Computer Security Symposium 2015
21 - 23 October 2015

Application of Graph Problem for Bitcoin Mining

Samiran Bag† Sushmita Ruj‡ Kouichi Sakurai†

†Department of Informatics
Kyushu University
{bag@inf,sakurai@csce}.kyushu-u.ac.jp

‡R. C. Bose Center for Cryptology and Security
Indian Statistical Institute
sush@isical.ac.in

Abstract In this work we propose a scheme that could be used as an alternative to the existing proof of work(PoW) scheme for mining in Bitcoin P2P network. Our scheme ensures that the miner must do at least a non-trivial amount of computation for solving the computational problem put forth in the paper and thus computing a PoW. Here, we have proposed to use the problem of finding the largest clique in a big graph as a replacement for the existing Bitcoin PoW scheme. In this paper, we have dealt with a graph having $O(2^{30})$ vertices and $O(2^{48})$ edges which is constructed deterministically using the set of transactions executed within a certain time slot. We have discussed some algorithms that can be used by any Bitcoin miner to solve the PoW puzzle. Our proposed scheme is better than the existing proof of work scheme that uses Hashcash, where a lucky miner could manage to find a solution to the proof of work puzzle by doing less amount of computation.

1 Introduction

The main downside of the Bitcoin mining scheme is that it does not ensure a non-trivial amount of computation for finding a proof of work. However, the work of Tromp [9] is proposed as a substitute for the current Bitcoin mining Algorithm. In this scheme the author proposed to use graph problems for Bitcoin proof-of-work. The paper proposes to construct graph depending upon a hash based computation and then examines the presence of a subgraph(an L cycle) in the original graph. In this paper we introduce another proof-of-work scheme that could be used as an alternative to the existing Bitcoin mining puzzle. Our scheme proposes to employ clique finding problem in a giant random graph for the proof of work puzzle. In this scheme a miner needs to find the largest possible clique in a graph containing exponential number of vertices. The construction of the random graph itself requires computation of exponentially many hashes. Thus, finding the largest clique in the graph becomes a hard problem. We also have proposed an Algorithm that could find the largest cliques in $O(2^{85.6})$ time which is not too higher than the capacity of Bitcoin network as of April 2015. We further improve the run-time to $O(2^{70.5})$. Our work is moti-

vated by Tromp's paper [9], but is more efficient than than [9]. The main downside of Tromp's scheme is that it requires all peers to agree upon a target subgraph H(typically an L-cycle). In a decentralized autonomous system such as Bitcoin where the network parameters may be subject to change in a periodic manner, it is not feasible for different peers to agree upon a common subgraph H which may not be the same for ever. Similarly, construction of the large graph is also a challenging task as it would require someone to select the keys of the hash function that is used in constructing the large graph. Thus the scheme of Tromp [9] is difficult to be implemented in a decentralized system. In our scheme, however the graph is constructed deterministically from the set of transactions conducted during a certain time slot. So, any peer of the Bitcoin network can construct the graph deterministically from the set of transactions. Thus, our scheme is more applicable to the decentralized Bitcoin networking scenario.

2 Related Work

We discuss these topics for our background study.

Cryptocurrency 1) Bitcoin: Bitcoin is a decentralized cash system that does not depend on a centralized server such as a Bank. Its users make online payments by digitally signing every transaction with their secret key. The corresponding public key can be used to publicly verify the authenticity of the transaction. Every transaction is broadcast over the entire network. Since there is no trusted server, Bitcoin network validates every transaction by maintaining a public ledger of all executed valid transactions. This ledger is called Bitcoin block chain. Every executed transaction is stored in the block chain. Thus, Bitcoin users are prevented from double spending a Bitcoin transaction. A Bitcoin block is constructed by miners and it requires one to execute a nontrivial amount of computation. For mining Bitcoin, a miner needs to find a nonce such that the hash of the nonce, the merkle root of all valid transactions as well as some other parameter starts with some predefined number of zeros. Since the hash function is preimage resistant, the only way of finding such a nonce is to do a brute force over all the possible values of the nonce until a nonce is found that gives the hash the requires number of zeros. This is called the Bitcoin mining puzzle and it needs to be solved by at least one miner every time a block is constructed. Typically it takes about ten minutes to construct a block in the Bitcoin network. The number of zeros which are required for the hash to be valid solution defines the difficulty level of the Bitcoin mining puzzle and is updated regularly to ensure that the time to construct a block remains nearly equal to ten minutes. A Bitcoin miner is given a reward of 50 Bitcoins when her solution is accepted by the network. This incentive attracts Bitcoin miners into expending their computational resources for solving the mining puzzle. At present the Bitcoin network computes 400 million Gigahashes per second. Since, mining takes nontrivial amount of computation, typically equivalent to the processing power of millions of desktop machines it is not possible for an individual miner to solve the mining puzzle on her own. Hence, many individual miners collude to form a big computational force by combining their processing power to crack down the mining puzzle together. The group of miners is called a mining pool and are administrated by a miner called pool operator. In every pool, the pool operator is solely responsible for distribution of jobs to individual miners as well as for sharing the reward if the pool as a whole can solve

the mining puzzle.

2) Clique problem : A clique is a complete subgraph of a graph. Clique problem involves finding two types of cliques viz. maximal clique and maximum clique. A maximal clique is one that cannot be extended to form a clique of bigger size i.e. there is no bigger clique in the graph that contains the former as a subgraph. Again, a maximum clique of graph is a clique that has the size equal to that of the largest clique in the same graph. Clique problem is defined as the problem of finding the largest clique in a graph or listing all maximal cliques in the graph. Moon et al. [6] showed that the number of maximal cliques of a graph could be $O(3^{n/3})$. The simplest Algorithm that can list all maximum clique of a graph is the Bron-Kerbosch Algorithm [2] that has a mean run time of $O(3^{n/3})$. A variant of this was proposed by Tomita et al. [8] and it has a worst case complexity of $O(3^{n/3})$. The Algorithm finds all the maximal cliques in the graph and returns the largest one. The best Algorithm known till today is by Robson [7] that has a run time of $O(2^{0.249n})$.

Cliques of a fixed size k can be found in polynomial time by examining all subsets of size k and checking whether each of them forms a clique or not. This would take $O(\binom{n}{k}k^2)$ time. There has been a lot of research done on finding all triangles in a graph. Chiba et al. [3] proposed an Algorithm that finds all triangles in $O(e^{\frac{3}{2}})$ time, e being the number of edges in the graph. The Algorithm can enumerate all triangles in a sparse graph faster. Alon et al. [1] improved the $O(e^{\frac{3}{2}})$ Algorithm for finding triangles to $O(e^{1.41})$ by using fast matrix multiplication.

3 Preliminaries

3.1 Random Graph

A random graph $\mathcal{G}_{n,p}$ is a graph (V, E) on a set of vertices V, $|V| = n$ such that

$$\forall v_1, v_2 \in V, Pr[\{v_1, v_2\} \in E] = p.$$

That is in a random graph with n vertices, the probability of occurrence of an edge between a pair of vertices is given by p. This model of construction of graph was first proposed by Erdős and Rényi and is named after them.

Theorem 1 *[4] Let $Z(n,p)$ denote the size of largest complete subgraph of a graph $\mathcal{G}_{n,p}$. The sequence $\{Z(n,p)\}$ of random variables satisfies*

$$\lim_{n\longrightarrow\infty} Pr[Z(n,p) \rightarrow \frac{2\log n}{\log\frac{1}{p}} + O(\log n)] = 1$$

Theorem 2 *[5] For $n \geq 1$, $0 < p < 1$, for $1 \leq k \leq n$, $\left\{\sum_{j=\max(0,2k-n)}^{k} \frac{\binom{n-k}{k-j}\binom{k}{j}}{\binom{n}{k}} p^{-j(j-1)/2}\right\}^{-1}$ $\leq Prob[Z(n,p) \geq k] \leq \binom{n}{k}p^{k(k-1)/2}$.*

The above Theorems show that for any random graph having high number of vertices the size of the largest subgraph approaches a certain value. In Theorem 2, Matula proved using computation that the density of $Z(n,p)$ is very spiked for a big random graph of. He computed that $Pr[Z(10^{10}, 1/4) = 30] > 0.9997$ [5]. If we have a random graph $\mathcal{G}_{n,p}$, then if n is very high, typically having exponential size, then the size of the largest complete subgraph will be $\approx \frac{2\log n}{\log 1/p} + O(\log n)$ with an very high probability. Due to this behavior of the density of $Z(n,p)$, it is possible to apply clique finding problem for solving Bitcoin proof of work.

3.2 Bitcoin Transactions

Bitcoin users, as mentioned above, make online payments by executing a transaction. A

Bitcoin transaction is hashed with the payee's public key and is signed by the payer to validate the payment. A transaction can have multiple input fields as well as output fields and the number of all of them determines the actual size of the transaction. Each transaction input requires at least 41 bytes for references to the previous transaction and other headers and each transaction output requires an additional 9 bytes of headers. Besides, each transaction has an at least 10 bytes long header. Thus, 166 bytes is the minimum size of a Bitcoin transaction.

4 Our Scheme

4.1 The First Scheme

Let us define 'epoch' to be the period of time between construction of two consecutive blocks. The time of construction of a block in the Bitcoin network happens to be 10 to 15 minutes. Therefore an epoch may last for a fixed time between 10 to 15 minutes so that all transactions executed after the commencement of an epoch and before the expiry of that epoch are included in the Bitcoin block for that epoch. Transactions which are executed sharply before the expiry of an epoch may not be included in the block for the current epoch due to the delay of the transaction in traveling across the Bitcoin network and therefore it could be safely accommodated in the next block if their validity is acknowledged by the miner. Without loss of generality we may assume that the number of transactions occurring in a particular block is a power of 2. It can be forcibly done either by accommodating some extra transactions to the next block or by using some predefined dummy transactions. Let, $\tau = \{T_i : 1 \leq i \leq 2^\nu\}$ be the set of transactions that are yet to be included in a Bitcoin block in a certain epoch ε_0. Let $\mathcal{T}$ be a set such that $\mathcal{T} = \{L_{(i-1)*2^{30-\nu}+j} : L_{(i-1)*2^{30-\nu}+j} = T_i||j, 1 \leq i \leq 2^\nu, 0 \leq j \leq 2^{30-\nu}-1\}$. Hence, $|\mathcal{T}| = 2^\nu * 2^{30-\nu} = 2^{30} = n$(say). We construct a graph $\mathcal{G}_{n,p} = (V, E)$ such that $V = \mathcal{T}, n = |\mathcal{T}|$. We define the set of edges as $E = \{(u, v) : u, v \in V, H(u||v) = 0^{12}||\{0,1\}^*\}$, where $H(\cdot)$ is a hash function. Hence, a single hash needs to be calculated to check the existence of an edge between a pair of vertices. Also, the probability of occurrence of an edge between any two vertices is given by $p = \frac{1}{2^{12}}$.

Our Proof-of-Work scheme is to find the highest number of largest complete subgraph in $\mathcal{G}_{n,p}$ constructed as above. We provide our PoW scheme in Algorithm 1. In this Algorithm, a miner tries to find the maximum cliques by searching the adjacency matrix of the graph $\mathcal{G}_{n,p}$. So, the problem of finding the maximum complete subgraph of $\mathcal{G}_{n,p}$ is reduced to the problem of finding the maximum submatrix of the adjacency matrix of $\mathcal{G}_{n,p}$ with all entries equal to '1' except the diagonal entries.

In Algorithm 1, the miners need to find as many cliques as possible and need to broadcast them throughout the Bitcoin network. Now, from Theorem 1, we can say that the size of the maximum clique should be 5 or 6. Now, the expected number of cliques of size 5 is $\binom{2^{30}}{5} p^{\frac{1}{2}\times 5\times 4} \approx 2^{30}$. On the other hand the expected number of cliques of size 6 is equal to $\binom{2^{30}}{6} p^{\frac{1}{2}\times 6\times 5} < 1$. In section 5, we discuss some algorithms that can be used for finding all maximum cliques of the graph. It can be noted that broadcasting all $O(2^{30})$ cliques throughout the entire network will not be feasible for a miner. Hence, the Bitcoin network may choose a predefined upper bound(say Z) for the number of maximum cliques that a miner may send for one particular block. If multiple miners send Z many cliques then the network peers may select the one that came earlier. Alternatively, the peers may store the blocks on separate branches. Ultimately, only a single branch will be extended as it does

- **Setup** $\tau = \{T_1, T_2, \ldots, T_{2^\nu}\}$ are the 2^ν transactions occurring in a particular epoch. $\mathcal{T} = \{L_{(i-1)*2^{30-\nu}+j} : L_{(i-1)*2^{30-\nu}+j} = T_i||j, 1 \leq i \leq 2^\nu, 0 \leq j \leq 2^{30-\nu} - 1\}$. $V = \mathcal{T}$.
- **Proof of Work Computation** In a scratch off attempt a miner computes a $V \times V$ matrix B as follows:

$$B[i][j] = B[j][i] = \begin{cases} * & \text{if } i = j \\ 1 & \text{if } H(v_i||v_j) \in \{0^{12}||\{0,1\}^*\} \\ 0 & \text{elsewhere} \end{cases}$$

  where $v_i, v_j \in V, i \leq j$. The user finds the largest square submatrix B'_0 of B such that each and every entry of B'_0 is a 1 except the diagonal elements. If she finds such a submatrix then she broadcasts it along with the index set I_0.
  Step $r = 1, \ldots$ do {
  if the epoch has ended then exit
  else find another square submatrix from B such that $[B'_r] = [B'_0]$. If such a matrix could be found then broadcast it along with the index set I_r.}
- **Verification** The verifier holds separate caches for storing the solutions from different miners. The verifier checks every submatrix for being a valid clique as shown in Algorithm 1. If the Verification is successful then the user caches the proof of work along with the index set which corresponds to the set of vertices The verifier rejects all PoW whose index sets have a non-null intersection with a PoW present in its cache. After the epoch has expired the verifier checks its cache and selects a miner who has so far sent the highest number of proofs(largest cliques). Ties are broken arbitrarily and the corresponding block is accepted and added to the Bitcoin block chain.

図 1: New proof of work scheme for mining in Bitcoin network

presently. However, as we have seen, the number of cliques will be very few in case the clique number happens to be 6.

5 Finding Maximum Cliques of a graph

Here, we discuss some techniques that a miner can use to find maximum cliques of the graph and thus can solve the Bitcoin PoW puzzle introduced in this paper. The Algorithm 2 uses a bottom up approach to construct the set of all largest cliques. Before discussing the actual algorithm, we shall prove Lemma 3.

Lemma 3 *The computation needed to find all cliques of size $\kappa + 1$, if it exists given the set of all cliques of size κ is $O(2^{30+6k(6-\kappa)}\kappa)$.*

Proof The expected number of cliques of size κ is $\omega = \binom{n}{\kappa} p^{\frac{1}{2}\kappa(\kappa-1)}$. In our case, $n = 2^{30}, p = \frac{1}{2^{12}}$. So, $\omega = \binom{2^{30}}{\kappa}/2^{6\kappa(\kappa-1)} \approx \frac{2^{30\kappa}}{2^{6\kappa(\kappa-1)}} = 2^{6\kappa(5-\kappa+1)} = 2^{6\kappa(6-\kappa)}$. Now, from this set of candidate cliques of size κ, a clique of size $\kappa + 1$ can be found using the above method. So, the total computation needed is $\omega * 2^{30} * \kappa$. Hence, proved. ■

We shall now show how the result of Lemma 3 could be used to construct an Algorithm that finds all maximum cliques in reasonable time. Let us define $\mathcal{C}_i = \{\langle v_1, v_2, \ldots, v_i\rangle\}$ to be the set of all i-cliques of the graph $\mathcal{G}_{n,p} = (V, E)$.

Lemma 4 *The computational complexity of Algorithm 2 is $O(2^{85.58})$*

Proof The Algorithm 2 takes a bottom up

Require: $\mathcal{C}_3$.
Ensure: List all largest clique of the random graph.
for $i := 3; \mathcal{C}_i \neq \emptyset; i++$ **do**
  while $\mathcal{C}_i$ is not empty **do**
    Choose a clique $\langle v_1, v_2, \ldots, v_i \rangle \in \mathcal{C}_i$
    for all $v \in V \setminus \{v_1, v_2, \ldots, v_i\}$ **do**
      if $\bigcup_{j=1}^{i} \{v_i, v\} \subset E$ **then**
        $\mathcal{C}_{i+1} = \mathcal{C}_{i+1} \cup \langle v_1, v_2, \ldots, v_i, v \rangle$.
      end if
    end for
    $\mathcal{C}_i = \mathcal{C}_i \setminus \langle v_1, v_2, \ldots, v_i \rangle$.
  end while
end for

図 2: An Algorithm to list all largest cliques

approach in computing the maximum clique. It first takes as input the set of all cliques of size 3. Then it tries to grow the size of the the cliques by checking if any of the other vertices of the graph creates a bigger clique with it. Thus in every iteration of the outer for loop of Algorithm 2, it uses the set of all cliques of size i to construct a set of all cliques of size $i+1$. Now, according to Lemma 3, the complexity in each iteration is given by $O(2^{30+6i(6-i)}i)$. Without loss of generality we may assume that we run the Algorithm until $\mathcal{C}_6$ is computed. So, the complexity in each iteration should be as follows;
For $i = 3$, the run time is $O(2^{84} * 3)$. For $i = 4$, the run time is $O(2^{78} * 4)$. For $i = 5$, the run time is $O(2^{60} * 5)$.
It is apparent that the complexity of the above Algorithm is $O(3 * 2^{84})$. Now, we can precompute $\mathcal{C}_3$ using the Chiba & Nishizeki Algorithm [3] in $O(2^{72})$ time. So, the total complexity of finding all cliques of size 6(if at least one exists) is $O(3 * 2^{84}) \approx O(2^{85.58})$. ■

Lemma 5 *The computational complexity of Algorithm 2 is* $O(2^{85.58})$

Proof The Algorithm 2 takes a bottom up approach in computing the maximum clique. It first takes as input the set of all cliques of size 3. Then it tries to grow the size of the the cliques by checking if any of the other vertices of the graph creates a bigger clique with it. Thus in every iteration of the outer for loop of Algorithm 2, it uses the set of all cliques of size i to construct a set of all cliques of size $i+1$. Now, according to Lemma 3, the complexity in each iteration is given by $O(2^{30+6i(6-i)}i)$. Without loss of generality we may assume that we run the Algorithm until $\mathcal{C}_6$ is computed. So, the complexity in each iteration should be as follows;
For $i = 3$, the run time is $O(2^{84} * 3)$
For $i = 4$, the run time is $O(2^{78} * 4)$
For $i = 5$, the run time is $O(2^{60} * 5)$.
It is apparent that the complexity of the above Algorithm is $O(3 * 2^{84})$. Now, we can precompute $\mathcal{C}_3$ using the Chiba & Nishizeki Algorithm [3] in $O(2^{72})$ time. So, the total complexity of finding all cliques of size 6(if at least one exists) is $O(3 * 2^{84}) \approx O(2^{85.58})$. ■

This is a reasonable amount of computation for Bitcoin miners. The present hash rate of Bitcoin network is $10^{22} \approx 2^{73}$ and soon it is expected to reach the order of 2^{85} when our Algorithm 2 will become feasible to be applied for minting Bitcoin.

The space complexity of Algorithm 2 is determined as $M = \max_{i=3}^{6}(|\mathcal{C}_i|) = O(\max_{i=3}^{6} 2^{6\kappa(6-\kappa)})$. It is easy to see that $M = O(2^{54})$. Hence, the space complexity is of the order of a petabyte. Hence, the mining pool requires to store the set $\mathcal{C}_i$ in a distributed fashion. Since, a mining pool may contain tens of thousands of miners it is possible to afford this much space for executing Algorithm 2.

Further reduction of the computational complexity of finding $\mathcal{C}_6$ can be done by computing all K_4s of the graph as follows: 1) Find all quadrangles of the graph using the Chiba & Nishizeki Algorithm [3]. 2) Find all K_4s using

the set of quadrangles. 3) Then using the set of all K_4s ($\mathcal{C}_4$) try to find $\mathcal{C}_6$ using Algorithm 2.

Lemma 6 *The method stated above improves the computational complexity of Algorithm 2 to* $O(2^{80})$.

Proof The set of all quadrangles could be found using Chiba & Nishizeki Algorithm in $2^{70.5}$ time [3]. We calculate the expected number of quadrangles as follows; the number of ways 4 vertices could be chosen out of 2^{30} vertices is $\binom{2^{30}}{4}$. Now, we can create 3 quadrangles from every set of 4 vertices, each of them with a probability of $(\frac{1}{2^{12}})^4$. So the expected number of quadrangles will be $\binom{2^{30}}{4}(\frac{1}{2^{12}})^4 * 3 < 2^{69.6}$. So, in order to check whether each of them is a subgraph of a clique, $O(2^{69.6})$ computations would be required. Thus, $\mathcal{C}_4$ could be generated. Thereafter we could follow Algorithm 2 to find $\mathcal{C}_6$. It is easy to see that thus the overall run time could be improved to $O(2^{80})$. ■

Theorem 7 *There exists an Algorithm that outputs the set of maximum cliques in* $O(2^{70.5})$ *time and* $O(2^{48})$ *space.*

Proof The Algorithm works as follows:
1) It finds all quadrangles using Chiba & Nishizeki Algorithm. Whenever a quadrangle is found the Algorithm checks whether it could be a part of a K_4 or not. This could be done by computing only two hashes per quadrangle. If a quadrangle could be extended to a K_4 store it in $\mathcal{C}_4$. So, the total amount of computation needed is bounded by the total number of quadrangles in the graph which is $O(2^{69.6})$.

2) Without loss of generality we may assume $\mathcal{C}_4$ stores all K_4s such that $\forall(u_1, u_2, u_3, u_4) \in \mathcal{C}_4$, $u_1 < u_2 < u_3 < u_4$. Store the set of K_4s $\mathcal{C}_4$ in a sorted array $A[]$ such that for every $i, j \in |\mathcal{C}_4|, i < j$ if $A[i] = \{v_1, v_2, v_3, v_4\}$ and $A[j] = \{v'_1, v'_2, v'_3, v'_4\}$, then $\exists l \in \{1, 2, 3, 4\}$ such that $v_m = v'_m, \forall 1 \leq m < l$ and $v_l < v'_l$. This would require $|\mathcal{C}_4| \log |\mathcal{C}_4|$ time. Since, $|\mathcal{C}_4| = 2^{48}$, the amount of computation needed to do this is $(48 * 2^{48}) \approx O(2^{54})$.

3) We can now use the sorted array $A[]$ to find all K_5s. For all $i, 1 \leq i < |\mathcal{C}_4|$, if $A[i] = \{u_1, u_2, u_3, v\}$ and $A[i + t] = \{u_1, u_2, u_3, v'\}$, where $t > 1$, check whether $\{u_1, u_2, u_3, v, v'\}$ is a clique or not. This could be checked by computing a single hash. If $\{u_1, u_2, u_3, v, v'\}$ is a K_5 add it to $\mathcal{C}_5$. This would take $O(|\mathcal{C}_4|)$ or $O(2^{48})$ computations (see Lemma 8). The space needed to store $\mathcal{C}_5$ is $O(2^{30})$.

4) Now, the set $\mathcal{C}_5$ could be used to compute the set $\mathcal{C}_6$ if it is nonempty. This could be done using the same method stated in step 2 and 3 or in Algorithm 2.

It could be checked that the time complexity of this Algorithm is determined by the time needed to find all quadrangles using Chiba & Nishizeki Algorithm which is $O(2^{70.5})$. Similarly, the space complexity is $O(|\mathcal{C}_4|)$ or $O(2^{48})$. ■

The time complexity of the Algorithm described in Theorem 7 is same as the average computation needed to solve the existing proof of work puzzle which is equivalent to $O(2^{73})$ hash computations.

Lemma 8 *Let* $A[]$ *be a sorted array of all* K_4*s such that for every* $i \in [|\mathcal{C}_4|]$, *if* $A[i] = \{v_1, v_2, v_3, v_4\}$, *then* $v_1 < v_2 < v_3 < v_4$ *and for every* $i, j \in |\mathcal{C}_4|, i < j$ *if* $A[i] = \{v_1, v_2, v_3, v_4\}$ *and* $A[j] = \{v'_1, v'_2, v'_3, v'_4\}$, *then* $\exists l \in \{1, 2, 3, 4\}$ *such that* $v_m = v'_m, \forall 1 \leq m < l$ *and* $v_l < v'_l$. *Then finding every pairs* $(i, j), i \neq j$ *such that* $A[i] = \{v_1, v_2, v_3, v_4\}$ *and* $A[j] = \{v_1, v_2, v_3, v'_4\}$ *takes* $O(|A|)$ *time.*

Proof For any arbitrary $i \in [|\mathcal{C}_4|]$, Let $A[i] = \{v_1, v_2, v_3, v_4\}$. Therefore, all other K_4s like $\{v_1, v_2, v_3, z\}$ if there are any will be adjacent to $A[i]$ in the array. If there are Δ such cliques $A[j]$ whose first three vertices are same as that of $A[i]$, then all such pairs (i, j) could be found

in $O(\Delta^2)$ time. Whether any of these pairs could be extended to form a K_5 can be ascertained by computing a single hash per such pair. Now, for any three vertices $\{u, v, w \in V\}$, $\Delta = \{i : i \in [|\mathcal{C}_4|], \{u, v, w\} \subset A[i]\}$. So, $E(\Delta) = 2^{30} * (\frac{1}{2^{12}})^3 = \frac{1}{2^6}$. Now, the total time needed to find every pairs $(i, j), i \neq j$ such that $A[i] = \{v_1, v_2, v_3, v_4\}$ and $A[j] = \{v_1, v_2, v_3, v_4'\}$ is $O\left(|A|\left\{\max(1, E^2(\Delta))\right\}\right)$ or $O(|A|)$. ■

6 Conclusion

In this paper we propose a new proof of work scheme for cryptocurrencies such as Bitcoin. This proof of scheme makes use of the set of transactions to construct a giant graph deterministically. The miners are required to find the largest clique of in this graph as a solution to the proof of work puzzle. We also have proposed an Algorithm that can be used to find a solution of this puzzle by performing $O(2^{70.5})$ hash calculations which is commensurate to the hashpower of the current Bitcoin network.

Acknowledgement

Concerning this work, the authors were partially supported by JSPS and DST under the Japan-India Science Cooperative Program of research project named:"Computational Aspects of Mathematical Design and Analysis of Secure Communication Systems Based on Cryptographic Primitives." The third author is partially supported by JSPS Grants-in-Aid for Scientific Research KAKEN-15H02711.

参考文献

[1] Noga Alon, Raphael Yuster, and Uri Zwick. Finding and counting given length cycles (extended abstract). In *Proceedings of the Second Annual European Symposium on Algorithms*, ESA '94, pages 354–364, London.

[2] Coen Bron and Joep Kerbosch. Algorithm 457: Finding all cliques of an undirected graph. *Commun. ACM*, 16(9), pages 575–577,September 1973.

[3] Norishige Chiba and Takao Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14(1), pages 210–223, February 1985.

[4] G. R. Grimmett and C. J. H. McDiarmid. On colouring random graphs. *Mathematical Proceedings of the Cambridge Philosophical Society*,volume 77, pages 313–324, 1975.

[5] D.W. Matula. On the complete subgraph of random graph. In *Comb. Math and its Appl*, pages 356–369, Chappel Hill, N.C., 1970.

[6] J.W. Moon and L. Moser. On cliques in graphs. *Israel Journal of Mathematics*, 3(1), pages 23–28, 1965.

[7] John M Robson. Finding a maximum independent set in time o (2n/4). Technical report, Technical Report 1251-01, LaBRI, Université de Bordeaux I, 2001.

[8] Etsuji Tomita, Akira Tanaka, and Haruhisa Takahashi. The worst-case time complexity for generating all maximal cliques and computational experiments. *Theor. Comput. Sci.*, 363(1),pages 28–42, October 2006.

[9] John Tromp. Cuckoo cycle: a memory bound graph-theoretic proof-of-work. Cryptology ePrint Archive, Report 2014/059, 2014. `http://eprint.iacr.org/`.