Journal of
Risk and Financial Management

MDPI

Article

Inflation Propensity of Collatz Orbits: A New Proof-of-Work for Blockchain Applications

Fabian Bocart

Independent Researcher, Jackson Height, NY 11372, USA; fabian.bocart@gmail.com

Received: 20 September 2018; Accepted: 19 November 2018; Published: 27 November 2018

Abstract: Cryptocurrencies such as Bitcoin rely on a proof-of-work system to validate transactions and prevent attacks or double-spending. A new proof-of-work is introduced which seems to be the first number theoretic proof-of-work unrelated to primes: it is based on a new metric associated to the Collatz algorithm whose natural generalization is algorithmically undecidable: the inflation propensity is defined as the cardinality of new maxima in a developing Collatz orbit. It is numerically verified that the distribution of inflation propensity slowly converges to a geometric distribution of parameter $0.714 \approx \frac{(\pi-1)}{3}$ as the sample size increases. This pseudo-randomness opens the door to a new class of proofs-of-work based on congruential graphs.

Keywords: geometric distribution; collatz conjecture; inflation propensity; systemic risk; cryptocurrency; blockchain; proof-of-work

MSC: 60E05; 62E17

JEL Classification: C46; C65; O39

1. Introduction

A decentralized electronic payment system relies on a ledger of transactions shared on a network. The decentralization of a transaction ledger raises the question of security and integrity of the ledger. In the original Bitcoin protocol, the problem of double-spending or alteration of the ledger is solved by the use of blockchain, a system that requires proof-of-work by a network of computers to confirm transactions. In cryptography, intensive computation as proof-of-work allows one party to verify with little computational effort that a counterparty has spent a large amount of computational effort. The concept was originally developed by Dwork and Naor (1992) as a spam prevention technique. Nakamoto (2008) used, for Bitcoin, a proof-of-work based on Back (2002). The protocol consists in finding a nonce value such that the application of the SHA-256 hashing algorithm to a combination of that nonce and a block of information gives a hash starting with series of zeroes by targetting a given threshold. The idea behind the proof-of-work is that participants have an incentive to cooperate rather than to cheat because the computational power required to cheat is too large. However, as cryptocurrencies became more popular and diverse, an over-reliance on mainstream proof-of-work protocols, such as hashcash-SHA256, Ethash (Wood 2014) or hashcash-Scrypt based proof-of-work (Percival 2009) creates a new type of systemic risk in which a cryptographic breakdown would jeopardize cryptocurrencies that rely on these standard proofs-of-work. A weakness of proofs-of-work in cryptocurrency applications is the threat that a single individual (or a coordinated group) would be able to generate blocks faster than 50% of the network. In that case, this entity would completely control the blockchain-based validation system of transactions. In practice, attacks on hash functions could prevent new transactions or alter past ones. In financial markets, exchanges have the possibility to cancel trades in case of infrastructure breakdown or malfunction. By opposition,

a systemic failure of the proof-of-work system in decentralized cryptocurrency markets could mean the destruction of the whole history of transactions. Potential risks clouding the proof-of-work system include innovation in technology, mathematics and cryptography that could compromise the existing protocols. Proofs-of-work entirely based on existing hash algorithms such as SHA-256 have been under stress in recent years. Rubin (2017) documented a well-known mining optimization ("ASIC-BOOST") that allowed to mine Bitcoin blocks faster than the network average by taking advantage of a technical flaw in SHA-256. A specific optimization of the mining instruments allowed reducing the problem's complexity by exploiting collision attacks on the SHA-256 hash algorithm. The multiplication of proofs-of-work help mitigate this type of hyper-specialized hardware attack. Bitcoin, Ethereum, Bitcoin Cash and Litecoin overwhelmingly dominate the market capitalization of minable coins. Such concentration of the volumes into a few cryptocurrencies represent equally a significant systemic risk. When looking at the top 25 cryptocurrencies by diluted market capitalization (see Table 1), eight of them use Scrypt as underlying hash algorithm for proof-of-work. Introducing new types of proof-of-work is needed to help networks diversifying the protocols in case of increased concentration of hyper-specialized computational power.

Table 1. The 25 top cryptocurrencies as of 15 October 2018 as can be seen on https://onchainfx.com/v/SMT45r.

Name	∼Fully Diluted (Y2050) Marketcap/15 October 2018	Underlying Algorithm
Bitcoin (BTC)	$134,308,812,450	SHA-256
Ethereum (ETH)	$29,787,293,584	SHA-3
Bitcoin Cash (BCH)	$9,225,442,784	SHA-256
Litecoin (LTC)	$4,430,985,913	Scrypt
Dash (DASH)	$2,956,683,098	X11
Monero (XMR)	$2,300,499,210	CryptoNight
ZCash (ZEC)	$2,262,517,311	Equihash
Ethereum Classic (ETC)	$2,161,731,159	SHA-3
Dogecoin (DOGE)	$1,402,169,807	Scrypt
Siacoin (SC)	$564,862,312	Blake-2b
Bitcoin Gold (BTG)	$526,927,423	Equihash
Digibyte (DGB)	$483,402,492	SHA-256 and others
ReddCoin (RDD)	$447,635,857	Scrypt
Bitcoin Diamond (BCD)	$343,664,370	X13
ZenCash (ZEN)	$275,245,426	SHA-3
Verge (XVG)	$229,929,732	Scrypt
Zcoin (XZC)	$194,506,940	Equihash
Monacoin (MONA)	$124,690,762	Scrypt
Syscoin (SYS)	$81,207,881	Scrypt
Zclassic (ZCL)	$67,149,925	Equihash
Vertcoin (VTC)	$53,917,212	Lyra2REv2
Bitcoin Private (BTCP)	$51,124,537	Equihash
LBRY Credits (LBC)	$41,220,511	LBRY
Einsteinium (EMC2)	$25,808,910	Scrypt
GameCredits (GAME)	$13,734,781	Scrypt

So, even though there exist hundreds of different hash functions already, more diversification of proofs-of-work could further mitigate cryptographic risks and improve robustness of the nascent crypto-economy. Several types of proof-of-work have been designed using new hash functions, such as prime numbers verification (King 2013), graph-theoretic proof-of-work (Tromp 2015) or proof-of-work based on the generalized birthday problem (Biryukov and Khovratovich 2017). Post-quantum algorithms are currently being developed in the field of security, see, for example, Bae et al. (2017). In particular, Kiktenko et al. (2018) propose a quantum-safe blockchain that utilizes quantum key distribution. The application presented in the following sections seems to be the first documented attempt to establish a number theoretic proof-of-work unrelated to primes. The hash proposed is based

on properties of the Collatz algorithm. In order to describe this algorithm, consider the following function from $\mathbb{N}_0$ to $\mathbb{N}_0$:

$$T(x) = \begin{cases} x/2 & \text{if } x \text{ is even} \\ (3x+1)/2 & \text{if } x \text{ is odd} \end{cases} \tag{1}$$

Now, apply the following iterate of T:

$$\begin{cases} T^0(x) = x \\ T^{(k+1)}(x) = T(T^k(x)) \end{cases} \tag{2}$$

The Collatz conjecture states that $\forall\, x \in \mathbb{N}_0, \exists$ a finite k such that $T^k(x) = 1$. Lagarias (2010) uses the following terminology: the "total stopping time" is defined as $\sigma_\infty(x) = \inf\{k : T^k(x) = 1\}$. The "stopping time" $\sigma(x)$ is $\inf\{k : T^k(x) < x\}$. The "gamma value" is defined as $\gamma(x) = \frac{\sigma_\infty(x)}{log(x)}$.

For instance, let us consider the case for $x = 3$:

$$\begin{cases} T^0(3) = 3, \\ T^1(3) = (3 \times 3 + 1)/2 = 5, \\ T^2(3) = (5 \times 3 + 1)/2 = 8, \\ T^3(3) = 8/2 = 4, \\ T^4(3) = 4/2 = 2, \\ T^5(3) = 2/2 = 1. \end{cases} \tag{3}$$

In this example, the Collatz sequence[1] is $\{3, 5, 8, 4, 2, 1\}$ and $\sigma_\infty(3)$ equals to 5 while $\sigma(3) = 4$. By definition, the value of $\sigma_\infty(x)$ depends on the starting point of the algorithm. For example $\forall\, \alpha \in \mathbb{N}_0$, $\sigma_\infty(2^\alpha) = \alpha$ as

$$T^\alpha(2^\alpha) = 1. \tag{4}$$

Analyzing the total stopping time $\forall\, x \in \mathbb{N}_0$ has proven challenging: the lack of clear patterns and the absence of an analytical shortcut to estimate $\sigma_\infty(x)$ have left practitioners with numerical methods to compute it and verify the conjecture. e Silva (2010) proved computationally that the conjecture holds up until $x = 20 \times 2^{58}$. Current computational capabilities have allowed confirming the conjecture for very large numbers. For example, Honda et al. (2017) introduced a GPU-based method to verify the Collatz algorithm. The authors could verify 1.31e12 64-bit numbers per second. A probabilistic approach is also a frequent workaround to justify the validity of the Collatz conjecture: assuming function $T^k(x)$ is "random enough", Crandall (1978) showed that half of the time, the next number in the sequence will be $(3x+1)/2$, then for the next iteration, $1/4$ of the time it will be $(3x+1)/4$, then for the next iteration, $1/8$ of the time it will be $(3x+1)/8$ and so on so that the average growth in the sequence will be $(\frac{3}{2})^{1/2}(\frac{3}{4})^{1/4}(\frac{3}{8})^{1/8}(\frac{3}{16})^{1/16}(\frac{3}{32})^{1/32}... = \frac{3}{4} < 1$. Terras (1976) demonstrated that the set of integers {x:- x has stopping time $\leq$ k} has a limiting asymptotic density $F(k)$ with $F(k) \to 1$ as $k \to \infty$. These elements tend to indicate that $T^k(x)$ does not diverge to infinity as k grows. Using Minsky (1961) machines, Conway (1972) showed that a problem generalizing the Collatz conjecture is not algorithmically decidable. Kurtz and Simon (2007) extended the proof to show that this generalization is Π_0^2 complete. If the problem is truly algorithmically undecidable, then no information about the future inflation of the Collatz map is passed from one step k to the next step $k+1$. To explore that hypothesis and the properties of this "pseudo-randomness", let us define

[1] also called "trajectory" or "forward orbit".

the *inflation propensity of order K* $\xi(x,K)$ as the cardinality of the set of steps that lead to a number strictly larger than all previous numbers in the same sequence:

$$\xi(x,K) = \mathbf{card}\left\{k : T^k(x) > \max(M_{x,k})\right\}, k = 1, ..., k, ...K, \tag{5}$$

where $M_{x,k} = \left\{T^0(x), T^1(x), ..., T^{k-1}(x)\right\}$. $\xi(x, \sigma_\infty(x))$ is a particular case. For the ease of notation: $\xi(x) = \xi(x, \sigma_\infty(x))$. In the above example of $x = 3$, $\xi(3) = 2$. Indeed, the set of numbers strictly larger than the previous maxima in the sequence are $\{5, 8\}$ so that $\xi(3) = \mathbf{card}\{5, 8\} = 2$. In the other example presented supra with $x = 2^\alpha$, $\xi(2^\alpha) = 0 \; \forall \; \alpha \in \mathbb{N}_0$ since no number in their sequences can be strictly larger than the initial one.

This research paper investigates the distribution of $\xi(x)$, the inflation propensity as a deterministic variable that resembles a random behavior. If past maxima anywhere in the sequence are independent from new maxima later computed in that orbit, we should have that $\xi(x) \sim \mathrm{G}(\rho)$, a geometric distribution of parameter ρ with density $f(\xi(x) = y) = \rho^y(1-\rho)$. The interests of fitting a density distribution to $\xi(x)$ are multiple. First, in absence of proof of the Collatz conjecture, numerical analysis of the problem stays relevant towards resolving the question. Second, by properly addressing the behavior of the series for large numbers, one can help anticipate the computational challenges related to exploring the orbits of the Collatz map. Third, identifying pseudo-random behavior of Collatz inflation propensity directly leads to a new class of proofs-of-work for blockchain applications. The remainder of this document is built as follows. The next section discusses the empirical distributions of $\sigma_\infty(x)$, $\sigma(x)$ and $\xi(x) \; \forall \; x \in \mathbb{N}_0$. The third section details the observed density of $\xi(x)$. The density parameter of a geometric distribution is estimated using all natural numbers up to 1×10^{11} as sample. The fourth section presents a new proof-of-work based on inflation propensity, while the last section is a conclusion.

2. Inflation Propensity

Lagarias (1985) describes the $3x + 1$ conjecture as "a deterministic process that simulates random behavior" and goes further to mention that the problem seems "structureless". Urvoy (2001) formally proves the non-regularity of the Collatz's graph. As a visual illustration of this "structurelessness", the total stopping time for the first 1×10^6 natural numbers as a function of their value is presented in Figure 1. The equally "structureless" empirical distribution of the total stopping time for the same numbers is presented in Figure 2. In this context, "structureless" means that it is impossible to anticipate the frequency of the total stopping time. This is unfortunate since it means observing the total stopping times over a region of $\mathbb{N}_0$ gives no information whatsoever on the Collatz problem apart from strictly verifying its convergence. The mean of the total stopping time totally depends on the region over which it is computed, and, even when considering a closed subset of $\mathbb{N}_0$, the distribution of the total stopping time appears to be erratic and does not seem to follow any regular pattern. As such, the total stopping time has no apparent statistical properties that could be useful in applications such as, for instance, generating random numbers.

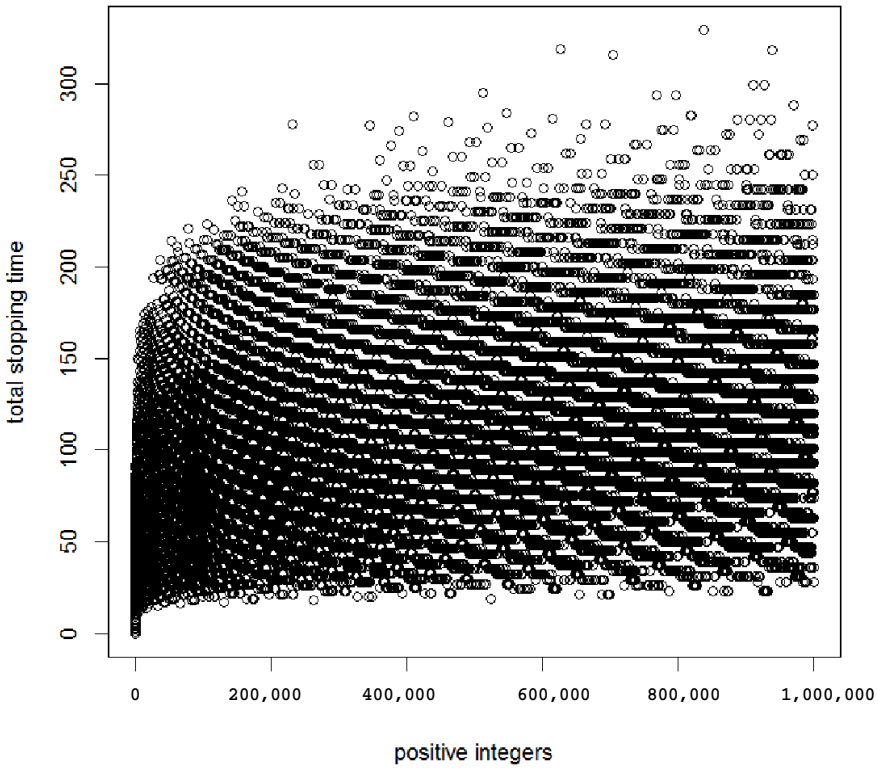

Figure 1. "Structureless" total stopping time for the first 1×10^6 natural numbers.

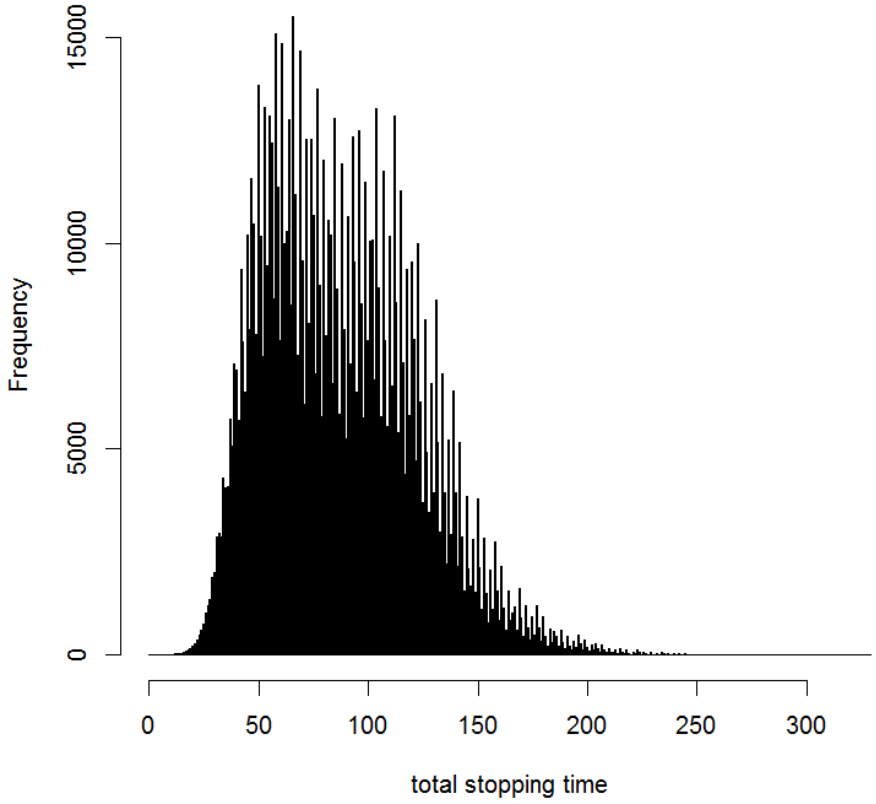

Figure 2. "Structureless" distribution of the total stopping time for the first 1×10^6 natural numbers.

Precisely because the Collatz graph is non-regular, its complexity gives rise to a pseudo-random behavior. Nichols (2018) and Kontorovich and Lagarias (2010) explore similarities between the Collatz model and the following dynamical system:

$$\log_2 T^K(x) \approx \log_2 x - K + b_3 \sum_{k=0}^{K} Y_k, \tag{6}$$

where b_3 is a constant and Y_k are IID (independent and identically distributed) Bernouilli random variables. The stochastic models predict that all orbits converge to a bounded set and that the total stopping time $\sigma_\infty(x)$ for the $3x+1$ map of random starting point x is about 6.95212 $\log x$ steps, as $x \to \infty$ have a normal distribution centered around that value. The authors point out that a suitable scaling limit for the trajectories is a geometric Brownian motion. This approach is extended in the current research in order to find a discrete metric that could exhibit some type of consistency and is independent from the starting point x. If a geometric Brownian motion can properly describe trajectories of large orbits, it means its Markov property can be exploited: each marginal step in the orbit is independent from the previous step. As a consequence, the probability to find new maxima after any random point $T^k(x)$ of a large orbit does not depend on how many new maxima were discovered before that point. In other words, for any $x >> 4 \in \mathbb{N}$:

$$P\Big(\xi(x) > M \mid \xi(x) \geq \xi(x,k)\Big) = P\Big(\xi(x) > M - \xi(x,k)\Big), \tag{7}$$

where $M > \xi(x,k)$ and $M \in \mathbb{N}$. If the inflation propensity is memoryless as described by Equation (7), it directly implies that the density $f(\xi(x) = y)$ follows a geometric distribution. It would mean that

$$f(\xi(x) = y) = \rho^y(1-\rho) \tag{8}$$

with $\rho \in]0;1[$ and $y \in \mathbb{N}$. The moment generating function is

$$\mu_n = Li_{-n}(\rho) - \rho Li_{-n}(\rho), \tag{9}$$

where $Li_n(\rho)$ is the nth polylogarithm of ρ and

$$\hat{\rho} = \frac{\mu_1}{1+\mu_1} \tag{10}$$

is the corresponding estimator of ρ based on Equation (9). It is also the maximum likelihood estimator. The empirical distribution of $\xi(x)$ defined in (5) is presented in Figure 3. The next step is to test the hypothesis that $\xi(x) \sim \mathrm{G}(\rho)$.

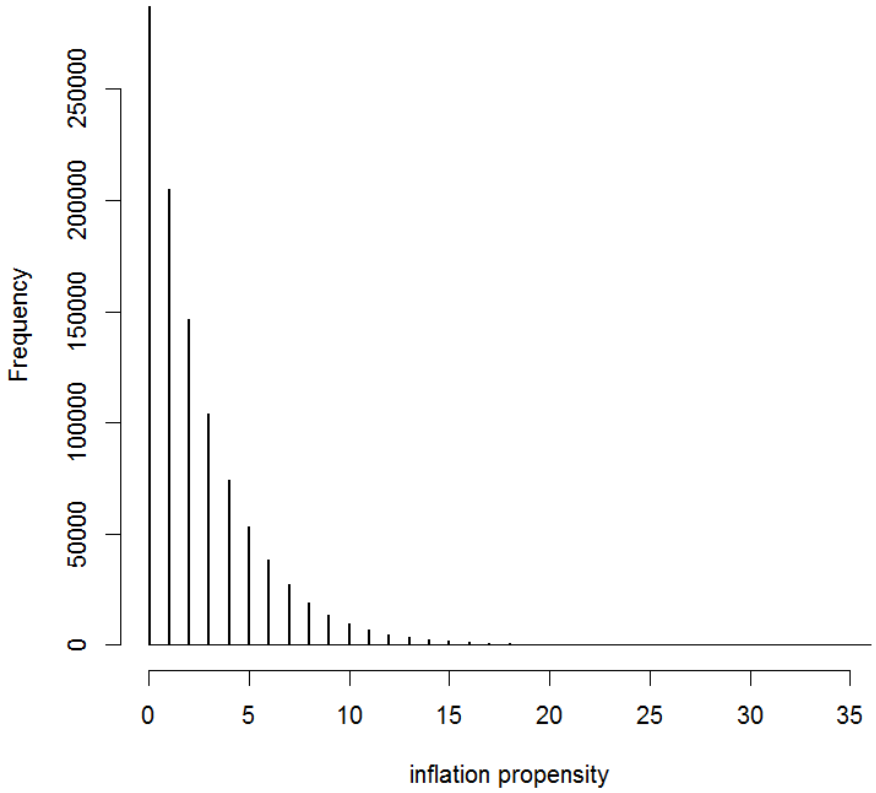

Figure 3. Quasi-geometric distribution of the inflation propensity for the first 1×10^6 natural numbers.

3. Empirical Results

The samples consist in the first 1×10^8 , 1×10^9, 1×10^{10} and 1×10^{11} positive integers. For each sample, the maximum likelihood estimator of ρ is computed, then tests are performed to see if elements of the distribution follow a geometric distribution of parameter ρ:

$$H_0 : P(\xi(x) = n) = (1-\rho)^{n-1}\rho \quad \forall n = 1, ..., q \tag{11}$$

$$H_1 : P(\xi(x) = n) \neq (1-\rho)^{n-1}\rho \quad \forall n = 1, ..., q \tag{12}$$

where $q \in [0, N]$ and N is the largest observed maximum in the sample. When $q = N$, the entire distribution is tested for goodness of fit with a geometric distribution of parameter $\hat{\rho}$. The tests are performed using Pearson's χ^2 test at a 10% confidence level. Table 2 summarizes the results of the tests. As the sample size increases, the hypothesis is not rejected when it comes to considering the first quantiles of the distribution. For the last sample (1×10^{11}), the hypothesis that the distribution of the inflation propensity follows a geometric distribution cannot be rejected up to the 91th percentile, compared to the 49th percentile for the 1×10^9 sample. Computational limitations prevent at this stage investigating larger sample sizes so that the geometric behavior of the inflation propensity over the entire domain ($\mathbb{N}_0$) needs to be conjectured. Interestingly, the estimator for ρ seems also to converge to a given value as the size of the sample increases and is very close to $\frac{\pi-1}{3}$, which is coincidentally the solution to the equation $3x + 1 = \pi$ (see Figure 4). Table A1 in Appendix A indicates the distribution of inflation propensities for the first 1×10^{11} integers.

Table 2. Pearson's χ^2 tests for goodness of fit with a geometric distribution.

		Sample 1×10^8	**Sample 1×10^9**	**Sample 1×10^{10}**	**Sample 1×10^{11}**
$\hat{\rho}$		**0.7133482**	**0.7135956**	**0.713667**	**0.713681**
q	**Percentile**	p**-Value**	p**-Value**	p**-Value**	p**-Value**
0	29	0.01	**0.15**	**0.70**	**0.64**
1	49	0.04	**0.19**	**0.70**	**0.14**
2	64	0.09	0.00	**0.23**	**0.25**
3	74	**0.15**	0.00	**0.36**	**0.39**
4	82	0.08	0.00	**0.11**	**0.35**
5	87	0.05	0.00	0.01	**0.37**
6	91	0.07	0.00	0.00	**0.13**
7	93	**0.11**	0.00	0.00	0.03
8	95	0.00	0.00	0.00	0.04
9	97	0.00	0.00	0.00	0.03
10	98	0.00	0.00	0.00	0.00

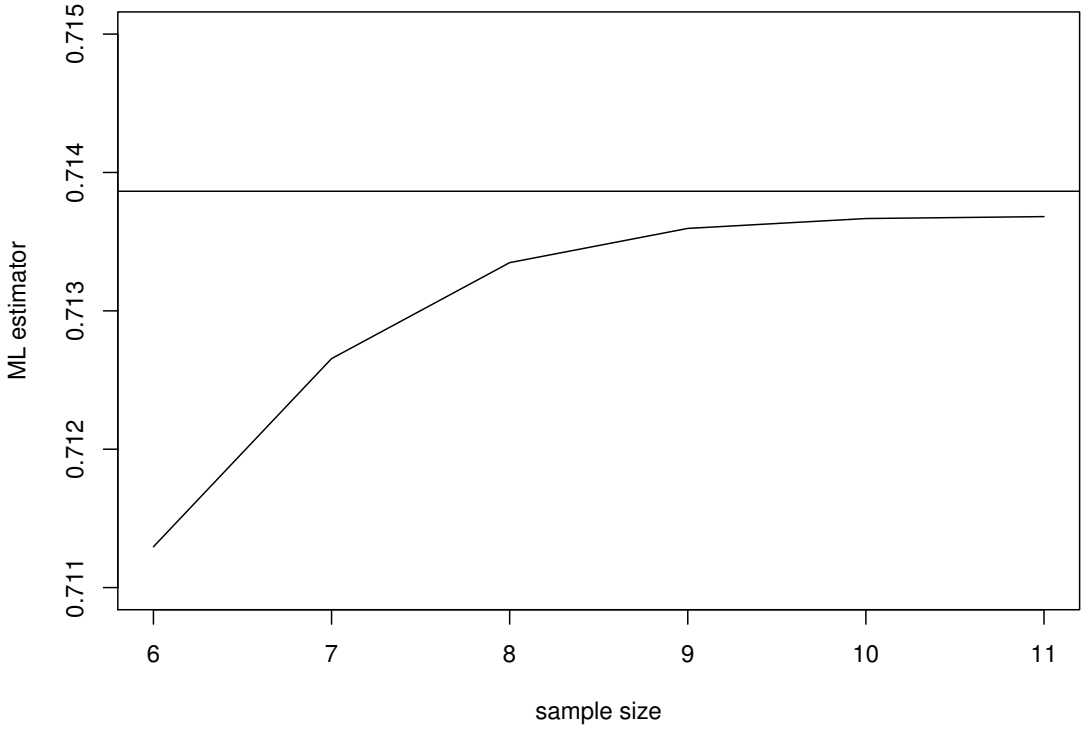

Figure 4. $\hat{\rho}$ as a function of the sample size (log10-scale).

4. Application

4.1. Collatz-Based Proof-of-Work

Because the distribution of the inflation propensity of Collatz orbits can be assumed to be geometric over large samples, and that a natural generalization of the Collatz algorithm has been proven to be undecidable, the inflation propensity can be considered as a new candidate to generate proofs-of-work, conjecturing the Collatz algorithm is also undecidable. Consider the following problem: find any set X made of n natural numbers $\{X_1, ..., X_i, ..., X_n\}$ whose values are between B and $B^* = B + \alpha$, a larger number, and that have inflation propensities of given values $\{Q_1, ..., Q_i, ..., Q_n\}$ with $n << \alpha$. In other terms, find a solution to the problem

$$Q_i = \xi(X_i) \quad \forall i \in \{1, ..., n\}, \tag{13}$$

where Q_i is known, $X_i \in [B, B^*]$ and $X_i \neq X_j \quad \forall i \neq j \in \{1, ..., n\}$. $\alpha, B, B^*, Q, X \in \mathbb{N}_0$. B is the unsigned integer value corresponding to a 256 bits block of hashed information. α is set to an arbitrarily large value, for example $\alpha = 2^{64}$. Note that this is still a fraction of the value for B so that pre-computation is virtually impossible in practice.

Since $P(\xi(X_i) = Q_i) \approx (\frac{\pi-1}{3})^{Q_i}(1 - \frac{\pi-1}{3})$ the difficulty to the problem can be designed in a straightforward manner: solutions for higher targets Q_i will be exponentially more difficult to find. Nevertheless, verifying the proof given inputs X and B is immediate, a desirable property for a proof-of-work. Once a valid solution set X has been found, the nounce ν is simply:

$$\nu = X - B, \tag{14}$$

which in practice is an array if X is a set and is an integer if X is a scalar. At the exception of the nonce and the target Q, the remainder of blockchain application based on Collatz is identical to the existing Bitcoin protocol. In practice, the target set Q can be selected by the network so that, similar to Bitcoin, six blocks are mined per hour. Every 2016 blocks, clients can compare the performance of the network and adjust the difficulty accordingly. Thanks to the geometric nature of the inflation propensity, a protocol for this adjustment is straightforward. Let us assume U_0 is the average amount of time required by the network to find any single value $\xi(x)$. Any total computational time $U_T \geq U_0$ can be easily selected by finding a set Q solving the following problem:

$$U_T = \sum_{q \in Q} \frac{1}{\rho^q} U_0 + \epsilon. \tag{15}$$

Two additional constraints must be considered for the protocol to be properly defined: the set Q must be chosen so that $0 \leq \epsilon \leq U_0$ and the cardinality of the set must be as small as possible.

4.2. Example: Bitcoin Genesis Hash

A new Bitcoin genesis hash is created using original inputs by Nakamoto (2008), but exploiting inflation propensity proof-of-work instead of hashcash. The inputs are: a hash merkle root that condenses all information related to the first Bitcoin transaction, a version number, a public key, a date, a time stamp that is used as coinbase parameter, and a target for complexity. A genesis block is the first block of a blockchain. Figure 5 illustrates the proof-of-work system. To create a genesis hash using inflation propensity as proof-of-work, only two adjustments to the Bitcoin protocol are required. First, the target for complexity is expressed with an integer, which is the targeted inflation propensity. This directly relates to a specific probability of occurrence. Second, the hashcash is replaced with the inflation propensity algorithm. In practice, the block header is hashed using SHA-256 then converted into an integer using hexadecimal encoding. This corresponds to B in Equation (14). The target set Q is arbitrarily set to a single value of 40 for the generation of this first hash, which corresponds to a probability of occurence of $\sim 4 \times 10^{-7}$. The value of B given Nakamoto's other initial inputs is of $\sim 2.52 \times 10^{76}$. The X nonce is then incrementally added to the integer B and inflation propensity is computed until the target of 40 is reached. The values obtained from each iteration are hereafter named "Xis". In the Python implementation of the algorithm, 2056 Xis are computed per second on an Intel Core i7-4700MQ CPU with 8×2.40 GHz. After 28 min of computation, the solution is found. Verification of the solution is done in $\approx 5 \times 10^{-4}$ seconds on the same machine. Table 3 describes diagnostics and results of the genesis hash. Using this first instance to calibrate the computational difficulty, the smallest set Q that solves Equation (15) that would yield an expected computational time of 10 min for the next block would be $\{2, 6, 16, 19, 22, 26, 31, 36, 41\}$. The Python code to generate the Genesis Hash is provided in Appendix B.

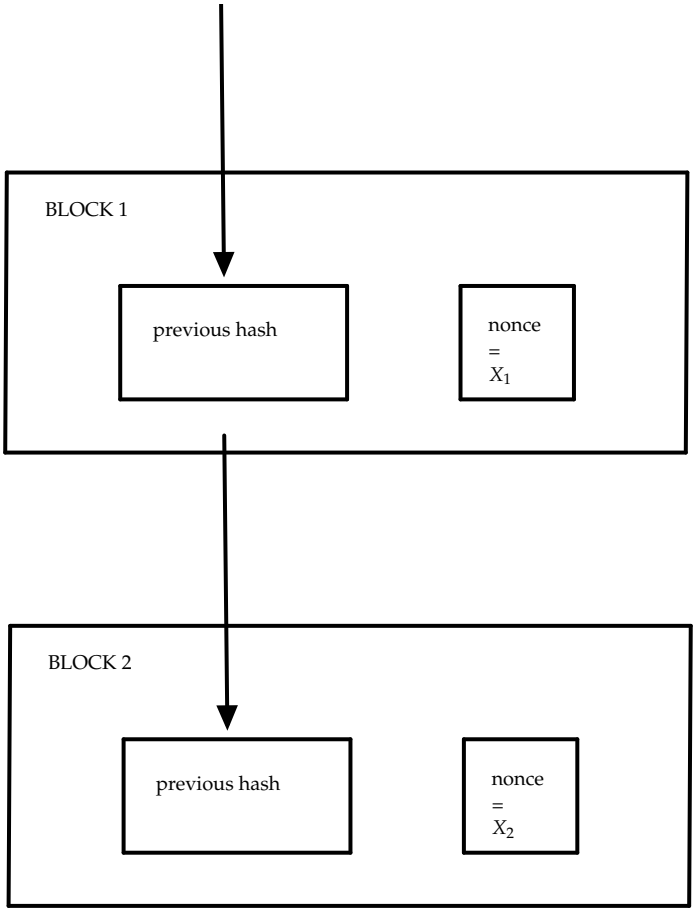

Figure 5. Proof-of-work system in the blockchain.

Table 3. A genesis hash based on original Bitcoin's inputs for genesis but using inflation propensity as proof-of-work.

block header hash	37d25f7f472fde7bb5b84f4bb319097c580383911b45eff10e68afa06073d6c0
corresponding integer	25248903652996148805237565338196318809513309980842754974279018460154571249344
merkle hash	4a5e1e4baab89f3a32518a88c31bc87f618f76673e2cc77ab2127b7afdeda33b
pszTimestamp	The Times 3 January 2009 Chancellor on brink of second bailout for banks
pubkey	04678afdb0fe5548271967f1a67130b7105cd6a828e03909a67962e0ea1f61deb649f6bc3f4cef38c4f 35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f
time	1231006505
inflation propensity target	40 (0×28)
nounce	3420991
genesis hash	9ed4d59e375c60e568524ac7fdfcce2c36dd8d449a20b0be8c9f6f9dbd2f8709
computational time	28 min

4.3. Advantages of the Collatz-Based Proof-of-Work

The advantages of a Collatz-based proof-of-work are many. From a practitioner perspective, the algorithm is easy to implement in code since the underlying problem is made of simple arithmetic operations, however, bigint arithmetics are needed in case values inflate beyond 2^{256}. Also, the natural generalization of the Collatz algorithm is known to be algorithmically undecidable. If this holds for Collatz algorithm, asymmetry is guaranteed: it is difficult to find the targeted value but easy to verify. From an engineering point of view, difficulty control based on a geometric distribution is significantly more complex than one based on hashcash, however, from a statistical perspective, the geometric distribution allows a very convenient tailoring of the computational complexity. It is very easy to adjust a specific targeted inflation-propensity, or a combination of targets. The same algorithm can

also be indefinitely extended to meet new computational improvements since the upper bound of the orbits is infinity. In addition to this scalability, it could be possible to generalize the $3x+1$ algorithm to other congruential graphs exhibiting the same properties (for example, the $5x+1$ graph). Provided further research confirms this hypothesis, such a feature could allow more possibilities to generate new proofs-of-work.

5. Conclusions

For the classical $3x+1$ map, it is conjectured that inflation propensity $\xi(x) = \mathbf{card}\left\{k : T^k(x) > \max(M_{x,k})\right\}, k = 1, ..., k, ...\sigma_\infty(x)$ has a geometric density distribution whose parameter's value $\rho \approx \frac{\pi-1}{3}$. This has been verified numerically for the first 1×10^{11} integers. The inflation propensity of Collatz orbits is a new metric that exhibits properties particularly well suited to be the base for new cryptography applications. A new proof-of-work is suggested: finding a set X of n integers greater than a hashed block of information B but smaller than a threshold B^* such that their inflation propensities be of n given values $Q_1, ..., Q_n$. Advantages of this approach are multiple, including an infinite scalability and the possibility to easily tune complexity of the algorithm. This work seems to be the first number theoretic proof-of-work unrelated to primes. Further research is needed to generalize this type of proof-of-work to a larger class of congruential graphs.

Funding: This research received no external funding.

Conflicts of Interest: The author declares no conflicts of interest.

Appendix A. Distribution of Inflation Propensity for the First 1×10^{11} Integers

Table A1. Distribution of inflation propensity $\xi(x)$ for the first 1×10^{11} integers.

$\xi(x)$	Observations
0	28,631,964,381
1	20,434,254,718
2	14,583,348,496
3	10,407,804,534
4	7,427,954,284
5	5,301,161,512
6	3,783,166,989
7	2,699,976,430
8	1,927,052,441
9	1,375,229,862
10	981,424,318
11	700,353,911
12	499,868,474
13	356,795,944
14	254,706,290
15	181,761,315
16	129,757,032
17	92,628,127
18	66,127,176
19	47,199,172
20	33,676,458
21	24,024,158
22	17,138,021
23	12,231,945
24	8,727,118
25	6,225,787

Table A1. *Cont.*

$\xi(x)$	**Observations**
26	4,432,544
27	3,162,432
28	2,251,004
29	1,599,248
30	1,139,341
31	814,975
32	583,455
33	416,994
34	298,683
35	212,914
36	150,443
37	106,613
38	76,749
39	55,452
40	39,947
41	28,495
42	20,259
43	14,253
44	10,396
45	7791
46	5431
47	3690
48	2640
49	1984
50	1448
51	1041
52	745
53	595
54	467
55	347
56	234
57	170
58	127
59	72
60	41
61	21
62	20
63	17
64	17
65	9
66	2
67	0
68	1
69	0

Appendix B. Python Code for Genesis Block

Modified from Hartikka (2017).

```
import hashlib, struct, binascii, time, sys, optparse

from construct import *

def main():
options = get_args()
input_script = create_input_script(options.timestamp)
output_script = create_output_script(options.pubkey)
tx = create_transaction(input_script, output_script,options)
hash_merkle_root = hashlib.sha256(hashlib.sha256(tx).digest()).digest()
print_block_info(options, hash_merkle_root)
block_header      = create_block_header(hash_merkle_root, options.time, options.bits, options.nonce)
genesis_hash, nonce = generate_hash(block_header, options.nonce, options.bits)
announce_found_genesis(genesis_hash, nonce)

def get_args():
parser = optparse.OptionParser()
parser.add_option("-t", "--time", dest="time", default=int(1231006505), type="int", help="the (unix) time
when the genesisblock is created")
parser.add_option("-z", "--timestamp", dest="timestamp", default=
"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks",
type="string", help="the pszTimestamp found in the coinbase of the genesisblock")
parser.add_option("-n", "--nonce", dest="nonce", default=0,
type="int", help="the first value of the nonce that will be incremented
when searching the genesis hash")
parser.add_option("-p", "--pubkey", dest="pubkey", default="04678afdb0fe5548271967f1a67130b7105cd6a828e03909
a67962e0ea1f61deb649f6bc3f4cef38c4f35504e51ec112de5c384df7ba0b8d578a4c702b6bf11d5f",
type="string", help="the pubkey found in the output script")
parser.add_option("-v", "--value", dest="value", default=5000000000,
type="int", help="the value in coins for the output, full value (exp. in bitcoin 5000000000
- To get other coins value: Block Value * 100000000)")
parser.add_option("-b", "--bits", dest="bits",
```

```
type="int", help="the target in compact representation, associated to a difficulty of 1")
(options, args) = parser.parse_args()
if not options.bits:
options.bits = 40
return options

def create_input_script(psz_timestamp):
psz_prefix = ""
if len(psz_timestamp) > 76: psz_prefix = '4c'
script_prefix = '04ffff001d0104' + psz_prefix + chr(len(psz_timestamp)).encode('hex')
print (script_prefix + psz_timestamp.encode('hex'))
return (script_prefix + psz_timestamp.encode('hex')).decode('hex')

def create_output_script(pubkey):
script_len = '41'
OP_CHECKSIG = 'ac'
return (script_len + pubkey + OP_CHECKSIG).decode('hex')

def create_transaction(input_script, output_script,options):
transaction = Struct("transaction",
Bytes("version", 4),
Byte("num_inputs"),
StaticField("prev_output", 32),
UBInt32('prev_out_idx'),
Byte('input_script_len'),
Bytes('input_script', len(input_script)),
UBInt32('sequence'),
Byte('num_outputs'),
Bytes('out_value', 8),
Byte('output_script_len'),
Bytes('output_script', 0x43),
UBInt32('locktime'))

tx = transaction.parse('\x00'*(127 + len(input_script)))
tx.version         = struct.pack('<I', 1)
```

```
tx.num_inputs      = 1
tx.prev_output     = struct.pack('<qqqq', 0,0,0,0)
tx.prev_out_idx    = 0xFFFFFFFF
tx.input_script_len = len(input_script)
tx.input_script    = input_script
tx.sequence        = 0xFFFFFFFF
tx.num_outputs     = 1
tx.out_value       = struct.pack('<q' ,options.value)
tx.output_script_len = 0x43
tx.output_script   = output_script
tx.locktime        = 0
return transaction.build(tx)

def create_block_header(hash_merkle_root, time, bits, nonce):
block_header = Struct("block_header",
Bytes("version",4),
Bytes("hash_prev_block", 32),
Bytes("hash_merkle_root", 32),
Bytes("time", 4),
Bytes("bits", 4),
Bytes("nonce", 4))

genesisblock = block_header.parse('\x00'*80)
genesisblock.version       = struct.pack('<I', 1)
genesisblock.hash_prev_block = struct.pack('<qqqq', 0,0,0,0)
genesisblock.hash_merkle_root = hash_merkle_root
genesisblock.time          = struct.pack('<I', time)
genesisblock.bits          = struct.pack('<I', bits)
genesisblock.nonce         = struct.pack('<I', nonce)
return block_header.build(genesisblock)

#Collatz inflation propensity
def inflation_propensity(x):
xMax=x
stepToMaximum=0
```

```
while x > 1:
if x % 2 == 0:
x = x / 2
else:
x = (3 * x + 1) / 2
if x > xMax:
xMax=x
stepToMaximum+= 1
return stepToMaximum

def generate_hash(data_block, start_nonce, bits):
print 'Searching for genesis hash..'
nonce          = start_nonce
last_updated   = time.time()
header_hash = generate_hashes_from_block(data_block)
print(binascii.hexlify(header_hash))
orbitTrajectory=int(header_hash.encode('hex_codec'), 16)
print(orbitTrajectory)
timeInit=time.time()
while True:
xi=inflation_propensity(orbitTrajectory)
last_updated = calculate_hashrate(nonce, last_updated, orbitTrajectory,timeInit)
if xi==bits:
return (generate_hashes_from_block(data_block), nonce)
else:
nonce     = nonce + 1
orbitTrajectory += 1
data_block = data_block[0:len(data_block) - 4] + struct.pack('<I', nonce)

def generate_hashes_from_block(data_block):
header_hash = hashlib.sha256(hashlib.sha256(data_block).digest()).digest()[::-1]
return header_hash

def calculate_hashrate(nonce, last_updated, orbitTrajectory, timeinit):
if nonce % 10000 == 0:
```

```
now           = time.time()
hashrate      = round(10000/(now - last_updated))
sys.stdout.write("\r%s Xis/s, Orbit: %s, Total time: %s minutes "
%(str(hashrate), str(orbitTrajectory), str((now-timeinit)/60)))
sys.stdout.flush()
return now
else:
return last_updated

def print_block_info(options, hash_merkle_root):
print "merkle hash: " + hash_merkle_root[::-1].encode('hex_codec')
print "pszTimestamp: " + options.timestamp
print "pubkey: "     + options.pubkey
print "time: "       + str(options.time)
print "bits: "       + str(hex(options.bits))

def announce_found_genesis(genesis_hash, nonce):
print "genesis hash found!"
print "nonce: "      + str(nonce)
print "genesis hash: " + genesis_hash.encode('hex_codec')

main()
```

References

Back, Adam. 2002. Hashcash-A Denial of Service Counter-Measure. Available online: http://hashcash.org/papers/hashcash.pdf (accessed on 25 November 2018).

Bae, Minyoung, Ju-Sung Kang, and Yongjin Yeom. 2017. A Study on the One-to-Many Authentication Scheme for Cryptosystem Based on Quantum Key Distribution. Paper presented at IEEE Conference on Platform Technology and Service (PlatCon), Busan, Korea, February 13–15, pp. 1–4.

Biryukov, Alex, and Dmitry Khovratovich. 2017. Equihash: Asymmetric proof-of-work based on the generalized birthday problem. *Ledger* 2: 1–30. [CrossRef]

Conway, John H. 1972. Unpredictable iterations. Paper presented at 1972 Number Theory Conference, Boulder, CO, USA, August 14–18, pp. 49–52.

Crandall, Richard E. 1978. On the 3x + 1 problem. *Mathematics of Computation* 32: 1281–92.

Dwork, Cynthia, and Moni Naor. 1992. Pricing via processing or combatting junk mail. Paper presented at Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16–20. Berlin and Heidelberg: Springer, pp. 139–47.

e Silva, T. Oliveira. 2010. Empirical verification of the 3x+ 1 and related conjectures. In *The Ultimate Challenge: The 3x+ 1 Problem*. Edited by Jeffrey C. Lagarias. Providence: American Mathematical Society, pp. 189–207. ISBN 978-0821849408.

Hartikka, Lauri. 2017. GenesisH0. Available online: https://github.com/lhartikk/GenesisH0 (accessed on 1 September 2018).

Honda, Takumi, Yasuaki Ito, and Koji Nakano. 2017. GPU-accelerated Exhaustive Verification of the Collatz Conjecture. *International Journal of Networking and Computing* 7: 69–85. [CrossRef]

King, Sunny. 2013. Primecoin: Cryptocurrency with Prime Number Proof-of-Work. *Bitcoin Magazine*, July 7.

Kiktenko, Evgeniy O., Nikolay O. Pozhar, Maxim N. Anufriev, Anton S. Trushechkin Ruslan R. Yunusov, Yuri V. Kurochkin, and Aleksey K. Fedorov. 2018. Quantum-secured blockchain. *Quantum Science and Technology* 3: 035004. [CrossRef]

Kontorovich, Alex V., and Jeffrey C. Lagarias. 2010. Stochastic models for the 3x+ 1 and 5x+ 1 problems and related problems. In *The Ultimate Challenge: The 3x+ 1 Problem*. Edited by Jeffrey C. Lagarias. Providence: American Mathematical Society, pp. 131–88. ISBN 978-0821849408.

Kurtz, Stuart A., and Janos Simon. 2007. The undecidability of the generalized Collatz problem. Paper presented at International Conference on Theory and Applications of Models of Computation, Shanghai, China, May 22–25. Berlin and Heidelberg: Springer, pp. 542–53.

Lagarias, Jeffrey C. 1985. The 3x+ 1 problem and its generalizations. *The American Mathematical Monthly* 92: 3–23. [CrossRef]

Lagarias, Jeffrey C., ed. 2010. The 3x+1 problem: An overview. In *The Ultimate Challenge: The 3x+ 1 Problem*. Providence: American Mathematical Society, pp. 3–30. ISBN 978-0821849408.

Minsky, Marvin L. 1961. Recursive unsolvability of Post's problem of "tag" and other topics in the theory of Turing machines. *Annals of Mathematics* 74: 437–55. [CrossRef]

Nakamoto, Satoshi. 2008. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Self-published.

Nichols, Daniel. 2018. Analogues of the 3x + 1 Problem in Polynomial Rings of Characteristic 2. *Experimental Mathematics* 27: 100–10. [CrossRef]

Percival, Colin. 2009. *Stronger Key Derivation via Sequential Memory-Hard Functions*. Self-published.

Rubin, Jeremy. 2017. *The Problem with ASICBOOST*. Self-published.

Terras, Riho. 1976. A stopping time problem on the positive integers. *Acta Arithmetica* 30: 241–52. [CrossRef]

Tromp, John. 2015. Cuckoo cycle: A memory bound graph-theoretic proof-of-work. Paper presented at International Conference on Financial Cryptography and Data Security, San Juan, PR, USA, January 26–30. Berlin and Heidelberg: Springer, pp. 49–62.

Urvoy, Tanguy. 2001. Regularity of congruential graphs. Paper presented at Mathematical Foundations of Computer Science 2000, MFCS 2000, Bratislava, Slovakia, August 28–September 1. Edited by Mogens Nielsen and Branislav Rovan. Lecture Notes in Computer Science. Berlin and Heidelberg: Springer, vol. 1893.

Wood, Gavin. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper* 151: 1–32.