# 2 Statistical Principles

We will study two phenomenon of random processes that are quite important, but possibly unintuitive. The goal will be to explore, formalize, and hopefully make intuitive these phenomenon.

- **Birthday Paradox:** To measure the expected collision of random events.

  *A random group of 23 people has about a 50% chance of having a pair with the same birthday.*

- **Coupon Collectors:** To measure the expectation for seeing all possible outcomes of a discrete random variable.

  *Consider a lottery where on each trial you receive one of $n$ possible coupons at random. It takes in expectation about $n(0.577 + \ln n)$ trials to collect them all.*

From another perspective, these describe the effects of random variation. The first describes *collision* events and the second *covering* events. Next lecture we will explore how long it takes for these events to roughly evenly distribute.

**Model.** For all settings, there is a common model of random elements drawn from a discrete universe. The universe has $n$ possible objects; we represent this as $[n]$ and let $i \in [n]$ represent one element (indexed by $i$ from $\{1, 2, 3, \ldots, n\}$) in this universe. The $n$ objects may be IP addresses, days of the year, words in a dictionary, but for notational and implementation simplicity, we can always have each element (IP address, day, word) map to a distinct integer $i$ where $0 < i \leq n$. Then we study the properties of drawing $k$ items uniformly at random from $[n]$ with replacement.

## 2.1 (Random) Hash Functions

A key tool, perhaps *the* tool, in probabilistic algorithms is a hash function. There are two main types of hash functions. The second (which we define in Lecture 6) is locality-preserving. But often one wants a hash functions which is uncorrelated, like in a hash table data structure.

A *random hash function* $h \in \mathcal{H}$ maps from one set $\mathcal{A}$ to another $\mathcal{B}$ (usually $\mathcal{B} = [m]$) so that conditioned on the random choice $h \in \mathcal{H}$, the location $h(x) \in \mathcal{B}$ for any $x \in \mathcal{A}$ is equally likely. And for any two (or more strongly, but harder to achieve, *all*) $x \neq x' \in \mathcal{A}$ the $h(x)$ and $h(x')$ are *independent* (conditioned on the random choice of $h \in \mathcal{H}$). It is important to state that for a fixed $h \in \mathcal{H}$, the map $h(x)$ is *deterministic*, so no matter how many times we call $h$ on argument $x$, it always returns the same result. The full independence requirement is often hard to achieve in theory, but in practice are often not a major problem. For the purposes of this class, most built-in hash functions should work sufficiently well.

Assume we want to construct a hash function $h : \Sigma^k \to [m]$ where $m$ is a power of two (so we can represent it as $\log_2 m$ bits) and $\Sigma^k$ is a string of $k$ of characters (lets say a string of numbers $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$). So $x \in \Sigma^k$ can also be interpreted as a $k$-digit number.

1. **SHA-1:** See `http://en.wikipedia.org/wiki/SHA-1`
   This secure hash function takes in a string of bits so $\Sigma = \{0, 1\}$ and $k$ is the number of bits (but many front ends exists that can let $\Sigma$ be all ASCII characters), and deterministically outputs a set of 160 bits, so $m = 2^{160}$.

   If one desired a smaller value of $m$, one can simply use any consistent subset of the bits. To create a family of hash functions $\mathcal{H}$ parameterized by some seed $a$, one can simply concatenate $a$ to all inputs, to obtain a hash $h_a \in \mathcal{H}$.

Functionality for this should be built in or available as standard packages for many programming languages.

2. **Multiplicative Hashing:** $h_a(x) = \lfloor m \cdot \mathsf{frac}(x * a) \rfloor$
   $a$ is a real number (it should be large with binary representation a good mix of 0s and 1s), $\mathsf{frac}(\cdot)$ takes the fractional part of a number, e.g. $\mathsf{frac}(15.234) = 0.234$, and $\lfloor \cdot \rfloor$ takes the integer part of a number, rounding down so $\lfloor 15.234 \rfloor = 15$. Can sometimes be more efficiently implemented as $(xa/2^q) \mod m$ where $q$ is essentially replacing the $\mathsf{frac}(\cdot)$ operation and determining the number of bits precision.

   If you want something simple and fast to implement yourself, this is a fine choice.

3. **Modular Hashing:** $h(x) = x \mod m$
   This roughly evenly distributed numbers, but numbers that are both powers of $m$ will *always* hash to the same location. We can use a random large prime $m' < m$. This will leave some bin always empty, but has less regularity.

   This is **not recommended**, but is a common first approach. It is listed here to advise against it.

## 2.2 Birthday Paradox

First, let us consider the famous situation of birthdays. Lets make formal the setting. Consider a room of $k$ people, chosen at random from the population, and assume each person is equally likely to have any birthday (excluding February 29th), so there are $n = 365$ possible birthdays.

The probability that any two (i.e. $k = 2$) people (ALICE and BOB) have the same birthday is $1/n = 1/365 \approx 0.003$. The birthday of ALICE could be anything, but once it is known by ALICE, then BOB has probability $1/365$ of matching it.

To measure that at least one pair of people have the same birthday, it is easier to measure the probability that no pair is the same. For $k = 2$ the answer is $1 - 1/n$ and for $n = 365$ that is about $0.997$.

For a general number $k$ (say $k = 23$) there are $\binom{k}{2} = k \cdot (k-1)/2$ (read as $k$ *choose* 2) pairs. For $k = 23$, then $\binom{23}{2} = 253$. Note that $\binom{k}{2} = \Theta(k^2)$.

We need for each of these events that the birthdays do not match. Assuming independence we have

$$(1 - 1/n)^{\binom{k}{2}} \quad \text{or} \quad 0.997^{253} = 0.467.$$

And the probability there is a match is thus 1 minus this number

$$1 - (1 - 1/n)^{\binom{k}{2}} \quad \text{or} \quad 1 - 0.997^{253} = 0.532,$$

just over 50%.

#### What are the problems with this?

- First, the birthdays may not be independently distributed. More people are born in spring. There may be non-negligible occurrence of twins.

  Sometimes this is really a problem, but often it is negligible. Other times this analysis will describe an algorithm we create, and we can control independence.

- Second, what happens when $k = n+1$, then we should always have some pair with the same birthday. But for $k = 366$ and $n = 365$ then

$$1 - (1 - 1/n)^{\binom{k}{2}} = 1 - (364/365)^{\binom{366}{2}} = 1 - (0.997)^{66795} = 1 - 7 \times 10^{-88} < 1.$$

Yes, it is very small, but it is less than 1, and hence must be wrong.

Really, the probability should be

$$1 - \left(\frac{n-1}{n}\right) \cdot \left(\frac{n-2}{n}\right) \cdot \left(\frac{n-3}{n}\right) \cdot \ldots = 1 - \prod_{i=1}^{k-1} \left(\frac{n-i}{n}\right).$$

Inductively, in the first round (the second person ($i = 2$) there is a $(n-1)/n$ chance of having no collision. If this is true, we can go to the next round, where there are then two distinct items seen, and so the third person has $(n-2)/n$ chance of having a distinct birthday. In general, inductively, after the $i$th round, there is an $(n-i)/n$ chance of no collision (if there were no collisions already), since there are $i$ distinct events already witnessed.

As a simple sanity check, in the $(n+1)$th term $(n-n)/(n) = 0/n = 0$; thus the probability of some collision of birthdays is $1 - 0 = 1$.

**Take away message.**

- There are collisions in random data!
- More precisely, if you have $n$ equi-probability random events, then expect after about $k = \sqrt{2n}$ events to get a collision. Note $\sqrt{2 \cdot 365} \approx 27$, a bit more than 23.

  Note that $(1 + \frac{\alpha}{t})^t \approx e^\alpha$ for large enough $t$. So setting $k = \sqrt{2n}$ then

  $$1 - (1 - 1/n)^{\binom{k}{2}} \approx 1 - (1 - 1/n)^n \approx 1 - e^{-1} \approx .63$$

  This is not exactly $1/2$, and we used a bunch of $\approx$ tricks, but it shows *roughly* what happens.
- This is fairly accurate, but has noticeable variance. Note for $n = 365$ and $k = 18$ then

  $$1 - (1 - 1/n)^{\binom{k}{2}} = 1 - (364/365)^{153} \approx .34$$

  and when $k = 28$ then

  $$1 - (1 - 1/n)^{\binom{k}{2}} = 1 - (364/365)^{378} \approx .64.$$

  This means that if you keep adding (random) people to the room, the first matching of birthdays happens $30\%$ $(= 64\% - 34\%)$ of the time between the 18th and 28th person. When $k = 50$ people are in the room, then

  $$1 - (1 - 1/n)^{\binom{k}{2}} = 1 - (364/365)^{1225} \approx .965,$$

  and so only about $3.5\%$ percent of the time are there no pair with the same birthday.

## 2.3 Coupon Collectors

Lets now formalize the famous coupon lottery. There are $n$ types of coupons, and we participate in a series of independent trials, and on each trial we have equal probability $(1/n)$ of getting each coupon. *We want to collect all toys available in a McDonald's Happy Meal.* How many trials ($k$) should we expect to partake in before we collect all coupons?

Let $r_i$ be the expected number of trials we need to take before receiving exactly $i$ *distinct* coupons. Let $r_0 = 0$, and set $t_i = r_i - r_{i-1}$ to measure the expected number of trials between getting $i - 1$ distinct coupons and $i$ distinct coupons.

Clearly, $r_1 = t_1 = 1$, and it has no variance. Our first trials always yields a new coupon.

Then the expected number of trials to get all coupons is $T = \sum_{i=1}^{n} t_i$.

To measure $t_i$ we will define $p_i$ as the probability that we get a new coupon after already having $i-1$ distinct coupons. Thus $t_i = 1/p_i$. And $p_i = (n-i+1)/n$.

We are now ready for some algebra:

$$T = \sum_{i=1}^{n} t_i = \sum_{i=1}^{n} \frac{n}{n-i+1} = n \sum_{i=1}^{n} \frac{1}{i}.$$

Now we just need to bound the quantity $\sum_{i=1}^{n}(1/i)$. This is known at the $n$th *Harmonic Number* $H_n$. It is known that $H_n = \gamma + \ln n + o(1/n)$ where $\ln(\cdot)$ is the natural log (that is $\ln e = 1$) and $\gamma \approx 0.577$ is the *Euler-Masheroni constant*. Thus we need, in expectation,

$$k = T = nH_n = n(\gamma + \ln n)$$

trials to obtain all distinct coupons.

**Extensions.**

- What if some coupons are more likely than others. *McDonalds offers three toys: Alvin, Simon, and Theodore, and for every 10 toys, there are 6 Alvins, 3 Simons, and 1 Theodore.* How many trials do we expect before we collect them all?

  In this case, there are $n = 3$ probabilities $\{p_1 = 6/10, p_2 = 3/10, p_3 = 1/10\}$ so that $\sum_{i=1}^{n} p_i = 1$.

  The analysis and tight bounds here is a bit more complicated, but the key insight is that it is dominated by the smallest probability event. Let $p^* = \min_i p_i$. Then we need about

  $$k \approx \left(\frac{1}{p^*}\right)(\gamma + \ln n)$$

  random trials to obtain all coupons.

- These properties can be generalized to a family of events from a continuous domain. Here there can be events with arbitrarily small probability of occurring, and so the number of trials we need to get all events becomes arbitrarily large (following the above non-uniform analysis). So typically we set some probability $\varepsilon \in [0, 1]$. (Typically we consider $\varepsilon$ as something like $\{0.01, 0.001\}$ so $1/\varepsilon$ something like $\{100, 1000\}$. Now we want to consider any set of events with combined probability greater than $\varepsilon$. (We can't consider all such subsets, but we can restrict to all, say, contiguous sets – intervals if the events have a natural ordering). Then we need

  $$k \approx \frac{1}{\varepsilon} \log \frac{1}{\varepsilon}$$

  random trials to have at least one random trial in any subset with probability at least $\varepsilon$. Such a set is called an *$\varepsilon$-net*.

**Take away message.**

- It takes about $n \ln n$ trials to get all items at random from a set of size $n$, not $n$. That is we need an extra about $\ln n$ factor to probabilistically guarantee we hit all events.
- When probability are not equal, it is the smallest probability item that dominates everything!
- To hit all (nicely shaped) regions of size $\varepsilon n$ we need about $(1/\varepsilon)\log(1/\varepsilon)$ samples, even if they can be covered by $1/\varepsilon$ items.