Probabilistic Analysis and Randomized Algorithms

Geoffrey Matthews

April 25, 2018

Goals

- Present difference between probabilistic analysis and randomized algorithms.
- Present technique of indicator random variables.
- Analysis of randomized algorithm.

The Hiring Problem

- You are using an employment agency to hire a new office assistant.
- The agency sends you one candidate each day.
- You interview the candidate and must immediately decide whether or not to hire that person and fire the current one.
- Cost to interview is c_i per candidate Cost to hire is c_h per candidate.
- Assume that $c_h > c_i$.
- You are committed to always have the best candidate seen so far.
- **Goal:** Determine what the price of this strategy will be.

Hire-Assistant

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

Costs

If there are n candidates and we hire m of them, cost is

$$O(nc_i + mc_h)$$

- Have to pay nc_i no matter what.
- Focus on mc_h.
- mc_h depends on the order of candidates.
- This is a common scenario.

Worst-case analysis

Worst-case analysis

- Candidates are sorted worst to best.
- We hire all candidates.
- Cost is
$$O(nc_i + nc_h) = O(nc_h)$$

Probabilistic analysis

- In general we have no control over the order.
- We could assume candidates come in random order.
- Assign a rank to reach candidate: $\text{rank}(i) \in \{1, 2, ..., n\}$. No ties.
- The list $(\text{rank}(1), \text{rank}(2), ...\text{rank}(n))$ is a permutation of $(1, 2, ..., n)$.
- The list of ranks is equally likely to be any one of the $n!$ permutations.
- The ranks form a **uniform random permutation**.

Problem of probabilistic analysis

- We must use knowledge of the distribution of inputs, or make assumptions about it.
- The expectation is over this distribution.
- The technique requires that we can make reasonable assumptions about the input.
- Also that we can successfully model the presumed input distribution.

Randomized algorithms

- We might not know the input distribution, or be able to model it.
- Instead, we randomize within the algorithm to impose a distribution.

Randomized-Hire-Assistant

Change the scenario:

- The employment agency sends us a list of all candidates in advance.
- On each day, we randomly choose a candidate from the list.
- Instead of relying on the input distribution, we impose a uniform random one.

What makes an algorithm randomized

- An algorithm is **randomized** if its behavior is determined in part by values produced by a **random-number generator**.
- `RANDOM(a,b)` returns an integer r, where $a \leq r \leq b$ and each of the $b - a + 1$ possible values of r is equally likely.
- In practice, `RANDOM` is implemented by a **pseudorandom-number generator**, which is a deterministic method returning numbers that "look" random and pass statistical tests.

Indicator random variables

- A simple yet powerful technique for computing the expected value of a random variable.
- Helpful in situations in which there may be dependence.
- Given a sample space and an event A, we define the indicator random variable

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

- **Lemma**
  For an event A, let $X_A = I\{A\}$. Then $E[X_A] = \Pr\{A\}$.
- **Proof**
  Letting $\overline{A}$ be the complement of A, we have

$$\begin{aligned} E[X_A] &= E[I\{A\}] \\ &= 1 \cdot \Pr\{A\} + 0 \cdot \Pr\{\overline{A}\} \\ &= \Pr\{A\} \end{aligned}$$

Simple example

- Determine expected number of heads if we flip a fair coin.
- Sample space: $\{H, T\}$
- $\Pr\{H\} = \Pr\{T\} = 1/2$
- $X_H = I\{H\}$.
- X_H counts number of heads in one flip.
- Since $\Pr\{H\} = 1/2$, lemma says $E[X_H] = 1/2$.

More complicated example

- Expected number of heads in n flips. Let X be a random variable for number of heads in n flips.
- $$E[X] = \sum_{k=0}^{n} k \cdot \Pr\{X = k\}$$
- Instead, define $X_i = I\{\text{the } i\text{th flip is } H\}$, so $X = \sum_{i=1}^{n} X_i$
- Lemma says $E[X_i] = \Pr\{H\} = 1/2$.

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

The hiring problem analysis

- Assume canddiates arrive in random order.
- Let X be the RV that is the number of times we hire someone.
- Define $X_i = I\{\text{candidate } i \text{ is hired}\}$
- Candidate i is hired iff i is better than $1, 2, \ldots, i-1$.
- $\Pr\{\text{candidate } i \text{ is best so far}\} = 1/i$
- $E[X_i] = 1/i$.

$$
\begin{aligned}
E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\
&= \sum_{i=1}^{n} E[X_i] \\
&= \sum_{i=1}^{n} 1/i \\
&= O(\lg n)
\end{aligned}
$$

The hiring problem

- The algorithm is deterministic:
  For any given input, the number we hire is always the same.
- The number of times we hire a new office assistant depends only on the input.
- In fact, it depends only on the ordering of the candidates ranks that it is given.
- Some rank orderings will always produce a high hiring cost. (Sorted by increasing quality.)
- Some will always produce a low hiring cost.
  (Any where the best candidate is first.)
- Some may be in between.

Randomizing the hiring problem

RANDOMIZED-HIRE-ASSISTANT(n)

randomly permute the list of candidates
HIRE-ASSISTANT(n)

- The randomization is now in the algorithm, not in the input distribution.
- Given a particular input, we can no longer say what its hiring cost will be. Each time we run the algorithm, we can get a different hiring cost.
- In other words, each time we run the algorithm, the execution depends on the random choices made.
- No particular input always elicits worst-case behavior.
- Bad behavior occurs only if we get "unlucky" numbers from the randomnumber generator.

Randomizing the hiring problem

RANDOMIZED-HIRE-ASSISTANT(n)
  randomly permute the list of candidates
  HIRE-ASSISTANT(n)

- The expected hiring cost is $O(c_h \lg n)$, regardless of input.

Randomly permuting an array

RANDOMIZE-IN-PLACE(A, n)

for $i = 1$ **to** n

  swap $A[i]$ with $A[\text{RANDOM}(i, n)]$

- **Goal:** Produce a uniform random permutation. (Each of the $n!$ permutations is equally likely.)
- In iteration i, choose $A[i]$ randomly from $A[i..n]$.
- Will never alter $A[i]$ after iteration i.
- $O(1)$ per iteration, so $O(n)$.

k-permutations

- Given a set of n elements, a **k-permuation** is a sequence containing k of the n elements.
- There are $n!/(n-k)!$ possible k-permutations.

Algorithm computes a uniform random permutation

RANDOMIZE-IN-PLACE(A, n)

for $i = 1$ **to** n

    swap $A[i]$ with $A[\text{RANDOM}(i, n)]$

- **Loop invariant:** Just prior to the ith iteration, for each possible $(i-1)$-permutation, $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$.

Algorithm computes a uniform random permutation

```
RANDOMIZE-IN-PLACE(A, n)
  for i = 1 to n
      swap A[i] with A[RANDOM(i, n)]
```

- **Loop invariant:** Just prior to the ith iteration, for each possible $(i-1)$-permutation, $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$.

- **Initialization:** Just before iteration 1, $A[1..0]$ contains the 0-permutation with probability 1.

Algorithm computes a uniform random permutation

RANDOMIZE-IN-PLACE(A, n)

for $i = 1$ **to** n
  swap $A[i]$ with $A[\text{RANDOM}(i, n)]$

- **Loop invariant:** Just prior to the ith iteration, for each possible $(i-1)$-permutation, $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$.
- **Maintenance:**

Algorithm computes a uniform random permutation

RANDOMIZE-IN-PLACE(A, n)

for $i = 1$ **to** n
    swap $A[i]$ with $A[\text{RANDOM}(i, n)]$

- **Loop invariant:** Just prior to the ith iteration, for each possible $(i-1)$-permutation, $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$.
- **Maintenance:**
  - Consider a i-permutation $\pi = (x_1, x_x, ...x_i)$.
  - It consists of $\pi' = (x_1, x_x, ...x_{i-1})$ followed by x_1.
  - Let E_1 be the event that π' is in $A[1..i-1]$.
  - Let E_2 be the event that x_i is put into $A[i]$.

$$\begin{aligned}
\Pr\{E_2 \cap E_1\} &= \Pr\{E_2 | E_1\} \Pr\{E_1\} \\
&= \frac{1}{n-i+1} \cdot \frac{(n-i+1)!}{n!} \\
&= \frac{(n-i)!}{n!}
\end{aligned}$$

Algorithm computes a uniform random permutation

RANDOMIZE-IN-PLACE(A, n)

for $i = 1$ **to** n
    swap $A[i]$ with $A[\text{RANDOM}(i, n)]$

- **Loop invariant:** Just prior to the ith iteration, for each possible $(i-1)$-permutation, $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$.
- **Termination:**

Algorithm computes a uniform random permutation

RANDOMIZE-IN-PLACE(A, n)

for $i = 1$ **to** n
  swap $A[i]$ with $A[\text{RANDOM}(i, n)]$

- **Loop invariant:** Just prior to the ith iteration, for each possible $(i-1)$-permutation, $A[1..i-1]$ contains this $(i-1)$-permutation with probability $(n-i+1)!/n!$.
- **Termination:**
- At termination, $i = n+1$, so $A[1..n]$ is a given n-permutation with probability

$$\frac{(n-n)!}{n!} = \frac{1}{n!}$$

The birthday paradox

- How many people must be in a room before there is a 50% chance of two of them having the same birthday? Assumptions:
- $$\Pr\{b_i = r\} = 1/n \text{ for } i = 1..k \text{ and } r = 1..n$$
- $$\begin{aligned}\Pr\{b_i = r \text{ and } b_j = r\} &= \Pr\{b_i = r\}\Pr\{b_j = r\} \\ &= 1/n^2\end{aligned}$$
- Hence