Unintended Effects of the Development of Quantum Computing on NP-Difficult Problems and the RSA Encryption Algorithm

Matt Fletcher

Abstract—The abstract goes here. This is an abstract. I'm going to keep writing an abstract. TODO write an actual abstract. I'm just filling up space.

Index Terms—Put in keywords here

1 Introduction

In 1955, mathematician John Nash penned a letter to the National Security Agency that would change the face of computing forever. In this letter, he presented a theory about the computational power required to solve different types of math problems. In particular, he focused on the art of cryptography, and the length of time necessary to crack an encryption. Up to that point, computer scientists who created algorithms focused on the required time to solve a particular length of problem. Nash suggested that instead of focusing on the length of time necessary to solve a problem of a particular length, scientists should instead focus on the rate of difficulty growth of the problem required to solve a problem as the length of the inputs grew at a linear rate. Furthermore, instead of focusing on computation time, a scientist should focus on number of computational steps required to solve the problem. A computational step is the number of state changes that the machine goes through in carrying out the set of steps to solve the problem. The shape of this growth curve would help classify the difficulty of the problem. For example, addition of numbers is a problem with linear growth. Doubling the number of digits to be added results in approximately double the computation time. This makes these problems grow at a linear rate[1]. However, problems such as multiplication grow at a slightly faster pace. Doubling the length of the inputs results in a growth of 2^2 times the number of computational steps. Tripling it results in a growth of 3^2 times the number of computational steps. Although this may seem like a fast growth, this is actually growing in what is known as polynomial time. The exponent of the growth rate is always constant. Polynomial growth can always be displayed as some combination of terms n^k, where n is a variable and k is a constant.

There is one more type of problem, however. Take for example cracking a password. For simplicity sake, assume the password is a PIN composed only of digits 0-9. In the worst case scenario, for a 1 digit PIN, the computer would take 10 guesses. For a 2 digit PIN, the computer would take 10^2 guesses. For a 4 digit pin, the computer would take 10^4 guesses. This is growing at a rate of 10^k, where k is some constant. This is known as exponential growth, where the variable is in the exponent. This curve is extremely sharp. For context, if a password was able to consist of any numbers, letters, or special characters on a standard US keyboard, with only an 8 digit password, the number of computational steps required skyrockets up to 82^8, or 2 with fifteen zeros following it.

Mathematicians had been noticing problems like this popping up all over in all different fields of math and science. So, they decided to classify these problems. If a problem could be solved in a polynomial amount of time, it was classified as difficulty P. Problems with difficulty P are both easy to solve, and easy to verify that the calculated answer is correct. Problems classified as NP are slightly different. NP stands for Nondeterministic Polynomial time. NP problems are difficult to solve, but easy to check a solution to in polynomial time. One of the most common examples of an NP problem is finding a subset of a list that satisfies a given requirement. This is often referred to as the hackey sack problem. If someone is given a collection of small sacks with given random weights, they are given the goal of finding a subset of those sacks that results in a given weight. The only way to approach this problem with current methods is a brute force algorithm, guessing a random subset, and checking the result. Yet another example of an NP problem is the factoring of a number into its prime factors. In his 1801 book *Disquisitiones Arithmeticae*, mathematician Frederick Gauss proved that any number has exactly 1 prime factorization[2]. It is extremely easy to check if a given factorization for a number is correct (the multiplication of said numbers is a P-difficulty problem). However, in order to find the unique factorization for a number, the only known method currently is a brute force approach, trying every single factor for that number. As the length of the target number grows, so does the length of computation time, but the computation time grows exponentially. As the growth of computation time is not of a polynomial form,

1. This is still technically a polynomial growth of the form n^k with $k = 0$.

2. This is according to the Fundemental Theory of Arithmetic.

the factorization problem is considered an NP-difficulty problem. But what application does this have in the real world?

When an online shopper enters their credit card details on an online store, they know their credit card details are safe if they see the little green lock icon in their URL bar. Less known is the fact that the only reason this lock icon exists is due to prime factorization being an NP problem. It all lies in the difficulty of RSA encryption. In 1978, three computer scientists, Ron Rivest, Adi Shamir and Leonard Adleman, publicly released an encryption algorithm titled after their names, RSA. The basis of this encryption technique[3] is two secret prime numbers (known as private keys) multiplied together, to form a public key. It is extremely easy to check whether the two numbers multiplied together result in the public key. However, in order to find the public key, the only way to approach is through a brute force approach [4]. For an idea of size, the public key length for most banking systems nowadays is over 600 digits long[5]. In order to brute force this, with current calculation methods, it would take somewhere on the order of 6.4 quadrillion years to brute force the hash.

Throughout this paper, I keep referring to the phrase "with current methods" in context of methods used to solve a problem. What I mean by this is the use a computing device known as a Turing computer to solve a problem. In 1936, English computer scientist and cryptologist Alan Turing proposed the idea of a theoretical computing machine that operates off of a memory tape with infinite length divided into discrete cells. Each cell contains a basic instruction. The reading head above the tape identifies the value of the cell on the memory tape, then moves the tape either to the left, to the right, or terminates the program. Although this may seem very simple, determining if an algorithm can be run on a Turing machine constitutes one of the most important problems in the field of computer science. A Turing computer is the most basic computer possible, taking in bits one at a time, and outputting a result based on the bit. These computers are powerful. However, they fail when it comes to NP problems. Because NP problems grow with non-polynomial growth, the computation time grows quickly to an unmanageable amount. However, this will change with the introduction of quantum computers. All modern day computers operate off of a common piece of electrical circuitry known as the transistor. A transistor is a micro-switch that can read either 1 (for on) or 0 (for off). They will exist in exactly one of these states at any given time. By combining these transistors into logical gates, basic functions can be performed, such as addition and multiplication. This is the basis of a computer.

A quantum computer operates off of a slightly different principle. Instead of operating off of a physical piece of hardware (the transistor) that produces a bit, it operates off of observation of a photon that represents a qubit. A qubit is a bit that can either represent a 1, a 0, or a superposition of a 1 and a 0. How can something exist in both a 1 or a 0? To understand this, one must first understand the principle behind Schrödingers cat.

The infamous Schrödinger's cat experiment is performed as follows. Imagine a living cat inside a sealed bunker, with a sealed beaker of poisonous material, a Geiger counter attached to a hammer, and a small piece of radioactive material. The radioactive material has a 50% chance of having one of its atoms decay in the next hour (which would set off the Geiger counter, smashing the hammer into the vial of gas and releasing it into the bunker, killing the cat). As long as the bunker is sealed and unobserved, the cat has a fifty percent chance of being dead and a fifty percent chance of being alive. When the bunker is opened, however, the cat must either be dead or alive. The instant before the bunker is opened, however, the cat exists in a quantum superposition of both dead and alive.

Just like the cat, as long as the qubit is unobserved, it exists in a quantum superposition both a 1 and a 0. As soon as the qubit is observed, though, it "snaps" to either a 1 or a 0.

I wish you the best of success.

mds
August 26, 2015

1.1 Subsection Heading Here

Subsection text here.

1.1.1 Subsubsection Heading Here

Subsubsection text here.

2 Conclusion

The conclusion goes here.

Appendix A
Proof of the First Zonklar Equation

Appendix one text goes here.

Appendix B

Appendix two text goes here.

Acknowledgments

The authors would like to thank...

References

[1] H. Kopka and P. W. Daly, *A Guide to LaTeX*, 3rd ed. Harlow, England: Addison-Wesley, 1999.

3. Please note that this is a highly simplified explanation of the encryption algorithm. To describe this in full detail would take far more in-depth explanations and mathematics than would be appropriate for this paper.

4. Although some algorithms exist to reduce the work slightly using methods to eliminate groups of guesses, it is still a polynomial reduction on an exponential growth, which means the reduction is almost unnoticeable.

5. A 617 digit long public key is the key length for 2048 bit encryption.

Michael Shell Biography text here.

John Doe Biography text here.

Jane Doe Biography text here.