BPP and the Chernoff Bound

In computability theory, a **decision problem** is any problem whose solution is a yes or a no (or a true/false or a zero/one or a whatever/whomever.) Examples include the following:

- Given two integers x and y, does x divide y?
- Given a computer program and an input to that computer program, will that program get stuck in an infinite loop?
- Is it actually the case that all true statements are provably true?
- Is my cat living a happy and fulfilled life?

Most problems which aren't binary in this way can be re-expressed in terms of a finite set of problems which are. For example, to solve the problem of finding the factors of a given number, we can simply ask whether or not every number up to that is a factor, and return the list of numbers corresponding to the 'yes' outputs of those decision problems.

Suppose we have a computer program (that is, a machine capable of executing sets of instructions) which solves a decision problem for us. Mathematically, what we really have is a function:

$$f : \mathbb{N} \to \{0, 1\} \tag{1}$$

That is to say, the computer program f takes in a positive integer (really a binary string) as an input, and outputs a 0 if the answer to the decision problem is a no and a 1 if the answer is yes. Really, f will usually answer an entire family of decision problems, based on the input. For example, if we are solving the decision problem of whether or not a given number has a non-trivial factor, then $f(n)$ will give us an answer for whatever n we plug in. A question such as "Will we ever find true happiness?" is not really a family of yes or no questions in the same way, but we can still view a computer program which answers it (supposing one existed) for us as a function of numbers, by simply taking it to be constant for all inputs. It will just equal the answer to the question, and it will be this same answer for all integer inputs.

The question of which problems are solvable by a computer and which aren't is truly fascinating, and deeply connected to the question of whether or not our universe is deterministic and the question of free will. (Determinism

and computability are *not* equivalent notions! Our universe could very well be deterministic but not computable. The converse, however, is false; If our universe is computable, then it is necessarily deterministic. So computability is a strictly stronger notion than determinism. This, however, is a discussion for another self indulgent writing venture.) It may or may not surprise you to learn that there are in fact problems which have been mathematically proven to *not* be computable. The halting problem is the classic example. The busy beaver function is another interesting case, as it can be quite easily defined, yet has been proven to grow faster than *any* computable function. (This beaver gets to *work*, lemme tell ya.) I'll leave these to your own google searches if you're interested. Mathematicians and computer scientists have made a surprising amount of concrete progress when it comes to computability in the last 60 or so years. Nevertheless, the question of computability is in general quite a difficult one.

There is, however, a related, more constrained and more approachable question, which, depending on context, might be just one step short of being equivalent to the question of computability. That question is this: Of those problems which *are* computable, which ones are *feasible* to execute? That is to say, which problems are solvable by a computer *efficiently*?

The value I put into this question is likely owed to my background as an engineering student. What use is it to have an algorithm for something, if it takes a length of time which is twice the age of the universe in order to execute it! In particular, I believe that we can, without too much loss of resolution, focus almost entirely on feasibility when it comes to one special case: The workings of the human mind. *If* the processes in our mind are in fact computable (a stance which I am personally biased against), then every algorithmic process thereof would also *have* to be efficient. Otherwise, I'd still be waiting to perform them! In my mind, it is a fruitful venture to look *past* the notion of computibility, and *towards* the notion of *efficient computability*. To approach this notion, we have to define what feasibility actually means. The generally agreed upon definition is the following:

A computer program f is **efficient** iff the number of steps it takes to produce a result is a polynomial function of the input (That is to say, $f = O(n^p)$ for some $p \in \mathbb{N}$).

If you're skeptical of this definition, you're not alone, but you would probably also be convinced of it's value after seeing enough of the results

which follow from it. In lieu of a drawn out discussion of whether or not this is the '*correct*' definition, I'll leave you with a quote:

"It should not come as a surprise that our choice of polynomial algorithms as the mathematical concept that is supposed to capture the informal notion of 'practically efficient computation' is open to criticism from all sides. Ultimately, our argument for our choice must be this: *Adopting polynomial worst-case performance as our criterion of efficiency results in an elegant and useful theory that says something meaningful about practical computation, and would be impossible without this simplification.*"
-Christos Papadimitriou

We say that a decision problem is **efficiently computable** iff there exists an efficient computer program which solves it. We denote the class of all such problems **P** (which stands for **polynomial time complexity**).

Now comes the probability theory. Suppose we have a computer which has the ability to simulate randomness, and use those random results within it's calculations. This new technology would allow us to design algorithms which might not always yield the same result for a given input. This allows us to ask a new question - What kinds of decision problems are efficiently solvable *with high probability?* Is it a larger class than **P**? We begin with a definition:

A decision problem is belongs to the class **BPP** (which stands for **bounded error probabilistic polynomial time complexity**) if there exists an efficient computer program which returns the correct answer with probability $\frac{1}{2} + \epsilon$ for some $\epsilon > 0$.

So a problem is **BPP** if we can come up with a program which returns the right answer more than half the time. Clearly

$$\mathbf{P} \subseteq \mathbf{BPP} \tag{2}$$

Since any problem in **P** has an associated program which solves the problem *deterministically*, that is to say, with probability 1. That is to say, any **P** problem is **BPP** with probability $\epsilon = \frac{1}{2}$. So now we reach the question which we are going to answer - Is **BPP** valuable? Let's see for ourselves.

Suppose that a problem is **BPP**. Who cares? What good is it to gain the correct answer 50.1% of the time? The value of **BPP** comes from the fact that, assuming our definition of efficiency is a good one, we can run the program *any number of times*, and have the overall process be efficient. Suppose I run an efficient program ten thousand times. By our definition of efficiency, the original program's number of steps is some polynomial function of the input $g(n)$. If $g(n)$ is a polynomial, then $10000g(n)$ is still a polynomial! This might sound like cheating, or it might shine some light on the initial controversy which surrounded this definition. Nonetheless, in nearly all practical cases this is actually fine. The current state of technology makes it decently reasonable to assume that any modern computer can execute any supposedly efficient program *any number of times* in the span of a single second. Don't worry, I know that the statement I just made is technically impossible. I don't need you to accept it as entirely true. This statement is only decently reasonable under a sufficient lack of assumed scrutiny, and as we'll shortly see, we'll never need to run a probabilistic **BPP** solution anywhere near ten thousand or even near one thousand times to make it functionally indistinguishable from a deterministic **P** solution.

Suppose that we run our program n times, and pick as our answer to the decision problem the output which occurred *most frequently*. What is the probability that we picked the wrong answer? Define the following random variables:

$$X_i = \begin{cases} 1 & \text{if the } i^{th} \text{ run of the program produces the correct answer} \\ 0 & \text{otherwise} \end{cases} \tag{3}$$

$$X_i \sim Bernoulli(p = \frac{1}{2} + \epsilon) \tag{4}$$

So these are of course identically distributed, and assuming our computer works properly, these X_i are all independent.

Since there are only two possible answers, the answer which occurs most frequently is the answer which occurred more than half of the times that I ran the program. If we're assuming that I picked the wrong answer, then the sum of all of our Bernoulli random variables must be smaller than half of n! (We'll even be pessimists and assume that if n is even and there is a tie, I'll

always pick the wrong answer.) So we have:

$$P(\text{Wrong answer is picked}) = P(\sum_{i=1}^{n} X_i \leq \frac{n}{2}) \tag{5}$$

Note that this is a sum of independent and identically distributed Bernoulli random variables. Thus,

$$\sum_{i=1}^{n} X_i \sim Binomial(n, \frac{1}{2} + \epsilon) \tag{6}$$

$$\implies P(\sum_{i=1}^{n} X_i = k) = \binom{n}{k} (\frac{1}{2} + \epsilon)^k (\frac{1}{2} - \epsilon)^{n-k} \tag{7}$$

$$\tag{8}$$

Now, a mode of a binomial random variable with parameters n and p always occurs at $\lfloor (n+1)p \rfloor$. Since our p is guarunteed greater than one half, we have

$$\left\lfloor \frac{n}{2} \right\rfloor \leq \left\lfloor (n+1)\frac{1}{2} \right\rfloor \leq \lfloor (n+1)p \rfloor \tag{9}$$

For the sake of making life more livable we'll assume that n is even so that $\frac{n}{2} = \left\lfloor \frac{n}{2} \right\rfloor$. (Since we're already being pessimists in the case that n is even, this isn't a big deal to assume.) The binomial distribution is a curve which rises up to a single peak, and then falls back down. So we know that it is an increasing function, at least until it passes the point(s) corresponding to it's mode(s). This is all to say,

$$P(\sum_{i=1}^{n} X_i = k) = \binom{n}{k} (\frac{1}{2} + \epsilon)^k (\frac{1}{2} - \epsilon)^{n-k} \tag{10}$$

$$\leq \binom{n}{k} (\frac{1}{2} + \epsilon)^{\frac{n}{2}} (\frac{1}{2} - \epsilon)^{\frac{n}{2}} \tag{11}$$

$$\tag{12}$$

For any $k \leq \frac{n}{2}$. Next we need to begin by finding an upper bound on how many terms there are exactly in this sum. That is, we need to ask

the question, how many different sequences of bits $(x_1, x_2, ..., x_n)$ contain at most $\frac{n}{2}$ ones? The exact answer to this question is a big messy sum of combinations, but if you just line up the boxes and think about it, you'll realize that at the very least, it's less than 2^n. (Since 2^n is the total number of sequences in which we can have any number of ones! In fact, I'm fairly sure that $\sum_k \binom{n}{k} = 2^k$, but I haven't bothered to confirm this algebraically so I'll just stick with $\leq$.) Finally, putting these two observations together, we have

$$P(\sum_{i=1}^{n} X_i \leq k) \leq 2^n (\frac{1}{2} + \epsilon)^{\frac{n}{2}} (\frac{1}{2} - \epsilon)^{\frac{n}{2}} \tag{13}$$

$$= 2^n [(\frac{1}{2} + \epsilon)(\frac{1}{2} - \epsilon)]^{\frac{n}{2}} \tag{14}$$

$$= 2^n [\frac{1}{4} - \epsilon^2]^{\frac{n}{2}} \tag{15}$$

$$= 2^n (\frac{1 - 4\epsilon^2}{4})^{\frac{n}{2}} \tag{16}$$

$$= 2^n \frac{(1 - 4\epsilon^2)^{\frac{n}{2}}}{4^{\frac{n}{2}}} \tag{17}$$

$$= (1 - 4\epsilon^2)^{\frac{n}{2}} \tag{18}$$

$$\leq (e^{-4\epsilon^2})^{\frac{n}{2}} \tag{19}$$

$$= e^{-\frac{4n\epsilon^2}{2}} \tag{20}$$

$$= e^{-2n\epsilon^2} \tag{21}$$

Where step (18) $\rightarrow$ (19) follows from the fact that

$$1 - x \leq e^{-x} \quad \text{for all x.}$$

(Just graph them on top of each other and you'll see it)

We have thus derived the **Chernoff Bound**, which is the statement

$$P(\text{Wrong answer is chosen}) \leq e^{-2n\epsilon^2} \tag{22}$$

This is an important result - it says that the probability that the most frequently occurring answer is incorrect decays *exponentially* with n, the number of repetitions!

To give you a sense of what this means, consider the following: Suppose we are running an efficient deterministic algorithm on a conventional computer, which finishes in less than a second. The probability that, during this

time, stray magnetic fields from the environment interfere with the electrical processes inside your computer and corrupt a bit, is on the order of 10^{-20}. In terms of number of zeroes past the decimal place, this is about halfway between the number of bits of storage space on a multi-terabyte hard drive and the scale at which quantum mechanics trumps classical mechanics. The point here is that **the extent to which I can trust the output of a *non-probabilistic* algorithm is in general no more than one minus this number.**

Now suppose I have a probabilistic efficient algorithm, which has a probability *34* of yielding the correct answer. (That is, $\epsilon = \frac{1}{4}$ How many times to I have to run the program such that the probability of the most frequent output being wrong is *smaller* than 10^{-20}? We can calculate this using the Chernoff Bound:

$$e^{-2n\frac{1}{4}^2} \leq 10^{-20} \implies e^{-\frac{n}{8}} \leq 10^{-20} \tag{23}$$

$$\implies -\frac{n}{8} \leq -20ln(10) \tag{24}$$

$$\implies n \geq 8 * 20ln(10) \approx 369 \tag{25}$$

So by running this probabilistic algorithm less than 400 times, the reliability of a computer generated the solution to a **BPP** problem is functionally *indistinguishable* from that of a **P** problem. It is entirely reasonable to run an efficient program 400 times on a modern computer with virtually no loss of efficiency compared to running it just once. These are equally valuable complexity classes, in all practical settings!