The Birthday Problem and Hash Collisions

Brian Powell
(Dated: August 5, 2015)

Birthday Problem

In flipping through a book on American history, you come to notice that two of our 44 presidents, Warren G. Harding and James K. Polk, were born on November 2. An acquaintance with British history would reveal that, of 52 prime ministers, two pairs share the same birthday. And similarly for Australia – of its 27 prime ministers, the 1st and the 24th were born on the same day. Something might be said for the peculiar birthday clustering of heads of state, if this wasn't also true for hockey teams (head coaches of the Montreal Canadiens, for one) or Academy Award-winning directors. These are relatively small groups of people – 44, 52, 27 – with at least a few members that share 1 out of 365 different birthdays. What's going on here?

Consider the following question: How large must a group of people be for there to be a 50% or greater chance that at least two of them have the same birthday? The answer, as we will see, is 23. With 57 people, we are almost guaranteed (99% probability) to find a match. These numbers are sufficiently low to challenge intuition; for this reason, this problem is also known as the Birthday paradox, although it is not a paradox in the logical sense. And it only seems paradoxical at first blush – we can make sense of it if we consider it with care. First, let's see why this result evades our naive expectation. It has to do with the fact that we are likely unconsciously mulling a *different* problem: How large must a group of people be for there to be a 50% or greater chance of at least two of them to share a *given* birthday? Say, your birthday?

We can answer this question without much trouble. We will call the probability of finding a match p, but we will consider instead the probability that there *is not* a match: $\bar{p} = 1 - p$. Assuming that all birthdays in the year are equiprobable, we find

$$\bar{p} = \left(\frac{364}{365}\right)^n \tag{1}$$

for the probability that none of a group of n people share your birthday. With $\bar{p} = p = 0.5$, there must be $n = 253$ people. Now this is about large enough to seem plausible. As we noted though, this is a different problem. We are not interested in matching a specific birthday, but instead matching *any* birthday. This too isn't difficult, with [1]

$$\bar{p} = \left(\frac{365}{365}\right)\left(\frac{364}{365}\right)\left(\frac{363}{365}\right)\cdots\left(\frac{365-n+1}{365}\right) = \frac{365!}{(365-n)!}\frac{1}{365^n} \tag{4}$$

giving the probability that no two out of n people share a birthday. Note that this probability is zero when $n > 365$, since then there *must* be a match (this eventuality is sometimes referred to as the

[1] We can also find p by summing the individual probabilities that exactly 2 people, exactly 3 people, and so on, share birthdays. For $k = 2$ people, we have

$$p(k=2) = \frac{1}{365^2}\frac{1}{364^{n-2}} \cdot 364 \cdot 363 \cdots (364-n-3) = \frac{1}{365^2}\frac{1}{364^{n-2}}\frac{364!}{(364-n-2)!} \tag{2}$$

Summing all cases, we have

$$p = \sum_{k=2}^{\infty} \frac{1}{365^k}\frac{1}{364^{n-k}}\frac{364!}{(364-n-k)!} \tag{3}$$

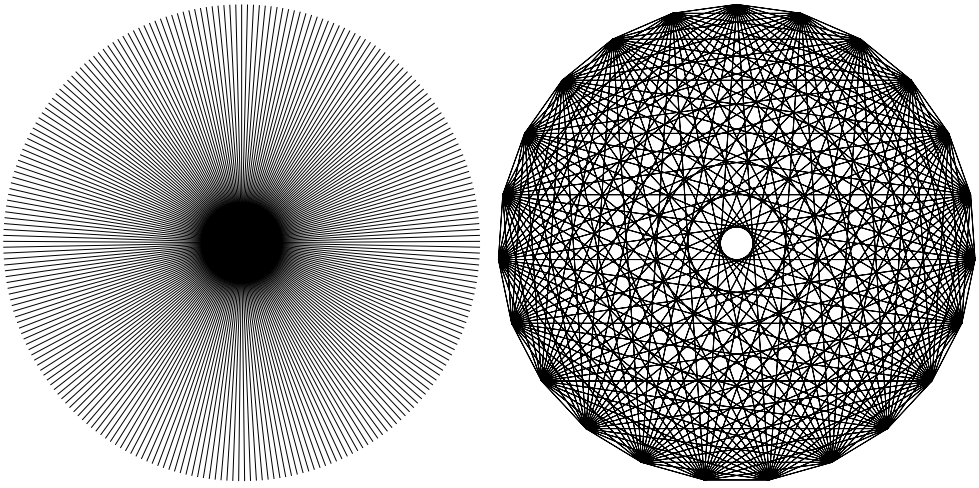
which is admittedly less wieldly than Eq. (5). It is nice, though, that the above infinite series sums to the closed form expression Eq. (5).

pigeonhole principle). Then,

$$p = 1 - \bar{p} = 1 - \binom{365}{n} \frac{n!}{365^n}. \tag{5}$$

The quantity $\binom{365}{n} n!$ gives the number of ways n distinct objects can be selected from a set of 365, without replacement, when the order is relevant. Solving Eq. (5) with $p = 0.5$ gives $n = 23$ people. We have succeeded in demonstrating the truth of the Birthday problem, but how are we to interpret it?

Consider that the number of different pairs that can be formed from a group of 23 people is $\binom{23}{2} = 253$. We've just seen this number – it's the number of people that we need to sample in order to have a 50% chance of finding one that shares our birthday. And so it's not the number of *people* that matter – it's the number of *comparisons*. In the first case that we analyzed, we were looking to match a given birthday. We found that 253 comparisons would be needed to furnish a 50% chance of a match, and since we are singling out one birthday, we must compare one person with 253 others. The next case, that addressed in the Birthday problem, does not fix the birthday. We still need 253 comparisons to find a match, but since any birthday will do, these 253 comparisons are distributed among the sample group of 23 people. Each person is compared against 22 others, and there are 23 such sets of comparisons. After dividing by two to account for double-counting (it's the same whether we compare Bob to Alice or Alice to Bob), we find $23 \times 22/2 = \binom{23}{2} = 253$.



(a) (b)

FIG. 1. (a) Topology of the first case considered, in which a specific birthday (center) is compared against 253 others (points around the circumference). (b) Topology of the case considered in the Birthday problem, in which the 253 comparisons are distributed among 23 people (points around the circumference). In both cases 253 comparisons are made.

Hash Collisions

The birthday problem has relevance to the security of hashing protocols. A hash function, h, is an algorithm that takes an arbitrary length input string (pre-image), m, and creates a fixed-sized output (image), $y = h(m)$. Cryptographic hash functions should be effectively *one-way*, in that given a hash, y, it should be computationally infeasible to find a pre-image, m, such that $y = h(m)$; this is known as *pre-image resistance*. This is effectively a statement about the infeasibility of "inverting" the hash. Hash functions should also be resistant to *collisions*, of which there are a few types:

Weak collision: Given a message m_1, another message, m_2, is found such that $h(m_1) = h(m_2)$. This is also known as a second-pre-image attack.

Strong collision: Any two messages, m_1 and m_2, are found that hash to the same value, $h(m_1) = h(m_2)$.

Chosen-prefix collision: Given two different prefixes, p_1 and p_2, messages m_1 and m_2 can be found such that their respective concatenations satisfy $h(p_1||m_1) = h(p_2||m_2)$.

Interestingly, the "one-way-ness" and collision resistance of a hash function are in tension: while a many-to-one function is hard to invert, the chance of collisions increases along with the number of messages that hash to the same image.

A weak collision was the subject of the first case that we analyzed: we had a fixed birthday that we were trying to match. Modern hash functions, however, have image spaces that are much larger than 365; for example, the hash algorithm MD5 produces a 128-bit hash, with $2^{128} = 3.4 \times 10^{38}$ possible values. For an N-bit hash function, one has a probability p of finding a weak collision after hashing n trial messages, where

$$n = \frac{\ln(1-p)}{\ln(1-2^{-N})}. \tag{6}$$

For a 128-bit hash, we need $n = 2.36 \times 10^{38}$ trials[2] to have a 50% chance of finding a weak collision, and so in general, an N-bit hash requires $\mathcal{O}(2^N)$ comparisons. As expected, this is of the same order of effort required to exhaust (brute-force) the full space of possibilities.

What about a strong collision? This is where the Birthday problem comes in: we don't care here what the matching hashes are – just that we find a match. The probability of finding a collision is given by Eq. (5) with 365 replaced with 2^N for an N-bit hash. We can find an approximation for the probability of finding a collision, p, as a function of the number of trials, by making some clever adjustments to the N-bit hash version of Eq. (4). First, we recast the product of fractions as

$$1 \times \left(\frac{2^N-1}{2^N}\right) \times \cdots \times \left(\frac{2^N-n+1}{2^N}\right) = \prod_{k=0}^{n-1} 1 - \frac{k}{2^N}. \tag{7}$$

For small k, the fraction $x = k/2^N \ll 1$, and so $1 - x \approx \exp(-x)$ and,

$$\begin{aligned} p \approx 1 - \prod_{k=0}^{n-1} \exp\left(-\frac{k}{2^N}\right) &= 1 - \exp\left(-\sum_{k=0}^{n-1} \frac{k}{2^N}\right) \\ &\approx 1 - \exp\left[-\frac{(n-1)n}{2^{N+1}}\right], \end{aligned} \tag{8}$$

where we have written the sum $\sum_{k=0}^{n-1} k = (n-1)n/2$. We can now obtain the number of trials needed to find a weak collision as a function of probability

$$n \approx \sqrt{-2\ln(1-p)}\,2^{N/2}. \tag{9}$$

For $p = 0.5$, the coefficient of $2^{N/2}$ is close to 1, with the result that for an N-bit hash only $2^{N/2}$ messages need to be sampled to have a 50% chance of finding a strong collision. [3] All of this assumes

[2] Note that a calculation of the difference $1 - 2^{-N}$ requires $\mathcal{O}(N)$-precision.

[3] An alternative, more boneheaded way to arrive at this result (and the way I originally derived it), is to use Stirling's approximation to get rid of the factorial terms: $\ln n! = n\ln n - n + \mathcal{O}(\ln n)$. This approximation is actually very good for large n, which is right where we wish to apply it. One finds,

$$p = 1 - \left(\frac{2^N}{n-2^N}\right)^x \exp(-n) \approx 1 - \exp\left[-\frac{(2n-1)n}{2^{N+1}}\right]$$

where $x = 2^N - n + 1/2$. This gives $n = \sqrt{\ln(1-p)}\,2^{N/2}$, which happens to be off by a factor of $\sqrt{2}$ from Eq. (9).

that the hash function is ideal – that it is a random mapping from the space of messages to the set of possible output values.[4] No real hash function, no matter how good, is truly random in this sense, and so the results we are obtaining here represent the greatest security we can expect out of a hash function. Any weakness of a real hash function might be leveraged to find a collision before the inherent limitation imposed by the Birthday problem is reached. For this reason, a hash function is said to be cryptographically *broken* if an attack exists that succeeds within fewer than $\mathcal{O}(2^N)$ rounds for a weak collision and $\mathcal{O}(2^{N/2})$ for a strong collision.

All of that said, for a 128-bit hash, we need at most $\mathcal{O}(10^{19})$ trials to have a good chance of finding a strong collision. How long will this take? It depends on the size of the message and the particulars of the hash function. For example, baseline implementations of MD5 can process around 300 MB/s on a modern 64 bit 1.8 GHz CPU [1], and the use of a GPU can increase this rate by a factor of 25 or so [2]. Suppose that we are seeking strong collisions of a 128-bit hash with the speed of MD5 among 1 KB text files: on a modern CPU, we'd be able to hash around 300,000 such messages per second. At this rate, it would take us a million years to have a 50% chance of finding a strong collision; a GPU would reduce this to tens of thousands years. Either way, this is infeasible. Of course, one could throw multiple GPU's at the problem: with a 100,000 GPU cluster, the time can be reduced to about half a year. A 100,000 GPU cluster is a thing of nations, not organized crime or professional hackers.

In summary, we have obtained limits on the numbers of trials necessary for having a good ($p = 50\%$) chance of finding weak and strong collisions of an ideal N-bit hash function: the number of trials needed increases exponentially with N for weak and $N/2$ for strong collisions. In designing a hash function, the specification should be based on the desired level of security; for example, the integrity of a certain document might need to be guaranteed for 50 years. Since the results obtained here apply to ideal functions, they should be interpreted as setting only upper limits on the computational resources required to perpetrate a successful attack. If the security requirement is a time limit as in the above example, we can vary not just the size of the hash but also the speed of the function. As we saw above, a perfect hash function with the size and speed specifications of MD5 will not succumb to a collision attack in a practical amount of time. However, an ideal 64-bit hash function with the same performance as MD5 will yield a strong collision in around 4 hours on a single CPU – easily achievable by unsophisticated attackers, especially given that a real hash function might have flaws that further reduce the required effort.

---

[1] `http://www.zorinaq.com/papers/md5-amd64.html`
[2] `http://www.golubev.com/about_cpu_and_gpu_2_en.htm`

---

[4] Keep in mind that we aren't randomly sampling the hash values directly, as we did for birthdays in the Birthday problem. In reality, we sample messages that we then hash; that an arbitrary sampling of messages implies a random "sampling" of hashes is an assumption only true for ideal hash functions.