An Introduction to Randomized algorithms

C.R. Subramanian

The Institute of Mathematical Sciences, Chennai.

Expository talk presented at the Research Promotion Workshop on "Introduction to Geometric and Graph Algorithms" at National Institute of Technology, Suratkal, January 10-12, 2012.

Randomized Algorithms

- A deterministic algorithm + auxiliary input of a sequence of unbiased and independent random bits.
- RA - a randomized algorithm to solve π.
- At every point during an execution of algorithm RA over I, the next move of A can possibly be determined by employing randomly chosen bits and is not uniquely well-defined.
- The execution and running time, intermediate steps and the final output computed could possibly vary for different executions of RA over the same I.

Why Randomization ?

- Randomness often helps in significantly reducing the work involved in determining a correct choice when there are several but finding one is very time consuming.
- Reduction of work (and time) can be significant on the average or in the worst case.
- Randomness often leads to very simple and elegant approaches to solve a problem or it can improve the performance of the same algorithm.
- Risk : loss of confidence in the correctness. This loss can be made very small by repeated employment of randomness.
- Assumes the availability of truly unbiased random bits which are very expensive to generate in practice.

Some Tail Inequalities

- X - random variable : $\mu = E[X]$; $Var(X) = E[X^2] - \mu^2$.
- Markov : X is nonnegative ;
- For every $t > 0$, $\mathbf{Pr}(X \geq t) \leq \mu/t$.
- Chebyschev : $\forall t > 0$, $\mathbf{Pr}(|X - \mu| \geq t) \leq Var(X)/t^2$.
- $\forall \epsilon \in (0, 1]$, $\mathbf{Pr}(|X - \mu| \geq \epsilon\mu) \leq Var(X)/\epsilon^2\mu^2$.
- Chernoff : $X = X_1 + \ldots + X_n$; $X_i \in \{0, 1\}$. independent.
- $\forall \epsilon \in (0, 1]$, $\mathbf{Pr}(X \leq \mu(1 - \epsilon)) \leq e^{-\epsilon^2\mu/2}$.
- $\forall \epsilon \in (0, 1]$, $\mathbf{Pr}(X \geq \mu(1 + \epsilon)) \leq e^{-\epsilon^2\mu/3}$.

Verifying matrix multiplication :

- $A, B, C \in F^{n \times n}$; Goal : To verify if $AB = C$.
- direct approach - $O(n^3)$ time.
- algebraic approach - $O(n^{2.376})$ time.
- Randomized Alg :
- Choose *u.a.r.* $r \in \{0,1\}^n$ and check if $ABr = Cr$.
- If so, output YES, otherwise output NO.
- If $AB \neq C$, then $\mathbf{Pr}(ABr = Cr) \leq 1/2$.
- requires $O(n^2)$ time.
- An example of a Monte-Carlo algorithm : can be incorrect but guaranteed running time and with a guarantee of confidence.

QuickSort(A, p, q):

- If $p \geq q$, EXIT.
- $s \leftarrow$ correct position of $A[p]$ in the sorted order.
- Move the "pivot" $A[p]$ into position s.
- Move the remaining elements into "appropriate" positions.
- Quicksort$(A, p, s-1)$;
- Quicksort$(A, s+1, q)$.

Worse-case Complexity of QuickSort :

- $T(n) =$ Worst-case Complexity of QuickSort on an input of size n. Only comparisons are counted.
- $T(n) = \max\{T(\pi) : \pi \text{ is a permutation of } [n]\}$.
- $T(n) = \Theta(n^2)$.
- Worst case input is : $\pi = \langle n, n-1, \ldots, 1 \rangle$.
- There exist inputs requiring $\Theta(n^2)$ time.

Randomized Version : RandQSort(A, p, q):

- If $p \geq q$, EXIT.
- Choose uniformly at random $r \in \{p, \ldots, q\}$.
- $s \leftarrow$ correct position of $A[r]$ in the sorted order.
- Move randomly chosen pivot $A[r]$ into position s.
- Move the remaining elements into "appropriate" positions.
- RandQSort$(A, p, s-1)$;
- RandQSort$(A, s+1, q)$.

Analysis of RandQSort :

- Every comparison is between a pivot and another element.
- two elements are compared at most once.
- rank of an element is the position in the sorted order.
- x_i is the element of rank i. $S_{i,j} = \{x_i, \ldots, x_j\}$.
- $X_{i,j} = 1$ if x_i and x_j are ever compared and 0 otherwise.
- $E[T(\pi)] = E[\sum_{i<j} X_{i,j}] = \sum_{i<j} E[X_{i,j}]$.
- $E[X_{i,j}] = \frac{2}{j-i+1}$.
- $E[T(\pi)] = \sum_{i<j} \frac{2}{j-i+1} \leq 2nH_n = \Theta(n(\log n))$.

Example of randomness improving the efficiency :

- Analysis holds for every permutation π.
- $T(n) =$ Maximum value of the Expected Time Complexity of RandQSort on an input of size n.
- $T(n) = \max\{E[T(\pi)] : \pi \text{ is a permutation of } [n]\}$.
- $T(n) = \Theta(n(\log n))$.
- For every π, $\mathbf{Pr}(T(\pi) > 8nH_n) \leq 1/4$.
- introducing randomness very likely improves the efficiency.
- An example of a Las Vegas algorithm : always correct but running time varies but with possibly poly *expected* time.

Las Vegas vs Monte-Carlo :

- Las Vegas $\rightarrow$ Monte-Carlo
- A - Las Vegas algo with $E[T_A(I)] \leq poly(n)$ for every I.
- By incorporating a counter which counts every elementary step into A and stopping after, say, $4poly(n)$ steps, one gets a poly time Monte-Carlo algorithm B with a guaranteed confidence of at least $3/4$.
- Monte-Carlo $\rightarrow$ Las Vegas
- A - Monte-Carlo alg with $poly(n)$ time and $1/poly(n)$ success probability. Suppose correctness of output can be verified in $poly(n)$ time.
- By running the alg A repeatedly (with independent coin tosses) until one gets a correct solution, we get a Las Vegas algo with poly expected time.

Randomization provably helps

- A simple counting problem :
- $A[1 \ldots n]$-array with $A_i \in \{1, 2\}$ for every i.
- $\mathrm{f(x)} = \mathrm{freq}(x) \geq n/5$ for each $x \in \{1, 2\}$.
- Goal : Given $x \in \{1, 2\}$ and an $\epsilon > 0$,
- determine *ans* : $ans \in [(1 - \epsilon)f(x), (1 + \epsilon)f(x)]$.
- Any deter. alg needs $\Omega(n)$ queries in the worst case for $\epsilon = 1/10$.
- $\exists$ rand. alg with $O(\log n)$ queries for every fixed ϵ.

Randomization provably helps

- **RandAlg**(A, x, ϵ) **:**
- $m = 20(\log n)/\epsilon^2$; $c = 0$.
- **for** $i = 1, \ldots, m$ **do**
- Choose uniformly at random $j \in \{1, \ldots, n\}$.
- **if** $A[j] = x$ **then** increment c.
- **endfor**
- Return $ans = nc/m$.
- **end**

Randomization provably helps

- **Analysis of RandAlg(A, x, ϵ) :**
- $X_i = 1$ if $A[j] = x$ for j chosen in the ith-iteration.
- $c = \sum_i X_i$; $E[X_i] = f(x)/n$.
- $\mu = E[c] = mf(x)/n \geq m/5$. $E[ans] = f(x)$.
- $\mathbf{Pr}(c \notin [(1-\epsilon)\mu, (1+\epsilon)\mu]) \leq 2e^{-\epsilon^2\mu/3} = o(n^{-1})$.
- $(1-\epsilon)f(x) \leq ans \leq (1+\epsilon)f(x)$ with probability $1 - o(1)$.
- No. of queries $= O((\log n)/\epsilon^2)$.
- No. of queries $= O(1)$ with success probability $\geq 3/4$.

Unbiased Estimator

- A is a randomized algorithm to approximate $\#I$.
- A outputs X such that $\mu = E[X] = \#I$.
- $\mathbf{Pr}(X \notin [(1 \pm \epsilon)\mu]) \leq Var(X)/\epsilon^2\mu^2$ by Chebyshev.
- $Var(X) = O(\mu) \Rightarrow reqd.prob = O(\epsilon^{-2}\mu^{-1})$.
- Example above : $Var(X) \leq \mu$, helps us !
- Often, X is not so nicely defined and $Var(X)$ may not be small compared to μ^2.

Boosting Success Probability - I

- Run m independent trials of $A(I, \epsilon)$.
- Take *ans* to be the numerical average of $\{X_1, \ldots, X_m\}$.
- $E[ans] = \mu$ and $Var(ans) = Var(X)/m$.
- $\mathbf{Pr}(ans \notin [(1 \pm \epsilon)\mu]) \leq Var(X)/m\epsilon^2\mu^2$.
- $\mathbf{Pr}(\text{success}) \geq 3/4$ provided $m \geq 4E[X^2]/\epsilon^2\mu^2$.
- a good approximation efficiently computable.

Boosting success probability - II

- A is a randomized algorithm to approximate $\#I$.
- A runs in time $poly(n, 1/\epsilon)$ and outputs ans :
- $\mathbf{Pr}((1-\epsilon)(\#I) \leq ans \leq (1+\epsilon)(\#I)) \geq 1/2 + \delta$.
- Run m independent trials of $A(I, \epsilon)$.
- Take ans to be the median of $\{ans_1, \ldots, ans_m\}$.
- $\mathbf{Pr}\left((1-\epsilon)(\#I) \leq ans \leq (1+\epsilon)(\#I)\right) \geq 1 - e^{-\delta^2 m/2}$.
- $\mathbf{Pr}(\text{success}) = 1 - o(n^{-1})$ provided $m \geq 4(\log n)/(\delta^2)$.
- a good approximation efficiently computable.

Approximating Frequency Moments

- Given $A = (a_1, \ldots, a_m)$, $a_j \in \{1, \ldots, n\}$.
- $m_i =$ frequency of i in A, $i \in \{1, \ldots, n\}$.
- Determine $F_k = \sum_i m_i^k$ using "small" space.
- $F_0 =$ number of distinct elements in A.
- $F_1 =$ length of the sequence A ; $F_2 =$ repeat rate of A.
- Determining F_k arises in Data Mining.
- Suppose we want to collect some statistical information from
- a large stream of data without having to store the data.
- $\Omega(n)$ bits needed for any deter. alg approximating F_k within a ratio of 1 ± 0.1.

Approximating F_2 - algorithm (Alon, Matias, Szegedy)

- $V = \{v_1, \ldots, v_h\}$, $h = O(n^2)$, each v_i - a n-vector of ± 1.
- V is four-wise independent. For $v \in_R V$, $\forall i_1 \leq \ldots \leq i_4$, $\forall(\epsilon_1, \ldots \epsilon_4) \in \{-1, 1\}^4$, $\mathbf{Pr}(\forall j, v(i_j) = \epsilon_j) = 1/16$.
- Choose $p \in_R \{1, \ldots, h\}$ and store it using $O(\log n)$ bits.
- $v_p(i)$ can be found (for a given i) using only $O(\log n)$ bits.
- $Z = \sum_i \epsilon_i m_i$. Z can be computed in one pass using $O(\log n + \log m)$ bits.
- Compute $X = Z^2$. space $= O(\log m + \log n)$.
- Take $s_1 = 16/\lambda^2$ independent samples $X_j = X$ and take their average Y.
- Take $s_2 = 2 \log(1/\epsilon)$ independent samples $Y_i = Y$ and output their median.

Approximating F_2 - analysis

- $E[X] = E[(\sum_i \epsilon_i m_i)^2] = \sum_i m_i^2 = F_2$.
- $E[X^2] = \sum_i m_i^4 + 6 \sum_{i<j} m_i^2 m_j^2$.
- $Var(X) = E[X^2] - E[X]^2 = 4 \sum_{i<j} m_i^2 m_j^2 \leq 2F_2^2$.
- $\mathbf{Pr}(|Y_i - F_2| \geq \lambda F_2) \leq \frac{2F_2^2}{s_1 \lambda^2 F_2^2} \leq 1/8$.
- $\mathbf{Pr}(|Y - F_2| \geq \lambda F_2) \leq \epsilon$.
- Total space complexity $= O(\log(1/\epsilon)(\log n + \log m)/\lambda^2)$ bits.

Randomized Rounding

- $V = \{x_1, \dots x_n\}$ boolean variables.
- $F = C_1 \wedge \dots \wedge C_m$; each C_j is a disjunction of literals.
- Eg : $C_j = x_1 \; v \; x_4^c \; v \; x_5$.
- MaxSat(F, V) : Given F over V, find an assignment satisfying the maximum number of clauses.
- This is a NP-hard problem.
- A simple deterministic alg satisfies at least $m/2$ clauses.
- Can we do better ?

Randomized Rounding

- Randomized Solution : Choose $f : V \rightarrow \{T, F\}$ *uar*
- $|C_j| \geq k \Rightarrow \mathbf{Pr}(f \text{ satisfies } C_j) \geq 2^{-k}$.
- Leads to an randomized approx alg which finds a f satisfying at least $3m/4$ clauses on the average if $|C_j| \geq 2$ for every j.
- $\forall j\ |C_j| \geq k \Rightarrow E[\#f] \geq m(1 - 2^{-k})$.
- What if we have a mixture of clauses of different sizes.
- $E[\#f] = \sum_k m_k(1 - 2^{-k})$ where $m_k = \{j : |C_j| = k\}$.
- Can we do better ?

LP-based Randomized Rounding

- ILP Formulation : Maximize $\sum_{1 \le j \le m} z_j$
- subject to : $\sum_{i \in C_j^+} y_i + \sum_{i \in C_j^-} (1 - y_i) \ \geq \ z_j \ \ \forall j$
- $y_i, z_j \in \{0, 1\}$ for every i and j.
- LP Relaxation : allow each $y_i, z_j \in [0, 1]$.
- An optimal solution $(y_i^*, z_j^*)_{i,j}$ of a LP can be found efficiently.
- Rand. Rounding : Independently and randomly set each $y_i = 1$ with probability y_i^*. Let g be the resulting assignment.
- $|C_j| = k \Rightarrow \mathbf{Pr}(C_j \text{ is satisfied by } g) \geq \beta_k z_j^*$ where $\beta_k = 1 - (1 - 1/k)^k \geq 1 - 1/e$.
- $E[\#f] \geq \sum_k \sum_{j : |C_j| = k} \beta_k z_j^* \ \geq (1 - 1/e) OPT(ILP)$.

LP-based Randomized Rounding

- $\beta_1 = 1$ and $\beta_2 = 0.75$ and $\beta_k < 1 - 2^{-k}$ for $k \geq 3$.
- Improved Algorithm B :
- Choose $f : V \rightarrow \{T, F\}$ *uar* and let X_1 be the number of clauses satisfied.
- Run the LP-based Randomized Rounding algo and let n_2 be the number of clauses satisfied.
- Return the best of the two solutions found which satisfies at least $\max\{n_1, n_2\}$ clauses.
- $\max\{E[n_1], E[n_2]\} \geq (0.75) \sum_j z_j^* \geq (0.75) OPT(ILP)$.

Randomized algorithms

- RA - a poly time rand algo for a decision problem π.
- each instance has only y/n answer.
- For $I \in YES(\pi)$, $\mathbf{Pr}(ans(RA, I) = y) \geq 1/2$;
- For $I \in NO(\pi)$, $\mathbf{Pr}(ans(RA, I) = n) = 1$;
- Number of bits used $r = r(n)$.
- $RP = \{L \subseteq \Sigma^* : \exists$ such a rand algo RA for $\pi\}$.
- $RP \subseteq NP$; Is $NP \subseteq RP$?
- RA is a RP algorithm.

Boosting Success Probability - I

- Suppose we want a confidence of $1 - \delta$.
- Algorithm RB :
-     Run m independent trials of $RA(I)$.
-     Output y if at least one trial says y ;
-     Otherwise, output n ;
- $T_{RB}(I) \leq mT_{RA}(I)$.
- For $I \in YES(\pi)$, $\mathbf{Pr}(ans(RB(I)) = n) \leq \delta$
-     provided $m \geq \log_2(1/\delta)$.
- To make error prob at most 2^{-m}, we need mr random bits.
- Can we do with less random bits ?

Boosting Success Probability - II

- Choose $R \in Z_p = \{0, \ldots, p-1\}$ u.a.r.
- $\mathbf{Pr}(A(x, R) = 1) \geq 1/2$ if $x \in L$ and is 0 otherwise.
- Choose $a, b \in Z_p$ uniformly and independently at random.
- Compute $R_i = ai + b(modp)$ for $1 \leq i \leq t$.
- Each R_i is uniformly distributed over Z_p.
- R_is are pairwise independent.
- $Z_i = 1$ if $A(x, R_i) = 1$ and is 0 otherwise. $Z = \sum_i Z_i$.
- $E[Z] \geq t/2$ and $Var(Z) = \sum_i Var(Z_i) \leq t/4$.
- By Chebyschev, $\mathbf{Pr}(Z = 0) \leq 1/t$.
- with just $2r$ independent bits, we get error prob at most $1/t$ as against $r(\log t)$ random bits required in independent trials.

Conclusions

- Employing randomness leads to improved simplicity and improved efficiency in solving the problem.
- However, assumes the availability of a perfect source of independent and unbiased random bits.
- access to truly unbiased and independent sequence of random bits is expensive and should be considered as an expensive resource like time and space. One should aim to minimize the use of randomness to the extent possible.
- assumes efficient realizability of any rational bias. However, this assumption introduces error and increases the work and the required number of random bits.
- There are ways to reduce the randomness from several algorithms while maintaining the efficiency nearly the same.