Conquering Generals NP-Hard Proof of Work for Blockchain Construction

conqueringgeneral@yandex.com
Royal Holloway University of London
Department of Mathematics
MSc Mathematics of Cryptography and Communications

Advisor: Private

September 2, 2017

Declaration

This dissertation is duly submitted as part of the requisite work for the award of the Masters of Science in Mathematics of Cryptography and Communications at Royal Holloway University of London. I declare that I have successfully completed the Avoiding Plagiarism course and fully understand and agree to abide by its contents. I can therefore declare that all work on this dissertation is my own in accordance to all rules and regulations referenced above.

Signature

Contents

1 Introduction **4**
- 1.1 History of Pricing, Cost and Proof of Work Algorithms . . . 4
- 1.2 Use of Proof of Work in the Context of Cryptocurrencies . . . 4
- 1.3 Proof of Work in Bitcoin . . . 4
  - 1.3.1 The 51% Attack . . . 5
- 1.4 Ambition to Improve Proof of Work . . . 5
- 1.5 Imperious and Humble Goals of this Research . . . 6
- 1.6 Document Layout . . . 7

2 Background Information and Definitions **7**
- 2.1 NP-Completeness and NP-Hardness . . . 8
  - 2.1.1 A Brief History of the Terminology . . . 8
  - 2.1.2 Formal-Language Theory of Computational Decision Problems . . . 8
  - 2.1.3 Computational Complexity Classes . . . 9
  - 2.1.4 Polynomial Time Reducibility . . . 10
- 2.2 The Travelling Salesman Problem . . . 11
  - 2.2.1 Assumptions for the TSP . . . 12
  - 2.2.2 Justification for Application in Proof of Work . . . 12
  - 2.2.3 History of the TSP . . . 13
  - 2.2.4 Combinatorial Optimization Algorithms for the TSP . . . 14
  - 2.2.5 Instances of the TSP . . . 15
  - 2.2.6 Is the Euclidean TSP NP-Complete? . . . 15
- 2.3 Desiderata of Cryptographic Hash Functions . . . 16
  - 2.3.1 Standard Security Properties of a Cryptographic Hash Function . . . 16
  - 2.3.2 The Random Oracle Model . . . 17
- 2.4 Definition: Byzantine Generals Proof of Useful Work . . . 18
  - 2.4.1 Overview of the Byzantine Generals Proof of Useful Work . . . 19
  - 2.4.2 Formal Definition: Byzantine Generals Proof of Useful Work . . . 19

3 Current Proof of Work Variants **21**

4 Proposal: The Conquering Generals Proof of Work **22**
- 4.1 Prerequisites for Proof of Work . . . 22
- 4.2 Main Stages of The Conquering Generals Proof of Work . . . 22
- 4.3 Insight to the Name “The Conquering Generals” . . . 25
- 4.4 Details, Benefits and Limitations of Proposed Proof of Work . . . 26
  - 4.4.1 Round 1 Step 1 Generate . . . 26
  - 4.4.2 Round 1 Step 2 Solve . . . 28
  - 4.4.3 Round 1 Step 3 Propagate . . . 28
  - 4.4.4 Round 1 Step 4 Verify . . . 28
  - 4.4.5 Round 1 Step 5 Sort . . . 28
  - 4.4.6 Round 2 Step 1 Generate . . . 29
  - 4.4.7 Round 2 Step 2 Solve . . . 35
  - 4.4.8 Round 2 Step 3 Propagate . . . 36
  - 4.4.9 Round 2 Step 4 Sort . . . 37
  - 4.4.10 Round 2 Step 5 Verify . . . 37
  - 4.4.11 Round 2 Step 6 Commit Transactions, Output and Loop . . . 38

5 Proof: The Conquering Generals is a Proof of Work Algorithm 38
5.1 Proof of Theorem 4 for Round 1 38
5.2 Proof of Theorem 4 for Round 2 39

6 Future Research 41

7 Final Comments 42

8 Glossary 43

Abstract

Proof of Work algorithms are ubiquitously used in cryptocurrencies in order to obtain universal consensus in a distributed system of peer-to-peer hosts which inherently share no trust. When employed in the context of cryptocurrencies, Proof of Work algorithms are exploited to prevent double-spending by creating an immutable data structure known as the blockchain. The most common Proof of Work algorithms currently use the SHA-256 based hashing algorithm to collectively prove that a certain amount of computational expenditure has been spent in order to solve a problem based on an adjustable difficultly target. The purpose of this paper is to explore a Proof of Work system which uses the SHA-256 cryptographic hashing algorithm to construct the NP-Hard Travelling Salesman Problem. The augmented goals of such a composition over a singular hash based proof of work are to:

- provide a computational energy multi-use model in pursuit to solve hitherto intractable and practical computational problems.
- to fiscally incentivize further research into the algorithms that aim to elucidate the optimization of the NP-Hard Travelling Salesman Problem.

1 Introduction

1.1 History of Pricing, Cost and Proof of Work Algorithms

The primary notions of Proof of Work algorithms were initially conceived as a mechanism to combat spam email in 1992 [25]. The terminology at the time was that of a *pricing function*, and the inception of the idea to propose the solving of a computational challenge prior to accessing a resource was contemplated. Further formalization of this objective was discussed in the Hashcash paper as a method to mitigate Denial of Service attacks. The terminology progressed into that of a *cost function* [5]. One of the earliest formalizations of the term *Proof of Work* was outlined in 1998 in the paper *"Proof of Work and Bread Pudding Protocols"*. Proof of Work was qualitatively defined as, *"In a PoW, a prover demonstrates to a verifier that she has performed a certain amount of computational work in a specified interval of time"* [35].

1.2 Use of Proof of Work in the Context of Cryptocurrencies

PoW algorithms were never industrialized to combat spam email or to mitigate Denial of Service attacks. The economic and security reasons for this flat utilization were succinctly quantified in the *"Proof of Work Proves Not to Work"* 2004 paper [41]. However, the pinnacle in the use case for PoW algorithms was yet to be fully realized in the disruptive revolution that underpinned the creation of cryptocurrencies. The earliest recorded proposal of a cryptocurrency was Wei Dai's b-money which is described as an anonymous, distributed electronic cash system. The term *Proof of Work* was not explicitly used, however the concept of solving a computational problem to create currency was explored [24]. In 2008, Nakamotos Bitcoin cryptocurrency proposal referred to b-money and Hashcash concepts and elected the use of the SHA-256 hashing algorithm for use in the PoW mechanism [45].

1.3 Proof of Work in Bitcoin

Proof of Work exists in Bitcoin to serve the two main purposes of minting new currency and providing consensus in a peer-to-peer network. Mining is the process by which new

Bitcoin is minted and added to the money supply [3]. However, it is worth noting that Bitcoin is a deflationary currency conditioned on the predictable constrained supply of the currency [3]. In approximately 2140, the last of the 21 million Bitcoins will have been minted, and thenceforth, mining will be fiscally incentivized solely by winning transaction fees for each block. This is in contrast to the current position, where the largest proportion of the mining incentive comes from minting new currency, and only a nominal measure comes from block transaction fees. PoW also generates an immutable data structure known as the blockchain to ensure that every transaction subsequent to the Genesis Block being established (on January 3, 2009 18:15:05 GMT) is irrevocably scribed. It is precisely because the blockchain provides a non-repudiatable, publicly distributed database of all historical transactions that fraudulent transactions known as double-spending, become computationally infeasible. (See Section 1.3.1 for a note regarding the latter).
In summary, Bitcoin, and other altcoin miners are a fiscally incentivized group of distributed computational resource consensually exploited to expend electrical energy in order to compete against each other to solve the computational problem presented by the Proof of Work difficulty target. This energy expenditure in turn provides consensus for the validity and immutability of the transactions in each block.
For an excellent review of the technical details of Bitcoin Proof of Work see *Mastering Bitcoin*, chapter 8 *Mining and Consensus* by Andreas Antonopoulos [3].

1.3.1 The 51% Attack

Bitcoin uses a Hashcash based PoW outlined by Back [5]. Any Hashcash based PoW's are susceptible to the 51% Attack. This attack allows a miner who possesses 51% or more of the computational resource available in the mining pool to *"cause deliberate 'forks' in the blockchain and double-spend transactions or execute denial of service attacks against specific transactions"* [3]. In January 2014, the Ghash.io mining pool was in possession of between 40-50% of the computational resource across all Bitcoin miners. The nature of Bitcoin ensures that there is no central regulation point, and in the case of the Ghash.io mining pool, only self-regulation pre-empted individual miners to leave this mining pool [16].

1.4 Ambition to Improve Proof of Work

In Wei Dai's 1998 b-money paper, under Section 1 *The Creation of Money*, he indicates the following: *"... The only conditions are that it must be easy to determine how much computing effort it took to solve the problem and the solution must otherwise have no value, either practical or intellectual"* [24].
Ten years later, Nakamotos 2008 Bitcoin paper, under Section 4 *Proof of Work*, he indicates the following: *"Proof of Work is essentially one-CPU-one-vote"* [45].
Neither of these visionaries could have predicted the exponential growth in competition between cryptocurrency miners. The competition between miners drove the requirements for higher hash rates per second, noted **H/s**. CPU mining was soon supplanted by GPU and FPGA mining and most recently by ASIC miners. There is currently debate about the possible environmental impact of cryptocurrency mining, due to the energy expenditure required. In an April 2013 article the following statistic was provided: *"on a given weekend in 2011 stated that in a 24 hour period approximately 930 megawatts of power were expended in the activity of bitcoin mining. This translated fiscally to approximately 140k USD, and into energy consumption as 'Thats enough to power roughly 31,000 US homes, or about half a Large Hadron Collider.' "* [2].

Beyond the debate about environmental impact there is an appetite to consider a revisionist view on Wei Dai's comments about having no practical or intellectual value as an auxiliary outcome of finding a solution to the computational challenges. To date there are currently altcoins in circulation such as Primecoin [39], Reicoin [49] and Gapcoin [27] based on number theoretic Proofs of Work which indeed challenge Wei Dai's notions of practical and intellectual value. Of academic significance, there have also been suggestions of other Proofs of Useful Work that seek to find solutions to instances of the Orthogonal Vector Problem [6] and also to base Proofs of Work on graph theoretic computational problems based on finding small cycles or other structures in large random graphs [54]. The former and latter have collectively provided the inception of the idea for the Proof of Work explored in this paper. There is also consideration for other schemes such as Proof of Stake, Proof of Space / Capacity and also hybrids of Proof of Stake and Proof of Work systems [14]. However, these proposals will not be within the scope of consideration for this work.

1.5 Imperious and Humble Goals of this Research

Proposals to augment the Bitcoin PoW and blockchain architecture are generally met with hostility. One of the concerns revolve around the exploiting the immutability of the Bitcoin blockchain by recording of other data types such as smart contracts [53] into the distributed database, thereby contributing to what is known as blockchain bloat [56]. Continuing on theme of the scalability of the current Bitcoin blockchain, which at current time of writing is in excess of 100GB, Bitcoin Core developers are at least entertaining conversations about revision to the SHA-256 based PoW [48]. Also, during the time of creating this paper, a monumental change to the Bitcoin currency occurred due to a disagreement between Bitcoin Core developers and prominent miners over a feature known as *SegWit* or *"Segregated Witness"* which enabled the transaction rate of the Bitcoin currency to increase by allowing each block in the blockchain to have an upper limit of 8MB, compared to the original 1MB limit. On August 1, 2017, a hardfork [9] was officially established and the Bitcoin Cash cryptocurrency was created [8].
This paper is capitalizing on the current appetite for revision and the possibility of another hard or soft fork that may result should a new PoW algorithm be considered and ultimately implemented.
Prevailing ASIC based miners have devoted considerable capital expenditure into their mining operations, so to remove the SHA-256 based PoW would most certainly be met with opposition. There would also arguably be an environmental impact of mass obsoleting of ASIC hardware.
In order to address the concerns of the present ASIC miners, this paper will investigate the construction of a hybrid hash based and the subsequent assembly of a NP-Hard graph theoretic PoW system based on the intractable Travelling Salesman Problem.
The imperious ambition of this paper is to propose a scheme for energy multi-use in high market capitalization, SHA-256 hash based PoW cryptocurrencies, such as Bitcoin, or Bitcoin Cash. The only way to truly address energy multi-use, is not to create yet another altcoin, but aim to win the hearts and minds of the current disruptive cryptocurrency pioneers. The ambition of this paper is to make a compelling case to disrupt the disruptors, and sow the seeds for consideration of the proposed PoW system. Failing to disrupt the enduring nature of the current immutable Bitcoin blockchain PoW, the humbler ambition of this paper is to advocate the concept of Proofs of Work systems such that construction from hash based functions to other NP-Hard problems becomes feasible. Additionally, with further collaboration with developers and computer

scientists, improvements to this concept could build on the growing ambition to exploit the computational power of cryptocurrency mining nodes to solve not only useful, but genuinely practical and hitherto intractable NP-Hard problems.
Lastly, an incidental sanguine consequence of this paper is to provide a widely distributed platform, namely that of cryptocurrency mining, for indirect research into the Clay Institutes **P** Versus **NP** Millennium problem [20]. Should it become feasible to operate the proposed PoW, the competition in mining will expand beyond raw computational hash rates to also encompass the creativity in algorithm design for the NP-Hard TSP. Any improvements in heuristic or approximation algorithms accuracy or exact algorithm efficiency for the TSP will inevitably provide more empirical research towards the **P** Versus **NP** problem.

1.6 Document Layout

This dissertation will proceed in the subsequent sections as follows:
The **Background Information and Definitions** section will provide an overview of NP-Complete decision problems and evidence that when modified to adjust such problems to related optimization problems they remain intractable. There will also be a brief discussion of the *Reducibility of Combinatorial Problems* [37]. The latter discussion will provide insight to the **Final Comments** section of the paper. Context and minutiae to the version of the intractable route optimization TSP will also be defined. Finally, we will merge definitions from Dwork, Back, Ball and Karp respectively to provide our formal definition of a particular type of PoW Algorithm, with a view later in the paper to prove that our proposed system duly satisfies this definition.
The **Current Proof of Work Variants** section will provide a summary of some of the prevailing algorithms in use in Bitcoin, altcoins and supplementary recommendations in other academic papers. Perceived benefits and limitations of each system will also be catalogued to provide the required context and incentive for this paper.
The section **Proposal: The Conquering Generals Proof of Work** will contain the original work related to this paper. Practical considerations will be outlined, as well as perceived limitations and proposed benefits of this PoW algorithm.
Penultimately, the section **Proof: The Conquering Generals is a Proof of Work Algorithm**, will seek to prove that the proposal in this paper does indeed meet the requirements of the formal definition of a Proof of Work algorithm as discussed in the **Background Information and Definitions** section.
The paper concludes with the **Future Research, Final Comments and Glossary** sections which are self-explanatory. We mark the end of each Definition or Theorem with the ♠ marker.

2 Background Information and Definitions

The information provided in this section will provide us with the necessary knowledge to provide insight into the rationale for the proposal of the PoW and its' benefits and limitations in Section 4.4. First, we provide history and definitions in computational complexity theory of NP-Completeness and NP-Hardness. Subsequently we present a mathematical and historical overview of the TSP. A terse overview is provided for a provable security model for hash functions known as the Random Oracle Model [7]. Finally we formulate the definition of the *Byzantine Generals Proof of Useful Work* system.

2.1 NP-Completeness and NP-Hardness

The goal of this section is to provide an overview with the appropriate references to convince the reader that the Travelling Salesman Decision Problem is NP-Complete in the strong sense. Importantly, as our proposal will not construct an instance of the Decision Problem, but instead the related Optimization Problem, we wish to convince the reader that the Travelling Salesman Optimization Problem is *'no easier than'* or *'at least as hard as'* the Decision Problem.

Next let us focus our discussions about NP-Completeness and NP-Hardness with a quote: *"When we demonstrate that a problem is NP-Complete, we are making a statement about how hard it is (or at least how hard we think it is), rather than about how easy it is. We are not trying to prove the existence of an efficient algorithm, but instead that no efficient algorithm is likely to exist"* [23].

2.1.1 A Brief History of the Terminology

The strongest notions of the term that we now refer to as NP-Completeness were essentially first illustrated in the early 1970's by Cook and Karp. In 1971, although Cook did not explicitly derive the term NP-Complete, he highlighted the significance of "polynomial time reducibility" [19] [29]. We will cover reducibility in greater detail in section 2.1.4.

Cooks' paper also proved that every problem in the computational complexity equivalence class of NP could be polynomially reduced to a base NP-Complete problem known as SATISFIABILITY, (also known as the SAT problem) [19] [29]. In other words, Cook provided the foundations of the first NP-Complete Problem SAT, from which all subsequent polynomial-time reductions, to ultimately prove NP-Completeness, would be based upon.

In 1972 Karp then proved that the Decision variants of 21 specific combinatorial problems were *'at least as hard as'* the SAT problem [37] [29]. However, although there was a reference to the term Complete Problems in Karps' paper, the specific term NP-Complete was still not explicitly stated.

Finally, in 1973 ahead of his (then) latest release of *The Art of Computer Programming*, Knuth conducted a poll amongst members of the research community to help determine the terms we now know as NP-Complete and NP-Hard. The request was to find a succinct term to represent the loquacious description *'at least as hard as the polynomial complete problems'*. In 1974 the terms NP-Complete and NP-Hard were collectively agreed. NP-Hard was to qualitatively mean *'as hard as the hardest problems in NP.'* [29].

2.1.2 Formal-Language Theory of Computational Decision Problems

Recall that a computational decision problem is one where there is a yes or no question regarding an input string x, and where an algorithm outputs either 1, to mean *yes*, or 0, to mean *no* in response to the question.

Formal language theory will allow us the means to quantify decision problems into respective complexity classes. A brief overview of a formal language is provided below.

Definition 1 *Formal Language Theory Overview* [23]
Let:
Σ be an *alphabet*, which is a finite set of symbols.
$\mathcal{L}$ be a *language*, which is a set of strings made up of symbols from Σ.
$\mathcal{Q}$ represent an algorithm that solves a decision problem.
x be an input string.
n be the length of a string, such that $|x| = n$, and $n \in \mathbb{Z}^+$.
$k \in \mathbb{Z}^+$ be a constant.
We say that Algorithm $\mathcal{Q}$:

- *accepts language* $\mathcal{L}$ *in polynomial time* if $\mathcal{L} = \{x \in \{0,1\}^* : \mathcal{Q}(x) = 1\}$, and for input $|x| = n$, where n is the length of the string, this happens in $O(n^k)$ time.
- *rejects language* $\mathcal{L}$ *in polynomial time* if $\mathcal{L} = \{x \in \{0,1\}^* : \mathcal{Q}(x) = 0\}$, and for input $|x| = n$, where n is the length of the string, this happens in $O(n^k)$ time.
- *decides language* $\mathcal{L}$ *in polynomial time* if $\exists\ k$ such that, for any input $|x| = n$, where n is the length of the string, it decides correctly whether $x \in \mathcal{L}$ or conversely if $x \notin \mathcal{L}$ in time $O(n^k)$. ♠

To consider the subtle difference between *accepting* and *deciding* a language $\mathcal{L}$, note that Algorithm $\mathcal{Q}$ will not necessarily *reject* a string just because an input $x \notin \mathcal{L}$. This condition could simply cause Algorithm $\mathcal{Q}$ to go into an endless loop, and this is not the same as explicitly *deciding* whether $x \in \mathcal{L}$ [23].

A trivial example of a decision problem is as follows:
INPUT: $x = 241$
QUESTION: does 2 divide x ?
It is clear to see that algorithm $\mathcal{Q}$, to test if the residue class of x is 0 or 1, would output $\mathcal{Q}(x) = 0$ in this case. Of course, despite the output being binary, the complexity of decision problems need not be *easy*, as we shall outline.

A trivial example of a language is:
$\Sigma = \{0, 1\}$
$\mathcal{L} = \{1, 10, 100, 1000, 10000, \ldots\}$, is the language of binary representation of all the numbers which are powers of 2.

2.1.3 Computational Complexity Classes

We provide an abridged view of computational complexity classes in this section, sufficient to outline the main goal of quantifying the difference between *easy / tractable* and *hard / intractable* problems.

The focus will be on four main computational complexity classes, namely: **P**, to stand for Polynomial time - considered *tractable*, **NP**, to stand for Nondeterministic Polynomial, **NPC**, to stand for Nondeterministic Polynomial Complete and **NPH**, to stand for Nondeterministic Polynomial Hard. **NPC** are considered *intractable* problems that are the *hardest* in **NP**. Recall that the term nondeterministic is in reference to the theoretical nondeterministic Turing machine. The definitions of each class in terms of formal language theory are outlined in the definition below.

Definition 2 *Computational Complexity Classes (Abridged)* [23]

- $\mathbf{P} = \{\mathcal{L} \subseteq \{0,1\}^* : \exists$ *an algorithm* $\mathcal{Q}$ *that decides* $\mathcal{L}$ *in polynomial time*$\}$.
  An alternative definition of **P** is also:
- $\mathbf{P} = \{\mathcal{L} :$ $\mathcal{L}$ *is accepted by a polynomial time algorithm*$\}$.
- **NP** is the class of languages that can be *verified* by a polynomial time algorithm.
  $\mathcal{L} \in \mathbf{NP}$, if and only if:
  $\mathcal{L} = \{x \in \{0,1\}^* : \exists$ *a certificate* y *with* $|y| = O(|x|^c)$ *such that* $\mathcal{V}(x,y) = 1\}$.
  Where c is a constant such that, $c \in \mathbb{Z}^+$. Note, that $\mathcal{V}$ is a *verification algorithm*, which is a two-argument algorithm where x is the input string, and y is a binary string called a *certificate*. $\mathcal{V}$ verifies x if $\exists$ a *certificate* y such that $\mathcal{V}(x,y) = 1$.
- **NPC** is the class of languages $\mathcal{L} \subseteq \{0,1\}^*$ such that:
  1. $\mathcal{L} \in \mathbf{NP}$ and
  2. $\mathcal{L}' \leq_P \mathcal{L}$ *for every* $\mathcal{L}' \in \mathbf{NP}$.
  We will discuss polynomial time reductions to clearly explain $\leq_P$ in the next section.
- **NPH** is the class of languages that satisfy **NPC** criteria 2. but not necessarily criteria 1. **NPH** can also encompass *search* and *optimization* problems, unlike **P**, **NP**, **NPC**, which we restrict to *decision* problems. ♠

We also provide the definition of an addendum to that of **NPC**.

Addendum Definition 2 *NP-Completeness in the strong sense* [29]

- **NPC in the strong sense**. Let Π be a decision problem. We say Π is **NPC in the strong sense** if it cannot be solved by a *pseudo-polynomial* time algorithm unless $\mathbf{P} = \mathbf{NP}$. ♠

Proofs of each definition are beyond the scope of this paper and can be found in [23] and [29] respectively.

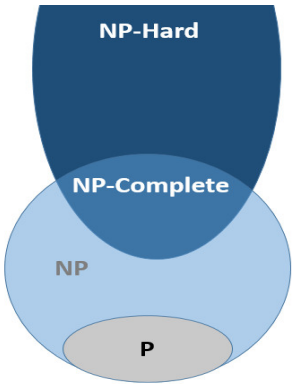

Figure 1: Euler Diagram of the complexity classes assuming $\mathbf{P} \neq \mathbf{NP}$. [30]

2.1.4 Polynomial Time Reducibility

We begin this section with another quote:
"*NP-Complete problems have the intriguing property that if any NP-Complete problem can be solved in polynomial time, then every problem in* ***NP*** *has polynomial time solution, that is* $\boldsymbol{P} = \boldsymbol{NP}$" [23].

In fact, as noted in the Introduction, the **P** Versus **NP** problem is one of the Clay Institutes Millennium problems. Returning to the definition, we say that polynomial time reductions provide a formal means for showing that one problem is at least as hard as another, to within a polynomial time factor [23].

The latter can be expressed as follows in terms of an *easy* problem:
Say we have two languages, $\mathcal{L}_1$ and $\mathcal{L}_2$.
Say we have a polynomial time algorithm $\mathcal{Q}$ to decide if string $x \in \mathcal{L}_2$.
If $\exists$ a polynomial time reduction from language $\mathcal{L}_2$ to language $\mathcal{L}_1$, then we know we can convert algorithm $\mathcal{Q}$ into a polynomial time algorithm to decide if $x \in \mathcal{L}_1$.
In other words, if language $\mathcal{L}_2 \in \mathbf{P}$, then language $\mathcal{L}_1 \in \mathbf{P}$, providing a polynomial time reduction exists.

We formalize these notions below [23]:

Theorem 1 If $\mathcal{L}_1,\ \mathcal{L}_2 \subseteq \{0,1\}^*$ are languages such that $\mathcal{L}_1 \leq_P \mathcal{L}_2$, then $\mathcal{L}_2 \in \mathbf{P}$ implies that $\mathcal{L}_1 \in \mathbf{P}$. ♠

Theorem 2 If any NP-Complete problem is polynomial time solvable, then $\mathbf{P} = \mathbf{NP}$. Equivalently, if any problem in **NP** is not polynomial time solvable, then no NP-Complete problem is polynomial time solvable. ♠

Proofs of both Theorem's can be found in [23].

The question at this point will likely be, "*but how do we prove a problem is NP-Complete?*" For this we return to the work of Cook and Karp. Recall that Cook provided proof of the first NP-Complete problem SAT and Karp provided the reductions for 21 combinatorial decision problems which could ultimately be reduced to the first proven NP-Complete problem. It should be at least qualitatively clear at this point, that to prove a problem is NP-Complete one must provide a reduction to a known NP-Complete problem. As one may say *all roads lead to Rome*, in the context of NP-Completeness one may instead say *all roads lead to SAT*. Again, we omit the technical details of NP-Completeness proofs as they go beyond the scope of this paper but can be found in [23] [19] [29] [37].

2.2 The Travelling Salesman Problem

Note: There are two different spellings of the word "travelling" and "traveling". In this document, we standardize all spellings to the former.
As our Proof of Work system will construct an instance of the Travelling Salesman Optimization Problem from the SHA-256 hash outputs of n of the total m cryptocurrency miners, we define the Decision and related Optimization problems in the formal language $\mathcal{L}$ introduced in **Definition 1** below.

Definition 3 *Travelling Salesman Decision Problem* [29]
INPUT: Set Ω of n cites $\{\omega_1, \ldots, \omega_n\}$, distance $d(\omega_i, \omega_j) \in \mathbb{R}^+$, where $1 \leq i, j \leq n$ and $i \neq j$, for each pair of cities $\omega_i, \omega_j \in \Omega$, positive integer B
QUESTION: Is there a tour of Ω having length B or less, i.e., a permutation $< \omega_{\pi(1)}, \omega_{\pi(2)}, \ldots, \omega_{\pi(n)} >$ *of* Ω such that

$$\left(\sum_{i=1}^{n-1} d(\omega_{\pi(i)}, \omega_{\pi(j)})\right) + d(\omega_{\pi(n)}, \omega_{\pi(1)}) \leq B \quad ? \spadesuit$$

Definition 4 *Travelling Salesman Optimization Problem*
INPUT: Set Ω of n cities $\{\omega_1, \ldots, \omega_n\}$, distance $d(\omega_i, \omega_j) \in \mathbb{R}^+$, where $1 \leq i, j \leq n$ and $i \neq j$, for each pair of cities $\omega_i, \omega_j \in \Omega$
QUESTION: What is the minimal tour length of a permutation $< \omega_{\pi(1)}, \omega_{\pi(2)}, \ldots, \omega_{\pi(n)} >$ *of* Ω, i.e. find a permutation that gives the following

$$\min\left\{ \left(\sum_{i=1}^{n-1} d(\omega_{\pi(i)}, \omega_{\pi(j)}) \right) + d(\omega_{\pi(n)}, \omega_{\pi(1)}) \right\} \spadesuit$$

If we choose the alphabet to be $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}$, then we say algorithm $\mathcal{Q}$ will decide if a string $x \in \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, .\}^*$ is in $\mathcal{L}$ if and only if $x \leq B$.

2.2.1 Assumptions for the TSP

We assume that both the Travelling Salesman Decision and Optimization Problem operate over a two-dimensional Euclidean space. This condition ensures that the Triangle Inequality applies.

Definition 5 *The Triangle Inequality*

For Cities $\omega_i, \omega_j, \omega_k$, where $1 \leq i, j, k \leq n$, the following property holds

$$d(\omega_i, \omega_j) \leq d(\omega_i, \omega_k) + d(\omega_k, \omega_j) \spadesuit$$

Addendum Definition 5 *Symmetric Travelling Salesman Problem* We also assume that the distances between any pair of cities is symmetric, that is

$$d(\omega_i, \omega_j) = d(\omega_j, \omega_i) \spadesuit$$

2.2.2 Justification for Application in Proof of Work

We justify our use of the Travelling Salesman Optimization Problem for use in a Proof of Work Algorithm with the following claims which attest to the *intractability* of seeking an optimal solution:

- **Claim 1:** There is a polynomial time reduction from Hamiltonian Cycle Problem to the TSP [37] [23].
- **Claim 2:** The Travelling Salesman Decision Problem is NP-Complete in the Strong Sense [29].
- **Claim 3:** It is possible to construct The Travelling Salesman Optimization Problem from the Travelling Salesman Decision Problem which is NP-Hard [29].

The formal proofs for these claims can be found in the respective references.

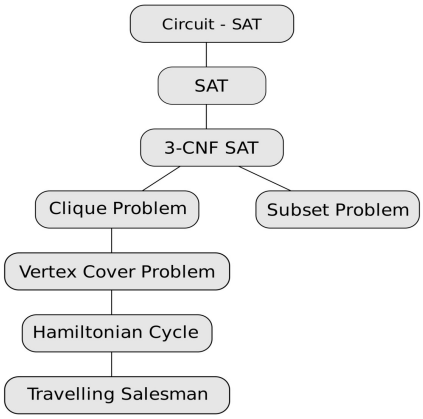

Figure 2: NP-Complete reductions from SATISFIABILITY to the HAMILTONIAN CYCLE problem as in Karps' original paper [37], and the final reduction to the TSP [23].

The run times of a polynomial versus exponential algorithm are given below. The figure below provides a general simplification that all polynomial time algorithms run in $O(n^2)$ and that all **NPC in the strong sense** problems run in $O(2^n)$.

Big O run time	$n = 10$	$n = 25$	$n = 50$	$n = 100$
n^2	0.00001 seconds	0.0002 seconds	0.0001 seconds	0.001 seconds
2^n	0.00001 seconds	0.03 seconds	13 days	40 trillion years

Figure 3: Running time on a 10^9 operations-per-second-computer [21].

2.2.3 History of the TSP

"Christos Papadimitriou told me that the travelling salesman problem is not a problem, it's an addiction" [38].

In this section, we provide a brief visual guide of the TSP and outline the progress made to date in attempting to solve the problem. Inextricably linked to the TSP is the relationship to graph theory and algorithm design. For an excellent review of this subject refer to William J. Cook's *In Pursuit of the Travelling Salesman* [21].

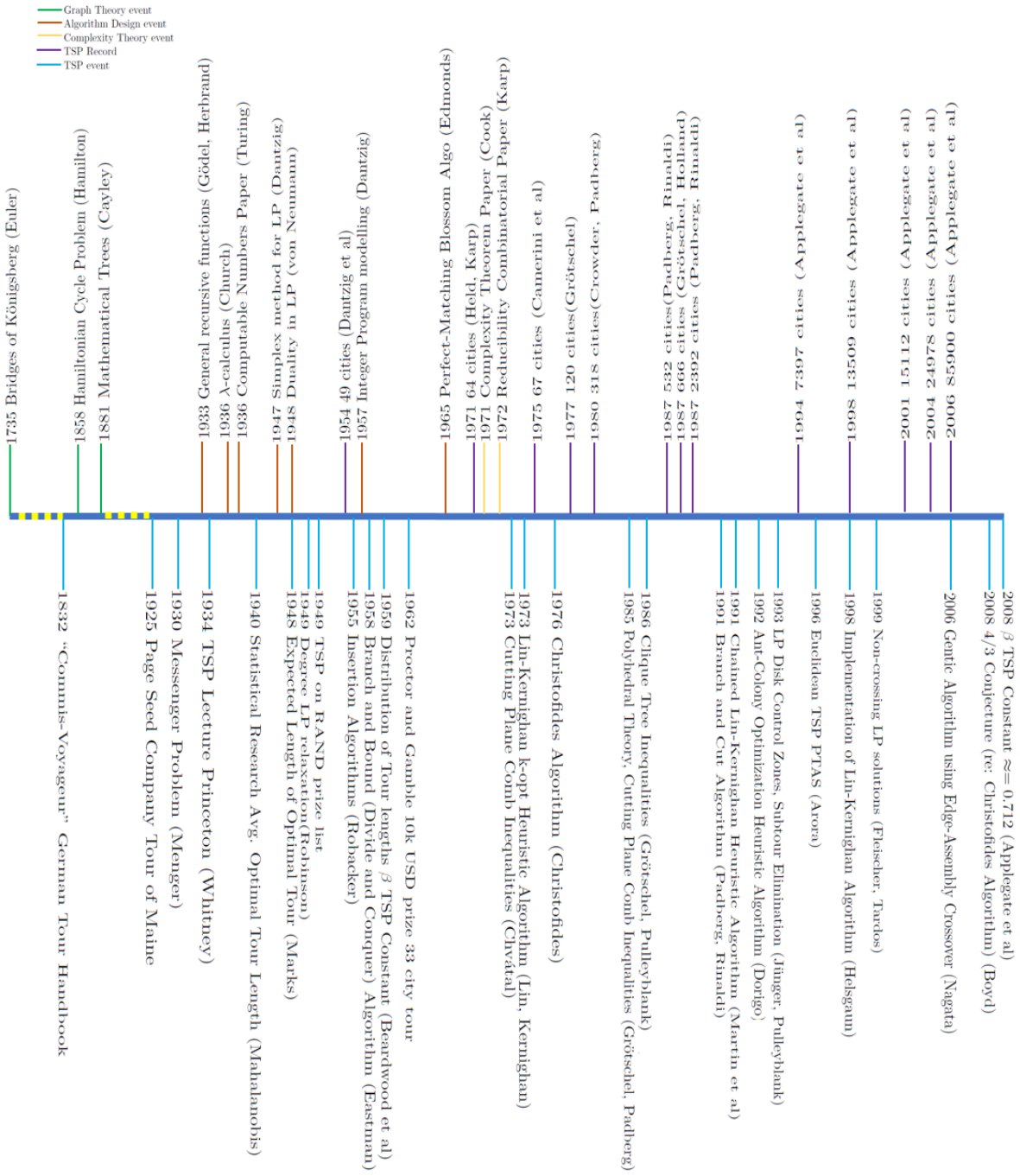

Figure 4: History of the TSP, based on "In Pursuit of the Travelling Salesman" by William J. Cook. (Not to scale) [21].

2.2.4 Combinatorial Optimization Algorithms for the TSP

Four algorithm classifications are used to solve combinatorial optimization problems, such as the TSP. Below is a table of the algorithm class and a known algorithm within the class to attempt to solve the TSP.

Algorithm Classification	Example Algorithm	Big O run time
Exhaustive Search	Näive Algorithm	$O(n!)$
Exact	Held-Karp Algorithm	$O(2^n n^2)$ [21]
Approximation	Christofides Algorithm	$O(n^4)$ [17]
Heuristic	LKH Algorithm	$O(n^{2.2})$ [33]

Figure 5: Algorithm classifications for sample TSP algorithms and respective run times.

In Section 4, the instances of TSP will have $n \leq 8000$, however we can see from the Figure 4 that in 2006 an $n = 85900$ city instance was solved. Some context is required for this solution. Similar to the RSA Factoring Challenge problems, a predefined set of sample TSP instances can be found on the TSBLIB, which was created in 1991 by Gerhard Reinelt [50]. The 85900-city challenge is an example, known in the TSPLIB by the name pla85900. The distances between cities was also rounded up to the next integer in each case using a ceiling function, denoted CEIL2D. Work for this TSP began in 1991. However, a focused effort on a large cluster of computers done by Applegate et al was made between February 2005 and April 2006. pla85900 was solved 136 CPU years later using heuristic, rather than exact methods [21].

The Concorde code is a library of 700 functions used to solve the TSP [22]. The Concorde code is known to find TSP tours on a Euclidean instance for $n \leq 100000$ in a matter of seconds with solutions that fall within 1-2% of the actual optimal value with a high degree of probability [21]. It includes several of the algorithms noted on Figure 4 such as the, Lin-Kernighan, Branch-and-Bound, and Subtour Elimination algorithms. The Lin-Kernighan-Helsgaun (LKH) heuristic algorithm holds the 85900-city tour record as seen on Figure 4. A discussion of the success of heuristic algorithms for finding nearly optimal tours on the TSP PoW will be discussed in Section 4.4.7.

2.2.5 Instances of the TSP

Different instances of the Travelling Salesman Problem exist, and these open the question of the possibility of different complexity classes for each variant. Returning to Figure 4, in 1998, Arora determined that a variant of the Metric TSP, know specifically as the Euclidean TSP (which we will construct in this paper) has a PTAS associated with it [4]. The paper outlines an α-approximation algorithm with a polynomial run time. However empirical evidence has shown that the algorithm design begins to falter as α approaches 1.0 because the run times sharply increase [21]. Arora also proved that for Metric TSP, which still obeys Definition 5, but does not use the ℓ_2 norm for calculating distances, that no PTAS could exist (unless $\mathbf{P} = \mathbf{NP}$). Regardless of instance, since 1976, *"no polynomial-time algorithm is known to have a better* ***worst-case*** *guarantee than Christofides' method"* [21]. The Christofides approximation algorithm is known to give a guarantee that the tour it provides is within 3/2 of the optimal tour length. Also in terms of exact algorithms, to work on *any* instance of the TSP, the Held-Karp algorithm run time (Figure 5) has not been improved upon. The impact of the current state of approximation and exact algorithms will also be discussed in Section 4.4.7.

2.2.6 Is the Euclidean TSP NP-Complete?

Claim 2 in Section 2.2.2 indicates that the TSP Decision problem is NP-Complete. However not all known subcases of the TSP are even provably in NP. In this paper

we construct the Euclidean TSP as well as the rectilinear or "Manhattan" TSP using distance measurements of the ℓ_2 and ℓ_1 norms respectively. Based on Papadimitriou's 1977 paper, we are left with a sense of confidence that both of these instances of TSP are indeed NP-Complete as we indicate in **Claim 2** [47]. However, Papadimitriou's paper assumes a simplification in the distance matrix that is constructed. He states that *"If we take it to be the (infinite precision) real-valued Euclidean metric, it is a nontrivial task to show that the resulting problem is in NP, since there is no obvious upper bound for the precision required in order to compare the length of a tour or path with a given integer. In what follows we will assume that the elements of the distance matrix are the integral parts of this metric"* [47]. Garey et al provided the insight as to why the Euclidean TSP was not known to be NP-Complete in their 1976 *"Some NP-Complete geometric problems"* paper [28]. The main problem in proving Euclidean TSP NP-Completeness is to do with the precision of the ℓ_2 norm, as this distance measurement involves taking square roots. It is still an outstanding problem to formally prove that the Euclidean TSP is NP-Complete because of the Sum of Square Roots Problem outlined in the latter paper. In terms of the ℓ_2 norm distance calculation for our construction of the TSP, Section 4.4.6 **Consideration 1** covers the details for attempting to deal with the precision requirements regarding taking square roots. As we opt for a construction similar to [47], albeit with precision exceeding that of just the integral portion of the distances, we have assurance that the rounded instance of the Euclidean TSP is indeed NP-Complete. Finally, referring back to **Claim 3**, the NP-Hardness of the Euclidean TSP is duly covered in by Garey et al in both [28] and [29].
Returning to the "Manhattan" TSP, this was shown to be NP-Complete and NP-Hard in [28] as the SSRP does not apply in this subcase of the problem.

2.3 Desiderata of Cryptographic Hash Functions

In this section we explore the *standard* and *idealized* security properties of cryptographic hash functions. The proposed PoW will rely heavily on the usage of SHA-256, therefore any cryptographic limitations and the consequences of such shortfalls must be examined in the context of its proposed implementation. We rely heavily on NIST guidance on the Secure Hash Standard as a justification for the usage in our Proof of Work.

2.3.1 Standard Security Properties of a Cryptographic Hash Function

As of August 2015, NIST guidance for cryptographic hash functions consider SHA-256 to be considered *secure* because [46]:

- It is computationally infeasible to find a message that corresponds to a given message digest.
- It is computationally infeasible to find two different messages that produce the same message digest.
- Any change to a message will, with a very high probability, result in a different message digest (This is known as the *Avalanche effect*) [26].

Let us formalize the notions above with a mathematical definition of a cryptographic hash function and its' desiderata.

Definition 6 A *hash family* is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where the following conditions are satisfied [52]:

- $\mathcal{X}$ is the finite or infinite set of possible *messages*. Where $|\mathcal{X}| \geq |\mathcal{Y}|$, or often the stronger condition that $|\mathcal{X}| \geq 2|\mathcal{Y}|$.
- $\mathcal{Y}$ is a finite set of possible *message digests* or *authentication tags*.
- $\mathcal{K}$, the *keyspace*, is a finite set of possible *keys*. For an *unkeyed* hash function we assume $|\mathcal{K}| = 1$.
- For each $K \in \mathcal{K}$, there is a *hash function* $h_k \in \mathcal{H}$. Each $h_k : \mathcal{X} \to \mathcal{Y}$. ♠

Definition 7 *(N,M)-hash family* [52]:

Let $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$ denote the set of all functions from $\mathcal{X}$ *to* $\mathcal{Y}$.
Suppose that $|\mathcal{X}| = N$ and $|\mathcal{Y}| = M$.
Then it is clear that $|\mathcal{F}^{\mathcal{X},\mathcal{Y}}| = M^N$.
Any hash family $\mathcal{F} \subseteq \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is termed a (N, M)-*hash family*. ♠

Definition 8 *Standard Security Properties of a Cryptographic Hash Function* [52]:

- **One-way or Preimage Resistant**.
  Given a hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $y \in \mathcal{Y}$.
  It must be *computationally infeasible* to find $x \in \mathcal{X}$ such that $h(x) = y$.
- **Second Preimage Resistant**.
  Given a hash function $h : \mathcal{X} \to \mathcal{Y}$ and an element $x \in \mathcal{X}$.
  It must be *computationally infeasible* to find $x^{'} \in \mathcal{X}$ such that $x^{'} \neq x$ and $h(x^{'}) = h(x)$.
- **Collision Resistant**.
  Given a hash function $h : \mathcal{X} \to \mathcal{Y}$.
  It must be *computationally infeasible* to find $x, x^{'} \in \mathcal{X}$ such that $x^{'} \neq x$ and $h(x^{'}) = h(x)$. ♠

Unlike 160-bit SHA-1, NIST has not deprecated the use of SHA-256 as a cryptographic hash function. As well as being exploited in PoW algorithms for cryptocurrencies it is also heavily utilized in cryptographic protocol suites such as IPSec, SSH and TLS, which arguably have stronger requirements for its compliance with security properties than PoW algorithms. In the Information Security properties of **CIAN**, *Confidentiality, Integrity, Availability* and most recently *Non-repudiation*, SHA-256 usage in the latter mentioned cryptographic suites form part of the requirement for protecting the *Confidentiality* and *Integrity* of data assets. However, in a cryptocurrency, the PoW's *raison d'être* is to prove that computational effort and ultimately that energy has been expended to arrive at a distributed consensus between a peer-to-peer network of miners that share no trust. Ultimately, the goal is still the *Integrity* of the blockchain, however the blockchain is, by design, publicly viewable in cleartext. In this context SHA-256 PoW is also there to ensure the *Non-repudiation* of the transactions in blockchain to avoid any double-spending of the cryptocurrency.

2.3.2 The Random Oracle Model

As previously noted, the proposed PoW in this paper will rely heavily upon the usage of the SHA-256 cryptographic hash function. It will therefore be felicitous to include an impact analysis of the perceived limitations regarding the randomness of the proposed system. As the Random Oracle Model is an idealized view of a hashing function, in

Section 4.4.6 **Consideration 5** we seek to survey the consequences to the integrity of the proposed system based on strictly extracting the 128 LSB's of the SHA-256 algorithm for the assembly of the TSP for the proposed PoW.
The Random Oracle Model was conceived by Bellare and Rogaway in 1993 as an idealized model of a cryptographic hash function that would subscribe to the required properties for the idea of *provable security* [7]. Practically, this model would be impossible to construct as the storage requirements and exponential look up times for each hash value would be impractical. Stinson and Green have each provided a concise summary of the Random Oracle Model as follows:

Definition 9 *The Random Oracle Model* [52] [31]:
We assume the *hash family* as in **Definition 6** and **Definition 7**.

Let:
$\mathcal{A}$ represent Alice, the encryptor.
$\mathcal{B}$ represent Bob, the decryptor.
$\mathcal{E}$ represent Eve, the adversary.
$\mathcal{O}$ represent the Oracle.

We assume $\mathcal{A}$ and $\mathcal{B}$ are honest, $\mathcal{E}$ is dishonest. We then choose a *hash function* $h : \mathcal{X} \to \mathcal{Y}$ randomly from $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$.

$\mathcal{A}, \mathcal{B}, \mathcal{E}$ are only permitted *oracle* access to the function h. Therefore, the only way any of them can compute the value $h(x)$ is to query $\mathcal{O}$.
$\mathcal{O}$ starts with an empty look up table. For any $x \in \mathcal{X}$, if $\mathcal{A}, \mathcal{B}$ or $\mathcal{E}$ ask her for the value of $h(x)$ she will duly choose a *truly* random value from $\mathcal{Y}$, update the look up table and respond to $\mathcal{A}, \mathcal{B}$ or $\mathcal{E}$ with the value she put in the look up table. Should $\mathcal{A}, \mathcal{B}$ or $\mathcal{E}$ ask her for the value of x at a later time, she will respond with the same value regardless of the time asked and regardless of which party were to ask her the value of $h(x)$. She is consistent with her responses.

Also, we assume that:

- The look up time of $h(x)$ is $O(1)$, assuming the value already exists in $\mathcal{O}$'s look up table.
- The time to generate a *truly* random $h(x)$ is $O(1)$, assuming it does not already exist in $\mathcal{O}$'s look up table. ♠

The definition of the *Random Oracle Model* implies the following theorem.

Theorem 3 [52]: Suppose that $h \in \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is chosen randomly, and let $\mathcal{X}_\prime \subseteq \mathcal{X}$. Suppose that the values $h(x)$ have been determined (by querying an oracle $\mathcal{O}$ for h) if and only if $x \in \mathcal{X}_\prime$. Then $\mathsf{Pr}\Big[h(x) = y\Big] = 1/M \quad \forall x \in \mathcal{X} \backslash \mathcal{X}_\prime \text{ and } \forall y \in \mathcal{Y}$. ♠

Proof of **Theorem 3** can be found in [52].

2.4 Definition: Byzantine Generals Proof of Useful Work

It will be imperative to prove that the proposed construction of the proposed PoW solution satisfies the criteria of our amalgamated definition of a *Byzantine Generals Proof of Useful Work*. In this section we provide both the informal intuition to the newly formulated term as well as a rigorous formal definition. Proof that the proposed PoW satisfies this definition will be covered in Section 5.

2.4.1 Overview of the Byzantine Generals Proof of Useful Work

The justification of the name *'Byzantine Generals Proof of Useful Work'* is based on the historical context of the terminology defined in the Hashcash cost functions paper [5] and verbose definition of *'Useful'* work provided by Ball et al [6]. Based on *The Byzantine Generals Problem* [40] and *Practical Byzantine Fault Tolerance* [13], cryptocurrency mining pools are seen to model the definitions of the distributed systems architecture defined in these papers. We assume henceforth because of the conformity of cryptocurrency mining pools to the Byzantine Generals model of distributed computing, that the Proof of Useful Work systems have the following properties noted informally in the following definition. [6] [5]:

Definition 10 *Overview of the Byzantine Generals Proof of Useful Work*

Constituent Algorithms: The subsequent algorithms must all comprise the system: Gen - which *generates* the *challenge*, Solve, which provides a *solution* to the *challenge*, Verify, which *verifies* the *challenge*, and Recon, which *reconstructs* a valid *solution.*

Efficiency: All constituent algorithms must run in polynomial time, with the exception of the Solve algorithm, (which may or may not run in polynomial time. This will depend on the decision to use an approximation, heuristic or exact algorithm to *solve* the given *challenge*).

Completeness: A complete algorithm is guaranteed to find a valid *solution* when there is one, else it returns a failure [36].

Soundness: A sound algorithm guarantees the correctness of a *solution* [36].

Hardness: If some Solve^* algorithm exists that takes less than the required amount of computational effort to allegedly complete the Proof of Work, then the *solutions* it provides against the *challenge* generated by the Gen algorithm will unlikely be the absolute optimal solutions to any NP-Hard Optimization Problem.

Usefulness: Valid *solutions* to the *challenges* constructed by the mining pool participants by the Gen algorithm can be verified and reconstructed in polynomial time by the Verify and Recon algorithms respectively.

Publicly-auditable: Any *solution* to the Proof of Work can be verified in polynomial time by *any* party without access to any private key, secret or trapdoor information.

Non-interactive: Participants in the PoW achieve distributed consensus about the generation of a pertinent *challenge* without any requirement to interact with a TTP. ♠

2.4.2 Formal Definition: Byzantine Generals Proof of Useful Work

Definition 11 *Constituent Byzantine Generals Proof of Useful Work Algorithms* [6]:

Let $m \in \mathbb{Z}^+$, and let $\{\alpha_1, \ldots, \alpha_m\}$ represent the set of all miners online at a given time.
Let $n \in \mathbb{Z}^+$, where n is the input size of the computational problem, and where $n \leq m$.
Let x be an instance of the computational problem, where $|x| = n$, and where we wish to learn some $f(x)$.
Let $t(n)$ represent a length of computational time.

- $\mathsf{Gen}(x)$ is a randomized algorithm constructed exclusively by $\{\alpha_1, \ldots, \alpha_m\}$, that takes an instance x and produces a *challenge* c_x.
- $\mathsf{Solve}(c_x)$is an **non-interactive** algorithm that solves the *challenge* c_x, producing a *solution sketch* s_x.
- $\mathsf{Verify}(c_x, s_x)$ is a **publicly-auditable**, possibly randomized algorithm that verifies the *solution sketch* s_x to *challenge* c_x.
- $\mathsf{Recon}(c_x, s_x)$ is an **publicly-auditable** algorithm that given a valid s_x for c_x reconstructs $f(x)$. ♠

Definition 12 *Properties of the Byzantine Generals Proof of Useful Work* [6] [5]. A $(t(n), \delta(n))$-Byzantine Generals Proof of Useful Work for f consists of the four algorithms noted in **Definition 11**, where $m, n \in \mathbb{Z}^+$. The algorithms must satisfy the following properties:

Efficiency: For any $|x| = n$

- For any x, $\mathsf{Gen}(x)$ runs in time $\tilde{O}(n^2)$.
- For any $c_x \leftarrow \mathsf{Gen}(x)$, $\mathsf{Solve}(c_x)$ runs in time $\tilde{O}(t(n))$.
- For any $c_x \leftarrow \mathsf{Gen}(x)$ and any s_x, $\mathsf{Verify}(c_x, s_x)$ runs in time $\tilde{O}(n)$.
- For any $c_x \leftarrow \mathsf{Gen}(x)$ and $s_x \leftarrow \mathsf{Solve}(c_x)$, $\mathsf{Recon}(c_x, s_x)$ runs in time $\tilde{O}(n)$♣.

Completeness: For any $c_x \leftarrow \mathsf{Gen}(x)$ and any $s_x \leftarrow \mathsf{Solve}(c_x)$,

$$\mathsf{Pr}\Big[\mathsf{Verify}(c_x, s_x) = accept\Big] = 1$$

Where the probability is taken over Verify's randomness.

Soundness: For any s and $c_x \leftarrow \mathsf{Gen}(x)$ such that $\mathsf{Recon}(c_x, s) \neq f(x)$,

$$\mathsf{Pr}\Big[c_x \leftarrow \mathsf{Gen}(x) \wedge \exists s\ \mathsf{Recon}(c_x, s) \neq f(x) \mid \mathsf{Verify}(c_x, s) = accept\Big] < neg(n)$$

Where the probability is taken over Gen's randomness and $|x| = n$.
(We also note that a **Sound** *solution sketch* need not be an optimal *solution sketch*, but merely a correct *solution sketch*, see **Definition 14**, **Round 2** Step 5 for an example).

Hardness: Assume $\mathbf{P} \neq \mathbf{NP}$. Let $1 \leq i \leq m$. For any $c_{x_i} \leftarrow \mathsf{Gen}(x_i)$, where $|x_i| = n$ and any $\mathsf{Solve}^{(\mathbf{P})}$, where the $(\mathbf{P})$ superscript indicates a polynomial time algorithm as noted in **Definition 1** and **2**, given the m *challenges* $x_1, \ldots, x_m$ and the corresponding **sound** *solution sketches* $s_1, \ldots, s_m$

$$\mathsf{Pr}\Big[(s_1, \ldots, s_m) \leftarrow \mathsf{Solve}^{(\mathbf{P})}(c_{x_1}, \ldots, c_{x_m}) \;\Big|\; \forall\ i\ s_i = s_{i(opt)}\Big] = \delta(n)$$

Where $s_{(opt)}$ is the absolute optimal solution to an NP-Hard (**Definition 2**) combinatorial optimization problem, and where
$\delta(n) = (number\ of$ **sound** *solution sketches*$)^m$. Therefore, the probability is taken over the possible number of **sound** *solution sketches* to the power of m.

We note that this definition implies (by combining *completeness* and *soundness*) the following notion of *usefulness*.

Usefulness: For any $c_x \leftarrow \mathsf{Gen}(x)$ and $s_x \leftarrow \mathsf{Solve}(c_x)$, we have

$$\mathsf{Recon}(c_x, s_x) = f(x).\spadesuit$$

♣ *Recall that* $\tilde{O}(n)$, *read"soft-oh"*, notation has the meaning of O, with the logarithmic factors ignored [23]:

$$\tilde{O}(t(n)) = \big\{g(n) \ : \ \exists\, c, k, \ and\ n_0 \in \mathbb{R}^+ \quad such\ that \quad 0 \leq g(n) \leq ct(n)log^k(n) \quad \forall\, n \geq n_0\big\}.$$

To provide an example of **Hardness** for the TSP we note that there are $\frac{(n-1)!}{2}$ possible tours of an n city instance. First note that if you start at city $\omega_{\pi(1)}$ there are $n-1$ options for the second city and $n-2$ options for the third city, etc. Also because of symmetric nature of the TSP as outlined in the **Definition 5 Addendum** we divide the $(n-1)!$ options by 2 [21]. Therefore for m *challenges* each of size n we have $\delta(n) = (\frac{(n-1)!}{2})^m$.

3 Current Proof of Work Variants

We provide the following table with a sample selection of current PoW variants, both academic and operational.

PoW Variant	**PoW Sub-Variant**	**Example Use Case**	**Perceived Benefits**	**Perceived Limitations**	**Market Cap (BTC)** [18]
Hash Based	SHA-256 output less than target value	Bitcoin [45]	original cryptocurrency PoW, widely used, matured	computational complexity hardness heuristic, energy consumption	16445250
Hash Based	Scrypt key derivation	Litecoin [42]	widely used, matured	computational complexity hardness heuristic only, energy consumption	1020188
Number Theoretic	Cunningham Chain, Bi-Twin Chain search	Primecoin [39]	academic application of solutions	low market cap	2909
Number Theoretic	Prime constellation search	Reicoin [49]	Reimann Hypothesis academic application	very low/ no market cap	0
Number Theoretic	Prime gap search	Gapcoin [27]	academic application of solutions	very low market cap	20
Graph Theoretic	Cycle and structure search in large graphs	academic only [54]	academic application, constructed problems have strongly conjectured computational complexity classes	currently only academic	0
Linear Algebra Based	Orthogonal Vector Problem	academic only [6]	academic application, verbose fine-grained computational complexity class conjecture provided	currently only academic	0

Figure 6: Table of current Proof of Work variants.

4 Proposal: The Conquering Generals Proof of Work

In this section we assume that use of the term *Proof of Work* refers explicitly to **Definition 12** provided in section 2.4.2. We describe the prerequisites for the proposed system, a high-level overview of each step, followed by an insight into the chosen name. We finally outline relevant details, benefits and limitations of the proposal.

4.1 Prerequisites for Proof of Work

As stated previously, Bitcoin mining is now almost exclusively cornered by ASIC hardware. Our PoW will require computations beyond that which a SHA-256 ASIC miner is capable. The computational prerequisites assumed for the proposed PoW are outlined below.

Definition 13 *Prerequisites for Proof of Work*

- miners must have synchronized clocks. Ideally all miners should synchronize to a predefined list of Stratum 1 NTP clocks.
- miners must have the computational resource to run the SHA-256 cryptographic hash function as well as being able to execute an algorithm of their choice to optimize the TSP. This implies the need for requisite memory to store a graph distance matrix - which will be a $n \times n$ two-dimensional array.
- miners must have the computational resource to run a sorting algorithm of their choice.
- miners must have the computational resource to run a floating-point arithmetic operation to find square roots using the Pythagorean theorem in order to calculate ℓ_2 norms.
- miners must know how many other miners are online in near real-time.

4.2 Main Stages of The Conquering Generals Proof of Work

Definition 14 *Main Stages of The Conquering Generals Proof of Work.*
The PoW will *deterministically* mint new currency at intervals of $[t_s, t_c]$, We propose that $t_c - t_s = 10$ minutes.
We use the details in **Definition 11** for the algorithms to be used.
Recall $n, m \in \mathbb{Z}^+$, $n \leq m$. The difficulty adjustment will also be based on value n.
In this case we define $c_{x_2} = \mathcal{G}$, where $\mathcal{G}$ represents a *graph distance matrix.*
We also assume h is the keyless hash function in **Definition 6**, which we propose to be NIST's SHA-256.
We also define η as a cryptographic nonce and ψ as a SHA-256 cryptographic hash output.
The subscripts are defined as 1 or 2 to represent the round, (i) to represent the identification of the miner, (j) to represent the block number in the blockchain and (k) to represent the counter number on the various attempted nonces.
The superscripts are defined as (ℓ), in Round 1 represents the *lowest* value in the set of all attempted hashes and in Round 2 represents the permutation of n cities giving the *lowest* sum. The final superscript $(\mathcal{L})$ in Round 2 represents the lowest of the (ℓ) lowest *solution sketches.* The main steps of the proposed PoW are as follows.

Round 1

1. $\mathsf{Gen}_1(x_1)$

   - The start time t_s for this round will be noted by all miners.
   - The set of $\{\alpha_1, \ldots, \alpha_m\}$ miners online will each randomly *generate* their own unique *challenge* $c_{x_1(i)(j)}$ by choosing $|x| = N$ *unconfirmed transactions* from the *memory pool*.
   - Once each *challenge* $c_{x_1(i)(j)}$ is *generated*, miners may purge the *solution sketches* from the $(j-1)$ blocks **Round 1** Step 5.

2. $\mathsf{Solve}_1(c_{x_1(i)(j)})$

   - Will be run by the set of $\{\alpha_1, \ldots, \alpha_m\}$ miners.
   - The miners will run the double iterated SHA-256 PoW as follows:
     Attempt 1: $h(h(c_{x_1(i)(j)}||\eta_{x_1(i)(j)(1)})) = s_{x_1(i)(j)(1)}$
     Attempt 2: $h(h(c_{x_1(i)(j)}||\eta_{x_1(i)(j)(2)})) = s_{x_1(i)(j)(2)}$
     $\vdots$
     Attempt at t_1: $h(h(c_{x_1(i)(j)}||\eta_{x_1(i)(j)(@t_1)})) = s_{x_1(i)(j)(@t_1)}$
   - Solve_1 will be run until time $t_1 = t_s + 3$ minutes.

3. Propagate *solution sketches*

   - At time t_1 each miner will propagate to all other $m-1$ miners their *solution sketch* $s^{(\ell)}_{x_1(i)(j)}$ from $\mathsf{Solve}_1(c_{x_1(i)(j)})$
   - We note that $s^{(\ell)}_{x_1(i)(j)} = \mathsf{min}\{s_{x_1(i)(j)(1)}, s_{x_1(i)(j)(2)}, \ldots, s_{x_1(i)(j)(@t_1)}\}$.
   - Miners must also propagate the corresponding nonce $\eta_{x_1(i)(j)(k)}$ for the lowest *solution sketch*. It will be required for verification in the next step.

4. $\mathsf{Verify}_1(c_{x_1(i)(j)}, s^{(\ell)}_{x_1(i)(j)})$

   - $\forall\ i$ where $1 \leq i \leq m$, all miners will verify the set of *solution sketches* $\{s^{(\ell)}_{x_1(1)(j)}, \ldots, s^{(\ell)}_{x_1(m)(j)}\}$.
   - The algorithm will run as follows for the appropriate (k) propagated in the previous step
     $\mathsf{Verify}_1(c_{x_1(i)(j)}, s^{(\ell)}_{1(i)(j)}) = h(h(c_{x_1(i)(j)}||\eta_{x_1(i)(j)(k)})) = s^{(\ell)}_{x_1(i)(j)}$
   - If $h(h(c_{x_1(i)(j)}||\eta_{x_1(i)(j)(k)})) \neq s^{(\ell)}_{x_1(i)(j)}$ then the particular *solution sketch* $s^{(\ell)}_{x_1(i)(j)}$ will be omitted from progression to the next step.
   - Verify_1 is a deterministic algorithm with no associated randomness.
   - The Recon_1 algorithm is the same as the Verify_1 algorithm and $f(x_1(i)(j)) = s^{(\ell)}_{x_1(i)(j)}$.

5. Sort *solution sketches*

   - Miners will sort the *solution sketches* into the *ascending ordered sequence* $| < \underbrace{s^{(\ell)}_{x_1(\sigma(a))(j)}, \ldots, s^{(\ell)}_{x_1(\sigma(b))(j)}}_{n \text{ lowest solution sketches}}, \ldots, s^{(\ell)}_{x_1(\sigma(c))(j)} > | = m$.

- The lowest n *solution sketches* and their corresponding miners, denoted by the set $|\{\alpha_{(\sigma(a))}, \ldots, \alpha_{(\sigma(b))}\}| = n$, are seen to have won **Round 1**, and will therefore progress on to **Round 2** of the PoW.
- The lowest *solution sketch* $s^{(\ell)}_{x_1(\sigma(a))(j)}$ will have special significance, as the transactions related to this *solution sketch* will be those that are scribed in the next block in the blockchain.
- For ease of notation we will denote it $\psi_{x_1(j)} = s^{(\ell)}_{x_1(\sigma(a))(j)}$.
- $a, b, c \in \mathbb{Z}^+$, $1 \leq a, b, c \leq m$ and $a \neq b \neq c$.
  $s^{(\ell)}_{x_1(\sigma(a))(j)} = \mathsf{min}\{s^{(\ell)}_{x_1(1)(j)}, \ldots, s^{(\ell)}_{x_1(m)(j)}\}$. This is the lowest in the set.
  $s^{(\ell)}_{x_1(\sigma(b))(j)} = \mathsf{min}^{n^{th}}\{s^{(\ell)}_{x_1(1)(j)}, \ldots, s^{(\ell)}_{x_1(m)(j)}\}$. This is the n^{th} lowest in the set.
  $s^{(\ell)}_{x_1(\sigma(c))(j)} = \mathsf{max}\{s^{(\ell)}_{x_1(1)(j)}, \ldots, s^{(\ell)}_{x_1(m)(j)}\}$. This the maximum or m^{th} lowest in the set.

Round 2

1. $\mathsf{Gen}_2(x_2)$

- The set of $|\{\alpha_{(\sigma(a))}, \ldots, \alpha_{(\sigma(b))}\}| = n$ miners with the lowest *solution sketches* from **Round 1** and will *generate* the *challenge* $c_{x_2(j)} = \mathcal{G}_{(j)}$ for the Travelling Salesman Optimization Problem as noted in **Definition 4**.
- $\{\alpha_{(\sigma(a))}, \ldots, \alpha_{(\sigma(b))}\} \subseteq \{\alpha_1, \ldots, \alpha_m\}$
- The *challenge* will take the form of a graph distance matrix as follows

$$c_{x_2(j)} = \mathcal{G}_{(j)} = \begin{bmatrix} 0 & d(\omega_1, \omega_2) & d(\omega_1, \omega_3) & \ldots & d(\omega_1, \omega_n) \\ d(\omega_2, \omega_1) & 0 & d(\omega_2, \omega_3) & \ldots & d(\omega_2, \omega_n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ d(\omega_n, \omega_1) & d(\omega_n, \omega_2) & d(\omega_n, \omega_3) & \ldots & 0 \end{bmatrix}$$

2. $\mathsf{Solve}^*_2(c_{x_2(j)})$

- The set of $\{\alpha_{(\sigma(a))}, \ldots, \alpha_{(\sigma(b))}\}$ miners will choose a Solve_2 algorithm of their choice as indicated by the $*$ superscript in order to attempt to find an optimal tour of cities for the constructed TSP *challenge* $c_{x_2(j)} = \mathcal{G}_{(j)}$.
- As the *solution sketch* will need to be sorted in ascending order, the value of the sum must be the first part of the data output.

$$s^{(\ell)}_{x_2(i)(j)} = \Bigg[\underbrace{\mathsf{min}^{[t_c]} \left\{ \left(\sum_{i=1}^{n-1} d(\omega_{\pi(i)}, \omega_{\pi(j)}) \right) + d(\omega_{\pi(n)}, \omega_{\pi(1)}) \right\}}_{lowest\ sum}, \underbrace{\left\langle \omega_{\pi(1)}, \omega_{\pi(2)}, \ldots, \omega_{\pi(n)} \right\rangle}_{permutation\ giving\ lowest\ sum} \Bigg].$$

- Therefore $s^{(\ell)}_{x_2(i)(j)}$ can be broken down with the superscripts (1) and (2) as follows:
  $s^{(\ell)(1)}_{x_2(i)(j)} = \mathsf{min}^{[t_c]} \left\{ \left(\sum_{i=1}^{n-1} d(\omega_{\pi(i)}, \omega_{\pi(j)}) \right) + d(\omega_{\pi(n)}, \omega_{\pi(1)}) \right\}$
  $s^{(\ell)(2)}_{x_2(i)(j)} = < \omega_{\pi(1)}, \omega_{\pi(2)}, \ldots, \omega_{\pi(n)} >$.
- Solve^*_2 will be run until time $t_c = t_s + 10$ minutes.
- Note that min indicates the absolute minimum, whereas $\mathsf{min}^{[t_c]}$ indicates the minimum obtained by the expiry of time t_c.

3. Propagate *solution sketches*

- At time t_c, each miner will propagate to the other $n-1$ miners, their optimal *solution sketch* $s^{(\ell)}_{x_2(i)(j)}$ from their chosen $\mathsf{Solve}^*_2(c_{x_2(j)})$ algorithm.

4. Sort *solution sketches*

- Miners $\{\alpha_{(\sigma(a))}, \ldots, \alpha_{(\sigma(b))}\}$ will then sort the *solution sketches* $\{s^{(\ell)}_{x_2(\sigma(a))(j)}, \ldots, s^{(\ell)}_{x_2(\sigma(b))(j)}\}$ in *ascending order*.
- The value which achieves $s^{(\mathcal{L})}_{x_2(i)(j)} = \mathsf{min}\{s^{(\ell)}_{x_2(\sigma(a))(j)}, \ldots, s^{(\ell)}_{x_2(\sigma(b))(j)}\}$, will tentatively win this round, subject to verification in Step 5.

5. $\mathsf{Verify}_2(c_{x_2(j)}, s^{(\mathcal{L})}_{x_2(i)(j)})$

- Miners $\{\alpha_{(\sigma(a))}, \ldots, \alpha_{(\sigma(b))}\}$ will then verify the *solution sketch* $s^{(\mathcal{L})}_{x_2(i)(j)}$ as follows.
- Step 1. Look up associated distances from $s^{(\mathcal{L})(2)}_{x_2(i)(j)}$, namely $d(\omega_{\pi(1)}, \omega_{\pi(2)}), d(\omega_{\pi(2)}, \omega_{\pi(3)}), \ldots, d(\omega_{\pi(n-1)}, \omega_{\pi(n)}), d(\omega_{\pi(n)}, \omega_{\pi(1)})$, directly from the constructed *challenge* $c_{x_2(j)} = \mathcal{G}_{(j)}$.
- Step 2. Sum over all the distances in Step 1.
- Step 3. IF *Sum in Step* 2 $= s^{(\mathcal{L})(1)}_{x_2(i)(j)}$, then,
  OUTPUT: *accept*
  ELSE OUTPUT: *reject*.
- If and only if $s^{(\mathcal{L})}_{x_2(i)(j)}$ is a valid *solution sketch* will the corresponding miner win the block and therefore mint the new currency and obtain the transaction fees.
- The $\mathsf{Recon}_2(c_{x_2(j)}, s^{(\mathcal{L})}_{x_2(i)(j)})$ algorithm is the same as Verify_2, and the $f(x)$ to *reconstruct* in this case is $f(x_2(i)(j)) = s^{(\mathcal{L})(1)}_{x_2(i)(j)}$.

6. Commit Transactions, Output and Loop

- The transactions associated with $\psi_{x_1(j-1)} = s^{(\ell)}_{x_1(\sigma(a))(j-1)}$ will be those scribed into the current block (j) (see Figure 7).
- The transactions associated with $\psi_{x_1(j)}$ will be scribed into the subsequent block $(j+1)$.
- The $\{\alpha_{(\sigma(a))}, \ldots, \alpha_{(\sigma(b))}\}$ miners will then output $h(h(c_{x_2(j)} || s^{(\mathcal{L})}_{x_2(i)(j)})) = \psi_{x_2(j)}$ to all $\{\alpha_1, \ldots, \alpha_m\}$ miners and then loop back to **Round 1** Step 1. ♠

We conclude this section with our final theorem, which we seek to prove in Section 5.

Theorem 4: The *Conquering Generals Proof of Work* outlined in **Definition 14** is a *Byzantine Generals Proof of Useful Work* as defined in **Definition 12**. ♠

4.3 Insight to the Name "The Conquering Generals"

The inspiration for the name "The Conquering Generals" came from merging the ideas from the TSP and the Byzantine Generals Problem. We imagine a group of generals with a conquering mission. **Round 1** can be anecdotally seen as the collection of their m officers performing reconnaissance on the cities they believe are strategically worth

conquering. Those with the best evidence will have their chosen city added to a list of those to be conquered. We liken the reconnaissance to the work performed by the officers, however in our actual design the cities are chosen randomly, rather than strategically. At the end of **Round 1** n cities chosen for attack are shortlisted. **Round 2** can be seen as the next part of the strategy. The generals must decide the most efficient route in which to launch their campaign - this is naturally how we overlay the TSP to the scenario, but instead of the innocuous Salesman, we have the Conquering Generals. Of course, the time limit for their planning is essential, they must begin their campaign with haste, so they cannot find the absolute optimal path, but must settle for the most optimal path they can define in a reasonable amount of time.

4.4 Details, Benefits and Limitations of Proposed Proof of Work

In this subsection we work through each of the steps in **Definition 14** to provide further details and examples to aid with the clarity of the system. We also discuss any perceived benefits and possible considerations of the proposed PoW.

4.4.1 Round 1 Step 1 Generate

- **Detail 1:** This step will *generate* this blocks *challenge*. Note that this blocks *challenge* chains to the previous rounds *challenges* and *solution sketches* and therefore continues adding to the blockchain structure. In Bitcoin, there is a *memory pool* of *unconfirmed transactions* awaiting inclusion into the next block. Whilst performing their PoW for the current block, miners also simultaneously prepare for the next block by selecting a number of these *unconfirmed transactions* in order to create a *candidate block*, which has a size limit of 1MB. The *block header* of the *candidate block* equates to this rounds *challenge* $c_{x_1(i)(j)}$. Selection of the *unconfirmed transactions* is an arbitrary decision made by each miner. However each miner must include any *high priority* transactions still outstanding into their candidate block. Each transaction has a *priority metric* associated with it. The *priority metric* is calculated as follows:
  $$\mathcal{P} = \frac{\sum_{i=1}^{n} \mathrm{v}_i \mathrm{a}_i}{\mathrm{S}}$$
  Where, $\mathcal{P}$ = priority metric, v = the value of in the input, (measured in Satoshi's), a = the age of the input, based on the number of blocks that have been committed to the blockchain since the transaction was recorded, s = the size of the transaction, measured in bytes, and n = number of inputs in each transaction.
  Below is a diagram outlining the details for the *challenge* $c_{x_1(i)(j)}$ for the proposed PoW. The marked difference from the current Bitcoin *challenge* is the addition of the $\psi_{x_2(i)(j-1)}$ value in the Block Header from **Round 2**.

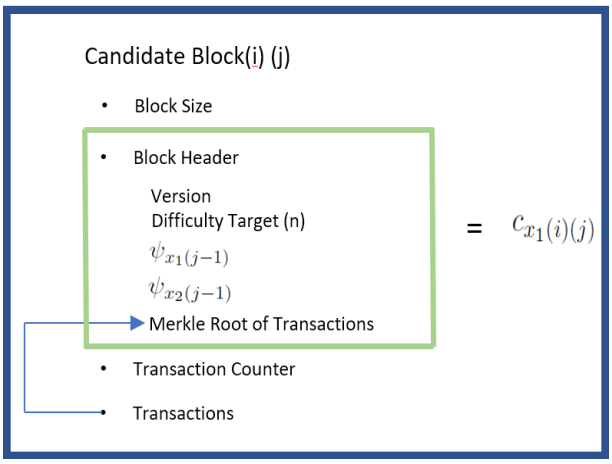

Figure 7: Relationship between each Candidate Block, and the Block Header = $c_{x_1(i)(j)}$.

For further detail on transactions and Merkle Trees see *Mastering Bitcoin*, chapters 7 and 8 [3].

- **Benefit 1:** In our proposal **Detail 1** can allow a mechanism in order to increase the current transaction throughput allowed in each mined block. As the August 1, 2017 hardfork with Bitcoin Cash has shown, transaction throughput is an important topic between Bitcoin Core developers and the mining community. In our proposal we indicate that n miners are seen to have won, **Round 1**. If an extra field were added to *mark* a transaction, miners creating a *candidate block* would have to ensure that all *unconfirmed transactions* were homogenously marked.

- **Example 1:** Say we have 2 bits in the transaction information to indicate a marking as follows

Marking Bits	**Marking Colour**
00	Red
01	Green
10	Blue
11	Yellow

Figure 8: Toy example of a transaction marking scheme for Bitcoin.

In the marking proposal above, each *challenge* would require another subscript to indicate which transaction marking it was going to include into the candidate block. The challenges would then become
$c_{x_1(i)(j)(\text{red})}, c_{x_1(i)(j)(\text{green})}, c_{x_1(i)(j)(\text{blue})}, c_{x_1(i)(j)(\text{yellow})}$.

If, *marking bits* $= \mathcal{M}$, and we also always assume that $|\mathcal{M}| < n$, then it is clear that up to $|\mathcal{M}|$ *candidate blocks* could be mined in each round, ultimately resulting in more transactions per round being committed to the blockchain.

4.4.2 Round 1 Step 2 Solve

- **Detail 1:** The $\mathsf{Solve}_1(c_{x_1(i)(j)})$ algorithm will be utilized until time t_1. The miners will compete to achieve the lowest cryptographic hash value by brute forcing different nonce values which are concatenated to the *challenge* $c_{x_1(i)(j)}$ *generated* in Step 1. We denote concatenation with the || symbol. This is exactly what is done with Bitcoin mining currently, however in this system, there will be no difficulty target. Miners must store their lowest hash value for submission at time t_1.

4.4.3 Round 1 Step 3 Propagate

- **Detail 1:** Currently in Bitcoin, the ability to propagate *solution sketches* is done via the peer-to-peer relationship formed at the bootstrap of each node which connects to TCP port 8333 of the nearest peers that are discovered by querying a predefined set of DNS domain names.

- **Consideration 1:** Currently Bitcoin miners have no requirement for storing their *solution sketches*. As soon as they find a nonce that achieves a hash value which is lower than the difficultly target, they submit this in real-time. However, if we opt for a system where we deterministically submit *solution sketches* at t_1 minutes, then this implies that memory or storage will have to be included in the hardware architecture and miners must adopt a store and forward method rather than a real-time submission method. This is non-trivial as it is not uncommon to have **PH/s** hash rates. ASIC miners could potentially set an internal difficultly target and only store a hash value beating that target, then perform a real-time comparison for any other hash in that also lower than the target, usurping the previous value if the latest value is lower. Finally at time t_1 they can submit their lowest *solution sketch*.

- **Consideration 2:** Whilst utilization of a Timestamp Server was noted in Nakamoto's original paper [45], the requirement to have synchronized time in a PoW with a deterministic mining interval will become even more paramount. Tolerance levels for time drifts between miners will need to be collectively agreed. The prerequisite for synchronizing to appropriate NTP servers was noted in **Definition 13**. Should a miner submit a *solution sketch* with a timestamp that exceeds a particular time tolerance or a system clock that has drifted to far from the other miners, then their submission must be excluded from further participation in the current round.

4.4.4 Round 1 Step 4 Verify

- **Consideration 1:** This verification step will require many more verifications than the current Bitcoin PoW. Bitcoin currently only requires one verification for each block committed to the blockchain, however as we have m miners online, we require m verifications. As ASIC hardware may have **PH/s** hash rates, this should not present any significant time overheads.

4.4.5 Round 1 Step 5 Sort

- **Detail 1:** For simplicity, we assume that all *solution sketches* in Step 4 are valid. However, if they are not, then an adjustment was made in the previous step to

exclude the invalid solutions. This will naturally work, by decrementing the value of m by 1 for each invalid solution. By removing invalid *solution sketches* in the previous step, we remove the requirement to waste computational effort in sorting them. Returning to the earlier assumption, once each miner has verified the m *solution sketches*, they will then run a sorting algorithm to sort the *solution sketches* in ascending order.

- **Consideration 1:** It is also worth noting that mining pools, which may consist of hundreds or even thousands of ASICs, are represented as one miner on the network, despite their hash rate being far greater than individual hobby miners. As there is a one-to-one correlation between mining pools to their respective outputs, this proxy design does not affect the construction of the PoW. The entire mining pool will output one solution, as if they were an individual hobby miner, so this complexity is of no consequence to the design considerations.

- **Consideration 2:** Currently Bitcoin mining is done on specific ASIC hardware. These ASIC's will not be designed to run sorting algorithms as they have been purposely engineered to exclusively run the SHA-256 cryptographic hashing algorithm. At this point, as indicated in **Definition 13** the miners will need to be able to return to computational architecture that can run a sorting algorithm.

- **Consideration 3:** In order to be competitive, miners should opt to use an optimal sorting algorithm of their choice, ideally with average case performance of $O(mlogm)$.

- **Consideration 4:** Clearly the computational overheads to sort m of the *solution sketches* goes beyond that which is currently required in Bitcoin. We hope that the other perceived benefits make this worthwhile.

4.4.6 Round 2 Step 1 Generate

- **Detail 1:** The construction of the *challenge* $c_{x_2(j)} = \mathcal{G}_{(j)}$, where $\mathcal{G}$ represents a graph distance matrix, will occur as follows:
For each of the n lowest *solution sketches* validated from **Round 1**, the 128 LSB's from each hash output will be laid over a two-dimensional Euclidean space, with dimensions $2^{64} \times 2^{64}$ where each block will be 1 *unit* $\times$ 1 *unit*. We also assume that the properties outlined in **Definition 5** and its **Addendum** apply.
Let us assume that one of the *solution sketches* from **Round 1** is $s^{(\ell)}_{x_1(i)(j)} = 0000000000000000B \ldots E360568F29EF54005F2AE8A787A28759E$.
Then we use the 128 LSB's as follows:
$\underbrace{360568F29EF54005}_{x-coordinate}\underbrace{F2AE8A787A28759E}_{y-coordinate}$.

- **Detail 2:** Once all of the cities $\{\omega_1, \ldots, \omega_n\}$ have been overlaid on the plane, then their distances must be calculated. With the TSP we assume that the graph is complete and there are $\frac{n(n-1)}{2}$ edges in which we must determine the distances between each of the cities. For each rounds constructed instance of the TSP, distance can either be calculated as the ℓ_1 or ℓ_2 norm. The distance measurement used will depend on the final digit of the lowest hash $\psi_{x_1(j)}$ as follows:

Final Digit	Distance measurement
1,3,5,7,9,B,D,F	ℓ_1 norm
0,2,4,6,8,A,C,E	ℓ_2 norm

Figure 9: Distance measurement based on final digit of $\psi_{x_1(j)}$.

Say we have city $\omega_i = (x_i, y_i)$ and city $\omega_j = (x_j, y_j)$. To calculate the ℓ_1 norm (also known as the Manhattan distance) between the cities the formula will be $d(\omega_i, \omega_j) = |x_i - x_j| + |y_i - y_j|$. However if the ℓ_2 norm is required then using Pythagoras, the formula to calculate the distance between each pair of cities is $d(\omega_i, \omega_j) = \sqrt{|x_i - x_j|^2 + |y_i - y_j|^2}$.

Once all $\frac{n(n-1)}{2}$ edges between the cities have been calculated we *generate* the *challenge* which was defined in **Definition 14**.

- **Detail 3:** Note that unlike **Round 1** Step 1, the *challenge* $c_{x_2(j)}$ is the same for all miners. In **Round 1** Step 1 each miner had the ability to create their own *candidate block*. However, as the $c_{x_2(j)}$ is linked explicitly to the lowest hashes from the **Round 1** Step 5, it is uniform, hence why no (i) subscript exists. Therefore, the randomness of the *generation* for the entire PoW construction comes solely from the Gen_1 algorithm.

- **Consideration 1:** If calculating the ℓ_2 norm form of distance, then $d(\omega_i, \omega_j) \in \mathbb{R}^+$. Therefore, the distance between each pair of cities will be found by taking square roots which is a floating point arithmetic operation. It is therefore recommended to follow the guidance given in section 4 *Attributes and rounding* of the IEEE 754-2008 Standard for Floating-Point Arithmetic standard [34]. Which rounding attributes to adhere by is beyond the scope of this paper but must be agreed by all miners should this PoW be adopted. Recall from Section 2.2.6, that the status of the Euclidean TSP was not known to be in NP if no rounding were to take place on the ℓ_2 norm, hence appropriate rounding for the construction of $c_{x_2(j)}$ is essential.

- **Consideration 2:** It is important to note that, because the *challenge* $c_{x_2(j)}$ will form part of the next **Round 1 Step 1** *challenge*, the data structure (down to the detail of the characters used) of $c_{x_2(j)}$ must be completely consistent amongst all miners. The reason for this was made clear in **Round 2** Step 6, as it formed part of a cryptographic hash output $h(h(c_{x_2(j)}||s^{(\mathcal{L})}_{x_2(i)(j)})) = \psi_{x_2(j)}$.

- **Consideration 3:** Similar to the **Round 1** Step 5 **Consideration 2** current ASIC mining hardware will not be designed to perform floating point arithmetic operations. However as **Definition 13** indicates this is a requirement and miners must be able to return to computational architecture that supports these operations in order to *generate* the **Round 2** *challenge*.

- **Consideration 4:** Similar to **Round 1** Step 5 **Consideration 4** there is a computational overhead when compared to current Bitcoin PoW in order to *generate* the proposed **Round 2** *challenge*. The hope is that the perceived benefits make the proposal compelling in spite of the extra complexity.

- **Example 2:** We provide a toy example below where $n = 5$, and instead of a $2^{64} \times 2^{64}$ grid we create a $2^4 \times 2^4$ grid. Also, instead of using the 128 LBS's we use only the 8 LSB's. Also note that the final digit of the lowest hash is A,

therefore we take the ℓ_2 norm for the distance measurement, therefore creating a Euclidean TSP instance.

Assume the n lowest *solution sketches* in *ascending* order from **Round 1** are:

$s^{(\ell)}_{x_1(\sigma(a))(j)} = 00000000000000008\ldots F6\underbrace{2}_{x_1}\underbrace{A}_{y_1}$. So $\omega_1 = (2, A)_{16} = (2, 10)_{10}$

$s^{(\ell)}_{x_1(\sigma(a+1))(j)} = 00000000000000013\ldots 8B\underbrace{7}_{x_2}\underbrace{D}_{y_2}$. So $\omega_2 = (7, D)_{16} = (7, 13)_{10}$

$s^{(\ell)}_{x_1(\sigma(a+2))(j)} = 0000000000000008A\ldots C8\underbrace{9}_{x_3}\underbrace{A}_{y_3}$. So $\omega_3 = (9, A)_{16} = (9, 10)_{10}$

$s^{(\ell)}_{x_1(\sigma(b-1))(j)} = 00000000000000436\ldots 50\underbrace{5}_{x_4}\underbrace{6}_{y_4}$. So $\omega_4 = (5, 6)_{16} = (5, 6)_{10}$

$s^{(\ell)}_{x_1(\sigma(b))(j)} = 00000000000003337\ldots 25\underbrace{C}_{x_5}\underbrace{2}_{y_5}$. So $\omega_5 = (C, 2)_{16} = (12, 2)_{10}$

Below is a visual of the construction above:

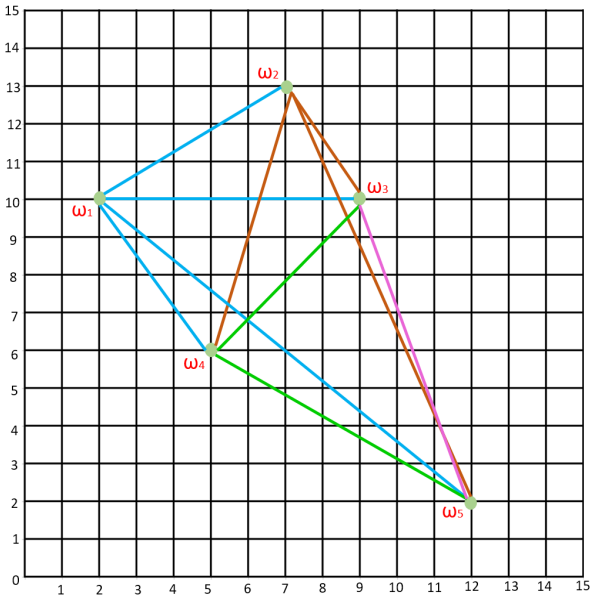

Figure 10: Visual of toy example of the Conquering Generals PoW **Round 2** Step 1 $c_{x_2(j)} = \mathcal{G}_{(j)}$ *generation.*

- **Consideration 5:** In order to *generate* this rounds *challenge* we depend on the use of the SHA-256 cryptographic hashing algorithm. Whilst NIST's Secure Hash Standard [46] provides us a certain level of assurance that the standard security properties outlined in **Definition 8** are adequately met we must note that the construction of this challenge is only pseudo-random, as SHA-256 is not a truly random idealized hash function as outlined in **Definition 9** *The Random Oracle Model* and therefore provides us no guarantee that **Theorem 3** applies. Furthermore, only a subset of the hash value is used in the construction, namely the 128 LSB's, and any cryptographic weaknesses when considering randomness extraction in this manner have not been scrutinized. However, if we return to Dworks' paper, the recommendation for a Pricing function to use the *cracked* Ong-Schnorr-Shamir signature scheme was explored. She notes that in the context of a Pricing function the consequences of using a *broken* function is not severe [25]. This does not absolve PoW algorithms from employing the use of

secure cryptographic hash functions, however the perceived magnitude of negative impact of using a non-idealized hash function in a PoW can be seen to be less caustic.

- **Consideration 6:** At first glance for a *reasonably* large n the possible number of complete graphs to be constructed for the TSP will be $\binom{n}{2^{128}}$. To provide context we note that $\binom{17}{2^{128}} > 2^{2048}$. As there are generally $m \approx 8000$ Bitcoin miners online, we can certainly choose n, where $n \leq m$, sufficiently large enough to ensure that the number of complete graphs make exhaustive construction and precomputation to optimize the TSP infeasible.
  There is however still a consideration to note when constructing this *challenge*. Let us first consider the complete unweighted graphs with adjacency matrices $\mathcal{G}_1$ and $\mathcal{G}_2$ each with n nodes. Recall that in order to prove that $\mathcal{G}_1 \cong \mathcal{G}_2$ we must find a permutation matrix P such that $\mathcal{G}_1 = \mathrm{P}\mathcal{G}_2\mathrm{P}^{\mathrm{T}}$. In the case of two complete graphs each with n nodes, this is trivial as each will have adjacency matrix

$$\mathcal{G}_1 = \mathcal{G}_2 = \begin{bmatrix} 0 & 1 & 1 & \dots & 1 \\ 1 & 0 & 1 & \dots & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & \dots & 0 \end{bmatrix}$$

  and it is clear to see that $\mathrm{P} = \mathrm{P}^{\mathrm{T}} = \mathrm{I_n}$. In other words, all complete graphs with n nodes can be trivially proven to be isomorphic with their adjacency matrices via the identity permutation matrix $\mathrm{I_n}$. In the context of the construction of the TSP, it should intuitive at this point, that although two complete graphs each with n cities are isomorphic, this will certainly not indicate that the *solution sketch* for each *challenge* will therefore be equal. Nonetheless, we need to acknowledge the probability that constructions of the challenge $c_{x_2(j)}$ may be geometrically congruent (a stronger notion than graph isomorphism). We provide three examples below, to demonstrate how graph isomorphism need not pose a problem in terms of precomputation of solutions, but how geometric congruence could *possibly* give one pause to scrutinize the efficacy of the *challenge* creation. We also provide an example of two graphs that are geometrically congruent, however as the final digit differs of the city representing the lowest hash, one takes the ℓ_1 norm and the other takes the ℓ_2 norm as indicated by Figure 9.

- **Example 3**
  We have $\mathcal{G}_1$ on the left and $\mathcal{G}_2$ on the right. We also show the ℓ_2 norm construction in each case for this example as the lowest hash of city ω_1 ends in digit 4 and 2 respectively on the diagrams.

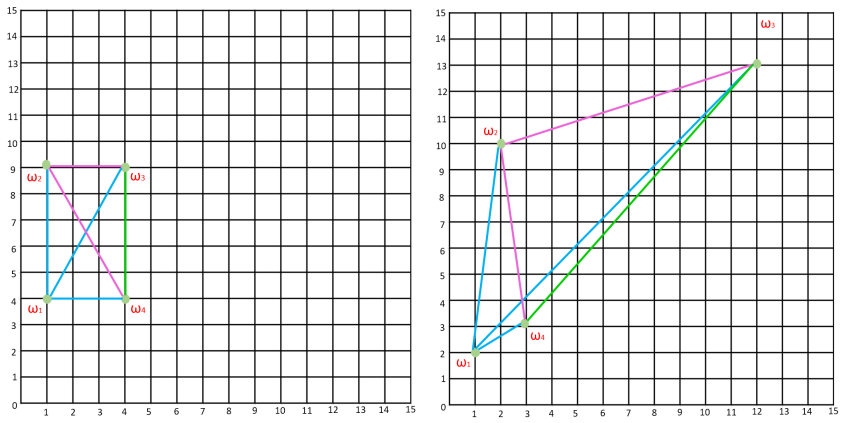

Figure 11: $\mathcal{G}_1 \cong \mathcal{G}_2$, however they are not geometrically congruent and have different solutions to the TSP.

- **Example 4**
  We have $\mathcal{G}_1$ on the left and $\mathcal{G}_3$ on the right. The ℓ_2 norm is taken on $\mathcal{G}_1$ as ω_1 has final digit 4, and on $\mathcal{G}_2$ as ω_1 has final digit $(12)_{10} = (C)_{16}$.

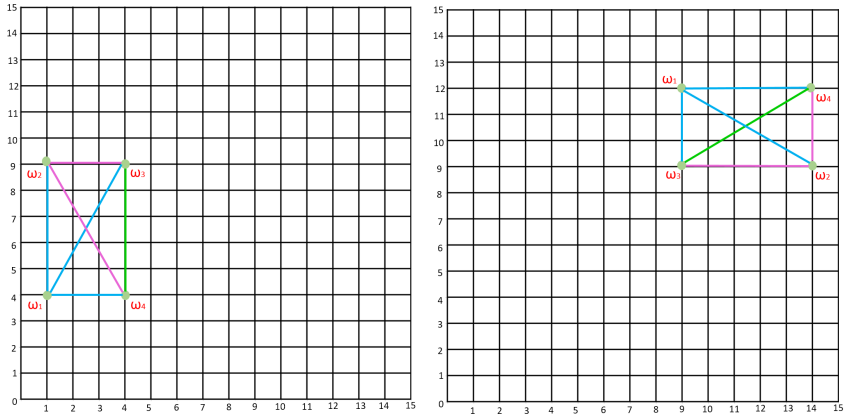

Figure 12: $\mathcal{G}_1 \cong \mathcal{G}_3$, are geometrically congruent and therefore have the same solution to the TSP despite having different hash values.

- **Example 5**
  We have $\mathcal{G}_4$ which provides a Euclidean TSP instance using the ℓ_2 norm as the distance measurement as the lowest hash of city ω_1 ends in digit 4, and $\mathcal{G}_5$ which is geometrically congruent to $\mathcal{G}_4$, but because the lowest hash associated with city ω_1 ends in 3, we measure distance using the ℓ_1 norm.

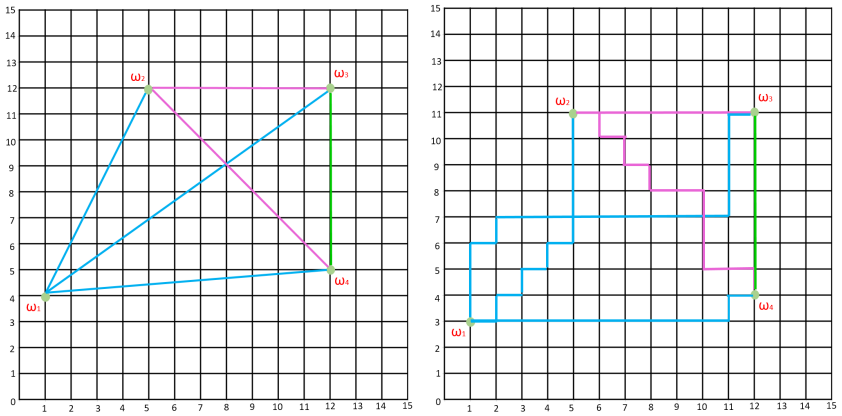

Figure 13: $\mathcal{G}_4 \cong \mathcal{G}_5$, are geometrically congruent but distance measurements differ based on the final digit of the lowest hash associated with city ω_1 in each instance

Note that whilst the ℓ_2 norm between two points on a Euclidean plane is unique, the ℓ_1 norm may take various paths with equal distance as indicated below.

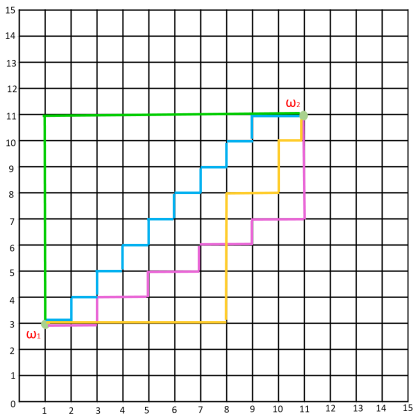

Figure 14: Various ℓ_1 norm paths between ω_1 and ω_2 which are equidistant.

We provide a heuristic argument for the infeasibility for the possibility of precomputation in the case of **Example 4** and **Example 5**. For a miner to be able to precompute all instances of the TSP in our described construction they must be able to:

1. Determine all possible complete graph *shapes* with n cities over the $2^{64} \times 2^{64}$ Euclidean plane and then calculate and construct their graph distance matrices for both ℓ_1 and ℓ_2 norm distance measurements.
2. Have the required amount of storage for all of the graph distance matrices of each possible complete graph *shape*.
3. Have the computational power to be able to perform a fast look up for a particular graph distance matrix upon the construction of the latest challenge $c_{x_2(j)}$.
4. Compute for each *shape*, the optimal solution for the TSP which is known to be NP-Hard as outlined in **Definition 2**.

For item 1 one could liken the infeasibility for a miner to determine every possible mapping of cities which form a unique *shape* on the Euclidean plane to that of performing an exhaustive key search in a block ciphers substitution-permutation network. For item 2 and 3, we have parallels with the Random Oracle Model, the storage requirements and look up times make this possibility infeasible. Finally, with item 4, we recall that the TSP is NP-Hard, so if using an exact algorithm, each computation for sufficiently large n becomes intractable (see Figure 3). If the miner were to opt to use an approximation or heuristic algorithm for item 4 with a polynomial run time, we return to the argument of item 1, proposing that the number of possible *shapes* still provides an astronomical barrier to considering this as a feasible option.

- **Benefit 1:** Recall from Section 2.2.2 that we are constructing an instance of the Travelling Salesman Optimization problem which is known to be a NP-Hard problem, and is furthermore derived from an NP-Complete decision problem in the strong sense. Also, as discussed in Section 2.2.6, our versions of the subcases of the Euclidean TSP and the "Manhattan" TSP are both known to be NP-Hard, and this is in contrast to the Hashcash based Proof of Work in Bitcoin which only has a heuristic belief of hardness based on SHA-256's pseudo-random behaviour [6].

4.4.7 Round 2 Step 2 Solve

- **Benefit 1:** In this step miners have the option to utilize a $\mathsf{Solve}_2^*(c_{x_2})$ algorithm of their choice in order to find the lowest *solution sketch* to the TSP as defined in **Definition 4**. Recall that there is a time constraint imposed on the miners, which expires in each Round at t_c. Therefore, miners will not be competing to find the absolute minimum for the constructed TSP, but rather the optimal tour that can be found until time t_c is reached. Miners may opt to use the current heuristic, approximation and exact algorithms already available (many of which can be found in the Concorde code), however they are also encouraged to create new algorithms or work to improve current ones. In this PoW construction competition for the optimal TSP solution will extend beyond raw computational power and fit into an energy multi-use model. Also, because of **Claim 2** and **Claim 3** in Section 2.2.2 any progression in algorithm design could provide further insight into the **P** versus **NP** Millennium Problem as indicated by **Theorem 1** and **Theorem 2** in Section 2.1.4. In a qualitative sense, as well as satisfying the requirements in **Definition 12**, which will be covered in Section 5, this NP-Hard PoW also requires "work" in an academic context. As Figure 3 illustrates, the more substantial gain to provide a competitive edge in this PoW is in algorithm design, rather than brute force.

- **Consideration 1:** Recall that only n miners compete in this step, whilst the other $m - n$ miners idle their computational power until the process loops back to **Round 1** Step 1. Whilst this may seem like a waste of energy, a computational pool which has the ability to work on the TSP could offer this *"idle"* period to help work on other unsolved TSPLIB instances with unknown optimal tours. Please see the Further Research section for a discussion on this consideration.

- **Consideration 2:** In Section 2.2.4 it was noted that heuristic algorithms currently hold precedent when finding *nearly* optimal tours for the TSP. In fact, it was stated that with a high degree of probability, some Euclidean instances find tours within 1-2% of the expected optimal tour length in the realm of seconds. This may put into question the entire PoW construction, considering that the LKH algorithm has a stated run time of $O(n^{2.2})$ as noted in Figure 5. However, the competition with heuristic algorithms will be in the finding marginal gains beyond the optimal tours that are found to be within 1-2% of the expected optimal tour length. Figure 15 below indicates the area of competition for heuristic algorithms currently used to attempt the TSP. For a cryptosystem used for ensuring the *confidentiality* of data, the success of heuristic algorithms may undermine the concepts of a provable security model required in the system. However, in the context of a PoW, using a heuristic algorithm still requires that energy was expended in order to find a near optimal solution.

- **Consideration 3:** Miners would also have the opportunity to use more than one $\mathsf{Solve}_2^*(c_{x_2})$ algorithm in this step. They may opt to use a collection of polynomial time heuristic algorithms known to be successful at finding TSPLIB tours such as the LKH or perhaps opt to use Nagata's Edge Assembly Crossover Genetic Algorithm as noted on Figure 4 known for heuristically solving the Mono Lisa 100000 city tour [44]. However, as heuristic algorithms only provide a probabilistic guarantee of finding the optimal tour, the Held-Karp exact algorithm as noted on Figure 5, would guarantee finding the optimal tour (albeit in an exponential run time). A miner may wish to supplement heuristic

algorithms with approximation and exact algorithms to hedge their PoW strategy.

- **Benefit 3:** Section 1.3.1 outlined the **51% Attack** risk faced by Hashcash based PoW's. Furthermore, in Section 1.4 the progression from CPU to ASIC based mining was also examined. A TSP based PoW can be seen to be *"ASIC resistant"* as any improvements in algorithm design could not be implemented on an ASIC once it was engineered and deployed. This design could still be vulnerable to the **51% Attack** as miners could still possess the majority of the ASIC based hashrate computational resource in **Round 1** Step 2, but they would also need to possess the majority of the non-ASIC based computational resource and also have the best algorithms in this **Round 2** Step 2 stage. Whilst this PoW doesn't fully address the **51% Attack**, it certainly mitigates it, as all 3 elements would be required to win each round, with the greatest emphasis being on the best algorithm for solving the TSP. This is clearly indicated in the asymmetry noted in Figure 7 of the contents of the Candidate Blocks' Block Header. $\psi_{x_1(j-1)}$ and $\psi_{x_2(j-1)}$ will only belong to an individual miner if the miner had the lowest cryptographic hash in **Round 1** and the lowest TSP tour in **Round 2**. There is no guarantee that the winner of the newly minted cryptocurrency and the miner whose Candidate Block becomes the next block in the blockchain are the same miner.

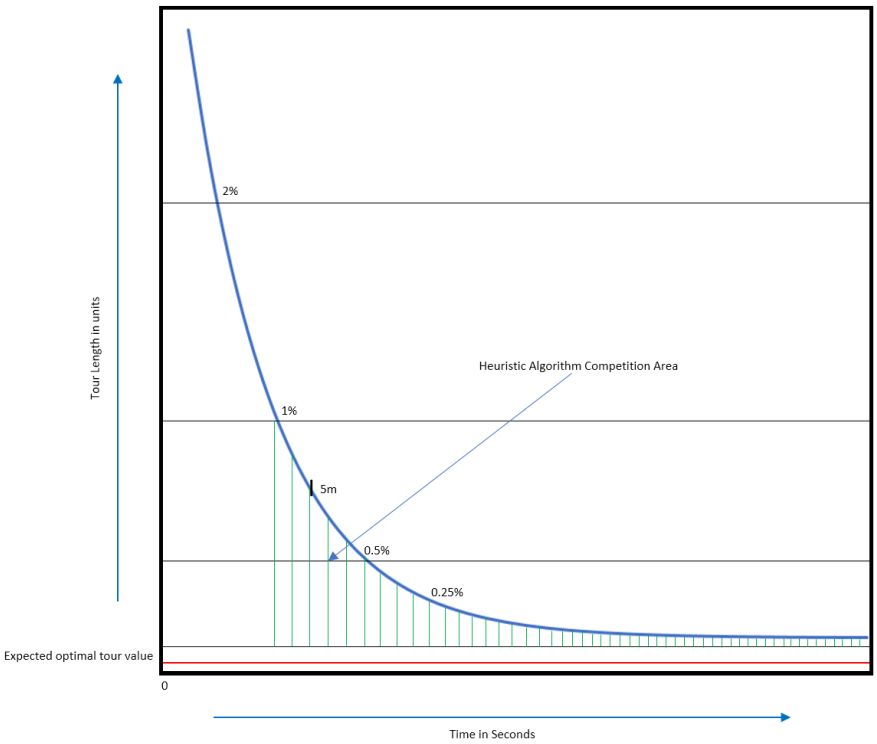

Figure 15: TSP heuristic algorithm solve time versus optimal tour length [21] and outlined competition area.

4.4.8 Round 2 Step 3 Propagate

- **Consideration 1:** Similar to **Round 1 Step 3** as we propose to submit *solution sketches* at a deterministic time interval time synchronization and the adherence to acceptable time drift tolerances for miners becomes essential.

- **Benefit 1:** Two of the current challenges with Bitcoin transactions currently, when compared to EMV payments, are the rate of transaction confirmation and the predictability of the confirmation interval. Deterministic mining rounds can stabilize the latter challenge by making confirmation intervals for outstanding transactions deterministic. One can see on June 8, 2017 the average confirmation time of a transaction was over 2500 minutes.

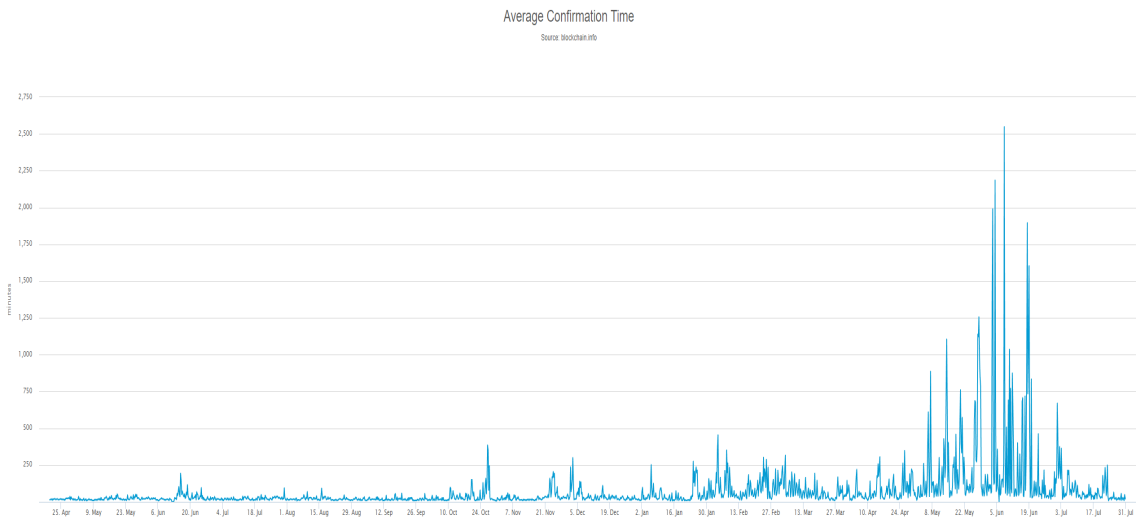

Figure 16: Bitcoin average block confirmation time [12].

Also, as the August 1, 2017, Bitcoin Cash hardfork has shown, transaction throughput and transaction confirmation predictability times are of chief importance to cryptocurrencies. The rate of transaction confirmation was addressed in **Round 1** Step 1 **Benefit 1**, whereas the predictability of confirmation intervals is explicit in the design of this PoW. Having a deterministic PoW interval $[t_s, t_c]$ is also reminiscent of the PoW originally outlined by Jakobsson [35].

4.4.9 Round 2 Step 4 Sort

- **Detail 1:** A discerning reader may have noted in **Round 1** we first verify and then sort, whereas in **Round 2** we reverse this order. The reason for this is in **Round 1** all *solution sketches* need to be verified in order to see if they should even be considered for sorting. Ruling out invalid *solution sketches* prior to sorting will potentially reduce the number of items to sort. Verification takes $O(m)$ time whereas sorting should take $O(mlogm)$ in the average case. However in **Round 2** there is only one winner, so verifying all *solution sketches* is not necessary. So long as the minimum $s^{(\mathcal{L})}_{x_2(i)(j)}$ is valid, there will be no need to waste computation on the verification of other *solution sketches*.

4.4.10 Round 2 Step 5 Verify

- **Detail 1:** This step guarantees the soundness of the *solution sketch* $s^{(\mathcal{L})}_{x_2(i)(j)}$ with the $\mathsf{Verify}_2(c_{x_2(j)}, s^{(\mathcal{L})}_{x_2(i)(j)})$ algorithm. If the solution $s^{(\mathcal{L})}_{x_2(i)(j)}$ is not a valid solution, then the next lowest value from **Round 2** Step 4, must go through the

verification attempt (this will continue until a the lowest *valid* solution is presented). If there is a tie (which could include no valid $s^{(\mathcal{L})}_{x_2(i)(j)}$ being output), the tiebreaking criteria will be the lowest hash from **Round 1**, $s^{(\ell)}_{x_1(i)(j)}$. If there is a tie in **Round 2** and a collision on the lowest hash from **Round 1** then the currency minted will be split between the miners that have won both rounds (or **Round 1** only in the case of no valid solutions being presented in **Round 2**). The probability of the latter scenario is highly unlikely as we assume SHA-256 is a collision resistance cryptographic hash function as indicated in **Definition 8**. These tie breaking conditions are here to ensure that **Round 2** will terminate in all cases.

4.4.11 Round 2 Step 6 Commit Transactions, Output and Loop

- **Consideration 1:** At **Round 2** Step 3 the *solution sketches* are propagated at t_c minutes. However, there are still two steps of sorting and verification prior to looping back to **Round 1** Step 1. As such, the allowed time for the previous steps must be considered prior to the timer resetting. The overhead in time for the previous 2 steps does not prevent transactions taking place, it simply adds a predictable time overhead to each round before looping back to generate the next $c_{x_1(i)(j)}$ challenge. Recall, as indicated in **Round 1** Step 1 **Detail 1**, miners simultaneously collect the transactions for the next *candidate block* whilst they perform the current PoW. This time overhead will also ultimately need to be deterministic, and would need to be defined and adjusted based on m the number of miners online and n the difficulty target and number of miners competing in **Round 2**.

5 Proof: The Conquering Generals is a Proof of Work Algorithm

This section will provide the proof of **Theorem 4**. As the *Conquering Generals Proof of Work* is split into two separate rounds, it is important to note that **Round 1** is a non-interactive Proof of Work, which can actually be defined as delegation of computation for the purposes of constructing the **Round 2** *challenge*. The reason for this is that **Round 1** has no difficulty target associated with it. According to Ball et al *"a PoW is a uPoW that doesn't require soundness (and thus usefulness), and a (non-interactive) verifiable delegation of computation scheme is a uPow that doesn't require hardness" [6].* Therefore, for **Round 1** we need only prove the constraints on **efficiency** and **completeness**. However as **Round 2** is the main departure from current Bitcoin Hashcash based mining, we must prove all constraints in **Definition 12** are satisfied.

5.1 Proof of Theorem 4 for Round 1

Efficiency
For each algorithm we proceed with a direct proof.

- $\mathsf{Gen}_1(x_1)$
  We assume that the number of *unconfirmed transactions* $= N$. For each *unconfirmed transaction* the miner must calculate the *priority metric* $\mathcal{P}$ (Section 4.4.1). As the calculation of the *priority metric* consists of only arithmetic

operations of addition, multiplication and division, this implies a run time of $\tilde{O}(N)$. $\square$

- $\mathsf{Solve}_1(c_{x_1(i)(j)})$
  This part of the PoW is effectively the same as Bitcoin, with the exception that the mining interval is constant, namely $[t_s, t_c]$, which we proposed to be 10 minutes. Also recall that no difficulty target was required in this construction. However for the sake of this proof, should a difficulty target be required for the $\mathsf{Solve}_1(c_{x_1(i)(j)})$ algorithm in this instance, the Hashcash based PoW has a heuristic assumption [6] to take time $\tilde{O}(t(k))$, where $t(k) = 2^k$ and k is the number of bits in the output of the cryptographic hash function [10], which we proposed to be SHA-256. Hence in this case $k = 256$. (As this part of the PoW is not the main body of original work in this paper, we accept the heuristic assumption of the $\tilde{O}(2^k)$ run time). $\square$

- $\mathsf{Verify}_1(c_{x_1(i)(j)}, s^{(\ell)}_{x_1(i)(j)})$
  We assume that there are n nonces associated with the lowest *solution sketches* because $|\{s^{(\ell)}_{x_1(1)(j)}, \ldots, s^{(\ell)}_{x_1(n)(j)}\}| = n$. Confirming if a cryptographic hash of $h(h(c_{x_1(i)(j)}||\eta_{x_1(i)(j)(k)})) = s^{(\ell)}_{x_1(i)(j)}$ runs in $O(1)$ time because the size of input of $(c_{x_1(i)(j)}||\eta_{x_1(i)(j)(k)})$ is generally the same, this implies a run time of $\tilde{O}(n)$. $\square$

- The Recon_1
  We assume that $\mathsf{Recon}_1 = \mathsf{Verify}_1$ which implies a $\tilde{O}(n)$ run time. $\square$

Completeness
We proceed with a direct proof.

- We assume that SHA-256 is a deterministic algorithm as indicated in the Secure Hash Standard [46]. This implies that SHA-256 has no associated randomness. If we also assume that $h(h(c_{x_1(i)(j)}||\eta_{x_1(i)(j)(k)})) = s^{(\ell)}_{x_1(i)(j)}$ this implies that $\mathsf{Pr}\Big[\mathsf{Verify}_1(c_{x_1(i)(j)}, s^{(\ell)}_{1(i)(j)}) = accept\Big] = 1$. $\square$

5.2 Proof of Theorem 4 for Round 2

Efficiency
For each algorithm we proceed with a direct proof.

- $\mathsf{Gen}_2(x_2)$
  We assume that constructing the *challenge* $c_{x_2(j)} = \mathcal{G}_{(j)}$ indicates a requirement to calculate the distances for $\frac{n(n-1)}{2}$ edges for a complete graph. We also assume that either the ℓ_1 or ℓ_2 norms represent each required distance calculation. As calculating either of these norms is an arithmetic operation consisting of addition and subtraction as well as taking square roots in the case of the ℓ_2 norm, this implies a $\tilde{O}(n^2)$ run time. $\square$

- $\mathsf{Solve}^*_2(c_{x_2})$
  Recall that for **Round 2** miners had the option to use a $\mathsf{Solve}^*_2(c_{x_2})$ algorithm of their choice. Figure 5 in Section 2.2.4 outlined a number of different algorithms for attempting to solve the TSP. Therefore the $O(t(n))$ times can be directly read from this table. $\square$

- Verify_2
  We assume that the Verify_2 algorithm requires n distance look ups directly from the *challenge* $c_{x_2(j)} = \mathcal{G}_{(j)}$. As each distance look up references an exact location on the given *challenge*, this implies as $O(1)$ look up time. As there are n edges to look up in each case followed by an arithmetic operation of the addition over all n distances, this implies a $\tilde{O}(n)$ run time. $\square$

- $\mathsf{Recon}_2(c_{x_2(j)}, s^{(\mathcal{L})}_{x_2(i)(j)})$
  We assume that $\mathsf{Recon}_2(c_{x_2(j)}, s^{(\mathcal{L})}_{x_2(i)(j)}) = \mathsf{Verify}_2(c_{x_2(j)}, s^{(\mathcal{L})}_{x_2(i)(j)})$ which implies a $\tilde{O}(n)$ run time. $\square$

Completeness
We proceed with a direct proof.

- We assume that Verify_2 algorithm is a deterministic algorithm with the steps outlined in **Definition 14** Step 5. This implies that Verify_2 has no associated randomness. If we also assume that $Sum\ in\ Step\ 2\ = s^{(\mathcal{L})(1)}_{x_2(i)(j)}$, this implies that $\mathsf{Pr}[\mathsf{Verify}_2(c_{x_2(j)}, s^{(\mathcal{L})}_{x_2(i)(j)}) = accept] = 1$. $\square$

Soundness
We proceed with an indirect proof.

- Assume, by way of contradiction, that s is a *solution sketch* such that $\mathsf{Recon}_2(c_{x_2(j)}, s) \neq f(x_2(i)(j))$ and that $\mathsf{Pr}\Big[c_{x_2} \leftarrow \mathsf{Gen}(x_2) \wedge \exists s\ \mathsf{Recon}(c_{x_2}, s) \neq f(x_2(i)(j)) \mid \mathsf{Verify}(c_{x_2}, s) = accept\Big] > neg(n)$.
  In the case of our PoW this indicates that as per **Round 2** Step 5 that $Sum\ in\ Step\ 2\ \neq s^{(\mathcal{L})(1)}_{x_2(i)(j)}$ implying that $\mathsf{Verify}_2(c_{x_2(j)}, s) = reject$, recalling also that Verify_2 is a deterministic algorithm. But then $\mathsf{Pr}\Big[c_{x_2} \leftarrow \mathsf{Gen}(x_2) \wedge \exists s\ \mathsf{Recon}(c_{x_2}, s) \neq f(x_2(i)(j)) \mid \mathsf{Verify}(c_{x_2}, s) = accept\Big] = 0$, giving us the required contradiction. $\square$

Hardness
We proceed with an indirect proof.

- Assume, by way of contradiction, that there is a $\mathsf{Solve}^{(\mathbf{P})}$, polynomial run time algorithm which finds the **sound** absolute minimal *solution sketches* $s_{x_2i(opt)}$ for all m *challenges* $c_{x_2(1)}, \ldots, c_{x_2(m)}$. In the case of the TSP, as per **Definition 4** $s_{x_2i(opt)} = \left[\mathsf{min}\left\{\left(\sum_{i=1}^{n-1} d(\omega_{\pi(i)}, \omega_{\pi(j)})\right) + d(\omega_{\pi(n)}, \omega_{\pi(1)})\right\}, < \omega_{\pi(1)}, \omega_{\pi(2)}, \ldots, \omega_{\pi(n)} >\right]$.
  Then we assume, by way of polynomial time reduction theory as outlined in Section 2.1.4, that $\mathsf{Solve}^{(\mathbf{P})}$ can also be transformed in polynomial time to solve the NP-Complete problem SAT and all other known NP-Complete problems as noted by **Theorem 1** and **Theorem 2**. But this is not possible, unless $\mathbf{P} = \mathbf{NP}$. $\square$

Usefulness

- The description of how the Recon_2 algorithm reconstructed the $f(x)$ in this PoW was given directly in **Definition 14** where,
  $f(x_2(i)(j)) = s^{(\mathcal{L})(1)}_{x_2(i)(j)} = \mathsf{min}^{[t_c]}\left\{\left(\sum_{i=1}^{n-1} d(\omega_{\pi(i)}, \omega_{\pi(j)})\right) + d(\omega_{\pi(n)}, \omega_{\pi(1)})\right\}$. $\square$

6 Future Research

- The **Round 1 Step 1 Generate, Benefit 1** transaction marking scheme could warrant future research as a means to increase the transaction throughput for Bitcoin. The details for how to include such a marking scheme into *candidate blocks* and the blockchain would require further technical analysis to ensure a seamless construction.

- The **Round 2 Step 1 Generate, Consideration 5 and Consideration 6** could both benefit from further research. One could research the impact of extracting randomness from only particular bits of a cryptographic hash value, rather than the entire hash, and see if under scrutiny this would still provide an appropriate source of pseudo-randomness when constructing an NP-Hard problem. Also the heuristic arguments in **Consideration 6** could benefit from a full combinatorial analysis to ensure that exhaustive search for each instance of the TSP is indeed computationally infeasible.

- The attempted use of NP-Hard problems for cryptosystems was most notably explored with the Merkle-Hellman knapsack cryptosystem in 1978 [43]. However, a polynomial time algorithm was found which could break the cryptosystem in 1984 by Shamir [51]. As stated by Dwork, and noted in Section 4.4.6, *broken* cryptosystems need not be precluded for use in PoW algorithms. Based on the construction in this paper, further research could be done to construct other NP-Hard problems based on the LSB's of cryptographic hash functions. If a collection of NP-Hard PoW's were created such as those in Figure 2 and beyond, the combinatorial search space would increase and different algorithms for specific problems could also be used in a competitive setting.

- **Round 2 Step 2 Consideration 1** noted that the $m - n$ miners that did not proceed to **Round 2** would *"idle"* and not perform any work to solve the constructed TSP. A recommendation in light of this *"idle"* time would be to provide an *"opt in"* to offer the miners a chance to utilize their computational resource to work on a currently unsolved TSPLIB instance. Also, if fiscal incentive was required, companies requiring solutions to genuine TSP instances for logistics, circuit board drilling, telescope calibration et al could be offered a method to pay miners to solve their instances of the TSP. Privacy concerns could be met by redacting or anonymizing the source of the data as required. The nature of Bitcoin payments makes anonymizing payment details trivial.

7 Final Comments

The underlying open conjecture noted in this research paper is that of the **P** Versus **NP** Problem. Whilst this paper did not seek to make explicit progress on this open problem, what it did explore was the incentive, via cryptocurrency mining, for others to pursue further research towards better optimizing the TSP. We hope that research bridges the gap between the accuracy of the heuristic and approximation algorithms and the efficiency of the exact algorithms.

We end with a quote from Scott Aaronson Professor of Computer Science at the University of Texas [1]:

"If $\boldsymbol{P = NP}$*, then the world would be a profoundly different place than we usually assume it to be. There would be no special value in 'creative leaps', no fundamental gap between solving a problem and recognizing the solution once its found. Everyone who could appreciate a symphony would be Mozart; everyone who could follow a step-by-step argument would be Gauss; everyone who could recognize a good investment strategy would be Warren Buffett. Its possible to put the point in Darwinian terms: if this is the sort of universe we inhabited, why wouldnt we already have evolved to take advantage of it? (Indeed, this is an argument not only for* $\boldsymbol{P \neq NP}$*, but for NP-complete problems not being efficiently solvable in the physical world.)"*

8 Glossary

- **altcoin** - Alternative Coins, Alternative Cryptocurrencies
- **ASIC** - Application-Specific Integrated Circuit
- **BTC** - ticker for Bitcoin Currency
- **CPU** - Central Processing Unit
- **DNS** - Domain Name System
- **EMV** - Europay, Mastercard, Visa
- **FPGA** - Field-Programmable Gate Array
- **GMT** - Greenwich Mean Time
- **GPU** - Graphics Processing Unit
- **IEEE** - Institute of Electrical and Electronics Engineers
- **IPSec** - Internet Protocol Security
- **LP** - Linear Programming
- **LSB** - Least Significant Bits
- **nonce** - Number Used Once
- **NIST** - National Institute of Standards and Technology
- **NTP** - Network Time Protocol
- **PoW** - Proof of Work
- **PTAS** - Polynomial Time Approximation Scheme
- **RAND** - Research and Development Corporation
- **Satoshi** - smallest denomination of a Bitcoin. 1 Satoshi = 10^{-8} Bitcoin
- **SHA** - Secure Hash Algorithm
- **SSH** - Secure Shell
- **SSRP** - Sum of Square Roots Problem
- **TCP** - Transmission Control Protocol
- **TLS** - Transport Layer Security
- **TSP** - Travelling Salesman Problem
- **TTP** - Trusted Third Party
- **USD** - ticker for United States Dollar

References

[1] Aaronson, Scott. *Reasons to believe*. Shtetl-Optimized, The Blog of Scott Aaronson, http://www.scottaaronson.com/blog/?p=122, 2006. Accessed June 2017.

[2] Anderson, Nate. *Mining bitcoins takes power, but it isn't an 'environmental disaster'*. Wired, http://www.wired.co.uk/article/bitcoin-environmental-disaster, 2013. Accessed June 2017.

[3] Antonopoulos, Andreas M. *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media, Inc, 2015.

[4] Arora, Sanjeev. *Polynomial Time Approximation Schemes for Euclidean TSP and other Geometric Problems*. Journal of the ACM, Volume 45, Issue 5, Pages 753-782, 1998.

[5] Back, Adam. *Hashcash - A Denial of Service Counter-Measure*. Tech. Rep, http://www.hashcash.org/papers/hashcash.pdf, 2002.

[6] Ball, Marshall. Rosen, Alon. Sabin, Manuel. Vasudevan, Prashant Nalini. *Proof of Useful Work*. In submission, 2017.

[7] Bellare, Mihir. Rogaway, Phillip. *Random Oracles are Practical: A Paradigm for Designing Efficient Protocols*. CCS '93 Proceedings of the 1st ACM conference on Computer and communications security, Pages 62-73, 1993.

[8] Bitcoin Cash. *Bitcoin Cash Peer-to-Peer Electronic Cash*. https://www.bitcoincash.org, 2017. Accessed August 2017.

[9] Bitcoin Developer Guide. *Block Height and Forking*. https://bitcoin.org/en/developer-guide#block-height-and-forking, 2017. Accessed July 2017.

[10] Bitcoin Wiki. *Hashcash*. https://en.bitcoin.it/wiki/Hashcash, 2017. Accessed August 2017.

[11] Bitnodes. *Global Bitcoin Nodes Distribution*. https://bitnodes.21.co, Accessed July 2017.

[12] Blockchain Info. *Average Confirmation Time*. https://blockchain.info/charts/avg-confirmation-time?timespan=2years, 2017. Accessed July 2017.

[13] Castro, Miguel. Liskov, Barbara. *Practical Byzantine Fault Tolerance*. OSDI '99 Proceedings of the third symposium on Operating systems design and implementation, Pages 173-186, 1999.

[14] Chepurnoy, Alexander. Duong, Tuyet. Fan, Lei. Zhou, Hong-Sheng. *TwinsCoin: A Cryptocurrency via Proof of Work and Proof-of-Stake*. In Cryptology ePrint Archive, Report 2017/232, 2017.

[15] Christofides, Nicos. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Technical Report 388, Grad School of Industrial Administration, CMU, 1976.

[16] Coin Desk. *CEX.IO Slow to Respond as Fears of 51% Attack Spread.* https://www.coindesk.com/cex-io-response-fears-of-51-attack-spread, 2014. Accessed August 2017.

[17] Cowen, Lenore. *Lecture 3: The Travelling Salesman Problem.* Comp 260: Advanced Algorithms, Tuffts University, 2011.

[18] Coin Market Cap. *CryptoCurrency Market Capitalizations.* https://coinmarketcap.com. Accessed July 2017.

[19] Cook, Stephen A. *The Complexity of Theorem-Proving Procedures.* STOC '71 Proceedings of the third annual ACM symposium on the Theory of Computing, Pages 151-158, 1971.

[20] Cook, Stephen A. *The P Versus NP Problem* Manuscript for the Clay Mathematics Institute Millennium Prize Problems, http://www.claymath.org/millennium-problems/p-vs-np-problem, 2000.

[21] Cook, William J. *In Pursuit of the Travelling Salesman - Mathematics at the Limits of Computation.* Princeton University Press, 2012.

[22] Cook, William J. *Concorde TSP Solver.* University of Waterloo, Department of Combinatorics and Optimization, http://www.math.uwaterloo.ca/tsp/concorde.html. Accessed August 2017.

[23] Cormen, Thomas H. Leiserson, Charles E. Rivest, Ronald L. Stein, Clifford. *Introduction to Algorithms - Third Edition.* The MIT Press, 2009.

[24] Dai, Wei *b-money.* http://www.weidai.com/bmoney.txt, 1998. Accessed June 2017.

[25] Dwork, Cynthia. Naor, Moni. *Pricing via Processing or Combatting Junk Mail.* CRYPTO '92 Proceedings on the 12th Annual International Cryptology Conference on Advances in Cryptology, Volume 740, Pages 139-147, 1992.

[26] Feistel, Horst. *Cryptography and Computer Privacy.* Scientific American, Volume 228, Issue 5, Pages 15-23, 1973.

[27] Gapcoin.org. *What is Gapcoin?.* http://gapcoin.org/index.php, 2014. Accessed July 2017.

[28] Garey, Michael R. Graham, Ronald L. Johnson, David S. *Some NP-Complete Geometric Problems.* STOC '76 Proceedings of the Eighth Annual ACM Symposium on Theory of Computing, Pages 10-22, 1976.

[29] Garey, Michael R. Johnson, David S. *Computers and Intractability: A Guide to the Theory of NP Completeness.* W.H. Freeman and Company, 1979.

[30] Geeks for Geeks. *NP-Completeness Set 1 (Introduction).* http://www.geeksforgeeks.org/np-completeness-set-1, 2013. Accessed July 2017.

[31] Green, Matthew. *What is the Random Oracle Model and why should you care?.* Cryptography Engineering Blog, https://blog.cryptographyengineering.com/2011/09/29/what-is-random-oracle-model-and-why-3, 2011. Accessed July 2017.

[32] Held, Michael. Karp, Richard. *A Dynamic Programming Approach to Sequencing Problems*. Journal of the Society for Industrial and Applied Mathematics, Vol 10, No 1, Pages 196-210, 1962.

[33] Helsgaun, Keld. *An Effective Implementation of the Lin-Kernighan Travelling Salesman Heuristic*. European Journal of Operational Research, Volume 126, Pages 106-130, 2000.

[34] Institute of Electrical and Electronics Engineers, Inc. *IEEE Standard for Floating-Point Arithmetic*. Revision of IEEE Std 754-1985, 2008.

[35] Jakobsson, Markus. *Proof of Work and Bread Pudding Protocols (Extended Abstract)*. CMS '99 Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks, Volume 23, Pages 258-272, 1999.

[36] Karaman, Sertac. *Soundness and Completeness of State Space Search*. https://ocw.mit.edu/courses/aeronautics-and-astronautics/16-410-principles-of-autonomy-and-decision-making-fall-2010/lecture-notes/MIT16_410F10_lec04.pdf, 2010. Accessed August 2017.

[37] Karp, Richard. *Reducibility Among Combinatorial Problems*. IBM Research Symposia Series, Complexity of Computer Computations, Pages 85-103, 1972.

[38] Kolata, Gina. *"Math Problem, Long Baffling, Slowly Yields"*. New York Times, http://www.nytimes.com/1991/03/12/science/math-problem-long-baffling-slowly-yields.html?pagewanted=all,
1991.
Accessed July 2017.

[39] King, Sunny. Nadal, Scott. *Primecoin: Cryptocurrency with prime number proof of work*. Tech. Rep, https://peercoin.net/assets/paper/peercoin-paper.pdf, 2013.

[40] Lamport, Leslie. Shostak, Robert. Pease, Marshall. *The Byzantine Generals Problem*. ACM Transactions on Programming Languages and Systems, Volume 4, Issue 3, Pages 382-401, 1982.

[41] Laurie, Ben. Clayton, Richard. *'Proof of Work' Proves Not to Work*. IN WEAS 04, 2004.

[42] Litecoin Wikipage. *Litecoin Scrypt Hashing*. https://litecoin.info/Scrypt, 2015. Accessed August 2017.

[43] Merkle, Ralph. Hellman, Martin. *Hiding information and signatures in trapdoor knapsacks*. IEEE Transactions on Information Theory, Volume 25, Issue 5, 1978.

[44] Nagata, Yuichi. *A Powerful Genetic Algorithm Using Edge Assembly Crossover for the Travelling Salesman Problem*. INFORMS Journal on Computing, Volume 25, Issue 2, 2013, Pages 346-363, 2013.

[45] Nakamoto, Satoshi. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Tech. Rep, https://bitcoin.org/bitcoin.pdf, 2008.

[46] National Institute of Standards and Technology. *Secure Hash Standard* Federal Information Processing Standards Publication,
http://dx.doi.org/10.6028/NIST.FIPS.180-4, 2015.

[47] Papadimitriou, Christos H. *The Euclidean Travelling Salesman Problem is NP-Complete*. Theoretical Computer Science, Volume 4, Pages 237-244, 1977.

[48] Redman, Jamie. *Bitcoin Developers Propose Changing Proof of Work Algorithm* Bitcoin News, https://news.bitcoin.com/bitcoin-developers-changing-proof-work-algorithm, 2017. Accessed July 2017.

[49] Reicoin.org *What is Reicoin?*. http://riecoin.org/index.html, 2014. Accessed July 2017.

[50] Reinelt, Gerhard. *TSBLIB95 - Travelling Salesman Library*. Ruprecht-Karls-Universität Heidelberg, https://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/tsp95.pdf, 2013. Accessed August 2017.

[51] Shamir, Adi. *A polynomial time algorithm for breaking the basic Merkle-Hellman cryptosystem*. SFCS '82 Proceedings of the 23rd Annual Symposium on Foundations of Computer Science, Pages 145-152, 1982.

[52] Stinson, Douglas R. *Cryptography Theory and Practice - Third Edition*. Chapman and Hall/CRC, 2006.

[53] Szabo, Nick. *Formalizing and Securing Relationships on Public Networks*. First Monday, Volume 2, Number 9, http://firstmonday.org/ojs/index.php/fm/article/view/548/469, 1997. Accessed July 2017.

[54] Tromp, John. *Cuckoo Cycle a memory bound graph-theoretic Proof of Work*. Lecture Notes in Computer Science, Financial Cryptography and Data Security, Volume 8976, Pages 49 - 62, 2015.

[55] van Wirdum, Aaron. *BIP 91 Has Locked In. Here's What That Means (and What It Does Not)*. Bitcoin Magazine, https://bitcoinmagazine.com/articles/bip-91-has-activated-heres-what-means-and-what-it-does-not, 2017.
Accessed July 2017.

[56] Wagner, Andrew. *Ensuring Network Scalability: How to Fight blockchain Bloat* Bitcoin Magazine, https://bitcoinmagazine.com/articles/how-to-ensure-network-scalibility-fighting-blockchain-bloat-1415304056, 2014.
Accessed July 2017.