find Pr(T). Ye can throw darts/ sample to find the area (Monte carlo!). Or construct a grid to calculate the volume of the irregular shape: but here the computation will be N^d, where N is number of grid points in each dimension.

4.2 Inference by sampling

4.2.1 Rejection sampling

To find $E[f(X)|E_1]$, Sample from distribution of X, reject samples for which $\neg E_1$, then use remaining points to find avg f(X).

4.2.2 Set counting techniques

Design sample space, sample m times to estimate event probability.

Maybe iteratively constrict sample space and multiply probabilities, derive number of events: Thus, one finds
$Pr(E_1), Pr(E_2|E_1)$.. finally finding $Pr(Y|E_1..)$; thence finding Pr(Y).

4.3 Inference quality

4.3.1 FPRAS for (eps, d) approx.

RAS: Randomized approximation scheme. Suppose $Pr(|m^{-1}\sum_{i=1}^{m}(X-\mu)| \geq \epsilon\mu) \leq d$. If m is FP (Fully Polynomial) in $\ln(\frac{2}{d}), \epsilon^{-2}, |x|$, we have a FPRAS. From Chernoff bound : $m \geq \frac{3\ln(\frac{2}{d})}{\epsilon^2\mu}$.

4.3.2 Fully poly almost uniform samplers (FPAUS)

[**Incomplete**]

5 Ways of making randomized algorithms

5.1 Algs for problems in the class RP

One sided error; success probability $p = Pr(f(x) = 1|x \in L) \geq 2^{-1}$.

5.1.1 Amplification of confidence of RP alg

5.1.1.1 'Monte Carlo' search alg

Sample a solution, check correctness, repeat t times, lower bound p. If trials n-wise independent: tn random bits; failure probability: $(1-p)^t \approx e^{-tp} \leq 2^{-tp} \to 0$ for large t. So, even if $p = (poly(n))^{-1}$; for t = poly'(n), success whp. If trials 2-wise independent: 2n random bits: t^{-1} using Chebyshev ineq. [**Check**]

5.1.1.2 Negligible success probability

Anything asymptotically
smaller than an inverse polynomial. Eg: $O(n^{-\ln n}), e^{-n}$. Else, could boost success rate.

5.1.2 'Las Vegas' search alg

Repeat MC alg till confirmed success.

5.1.3 Disperser

Uneven bipartite graph $(V = L \cup R, E)$; Min degree of v in L: D; N(v): nighbors of v; $\forall S \subseteq L, |S| = k$, expand: connect to $\geq \frac{|R|}{2}$ verteces. How small be D? Solving Pr(G not disperser) < 1: $D \geq \log|L| + 1$.

5.1.3.1 RP amplification

Make disperser with D=t; pick v in L; use N(v) as random bits. [**Incomplete**]

5.2 BPP alg

2-sided error. Repeat many times, take majority, use Chernoff.

5.3 Random projections

Solve problem in $\mathbb{R}^D$ by projecting points in V to random $\mathbb{R}^d$, $d < D$; distance preserved approx whp! Eg: Approx nearest neighbor, clustering; proj points in $\mathbb{R}^2$ to most lines. Take $x = (a-b) \in \mathbb{R}^D$, change bases to get $y = (a-b) \in \mathbb{R}^D$, project to first d directions: drop D-d coords, get $z \in \mathbb{R}^d$, renormalize (mult by $(\frac{D}{d})^{0.5}$) as projection reduces length.
(Johnson, Lindenstrauss): Let $|V| = n$; $\epsilon \in (0, 0.5), d = \frac{18}{\epsilon^2} \ln n$: $\exists$ `whp` f : $\mathbb{R}^D \to \mathbb{R}^d$ $\forall (a,b) \in V$, $\frac{\|f(a)-f(b)\|-\|a-b\|}{\|a-b\|} \leq \epsilon$: Proj to rand plane $\equiv$ proj rand unit vector y to get z. As $Pr(\|z\|^2 \leq k(\frac{d}{D})) \leq exp(\frac{d}{2}(1-k+\log k))$: Chernoff style proof; show f(a)-f(b) close to a-b; then use union bound to cover all pairs in V.
Tight up to a factor $\log(1/\epsilon)$: $\exists$ points V: $\Omega(\frac{\log n}{\epsilon^2 \log(\frac{1}{\epsilon})})$ to preserve distances.

5.4 Fingerprinting

BigObj1 = BigObj2 $\Leftrightarrow$ SmallObj1 = SmallObj2.

5.4.1 Matrix multiplication check

Problem: AB ?= C. Ordinary algorithm: Finding AB costs $O(n^{2.4})$.
Randomized algorithm: Sample r from $\{0,1\}^n$, check if $ABr = Cr$: $O(n^2)$.
Analysis: Let D = AB-C; $AB \neq C \implies [Pr(Dr = 0) \leq 2^{-1}]$ using principle of deferred decisions. Also from Schwartz Zippel Thm (see Algebra ref).

5.4.2 Codes and communication complexity

See Information and coding theory ref.

5.4.3 String search

Find pattern vector $y \in \{0,1\}^m$ in text vector $x \in \{0,1\}^n$; x(j) is $x_j..x_{j+m-1}$. Trivial alg: O(nm). Optimal det alg: (Boyer Moore) O(n+m) flops. Rand alg: compare fingerprints. Take rand prime p; use $F_p(x) = x mod p$. Find $F_p(x(j+1))$ from $F_p(x(j))$ in O(1) flops: precompute $F_p(2^{m-1})$; $x(j+1) =$

$2(x(j)-x_j 2^{m-1})+x_{j+m}$; So O(n+m) flops. From $\cup$ bound, prime num theorem, Pr(false match) $\leq (n-m+1)\frac{m}{\Pi(mn^3)} = O(\frac{\log n}{n^2})$. [**Incomplete**]

6 Derandomization, explicit construction

6.1 Derandomization

Design algorithm which uses random choices; Replace random choice with choice which maximizes conditional expectation; Eg: find $\geq \frac{|E|}{2}$ cut in G=(V, E). Devise rand alg which uses n 2-wise ind bits; enumerate all values for $\log n$ truly ind bits, make n 2-wise ind bits and run rand alg.

6.2 Explicit construction algorithms

Sample values for some variables, show that desired object still exists, use exhaustive search for other variables.

6.3 Sampling uniformly from unit sphere

Pick each point according to $N(0, n^{-1})$. [**Find proof**]

7 Approximation algorithms

7.1 LP based Approximation algorithms

Rephrase (maybe NP hard) problem as Integer Programming problem; Make LP relaxation; solve in polytime; translate solution by rounding; make (δ, ϵ) approximation guarantees. Rounding choices: To closest integer, or randomized rounding.

7.1.1 Vertex cover problem

G=(V,E). IP: Vars $v_i = 0$ or 1, $\forall (i,j) \in E, v_i + v_j \geq 1$, $\min \sum v_i$ =?; LP relaxation: $\hat{v}_i \in [0,1]$; solution $\sum \hat{v}_i \leq opt \leq \sum v_i$; round to nearest int to get approx soln $\{v_i\}$; as $v_i \leq 2\hat{v}_i$: $\sum v_i \leq 2opt$.
Max SAT.
[**Incomplete**]

7.2 Semidefinite programming based approximation algs

[**Incomplete**]

8 Applications

For number theory applications, see number theory ref.

8.1 Perfect matchings

8.1.1 Bipartite graph

G = (U, V, E); $|U| = |V| = n$. Matching: $\{e\} \in E$ not sharing endpoints. Perfect matching (pm) has size n. Naive alg takes $O((n!)n)$ time.
Make symbolic matrix with $A_{i,j} = x_{i,j}$ if $(u_i, v_j) \in E$, else 0. Let Q(x) = det(A) : n^2-nomial, 'symbolic determinant'. G has pm iff $\exists r : Q(r) \neq 0$. $deg(Q) \leq n$. Take prime $p > 2n$, pick r from Z_p; by Schwartz Zippel Thm, $Pr(Q(r) = 0 | \exists r' | Q(r') \neq 0) < 2^{-1}$.
If pm unique, find pm in polytime: Repeat: $E = E - \{e\}$; see if G still has pm. (Parallelizable.)

8.1.1.1 If pm not unique

Take uniformly random weight fn $w : [1, m] \to [1, 2m]$; let $W(S) = \sum_{d \in S} w(d)$.

Isolation lemma Let $S_i \in [1, m]$; take k S_i. $Pr(\exists minWt\ S_i, S_j | e \in S_i, e \notin S_j) \leq (2m)^{-1}$: by principle of deferred decisions: Pick W(e) last to $W(S_j) - W(S_i - \{e\})$. So, $\cup$ bound: Pr($\exists$ unique S_i with $W(S_i)$ min) $\leq 2^{-1}$.
Get w(e) $\forall e = (u_i, v_j) \in E$. Get matrix $B_{i,j} = 2^{w((u_i, v_j))}$. Then $det(B) = \sum_{pm\ M} (\pm 2^{W(M)})$. So find pm by finding highest power of 2 dividign det(B).

8.1.2 General graph

(Tutte): G = (V, E); Make Tutte matrix: $\forall i < j$ if $(u_i, v_j) \in E$, $A_{i,j} = x_{i,j}$ and $A_{j,i} = -x_{i,j}$, else 0. Let $|A| = Q(x)$ multinomial; G has pm iff $\exists x Q(x) \neq 0$: $det(A) = \sum_\pi sgn(\pi) \prod A_{i,\pi(i)}$; if some $\forall \pi : \prod A_{i,\pi(i)} = 0$, no pm. Else, Take prime $p > 2n$, pick r from Z_p, take Q over Z_p; by Schwartz Zippel, $\exists x$ where $Q(x) \neq 0$.

9 Analysis of randomized algorithms

See probabilistic analysis ref.

9.1 General Tricks

Running time of **MDP-algorithm**: Make Markov chain, make RV Z_i steps for going to absorbing state from i, get and solve recurrance eqns for $E[Z_i]$.

9.2 Results

2-SAT rand alg needs $O(n^2)$ time. 3-SAT rand alg (without restart) needs $O(n^{3/2}(\frac{4}{3})^n)$ time as Pr(moving forward in Markov chain) $< 1/2$. Max load Y when hash function from k-universal family used: $Pr(Y > \sqrt[k]{2n}) < 2^{-1}$ (bounding expected number of collisions, use Markov).

Bibliography

[1] *Probability and Computing: Randomized Algorithms and Probabilistic Analysis.* Cambridge University Press, 2005.