Advanced Algorithms **Fall 2015**

Lecture 8: Random Variable, Dec. 21, 2015

Faculty: K.R. Chowdhary *: Professor of CS*

8.1 Random variable

A random variable on a probability space Ω is just a function with domain Ω. Rather than writing a random variable as $f(\omega)$ everywhere, the convention is to write a random variable as a capital letter (X, Y, S, etc.) and make the argument implicit: X is really $X(\omega)$. Variables that are not random (or are not variable) are written in lowercase.

For example, consider the probability space corresponding to rolling two independent fair six-sided dice. There are 36 possible outcomes in this space, corresponding to the 6×6 pairs of values $\langle x, y\rangle$ we might see on the two dice. We could represent the value of each die as a random variable X or Y given by $X(\langle x, y\rangle) = x$ or $Y(\langle x, y\rangle) = y$, but for many applications, we do not care so much about the specific values on each die. Instead, we want to know the sum $S = X + Y$ of the dice. This value S is also random variable; as a function on Ω, it is defined by $S(\langle x, y\rangle) = x + y$.

8.1.1 Formal approach used

We think of a randomized algorithm as a machine M (or function) that computes $M(x, r)$, where $|x| = n$ is the input and r is the sequence of random bits. The machine model is the usual RAM model, where we have a memory space that is typically polynomial in the size of the input x, and in constant time we can read a memory location, write a memory location, or perform arithmetic operations on integers of up to $O(\log n)$ bits. For example, if $|x| = 1024$, then we represent it by $10-$bits.

In this model, we may find it easier to think of the random bits, supplied as needed by some subroutine, where generating a random integer of size $O(\log n)$ takes constant time. The justification for this assumption is that it takes constant time to read the next $O(\log n)$-sized value from the random input.

Because the number of these constant-time operations, and thus the running time for the algorithm as a whole, may depend on the random bits, it is now a *random variable* - a function on points in some probability space. The probability space Ω consists of all possible sequences r, each of which is assigned a probability $P_r[r]$ (typically $2^{|r|}$, and the running time for M on some input x is generally given as an expected value $E_r[time(M(x, r))]$, where for any X,

$$E_r[X] = \sum_{r\in\Omega} X(r).P_r[r]. \tag{8.1}$$

We can now quote the performance of M in terms of this expected value: where we would say that a deterministic algorithms runs in time $O(f(n))$, where $|x| = n$ is the size of the input, we instead say that our randomized algorithm runs in expected time $O(f(n))$, which means that $E_r[time(M(x, r))] = O(f(|x|))$ for all inputs x.

This is distinct from traditional worst-case analysis, where there is no expectation, and average-case analysis. The following trivial example shows the distinction.

Example 8.1 *Associate random variable X for flipping of a coin.*

Solution. Let sample space $S = \{H, T\}$. A random variable X_H, for coin coming as heads, i.e., event H. So,

$$X_H = \begin{cases} 1, & \text{if head comes,} \\ 0, & \text{if tail comes.} \end{cases} \tag{8.2}$$

The expected number of heads obtained in flip of a coin is,

$$\begin{aligned} E[X_H] &= 1.P_r\{H\} + 0.\{P_r\{T\} \\ &= 1.\frac{1}{2} + 0.\frac{1}{2} \\ &= \frac{1}{2} \end{aligned}$$

So the expected value of random variable associated with an event A is probability that event A occurs. Using of expected value is useful when repeated trials are performed. For example, to compute the number of heads in n coin flips by considering separately, the probability of heads, 1 heads, 2 heads, etc. Let X_i is random variable associated with the event in which the i-th flip comes up heads. Let X is the random variable denoting total number of heads in the n coin flip. So,

$$X = \sum_{i=1}^{n} X_i. \tag{8.3}$$

To compute expected number of heads, we take expectation in both sides,

$$\begin{aligned} E[X] &= E[\sum_{i=1}^{n} X_i] \\ &= \sum_{i=1}^{n} E[X_i] \\ &= \sum_{i=1}^{n} \frac{1}{2} \\ &= \frac{n}{2}. \end{aligned}$$

$\square$

8.1.2 Random variables and events

Any random variable X allows us to define events based on its possible values. Typically these are expressed by writing a predicate involving the random variable in square brackets. An example would be the probability that the sum of two dice is exactly 11 : $[S = 11]$; or that the sum of the dice is less than 5 : $[S < 5]$. These are both sets of outcomes; we could expand $[S = 11] = \{\langle 5, 6\rangle, \langle 6, 5\rangle\}$ or $[S < 5] = \{\langle 1, 1\rangle, \langle 1, 2\rangle, \langle 1, 3\rangle, \langle 2, 1\rangle, \langle 2, 2\rangle, \langle 3, 1\rangle\}$. This allows us to calculate the probability that a random variable has particular properties: $Pr[S = 11] = \frac{2}{36} = \frac{1}{18}$ and $P_r[S < 5] = \frac{6}{36} = \frac{1}{6}$.

The *probability mass function* of a random variable gives $P_r[X = x]$ for each possible value x. For example, our random variable S has the probability mass function shown in Table 8.1.

Table 8.1: Probability mass function for the sum of two independent fair six-sided dice.

S	Probability
2	1/36
3	2/36
4	3/36
5	4/36
6	5/36
7	6/36
8	5/36
9	4/36
10	3/36
11	2/36
12	1/36

For a discrete random variable X, the probability mass function gives enough information to calculate the probability of any event involving X, since we can just sum up cases using countable additivity. This gives us another way to compute $P_r[S < 5] = P_r[S = 2] + P_r[S = 3] + P_r[S = 4] = \frac{1+2+3}{36} = \frac{1}{6}$ (see table 8.1).

For two random variables, the *joint probability mass function* gives $P_r[X = x \wedge Y = y]$ for each pair of values x and y (this generalizes in the obvious way for more than two variables). To the extent that we can, we will try to avoid continuous random variables (opposite of discrete, e.g., coin flip and dice flip).

Two or more random variables are independent if all sets of events involving different random variables are independent. In terms of probability mass functions, X and Y are independent if $P_r[X = x \wedge Y = y] = P_r[X = x]P_r[Y = y]$.

For *discrete probability spaces*, any function on outcomes can be a random variable. The reason is that any event in a discrete probability space has a well-defined probability. For more general spaces, in order to be useful, events involving a random variable should have well-defined probabilities. For *discrete random variables* that take on only countably many values.

8.1.3 Expectation

The *expectation* or *expected value* of a random variable X is given by $E[X] = \sum_x xP_r[X = x]$. This is essentially an average value of X weighted by probability, and it only makes sense if X takes on values that can be summed in this way (e.g., real or complex values, or vectors in a real- or complex-valued vector space). Even if the expectation makes sense, it may be that a particular random variable X does not have an expectation, because the sum fails to converge.

For example, if X and Y are two independent fair six-sided dice, then

$$\begin{aligned} E[X] = E[Y] &= \sum_{i=1}^{6} i(\frac{1}{6}) \\ &= \frac{21}{6} \\ &= \frac{7}{2}, \end{aligned} \tag{8.4}$$

while $E[X+Y]$ from table 8.1 is:

$$\begin{aligned} \sum_{i=2}^{12} i.P_r[X+Y=i] &= \frac{2.1+3.2+4.3+5.4+6.5+7.6+8.5+9.4+10.3+11.2+12.1}{36} \\ &= \frac{252}{36} = 7 \end{aligned} \tag{8.5}$$

The fact that $7 = \frac{7}{2} + \frac{7}{2}$ here is not a coincidence.

The choice of appropriate random variable lies at the heart of the analysis of any randomized algorithm. An idea which is most often used is the linearity of *expectations*. Let X and Y be random variables. Then we have the following equality:

$$E[X+Y] = E[X] + E[Y] \tag{8.6}$$

It is important to note that this equation does not require independence between X and Y. This idea lets us simplify calculations that would be quite complex otherwise.

Example 8.2 *Suppose there are 26 students in a classroom and the professor has alphabet cards with him - each card has a different letter of the alphabet on it. Suppose the professor randomly gives one card per student. Then what is the expected number of people who get a card with the same letter as the first letter of their name?*

Solution: To answer this question, consider the random variables: $X_1, X_2, \ldots, X_{26}$, where for each i, $1 \leq i \leq 26$, we define $X_i = 1$ if ith person gets the correct letter, 0 otherwise. Since the cards were distributed randomly, the probability that the person gets the correct letter is $\frac{1}{26}$. Therefore, we have: $E[X_i] = \frac{1}{26}$ for all i. Finally, the expected number of people who get correct letters is given by $E[X_1 + X_2 + \cdots + X_{26}] = 1$. Note that there was no assumption made whatsoever about the first letters of peoples names. $\square$

Why Randomization works? The reason why randomized algorithms are successful in finding the solution is that often the problem instance has a lot of witnesses for yes or no. For example, in the case of checking whether a multi-variate polynomial is the zero polynomial, when the input polynomial is indeed a zero polynomial, then it evaluates to zero at all the points. When the polynomial is not zero, then there are a lot of points where the polynomial is non-zero. Hence, finding one witness out of the many is achieved easily by random sampling.

8.1.4 Max-Cut

Given a graph $G = (V, E)$, partition the vertex set into two sets (A, B). An edge is said to be cut by this partition, if its end points lie on different sides of the partition. Find a partition that maximizes the number of edges cut (see figure 8.1).

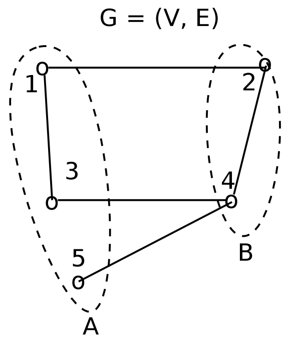

Figure 8.1: Finding a max-cut.

This problem is NP-hard. We will give a simple 0.5-approximation algorithm for this problem through randomization. This was the best known approximation guarantee for this problem for several years.

Let $|V| = n$ and $|E| = m$.

Claim 1. There exists a partition $(A, V \setminus A)$ of the vertex set such that number of edges cut $\geq \frac{|E|}{2} = \frac{m}{2}$.

Proof: We build a set A (i.e., one side of the partition) by including each vertex in A with probability 0.5. Now we ask the question: what is the expected number of edges cut by the partition $(A, V \setminus A)$. To answer this question, for every edge $(i, j) \in E$, we define $X_{ij} = 1$ if the edge (i, j) is cut by the partition, 0 otherwise.

We can write the following expression for the number of edges cut (X):

$$X = \sum_{(i,j) \in E} X_{ij} \tag{8.7}$$

The question we want to answer is: $E[X] =$? We use the linearity of the expectations. First convince your self that $E[X_{ij}] = \frac{1}{2}$.

This holds since, conditioned on i being in the set A or not, there is exactly one choice for j, which makes the edge (i, j) a cut. Hence, we get the following:

$$E[X] = \sum_{(i,j) \in E} E[X_{ij}] = \frac{m}{2}. \tag{8.8}$$

8.2 Sorting and Selection by Random Sampling

Some of the earliest randomized algorithms included algorithms for sorting a set (S) of numbers, and the related problem of finding the kth smallest element in S. The main idea behind these algorithms is the use of random sampling: a randomly chosen member of S is unlikely to be one of its largest or smallest elements; rather, it is likely to be "near the middle." Extending this intuition suggests that a random sample of elements from S is likely to be spread "roughly uniformly" in S. We now describe randomized algorithms for sorting and selection based on these ideas.

Algorithm RQS:

Input: A set of numbers S.

Output: The elements of S sorted in increasing order.

1. Choose an element y uniformly at random from S: every element in S has equal probability of being chosen.
2. By comparing each element of S with y, determine the set S_1 of elements smaller than y and the set S_2 of elements larger than y.
3. Recursively sort S_1 and S_2. Output the sorted version of S_1, followed by y, and then the sorted version of S_2.

Algorithm RQS is an example of a randomized algorithm-an algorithm that makes random choices during execution. It is inspired by the Quicksort algorithm. We assume that the random choice in Step 1 can be made in unit time. What can we prove about the running time of RQS?

We now analyze the expected number of comparisons in an execution of RQS. Comparisons are performed in Step 2, in which we compare a randomly chosen element to the remaining elements. For $1 \leq i \leq n$, let $S_{(i)}$ denote the element of rank i (the ith smallest element) in the set S. Define the random variable X_{ij} to assume the value 1 if $S_{(i)}$ and $S_{(j)}$ are compared in an execution, and the value 0 otherwise. Thus, the total number of comparisons is

$$\sum_{i=1}^{n} \sum_{j>i} X_{ij}. \tag{8.9}$$

By linearity of expectation the expected number of comparisons is

$$E[\sum_{i=1}^{n} \sum_{j>1} X_{ij}] = \sum_{i=1}^{n} \sum_{j>1} E[X_{ij}]. \tag{8.10}$$

Exercises

1. We flip a fair coin ten times. Find the probability of the following events.
   (a) The number of heads and the number of tails are equal.
   (b) There are more heads than tails.
   (c) The i-th flip and the $(11 - i)$-th flip are the same for $i = 1, \ldots, 5$.
   (d) We flip at least four consecutive heads.
2. The following approach is often called reservoir sampling. Suppose we have a sequence of items passing one by one. We want to maintain a sample of one item with the property that it is uniformly distributed over all the items that we have seen so far. Assume that we do not know the total number of items in advance. How can we sample without storing all the items that we see.

Consider the following algorithms which just stores one item in memory at all times. When the first item appears, it is stored in memory. When the kth item appears it replaces the item in memory with probability $\frac{1}{k}$. Explain why this algorithm solves the problem.

3. Consider the following algorithm for computing the k-smallest of a set of n elements (k-median):

   Input: a set S of n different numbers and a number k between 1 and n

   Output: the k smallest element of S

   Algorithm: $Find(S, k)$

   (a) Choose y from S uniformly at random
   (b) $S_1 = \{x \in S \mid x < y\}; S_2 = \{x \in S \mid x > y\}$;
   (c) If $|S_1| \geq k$ then $Find(S_1, k)$
   (d) If $|S_1| = k - 1$ then $Output\ y$
   (e) If $|S_1| < k - 1$ then $Find(S_2, k - |S_1| - 1)$

   Show that the expected running time of the algorithm is $O(n)$. (Hint: Try to write the running time as a recurrence and solve it).

4. Given a circle with circumference 1 and a marked point on the circle. Choose n additional points on the circle uniformly and independently at random. The random points partition the circle into n intervals.

   - What is the average interval length?
   - What is the expected interval length of the interval containing x.

5. Consider the interval $[0, 1]$ on the real line. Choose a point of the interval uniformly at random. This point partitions the line into 2 intervals.

   - What is the expected length of the longer (shorter) interval?
   - What is the expected length of the interval containing the median (i.e., the point 0.5)

References

[1] Thomas H. Corman, et al., "Introduction to Algorithms, 3rd ed." *PHI*, 2009.

[2] Mikhail J. Atallah and Marina Blanton, "Algorithms and Theory of Computation handbook, Second Edition, Special Topics and Techniques", CRC Press, Taylor and Francis Group - A Chapman and Hall Book, 2010.

[3] Allen B. Tucker, Jr., "The computer Science and Engineering Handbook," *CRC Press*, 1997.