Basic probability: axioms, conditional probability, random variables, distributions

Application: Verifying Polynomial Identities

- Computers can make mistakes:
  - Incorrect programming
  - Hardware failures

➔ sometimes, use randomness to check output

- Example: we want to check a program that multiplies together monomials

E.g: $(x+1)(x-2)(x+3)(x-4)(x+5)(x-6) \mathrel{?}= x^6-7x^3+25$

  - In general check if $F(x) = G(X)$?

- One way is:
  - Write another program to re-compute the coefficients
  - That's not good: may goes same path and produces the same bug as in the first

How to use randomness

- Assume the max degree of F & G is d. Use this algorithm:
  - Pick a uniform random number from: $\{1,2,3, \ldots 100d\}$
  - Check if $F(r)=G(r)$ then output "equivalent", otherwise "non-equivalent"
- Note: this is much faster than the previous way – $O(d)$ vs. $O(d^2)$
- One-sided error:
  - "non-equivalent" always true
  - "equivalent" can be wrong
- How it can be wrong:
  - If accidentally picked up a root of $F(x)-G(x) = 0$
  - This can occur with probability at most 1/100

Axioms of probability

- We need a formal mathematical setting for analyzing the randomized space
  - Any probabilistic statement must refer to the underlying probability space
- Definition 1: A probability space has three components:
  - A sample space Ω, which is the set of all possible outcomes of the random process modeled by the probability space
  - A family of sets Φ representing the allowable events, where each set in Φ is a subset of the sample space and
  - A probability function Pr: $\Phi \rightarrow R$ satisfying definition 2 below

  An element of Ω is called a *simple* or *elementary* event
- In the randomized algo for verifying polynomial identities, the sample space is the set of integers {1,…100d}.
  - Each choice of an integer r in this range is a simple event

Axioms

- Def2: A probability function is any function Pr: $\Phi \rightarrow R$ that satisfies the following conditions:
  1. For any event E, $O \leq Pr(E) \leq 1$;
  2. $Pr(\Omega) = 1$; and
  3. For any sequence of pairwise mutually disjoint events $E_1, E_2, E_3 \ldots$,

$$Pr(\cup_{i\geq 1} E_i) = \Sigma_{i\geq 1} Pr(E_i)$$

  - events are sets ➔ use set notation to express event combination
- In the considered randomized algo:
  - Each choice of an integer r is a simple event.
  - All the simple events have equal probability
  - The sample space has 100d simple events, and the sum of the probabilities of all simple events must be 1 ➔ each simple event has probability 1/100d

Lemmas

- Lem1: For any two events E_1, E_2:

  $Pr(E_1 \cup E_2) = Pr(E_1) + Pr(E_2) - Pr(E_1 \cap E_2)$

- Lem2(Union bound): For any finite of countably infinite sequence of events $E_1, E_2, E_3 \ldots,$

  $Pr(\cup_{i\geq 1} E_i) \leq \Sigma_{i\geq 1} Pr(E_i)$

- Lem3(inclusion-exclusion principle) Let $E_1, E_2, E_3 \ldots$ be any n events. Then

  $Pr(\cup_{i=1,n} E_i) = \Sigma_{i=1,n} Pr(E_i) - \Sigma_{i<j} Pr(E_j \cap E_j) +$

  $\Sigma_{i<j<k} Pr(E_j \cap E_j \cap E_k) - \ldots$

  $+(-1)^{l+1} \Sigma_{i_1 \leq \ldots \leq i_r} Pr(\cap\ r_{=1,l} E_{i_r}) + \ldots$

Analysis of the considered algorithm

- The algo gives an incorrect answer if the random number it chooses is a root of polynomial F-G
- Let E represent the event that the algo failed to give the correct answer
  - The elements of the set corresponding to E are the roots of the polynomial F-G that are in the set of integer {1,...100d}
  - Since F-G has degree at most d then has no more than d roots ➔ E has at most d simple events
- Thus, Pr(algorithm fails) = Pr(E) $\leq$ d/(100d) = 1/100

How to improve the algo for smaller failure probability?

- Can increase the sample space
  - E.g. {1,..., 1000d}
- Repeat the algo multiple times, using different random values to test
  - If F(r)=G(r) for just one of these many rounds then output “non-equivalent”
- Can sample from {1,...100d} many times with or without replacements

Notion of independence

- Def3: Two events E and F are independent iff (if and only if)

  $$Pr(E \cap F) = Pr(E) \,.\, Pr(F)$$

More generally, events $E_1, E_2, \ldots, E_k$ are mutually independent iff for any subset $I \subseteq [1,k]$: $Pr(\cap_{i \in I} E_i) = \Pi_{i \in I} Pr(E_i)$

- Now for our algorithm samples with replacements
  - The choice in one iteration is independent from the choices in previous iterations
  - Let E_i be the event that the i*th* run of algo picks a root r_i s.t. $F(r_i)-G(r_i)=0$
  - The probability that the algo returns wrong answer is

  $Pr(E_1 \cap E_2 \cap \ldots \cap E_k) = \Pi_{i=1,k} Pr(E_i) \le \Pi_{i=1,k} (d/_{100d}) = (1/_{100})^k$
- Sampling without replacement:
  - The probability of choosing a given number *is conditioned on* the events of the previous iterations

Notion of conditional probability

- Def 4: The condition probability that event E occurs given that event F occurs is

$$\Pr(E|F) = {}^{\Pr(E\cap F)}/_{\Pr(F)}$$

  - Note this con. pro. only defined if Pr(F)>0
  - When E and F are independent and Pr(F)>0 then

$$\Pr(E|F) = {}^{\Pr(E\cap F)}/_{\Pr(F)} = {}^{\Pr(E).\Pr(F)}/_{\Pr(F)} = \Pr(E)$$

  - Intuitively, if two events are independent then information about one event should not affect the probability of the other event.

Sampling without replacement

- Again assume $F \neq G$
  - We repeat the algorithm k times: perform k iterations of random sampling from [1,...100d]
  - What is the prob that all k iterations yield roots of F-G, resulting in a wrong output by our algo?
  - Need to bound $\Pr(E_1 \cap E_2 \cap \ldots \cap E_k)$

$\Pr(E_1 \cap E_2 \cap \ldots \cap E_k) = \Pr(E_k | E_1 \cap \ldots \cap E_{k-1}) \,.\, \Pr(E_1 \cap E_2 \cap \ldots \cap E_{k-1})$

$= \Pr(E_1).\ \Pr(E_1|E_2).\ \Pr(\Pr(E_3|E_1 \cap E_2).\ \ldots\ \Pr(E_k|E_1 \cap \ldots \cap E_{k-1})$

- Need to bound $\Pr(E_j|E_1 \cap \ldots \cap E_{kj1})$: $\leq {}^{d-(j-1)}/_{100d-(j-1)}$

So $\Pr(E_1 \cap E_2 \cap \ldots \cap E_k) \leq \Pi_{j=1,k}\ {}^{d-(j-1)}/_{100d-(j-1)} \leq ({}^{1}/_{100})^k$, slightly better

- Use d+1 iterations: always give correct answer. Why? Efficient?

Random variables

- Def 5: A random variable X on a sample space Ω is a real-valued function on Ω; that is X: $\Omega \rightarrow R$. A discrete random variable is a random variable that takes on only finite or countably infinite number of values
  - So, "X=a" represents the set $\{s \in \Omega \mid X(s)=a\}$
  - $Pr(X=a) = \sum_{X(s)=a} Pr(s)$

Eg. Let X is the random variable representing the sum of the two dice. What is the prob of X=4?

Random variables

- Def6: Two random variables X and Y are independent iff for all values x and y:

  $\Pr((X=x)\cap(Y=y)) = \Pr(X=x) . \Pr(Y=y)$

Expectation

- Def 7: The expectation of a discrete random variable X, denoted by E[X] is given by $E[X] = \Sigma_i i\Pr(X=i)$
  - where the summation is over all values in range of X
  - E.g Compute the expectation of the random variable X representing the sum of two dice

Linearity of expectation

- Theorem:
  - $E[\Sigma_{i=1,n}X_i] = \Sigma_{i=1,n}E[X_i]$
  - $E[c\ X] = c\ E[X]$ for all constant c

Bernoulli and Binomial random variables

- Consider experiments that succeeds with probability p and fails with probability 1-p
  - Let Y be a random variable takes 1 if the experiment succeeds and 0 if otherwise. Called a Bernoulli or an indicator random variable
  - $E[Y] = p$
  - Now we want to count X, the number of success in n tries
- A binomial random variable X with parameters n and p, denoted by B(n,p), is defined by the following probability distribution on j=0,1,2,..., n:
  - $Pr(X=j) = (n \text{ choose } j)\, p^j (1-p)^{n-j}$
  - E.g. used a lot in sampling (book: Mit-Upfal)

The hiring problem

```
HIRE-ASSISTANT(n)
best ← 0
candidate 0 is a least-qualified dummy candidate
for i ← 1 to n
    do interview candidate i
        if candidate i is better than candidate best
          then best ← i
                 hire candidate i
```

Cost Analysis

- We are not concerned with the running time of HIRE-ASSISTANT, but instead with the cost incurred by interviewing and hiring.
- Interviewing has low cost, say c_i, whereas hiring is expensive, costing c_h. Let m be the number of people hired. Then the cost associated with this algorithm is $O(nc_i + mc_h)$. No matter how many people we hire, we always interview n candidates and thus always incur the cost nc_i, associated with interviewing.

Worst-case analysis

- In the worst case, we actually hire every candidate that we interview. This situation occurs if the candidates come in increasing order of quality, in which case we hire n times, for a total hiring cost of $O(nc_h)$.

Probabilistic analysis

- *Probabilistic analysis* is the use of probability in the analysis of problems. In order to perform a probabilistic analysis, we must use knowledge of the distribution of the inputs.
- For the hiring problem, we can assume that the applicants come in a random order.

Randomized algorithm

- We call an algorithm *randomized* if its behavior is determined not only by its input but also by values produced by a *random-number generator*.

Indicator random variables

The indicator random variable *I[A]* associated with event A is defined as

$$I[A] = \begin{cases} 1 & \text{if A occurs} \\ 0 & \text{if A does not occur} \end{cases}$$

- Lemma: Given a sample space Ω and an event A in the sample space Ω, let $X_A = I\{A\}$. Then $E[X_A] = Pr(A)$.

Analysis of the hiring problem using indicator random variables

- Let X be the random variable whose value equals the number of times we hire a new office assistant and X_i be the indicator random variable associated with the event in which the ith candidate is hired. Thus,

$$X=X_1+X_2+\ldots+X_n$$

By the lemma above, we have

$E[X_i]=\Pr\{\text{ candidate } i \text{ is hired}\}=1/i$. Thus,

$$E[X]=1+1/2+1/3+\ldots+1/n=\ln n+O(1)$$

Randomized algorithms

```
RANDOMIZED-HIRE-ASSISTANT(n)
randomly permute the list of candidate
best←0
for i←1 to n
    do interview candidate i
        if candidate i is better than candidate best
            then best←i
                hire candidate i
```

```
PERMUTE-BY-SORTING(A)
n ← length[A]
for i ← 1 to n
    do P[i] ← RANDOM(1, n^3)
sort A, using P as sort keys
return A
```

- *Lemma:* Procedure PERMUTE-BY-SORTING produces a uniform random permutation of input, assuming that all priorities are distinct.

RANDOMIZE-IN-PLACE(A)

1 $n \leftarrow length[A]$

2 for $i \leftarrow 1$ to n

3 do swap $A[i] \longleftrightarrow A[\text{RANDOM}(i,n)]$

- Lemma: Procedure RANDOMIZE-IN-PLACE computes a uniform random permutation.