CHRISIMOS: A USEFUL PROOF-OF-WORK FOR FINDING MINIMAL DOMINATING SET OF A GRAPH

A PREPRINT

Diptendu Chatterjee
Computer Science and Information Systems Department
BITS Pilani, K K Birla Goa Campus, India
diptenduc@goa.bits-pilani.ac.in

Prabal Banerjee
Avail and Indian Statistical Institute Kolkata, India
mail.prabal@gmail.com

Subhra Mazumdar
TU Wien, and CDL-BOT, Vienna, Austria
subhra.mazumdar@tuwien.ac.at

July 26, 2023

ABSTRACT

Hash-based proof of work (PoW) used in the Bitcoin blockchain leads to high energy consumption and resource wastage. In this paper, we aim to re-purpose the energy by replacing the hash function with real-life problems having commercial utility. We propose *Chrisimos*, a useful proof-of-work based where miners are required to find a *minimal dominating set* for real-life graph instances. A miner who is able to output the *smallest dominating set* for the given graph within the block interval time wins the mining game. We also propose a new chain selection rule that ensures the safety and liveness of the mechanism. Thus our protocol also realizes a decentralized *minimal dominating set* solver for any graph instance. We provide formal proof of correctness and show via experimental results that the block interval time is within feasible bounds of hash-based PoW.

Keywords Blockchain, Useful Proof-of-Work, Graphs, Minimum Dominating set, NP-complete problem, Heuristic methods

1 Introduction

Cryptocurrencies have revamped the Banking and Financial Sector. Parties can execute transactions without relying on any trusted entity. Blockchain forms the backbone of these cryptocurrencies or tokens, and any transaction added to the blockchain is secure and tamper-proof Xiao et al. (2020). In permissionless blockchains, anyone can join and partake in the consensus without needing any approval Tasca and Tessone (2017).

Permissionless networks are prone to Sybil attacks. An attacker creates an artificially large number of fake identities Douceur (2002) with the intention of controlling the network and skewing the results of majority decisions on the ordering of transactions. Thus, permissionless blockchains use a Sybil-resistant mechanism called Proof of Work (PoW) Nakamoto (2008) that capitalizes on the computational resource invested by a participant to reach a consensus. PoW uses cryptographic primitives like hash hence the name hash-based PoW. Hash functions are computationally difficult with efficient verification. It becomes difficult for the attacker to interfere with the consensus without solving such puzzles. This procedure is also called mining. However, PoW-based mining is computationally expensive. Bitcoin is heavily criticized for the enormous energy consumption needed for mining Platt et al. (2021). As per the report in January 2023 Huestis (2023), Bitcoin is estimated to have an annual power consumption of 127 `terawatt-hours (TWh)` - more than many countries, including Norway. It is estimated to emit from `25 to 50 million tons` of CO_2 each year. This is on par with the annual emissions from diesel fuel used by US railroads. The large energy usage associated with PoW mining is an inevitable consequence of the fundamental nature of the algorithms.

Alternate consensus protocols like Proof-of-Stake King and Nadal (2012); Kiayias et al. (2017), Proof-of-Space Dziembowski et al. (2015); Park et al. (2018), Proof-of-Spacetime Moran and Orlov (2019), Proof-of-Authority Igor Barinov (2018), Proof-of-Burn Karantias et al. (2020a), etc overcome the problem by quantifying work done based on the utilization of other resources that have less or no adverse impact on nature. However, an alternate consensus has its own problems. Proof-of-Stake and Proof-of-Authority Farrington (2022) coins eliminate democracy and move the power to the "richest in the room". The problem with Proof-of-Space de Jong (2023) is that more miners are added to the network, the more storage space is required. The disadvantage of Proof-of-Spacetime is that it requires users to lock up a certain amount of coins in order to store data on the blockchain, which can limit participation in the network and create a barrier to entry for new users de Jong (2023). Proof-of-Burn Karantias et al. (2020b) still wastes resources needlessly. It is also questioned that mining power simply goes to those who are willing to burn more money. On the other hand, hash-based PoW aids in the decentralization of transactions while removing intermediaries and assuring transaction validity. The mining consensus offers equal opportunity to everyone and new miners are incentivized to add more hardware and spend more energy to receive their share of the mining rewards.

Instead of changing the underlying consensus mechanism, we try the approach of redirecting the energy consumed in PoW in solving a problem that has some utility in the real world. Proof of useful work (PoUW) Ball et al. (2017) utilizes the energy of the miners to perform some useful task that has either commercial utility or academic purpose. Such systems generally focus on NP-complete problems. This class of problems perfectly fits into blockchain systems because identifying the solution in the first place has no known polynomial algorithm but the solution to the problem can be verified in polynomial time Oliver et al. (2017). Early designs and implementations like Primecoin King (2013) and Gapcoin King (2014) are based on number theoretic hardness but these protocols do not solve any problem of utility. Also, the hardness of these problems can be manipulated by the miners compromising the security of the Blockchain. Some other works based on graph-theoretic PoW Tromp (2015), orthogonal-vector-based PoW Ball et al. (2017), and PoW based on the generalized birthday problem Biryukov and Khovratovich (2017) exist but these problems do not have much utility apart from academic use. Our proposed PoW solves a graph-theoretic problem of practical utility and the miner has no capability of manipulating the problem instance. Note that our work does not intend to make existing PoW environment-friendly. Our objective is to redesign PoW so that the energy consumed in mining a block is used for solving problems that have real-world value.

1.1 Contributions

We summarize the contributions as follows:

(i) We propose *Chrisimos*, a useful PoW where miners find a minimal dominating set of a real-life graph instance instead of generating nonce for hash. Our protocol leverages the computational hardness of *Minimum Dominating Set* (MDS) problem. There is no exact method that outputs the minimum dominating set of a graph instance in polynomial time. A miner chooses the heuristic of its choice that returns a minimal dominating set of a graph.

(ii) We use the greedy heuristic for finding a minimal dominating set discussed in Alon and Spencer (2016) on publicly available graph datasets to estimate the time taken for mining a block using our proposed PoW. The estimated time corresponding to a graph instance is recorded in a publicly available lookup table. We use this lookup table to estimate *block interval time* for any new graph instance. Any miner submitting the smallest dominating set in a given block interval time wins a block reward and adds his block to the chain. This induces competition among miners to return the best solution in the given block interval time. Thus, our protocol simulates a decentralized minimal dominating set solver.

(iii) We define a new chain selection rule to select the correct chain in the event of a fork. Instead of following the standard longest chain rule Nakamoto (2008), we define the work done in a chain as the summation of the work done in individual blocks. The work done in a block depends on the size of the graph as well as the computation effort put in by the miner in solving the dominating set problem. We prove that our chain selection rule guarantees the safety and liveness of the proposed scheme.

(iv) We run the greedy heuristic on certain datasets and construct the lookup table for block interval time. We observe that the block interval time is within feasible bounds of hash-based PoW, demonstrating the practicality of our approach.

1.2 Outline

The rest of the paper is organized as follows: in Section 2 we provide the background needed for comprehending our work, in Section 3 we provide a detailed description of *Chrisimos*, correctness, and complexity analysis of the scheme are discussed in Section 4, we discuss a new chain selection rule in Section 5, in Section 6 we discuss the security of

our proposed PoW scheme, experimental analysis in Section 7 shows that *Chrisimos* adds a block to the chain within a bounded time similar to hash-based PoW, we review the relevant literature in Section 8, and finally we conclude the paper in Section 9.

2 Background and Notations

We state the necessary background required for comprehending our work. If we mention network then it refers to the Bitcoin P2P network and if we mention graph then it refers to the input instances on which the miner needs to find a minimal dominating set.

2.1 Hash-based PoW

Bitcoin P2P network uses Adam Back's Hashcash Back et al. (2002) for PoW. A mining node constructs a block that has a transaction and a header. A difficulty target is specified in the block header that defines the required proof of work difficulty to make this a valid block. A miner has to find a nonce value, such that the result of applying the `SHA256` hash function to a tuple consisting of her public key, the hash of the previous block, the hash of the current block that is being added, and this nonce value, satisfies the difficulty target. The hash function is denoted as $\mathcal{H}$. Finding the nonce is akin to rolling a many-sided die and getting the result. Given the one-way property of hash functions, the only strategy for finding the valid nonce is to repeatedly try randomly-generated nonces until one of them solves the problem. At least one miner must have found the desired nonce whenever a block gets added to the chain Antonopoulos (2014). On average, it takes about ten minutes to construct a block in the Bitcoin network. The difficulty target is defined as the number of zeros that are required for the hash to be a valid solution and it is updated regularly to ensure that the time to construct a block remains nearly equal to ten minutes. Miners are rewarded with the block mining fee upon finding the correct nonce.

Since mining takes a nontrivial amount of computation and the use of specialized hardware, it may not be possible for an individual miner to find the nonce on her own. Usually, miners join any existing mining pool. If any member of the pool succeeds in finding the nonce, others earn a fraction of the block reward based on the computation power invested. The mining pool is administrated by a miner called a pool operator. The pool operator is solely responsible for the distribution of jobs to individual miners as well as for sharing the reward if the pool succeeds in adding a block to the chain Antonopoulos (2014).

2.2 Dominating Set Problem

A dominating set D of a graph $G(V, E)$ can be defined as a subset of vertices $V' : V' \subseteq V$, and every vertex in G is either in V' or is adjacent to some vertex in V' Chlebík and Chlebíková (2004).

A set D is minimal dominating set of the graph G if it does not contain any other dominating set as a proper subset. A minimal dominating set of G of minimum cardinality is called *minimum dominating set* (MDS). Computing an MDS is NP-hard. The decision version of the problem i.e. *dominating set problem* is NP-complete. It is defined as *"Given a graph $G(V, E)$ and an integer k, does G have a dominating set of size less than k?"*Garey and Johnson (1979)

Dominating sets of a graph has a lot of applications in real life Rao et al. (2020). Nodes in Wireless Sensor Networks (WSNs) are energy constrained. Backbone nodes are subset of nodes in such network that reduces the communication overhead, increases the bandwidth-efficiency, and decreases the overall energy consumption Asgarnezhad and Torkestani (2011). Such set of nodes also forms the dominating set. Online social network sites like Facebook, MySpace and Twitter are among the most popular sites on the Internet. Dominating sets have an important role social networks to spread ideas and information within a group. It can be used for targeted advertisements, alleviate social problems in the physical world like drinking, smoking, drug use and so on Wang (2014). Due to the financial limitation of the budget, it becomes important to effectively select the nodes to realize the desired goal.

3 Our Proposed Solution

In this section, we describe the protocol where the miner needs to find out the dominating set of a graph instance instead of finding a nonce for a hash.

3.1 System and Adversarial Model

There are two types of participants in our protocol: (i) any utility company supplying a graph instance and (ii) miners of the Bitcoin network. Each miner can mine and verify the blocks. By any utility company, we mean it could be any social networking company or company providing telecommunication services. Such networks change quite frequently with time so there remains a steady flow of input to the Bitcoin network. Utility companies earn high profits by utilizing the dominating sets to realize their objective. Thus the company shares a portion of their profit as remuneration to the miner who solve the dominating set for the given graph instance. We assume that any utility company submits its graph instances to a common public platform. The identity of the utility company is not revealed. Instead of storing the graph instance, miners can download the graph instances from that platform. Each graph instance is assigned an ID and IDs are assigned serially in increasing order. So the latest graph will have the highest numeric ID. A representative of the public platform announces the graph instance to the Bitcoin network.

3.2 Phases of the protocol

We define the different phases of *Chrisimos*:
(i) *Transaction Selection for Block*: A miner creates a block $\mathcal{B}$ with a set of valid transactions it receives along with the coinbase transaction and reward set by the utility company to create set $\mathcal{T}$.
(ii) *Preprocessing Phase*: Miner receives a graph $G(V, E)$, graph id G_{id}, hash of the graph $\mathcal{H}(G)$, minimum degree of the graph δ, deadline T_{max} within which a new block must be added to the chain, and signature on this hash $\mathcal{H}(G)$ as input. The signature on the hash ensures that the graph instance has been provided from the designated address supplying problem instances.

a. *Verification of Input*: The miner verifies the source of the graph and checks the correctness of G by comparing it with the signed hash of the graph. Additionally, it checks if the id G_{id} is higher than the graph id of the previous instance solved and added in the last block.

b. *Input Transformation*: Given the graph G, the miner transforms it to $G_T(V_T, E_T)$. The transformation is dependent on the transaction set of the current block and the hash of the previous block.

(iii) *Mining of Block*: The miner finds the dominating set of this extended graph G_T. It adds the dominating set to the block $\mathcal{B}$ and broadcasts it to the rest of the network.
(iv) *Block Verification*: A miner checks if the cardinality of the dominating set in $\mathcal{B}$ is less than the solution already stored in the given block interval time. If so, then the miner checks whether the dominating set in the block $\mathcal{B}$ is correct or not and updates the stored solution as well as updates the last stored block to $\mathcal{B}$. Once the block interval time elapses, the miner adds the last stored block to the blockchain.

We discuss the phases (ii.b.) *Input Transformation*, (iii) *Mining of the Block* and (iv)*Block Verification* in detail.

Input Transformation

We had mentioned that the graph instance must be transformed so that it becomes unique to the particular block. Since all miners work on the same instance at a given epoch, transforming the graph prevents another miner from stealing the solution fetched by a miner during block propagation. Also, miners may try to exploit the system by reusing a solution of a previous graph instance and adding a new block without expending any effort. We need the problem instance to be dependent on the content of the block. Each miner has to form a block, include its own set of transaction and include its own coinbase transaction, form the merkle tree, compute the merkle root. This merkle root and hash of the previous block forms the basis of transforming the block. So even if a miner M has transformed graph G to G_T and found a solution for G_T, another miner M' can definitely get the solution of G using the reverse transformation specified by M. But it has to send its own block in order to claim the reward. The block is valid provided M' has found a dominating set on its own transformed graph. Even if transaction set used by M and M' match for creating the block, the coinbase transaction of M and M' will be different. Even though M' knows the minimum dominating set on graph G, it will again need to recompute the dominating set on the transformed graph.

In our protocol, the transformation involves adding additional vertices and edges to G and extending it. We extend the graph instance G by adding another $|V|$ number of vertices. The new nodes are connected to the graph G based on a rule that depends on the Merkle root of the transaction set of the block and the hash of the previous block. The method for extension is discussed as follows and the pseudocode for *Extend* function is mentioned in Algorithm 1.

(a) Miner gets hashes h_1 and h_2, where h_1 is the hash of the previous block and h_2 is the hash of the root of the Merkle tree formed by the transaction set $\mathcal{T}$.

Algorithm 1: Extend

Input: G, h_1, h_2
$L \leftarrow K(h_1, h_2)$
Sort V in descending order of degree
Introduce $W = \{w_1, w_2, \ldots, w_{|V|}\}$
$j = 0$
$E_T = E$
for $v \in V$ **do**
  for i *in* $L[0 : |N(v)| - 1]$ **do**
    if *L(i) is 1* **then**
      Add edge e between v and w_{j+1}
      $E_T = E_T \cup \{e\}$
    end
  end
  $j = j + |N(v)| - 1$
  $L = L + |N(v)|$
end
$AdjList_w \leftarrow GetList(|V|, \delta)$
$temp_j = j = 2$
$k = |W| - 1$
for $w \in W$ **do**
  for $bit \in AdjList_w[0 : k]$ **do**
    if *bit is 1* **then**
      Connect w and w_j with edge e
      Add e to E_T
    end
    $j = j + 1$
  end
  $AdjList_w = AdjList_w + k + 1$
  $k = k - 1$
  $temp_j = temp_j + 1$
  $j = temp_j$
end
$V_T = V \cup W$
return $G_T(V_T, E_T)$

(b) We propose a function K which serves as the basis for extending the graph from G to G_T. The function K is defined as follows: it has pair of hashes h_1 and h_2 as inputs and gives an output of size $2|E|$ so $K : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^{|2E|}$, where λ is the security parameter. The output has an equal number of 0's and 1's, and we state the steps for generating the output:

(i) If $2|E| \leq \lambda$, output first $2|E|$ bits of $K(h_1, h_2)$.

(ii) If $2\lambda \geq 2|E| > \lambda$, the output of $K(h_1, h_2)$ is defined as follows: first λ bits is $\mathcal{H}(h_1, h_2)$, denoted as L_1, and rest of the $min(2|E| - \lambda, \lambda)$ bits are the $1's$ complement of L_1.

(iii) If If $2|E| > 2\lambda$, we generate the output of $K(h_1, h_2)$ follows: the first 2λ bits are generated by following steps (i) and (ii), the rest $2|E| - 2\lambda$ bits by concatentating the sequence of 2λ bits for $\lfloor \frac{2|E|}{2\lambda} \rfloor$ times. The last block has the first $|2E| - (2\lambda \lfloor \frac{2|E|}{2\lambda} \rfloor)$ bits from the sequence of 2λ bits concatenated, so we have an output L of size $2|E|$. Since L has an approximately equal number of 0's and 1's, the number of edges connecting V to $V_T \setminus V$ is $|E|$.

(c) The graph G is represented as a set of n ordered pairs, i.e., $G = \{(v_i, N_i) | v_i \in V \text{ and } N_i = N(v_i)\}, \forall i \in \{1, \ldots, n\}$, where $|N_i| \geq |N_j| \ \forall i < j$ where $N(v_i)$ is the set of neighbors of v_i. This implies that the vertices of G are sorted in decreasing order of degree, where v_1 is the vertex with the maximum degree, followed by v_2 whose degree is the second maximum, and this ends with the vertex v_n having the minimum degree.

(d) We introduce $|V|$ new vertices labeled $w_i, \forall i \in \{1, \ldots, |V|\}$. These vertices form part of the extended graph $G_T(V_T, E_T)$ where $V_T = 2|V|$, but they are assigned to the set $V_T \setminus V$.

(e) Connect vertices in V and $V_T \setminus V$, where connections of w_i with neighbours of $v_i, 1 \leq i \leq |V|$ will be dictated by specific $|N_i|$ bits of L ranging from $[\sum_{m=1}^{i-1} |N_m| + 1, \sum_{m=1}^{i} |N_m|]$, the connection rule is as follows: *For $1 \leq i \leq |V|$, if the l^{th} bit among these $|N_i|$ bits of L is 1, then w_i will have an edge with the l^{th} element of the array N_i, where N_i has the neighbors of v_i as its element. If the l^{th} bit is 0, then w_i is not connected to l^{th} neighbor of v_i.*

(f) *Procedure GetList($|V|, \hat{\delta}$)*: To connect the vertices w in $V_T \setminus V$, it must be ensured that the probability an edge exists between pair of vertices is $p > \frac{\delta}{n}$ where δ is the minimum degree in G. We justify the constraint on p in Lemma 6. We choose $p = \frac{\hat{\delta}}{n}$ where $\hat{\delta} = 2\delta$ and $\hat{\delta} \in \mathbb{N}$. The miner must therefore generate an adjacency list, where at least $p\binom{n}{2}$ edges exist. It is possible for the miner to extend the graph from G to G_T and send it as part of the block header. However, a graph may have a size of a few MBs and in bitcoin, the block size itself has a size of 1 MB. Blowing up the size of the block header from 80 B to a few MBs will lead to larger-sized blocks and introduce propagation delay. Hence the miner must send out the minimum information needed to generate graph G_T. Miner fixes a rule that allows generating a bit-sequence of length $\binom{n}{2}$ encoding adjacency list of graph $G_T \setminus G$. So first vertex w_1 will have a list encoding with the rest of $(n-1)$ vertices, i.e., $w_2, w_3, \ldots, w_n$, for w_2 the list size is $(n-2)$ encoding connections with $w_3, \ldots, w_n$ and so on. A bit 1 denotes the existence of an edge. Along with this, the miner needs to ensure that the number of 1's in the bits-sequence must be at least $p\binom{n}{2}$. We discuss a method by which a miner/verifier can easily generate the connections between vertices in $G_T \setminus G$.

- $\lfloor \frac{n}{\hat{\delta}} \rfloor$ no. of bits for each chunk is selected and next we follow the following rule. To set the first chunk :
  - If $\hat{\delta}$ is odd, set first $\lfloor \frac{n}{\hat{\delta}} \rfloor - 1$ bits as 0s and last bit 1, or $x = (\lfloor \frac{n}{\hat{\delta}} \rfloor - 1)'s 0|1$.
  - If $\hat{\delta}$ is even, set first bit as 1 and rest $\lfloor \frac{n}{\hat{\delta}} \rfloor - 1$ bits as 0s , or $x = 1|(\lfloor \frac{n}{\hat{\delta}} \rfloor - 1)'s 0$.
- The second chunk of $\lfloor \frac{n}{\hat{\delta}} \rfloor$ bits are the mirror image of x, the third chunk of $\lfloor \frac{n}{\hat{\delta}} \rfloor$ bits is left cyclic shift of x, the fourth chunk is mirror image of third chunk, this goes on till we generate $\binom{n}{2}$ bits.
- So the generalized rule is any $(2i+1)^{th}$ chunk is left cyclic shift of $(2i-1)^{th}$ chunk, and $(2i+2)^{th}$ chunk is mirror image of $(2i+1)^{th}$ chunk till we get $\binom{n}{2}$ bits. This ensures we have at least $\frac{\hat{\delta}(n-1)}{2}$ number of 1's in total.
- We generalize the indices having bit 1 in the bit-sequence depending on whether δ is even or odd:
  (a) Indices that have bit 1 if $(\hat{\delta} \mod 2 = 1)$ are $(2m+1) * \frac{n}{\hat{\delta}}$ and $(2m+1) * \frac{n}{\hat{\delta}} + 1$ where $0 \leq m \leq \lfloor \frac{(n-1)\hat{\delta}-1}{2} \rfloor$.
  (b) Indices that have 1 if $(\hat{\delta} \mod 2 = 0)$ are $1 + 2m * \frac{n}{\hat{\delta}}$ and $2(m+1) * \frac{n}{\hat{\delta}}$ where $0 \leq m \leq \lfloor \frac{(n-1)\hat{\delta}-1}{2} \rfloor$.

The extend function returns the graph $G_T(V_T, E_T)$ where $n' = |V_T| = 2|V|$ and $|E_T| = 2|E| + p\binom{n}{2} = 2|E| + \frac{\hat{\delta}(|V|-1)}{2}$. We also illustrate the steps of input transformation via an example.

Example 1 *We illustrate our proposed mining procedure through Fig.1. The block B_{k+1} uses hash of block B_k and Merkle root* 12345678 *for transforming the input instance G with 9 vertices $v_1, v_2, \ldots, v_9$ to G_T with 18 vertices $v_1, v_2, \ldots, v_9, w_1, w_2, \ldots, w_9$. The miner outputs a minimal dominating set of G_T as $\langle v_1, v_4, v_5, v_6, w_2, w_6, w_8 \rangle$. We discuss in detail the steps for transforming the input G to G_T, estimating the block interval time T^G_{max}, and finally, retrieving a dominating set of G_T.*

Mining of the Block

The miner finds the dominating set of size either equal or lower than the permissible size of dominating set within a given block interval time T^G_{max}, prespecified in the lookup table. We provide the pseudocode in Algorithm 2.

It broadcasts a block $\mathcal{B}$ containing the transaction set $\mathcal{T}$ (includes the coinbase transaction and reward for finding dominating set of G_T), the hash of previous block h_1, Merkle root of the transaction set $\mathcal{T}$ denoted as h_2, address for fetching the graph G along with the id of the graph, $\mathcal{H}(G)$, signature on $\mathcal{H}(G)$, minimum degree δ, and the dominating set of G_T.

Block Verification

When a verifier starts receiving blocks that have mined graph $G(V, E)$, it sets a variable $past\_size_{DS} = 2|V|$. At the very beginning, it will continue to store the solution and check until it gets a block that has provided a valid dominating

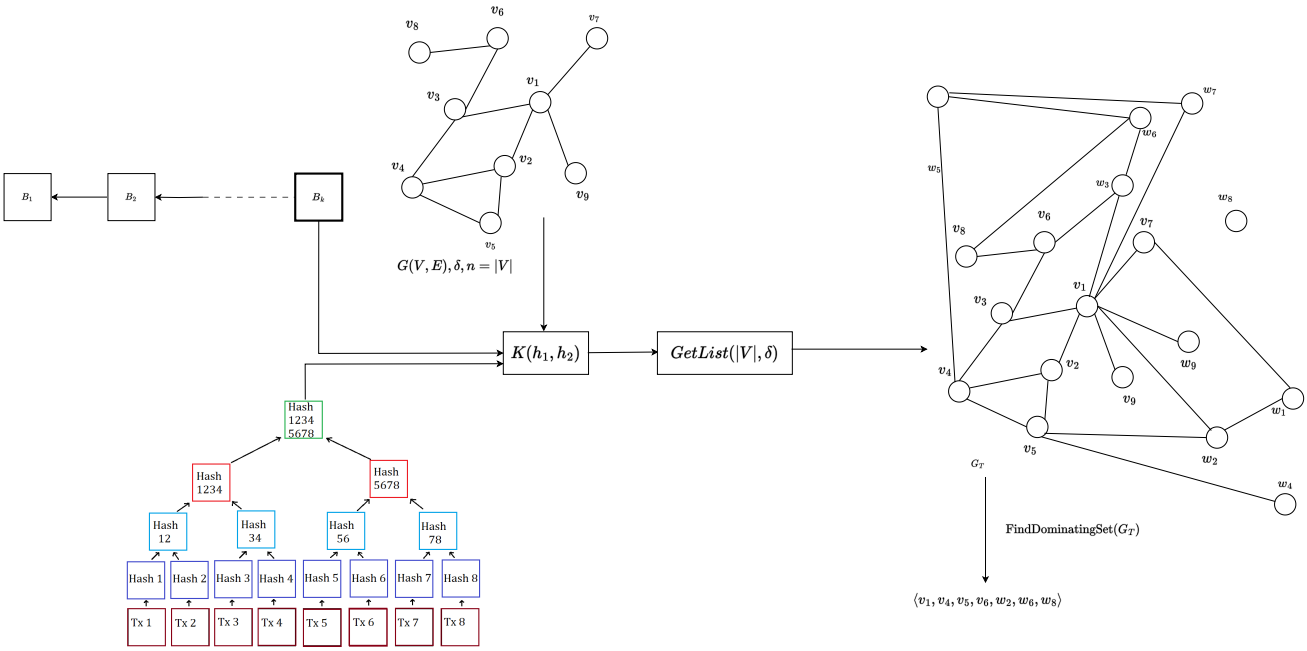

Figure 1: An instance of *Chrisimos* for adding a new block

Algorithm 2: Block Generation

Input: $G(V, E), G_{id}, H, \sigma_H, pk_G, \mathcal{T}, h_1$
if $H \neq \mathcal{H}(G)$ *or* $SigVerify(\sigma_H, H, pk_G) \neq 1$ *or* $G_{id} \leq GetPrevBlockGraphid()$ **then**
  abort
end
Compute $h_2 = MerkleRoot(\mathcal{T})$
Compute $G_T = extend(G, h_1, h_2)$
$\mathcal{D} = ComputeBound(|V_T|, \delta_{G_T})$
$DS' = V$
while $|DS'| > \mathcal{D}$ *and time elapsed* $< T^G_{max}$ **do**
  $DS(G_T) \leftarrow$ FindDominatingSet(G_T)
  if $|DS(G_T)| \leq \mathcal{D}$ **then**
    $DS' \leftarrow DS(G_T)$
  end
end
if *time elapsed* $< T^G_{max}$ *and* $DS' \leq \mathcal{D}$ **then**
  block_header $= HeaderGen(h_1, h_2, DS')$
  $\mathcal{B} \leftarrow BlockGen($block_header$, \mathcal{T})$
  return $\mathcal{B}$
end
else
  abort
end

set. $past\_size_{DS}$ is assigned to the cardinality of this dominating set. Now the verifier will cache the block until it gets a block with dominating set of sizes lower than the previous solution. The verifier will continue to verify solutions till the deadline T^G_{max} elapses. After the timeperiod for G is over, it will accept the last block it had stored and solutions appearing after T^G_{max} will be rejected. Miners who verify the block reach a consensus on the best result in the given epoch and a new block is added to the blockchain.

Given a block $\mathcal{B}$, the verifier scans for the transaction set $\mathcal{T}$ in the block. From the block header, the verifier gets the dominating set $DS(G_T)$ of the extended graph G_T, id of the graph G denoted as G_{id}, hash $\mathcal{H}(G)$, signature σ_H on

Algorithm 3: Block Verify

Input: $\mathcal{B}$
Parse $\mathcal{B}$ to get block header and transaction set $\mathcal{T}$
Get $h_1, h_2, G_{id}, H, \sigma_H, \delta$, dominating set $DS(G_T) = V_{ds} \cup W_{ds}$ and download graph G using G_{id}
Set $visited\_set = \phi$
if $h_2 \neq MerkleRoot(\mathcal{T})$ *or* $H \neq \mathcal{H}(G)$ *or* $SigVerify(\sigma_H, H, pk_G) \neq 1$ *or current time* $\geq T^G_{max}$ *or* $G_{id} \geq GetPrevBlockGraphid()$ *or* ($past\_size_{DS} < 2|V|$ *and* $past\_size_{DS} < |DS(G_T)|$) **then**
  reject solution
end
else
  $M \leftarrow \mathcal{H}(h_1, h_2)$
  $Indices_M \leftarrow calc\_index(M)$
  for v *in* V_{ds} **do**
    Mark v as visited
    Add v to $visited\_set$
    for $v' \in N(v)$ **do**
      If v' is not visited then mark it visited
      Add v' to $visited\_set$
    end
    for $v'' \in Indices_M(N(v))$ **do**
      Mark neighbor $v'' \in V_T \setminus V$ as visited if not visited and add v'' to $visited\_set$
    end
  end
  $AdjList_w \leftarrow GetList(|V|, \hat{\delta})$
  for v *in* W_{ds} **do**
    Mark v as visited
    Add v to $visited\_set$
    for $w \in AdjList_w(N(v))$ **do**
      Mark neighbor w as visited if not visited and add w to $visited\_set$
    end
    for v *in* $V \setminus visited\_set$ **do**
      for $v'' \in Indices_M(N(v))$ **do**
        Mark v'' as visited if not visited and add v'' to $visited\_set$
      end
    end
  end
  if $|visited\_set| = 2|V|$ *and* $past\_size_{DS} > |DS(G_T)|$ **then**
    $past\_size_{DS} = |DS(G_T)|$
  end
  else
    reject solution
  end
end

$\mathcal{H}(G)$, hash of the previous block h_1, and Merkle root of the transaction set $\mathcal{T}$ denoted as h_2. It downloads G from the public domain using G_{id}, retrieves the deadline T^G_{max}, checks if graph G provided is correct and whether G_{id} is greater than the instance id of the graph solved previously. Verifier checks if $\mathcal{T}$ is correct, constructs the Merkle tree using $\mathcal{T}$, and checks the correctness of h_2. The solution $DS(G_T)$ can be written as $V_{ds} \cup W_{ds}$. This is because some vertices of graph $G(V, E)$, denoted as V_{ds}, and some vertices in $G_T \setminus G$, denoted as W_{ds} will cover the graph.

The verifier first checks whether the vertices in V_{ds} cover the rest of the vertices G. If not all vertices are covered then set the variable *uncovered* to true. Next, the verifier calculates the indices that show which vertices in $G_T \setminus G$ are connected to vertices in G and checks how many vertices in $V_T \setminus V$ are also covered by V_{ds}. To calculate the indices, the verifier generates an output of size λ by hashing over inputs h_1 and h_2. Let the output be M. The verifier can now easily infer the positions of 1 in the bit sequence of size $2|E|$ if he knows the position of 1 in the bit sequence of size λ. We use the method $calc\_index$ on input M for calculating the indices and store it in $Indices_M$. Once the verifier

gets the indices, it will figure out which vertices in $V_T \setminus V$ got covered. We explain the function $calc\_index$ with an example as follows.

Example 2 *For ease of analysis, we consider the hash* $|H(h_1, h_2)|$ *as* 10*. If we have* 2|E| *of size* 200*, and* $\mathcal{H}(h_1, h_2)$ *generates a* 10-bit *hash* 0101010110*, so at indices* 2, 4, 6, 8, 9 *the bit is* 1*. The next block of* 10-bit *is the complement of the first block, and that will be* 1010101001*. As per the rule, if the position* i *in this binary string of length* k=10 *is* 1*, then all the bits* $mk + i$ *will be* 1*, where* $m \mod 2 = 0$ *and* $mk + i < 2|E|, m \in \mathbb{N}$*. So bits at positions* 2, 22, . . . , 182*,* 4, 24,. . . , 184*,* 6, 26,. . . , 186*,* 8, 28,. . . , 188*,* 9, 29,. . . , 189 *will be 1. So a total of* 50 *indices can be calculated where the bit will be* 1*. We had mentioned that the bits from* $k + 1$ *to* $2k$ *are the complement of the first* k *bits. So if a bit* i *is 0 in the binary string of size* k*,* $zk + i$ *will be* 1 *where* $z \mod 2 = 0$*. Since bit positions* 11, 13, 15, 17, 20 *are* 1*, so bits at positions* 11, 31,. . . , 191*,* 13, 33,. . . , 193*,* 15, 35,. . . , 195*,* 17, 37,. . . , 197*,* 20, 40,. . . , 200 *will be* 1*, so in total* 100 *indices are calculated where the bit will be* 1*. The verifier has to generate a binary sequence of size* k *and then using this rule, he can extend it to size* $2|E|$*.*

If the variable *uncovered* is true, the verifier checks if the vertices in W_{ds} cover the rest of the uncovered vertices in G. After this step, the verifier checks if the remaining uncovered vertices in $V_T \setminus V$ get covered by checking the connection between the vertices in $V_T \setminus V$. A bit-sequence of length $\binom{n}{2}$ encoding adjacency list of graph $G_T \setminus G$ has been discussed previously. Since the number of 1's in the bit string is $\hat{\delta}\frac{(n-1)}{2}$, it suffices for the verifier to calculate these indices. If the verifier finds that not all vertices are covered then the solution is discarded. The pseudocode is mentioned in Algorithm 3.

Block Reward: A miner who managed to submit a dominating set of least cardinality within the given time bound wins the mining game and gets the block reward. The block reward comprises three parts: fee from the coinbase transaction, the fee from the transactions, and the fee provided by the utility company for solving the dominating set of the graph. The lookup table provides an estimate of the block interval time for a graph instance, indirectly providing some insight into the hardness of the problem. Thus, we expect a fair utility company to decide on the remuneration directly proportional to the hardness of the problem. Only the miner whose block is added to the chain obtains the reward. Thus any rational miners have the incentive to stick to the protocol and compete for finding the best solution. One could question whether it be would more efficient to simply solve the problem directly, or release the problem in a marketplace to solve it, rather than trying to provoke competition among many miners. But this does not serve our purpose. Our motive is to combine the two individual problems in one system and realize a useful proof of work consensus. The dominating set problem ensures the security of the Bitcoin Blockchain. Since it is a permissionless network, any participant is free to join or leave anytime they want, and no single entity forces other participants to be a part of it.

3.3 Retrieval of solution

A miner executes the search for dominating set on the extended graph G_T but not on graph G. But the utility company wants the dominating set on G. It would be wasting coin if the solution for G_T cannot be mapped into a solution of G. We discuss how the dominating set for G can be easily calculated from $DS(G_T)$. The solution is union of V_{ds} and W_{ds}, where V_{ds} are the vertices from G. The utility company checks if all vertices in V_{ds} cover G. It could also be the case that not all vertices of V_{ds} are required for covering G. Then those redundant vertices can be eliminated and the rest are added to $DS(G)$. If G gets fully covered then W_{ds} is discarded. If not all the vertices of G are covered, then check which vertices in W_{ds} are covering the remaining vertices in G. If a vertex w_i is used for covering some vertices of G then corresponding v_i is added to the dominating set $DS(G)$. The reason behind this logic is already explained in *Extend* function. A neighbor of v_i can be connected to w_i with probability $\frac{1}{2}$. We provide the pseudocode in Algorithm 4 and argue in Section 4.2 why this mapping would result in a good solution for G in expectation.

3.4 Constructing the Lookup Table

In hash-based PoW, a difficulty target is set and adjusted to stabilize latency between blocks Antonopoulos (2014). Robustness of the system is ensured as the computational power of the network varies with miners joining and leaving the network. However, in our protocol, real-life problem instances are submitted to be solved by the miners. The instances may significantly differ in the difficulties. In addition, the actual difficulty of each particular instance may not be known. Thus, it is necessary to estimate the time taken to generate and verify the block, which is the *block interval time*. This is denoted as T^G_{max} and a new block in added within this time interval upon solving dominating set for the graph instance G.

Algorithm 4: Find $DS(G)$

Input: Block B
Parse block header of B to get $DS(G_T), G_{id}, h_1, h_2, \delta$ and retrive transaction set $\mathcal{T}$ from B
Parse dominating set $DS(G_T) = V_{ds} \cup W_{ds}$
Set $visited\_set = \phi$
Set $DS(G) = \phi$
for v *in* V_{ds} **do**
  Mark v as visited
  Add v to $visited\_set$
  Add v to $DS(G)$
  for $v' \in N(v)$ **do**
    If v' is not visited then mark it visited
    Add v' to $visited\_set$
  end
  if $|visited\_set| = |V|$ **then**
    return $DS(G)$
  end
  $M \leftarrow \mathcal{H}(h_1, h_2)$
  $Indices_M \leftarrow calc\_index(M)$
  for v *in* $V \setminus visited\_set$ **do**
    for $w \in Indices_M(N(v))$ **do**
      Mark w as visited if not visited and add w to $visited\_set$
      $v' \leftarrow map\_index(w, G)$
      Add v' to $DS(G)$
    end
  end
end

Estimating Block Interval Time

The block interval time comprises (a) block generation and (b) block verification. We had discussed previously that the crux of block generation is solving dominating set for the given graph instance. In this section, we discuss how a miner solves the dominating set problem.

(a) *Block Generation*: In Algorithm 2, we have introduced the function `FindDominatingSet` over the extended graph G_T without defining it. The miners are allowed to use an algorithm of their own choice. It computes the upper bound on the size of dominating set as stated in Theorem 1 and uses a method for estimating a solution that would lie within this bound. We have already proved in Theorem 3 that the greedy heuristic returns such a solution within polynomial time. Also, we have used the greedy heuristic while designing the lookup table for estimating the block interval time. Hence, a miner needs to find a threshold on the size of the solution and based on that, plan its mining strategy. Since the block interval time is larger than the estimated time taken to run the greedy heuristic, we assume the mining procedure to work as follows:

- Miner runs greedy heuristic on the extended graph G_T, and gets a solution of cardinality k. The problem miner tries to solve after the previous step: *Does there exist a dominating set for G_T of size less than k?*
- Given that the miner has sufficient time to report the solution, it applies all possible methods to obtain a solution having cardinality as low as possible.

(b) *Block Verification*: The verifier receives the solution and chooses the lowest cardinality result. It never publicly announces the state of the solution received from the other miners which could have acted as feedback. A miner has no information regarding the cardinality of dominating set that would maximize its chance of winning the mining game. Thus, the mining game induces competition among the miners leading the system to act like a decentralized minimal dominating set solver.

Once a miner adds the block to the chain, the next graph instance is provided after T^G_{max} elapses. We use certain benchmark graph datasets and estimate the time taken to solve the minimal dominating set using a greedy heuristic and verify the result. The estimated time is pre-computed and maintained in a look-up table for all the graph instances. We prove that there exists a dominating set within a feasible bound for any graph provided as dominating set problem

is NP-complete. The proof of the existence of a dominating set justifies that any miner will surely be able to find the solution for a given instance.

Theorem 1 *For a graph G_T with n' vertices and minimum degree δ' there exists a dominating set of size less than $\frac{n'(1+\ln(1+\delta'))}{1+\delta'}$ Alon and Spencer (2016).*

Proof 1 *The proof is provided in Alon and Spencer (2016) and we use it here for justifying the existence of the solution. Given the extended graph $G_T(V_T, E_T)$ with $n' = |V_T|$ vertices and minimum degree δ', we create a dominating set X of G_T by randomly selecting vertices from V_T with probability p. The $E(|X|) = n'p$. Now if we look into the set of vertices in $Y = V_T \setminus X$, then the probability for a vertex $y \in Y$ not to be covered by X is at most $(1-p)^{1+\delta'}$ as the vertex y and all its neighbors (at least δ' in number) can not be a neighbor of any of the vertices in X. We include all such vertices in a set Y_X. So, $E(|Y_X|) = n'(1-p)^{1+\delta'}$ Naturally, $X \cap Y_X = \phi$ and $X \cup Y_X$ give a dominating set of G_T. Here $|X \cup Y_X| = n'p + n(1-p)^{1+\delta'}$. Now we set the value of p to minimize the size of this dominating set.*

$$|X \cup Y_X| = n'p + n'(1-p)^{1+\delta'} \approx n'p + n'e^{-p(1+\delta')} \tag{1}$$

We denote the dominating set of G_T as $DS(G_T) = |X \cup Y_X|$. Taking first-order differentiation of $DS(G_T)$ w.r.t p,

$$\frac{\mathrm{d}DS(G_T)}{\mathrm{d}p} = n'(1-(1+\delta')e^{-p(1+\delta')}) = 0 \implies p = \frac{\ln(1+\delta')}{1+\delta'} \tag{2}$$

Substituting $p = \frac{\ln(1+\delta')}{1+\delta'}$ in Eq.1, the bound on the dominating set is $\frac{n'(1+\ln(1+\delta'))}{1+\delta'}$.

A tighter bound has been proposed in Rad (2019) but we stick to the result stated in Alon and Spencer (2016). Choosing a small bound on the size of dominating set allows a utility company to get a good solution for a given budget. The lower the size of dominating set, the more effective it is in terms of the cost of implementing a certain objective in the graph instance. We analyze the runtime needed for doing an exhaustive search on the solution space for finding a dominating set in $G_T(V_T, E_T)$ of size $\frac{n'(1+\ln(1+\delta'))}{1+\delta'}$.

Theorem 2 *Finding a dominating set of size $k \leq \frac{n'(1+\ln(1+\delta'))}{1+\delta'}$ using exhaustive search for graph $G_T(V_T, E_T)$, where $n' = |V_T|$ and δ' is the minimum degree of the graph has time complexity $\mathcal{O}(e^{n'})$.*

Algorithm 5: Greedy Heuristic for Dominating Set

Input: $G_T(V_T, E_T)$
Set $S = \phi$
Sort V_T in descending order of degree
while $V_T \neq \phi$ **do**
  Select vertex v from V_T
  $S = S \cup \{v\}$
  for $w \in Neighbor(v)$ **do**
    $V_T = V_T \setminus \{w\}$
  end
  $V_T = V_T \setminus \{v\}$
end
return S

Proof 2 *The miners perform an exhaustive search to find the dominating set of size $k \leq \frac{n'(1+\ln(1+\delta'))}{1+\delta'}$, where δ' is the minimum degree of G_T. The following inequality holds from Sterling's second approximation: $(\frac{n'}{k})^k \leq \binom{n'}{k} \leq (\frac{en'}{k})^k$*

When k is the maximum value, the number of subsets of vertices of desired size will be $\binom{n'}{\frac{n'(1+\ln(1+\delta'))}{1+\delta'}}$ and thus we have:

$$(\frac{1+\delta'}{1+\ln(1+\delta')})^{\frac{n'(1+\ln(1+\delta'))}{1+\delta'}} \leq \binom{n'}{\frac{n'(1+\ln(1+\delta'))}{1+\delta'}} \leq (\frac{e(1+\delta')}{1+\ln(1+\delta')})^{\frac{n'(1+\ln(1+\delta'))}{1+\delta'}} \tag{3}$$

We assign $w = \frac{1+\delta'}{1+\ln(1+\delta')}$ which increases with δ' and the inequality becomes $w^{\frac{n'}{w}} \leq \binom{n'}{\frac{n'}{w}} \leq (ew)^{\frac{n'}{w}}$, where $(ew)^{\frac{1}{w}}$ attains highest value e when $w = 1$. Thus we have,

$$\frac{1+\delta'}{1+\ln(1+\delta')} = 1 \implies \delta' = \ln(1+\delta') \tag{4}$$

But this holds true if $\delta' = 0$. For any graph, the minimum degree $\delta' \geq 1$ and $(ew)^{\frac{1}{w}}$ monotonically decreases as w increases beyond 1. Thus for any $\delta' \geq 0$, $(ew)^{\frac{n'}{w}} \leq (e)^{n'}$. So the time complexity of the exhaustive search for a dominating set in a graph of order n' is bounded by $e^{n'}$.

The above proof shows that searching exhaustively would result in an infeasible runtime for block generation. Another paper uses the *Measure and Conquer* approach on NP-hard problem of finding minimum dominating set obtaining a tighter bound $\mathcal{O}(2^{0.598n'})$ Fomin et al. (2009) but this is still exponential in the size of input. To provide an estimate of the runtime that is polynomial in the size of the input, we apply a greedy heuristic in the module `FindDominatingSet` of Algorithm 2 of block generation on several synthetically generated datasets and generate the lookup table. The greedy heuristic works as follows: select the maximum-degree vertex, say v, as the first element of the dominating set S, and discard all the neighbors of v from V_T. Repeat the process by selecting the next maximum-degree vertex in the set $V_T \setminus S$ until $V_T = \phi$. We define it in Algorithm 5. The greedy heuristic returns a dominating set $S : |S^*| \leq |S| \leq \ln \Delta |S^*|$ where $|S^*|$ is the size of the optimal dominating set. It has been proved in Alon and Spencer (2016) that size of the dominating set fetched by the greedy heuristic is within the upper bound.

Theorem 3 *The size of the dominating set fetched by the Greedy Heuristic is within the bound $\frac{n'(1+\ln(1+\delta'))}{1+\delta'}$ Alon and Spencer (2016).*

Proof 3 *We choose the vertices for the dominating set one by one, when in each step a vertex that covers the maximum number of yet uncovered vertices are picked. If we pick a vertex $v \in V_T$, it will cover at least $\delta' + 1$ vertices (including itself). Let's designate this set of vertices as $C(v)$.*

Suppose we select few such v vertices sequentially and the vertices in the union of corresponding $C(v)$'s are covered. Let the number of vertices that do not lie in this union be z. Let's denote such a vertex with u. All such u have their corresponding $C(u) \geq (\delta' + 1)$. If we take the sum over all such $C(u)$, we get at least $z(\delta' + 1)$. This will double count the connections of these z vertices with the other vertices. But these connections can go to at most n' vertices. Using averaging principle we can say that there exists one vertex which is included in at least $\frac{z(\delta'+1)}{n'}$ such $C(u)$ sets. If we select one such vertex out of the z vertices, it will cover at least $\frac{z(\delta'+1)}{n'}$ vertices. The number of uncovered vertices will be at most $z\Big(1 - \frac{\delta'+1}{n'}\Big)$. So we can say that at each step of selection, the number of uncovered vertices is reduced by a factor of $\Big(1 - \frac{\delta'+1}{n}\Big)$.

In the Greedy Heuristic, we start with n' uncovered vertices, thus $z = n'$. After selection of $\frac{n' \ln(\delta'+1)}{\delta'+1}$ vertices sequentially, the number of uncovered vertices will be,

$$n'\Big(1 - \frac{\delta'+1}{n'}\Big)^{\frac{n' \ln(\delta'+1)}{\delta'+1}} < n' e^{-\ln(\delta'+1)} = \frac{n'}{1+\delta'} \tag{5}$$

Thus after having $k = \frac{n' \ln(\delta'+1)}{\delta'+1}$ in the dominating set, we are still yet to select $\frac{n'}{\delta'+1}$ remaining uncovered vertices and form the dominating set. The size of the dominating set will be at most $\frac{n'(1+\ln(1+\delta'))}{1+\delta'}$.

The upper bound for the dominating set provided by Theorem 3 is indeed loose. We can safely claim that the result returned by the greedy heuristic will have a cardinality lower than this.

We describe the procedure to estimate the block interval time for a new graph instance from the lookup table:
(a) If the block generation upon using greedy heuristic on a benchmark instance takes a run-time τ units and the size of dominating set is $k' : k' \leq k$, the miner can possibly find the dominating set of size k' within time $t : 0 < t < \tau$ using another efficient algorithm. If more time is provided, a miner may return a dominating set of cardinality less than k'. We set an interval of $l\tau : l \in \mathbb{R}^+, l > 1$ so that both miner and verifier get enough time to propose and reach a consensus on the block to be added to the existing chain.
(b) When a graph instance $G''(V'', E'')$ with minimum degree δ'' is provided to the network, we check if the vertex count $|V''|$ of the instance matches with an entry of the lookup table. If entry $G'(V', E')$, minimum degree δ', in the lookup table where $|V'| = |V''|$, then that entry is selected. The edge count $|E'|$ of the entry is recorded and the time taken for proposing a new block for the given graph having edge count $|E''|$ will scale up (or scale down) by factor $\frac{(2|E''|+\delta''(|V''|-1))\times|V''|}{(2|E'|+\delta'(|V'|-1))\times|V'|} = \frac{2|E''|+\delta''(|V''|-1)}{2|E'|+\delta'(|V'|-1)}$. The edge count is considered with respect to the extended graphs of both G' and G''.
(c) If there is no entry that matches the vertex count $|V''|$ then the entry with maximum vertex count and less than $|V''|$ is selected. Let that instance be $\hat{G}(\hat{V}, \hat{E})$, with minimum degree $\hat{\delta}$, where vertex count of the entry is $|\hat{V}|$ and edge

count of the entry is $|\hat{E}|$. Time taken for proposing a new block for the given graph having edge count $|E''|$ will scale up (or scale down) by factor $\frac{(2|E''|+\delta''(|V''|-1))\times|V''|}{(2|\hat{E}|+\hat{\delta}(|\hat{V}|-1))\times|\hat{V}|}$.

This method for estimation works because the time taken for solving the dominating set of the extended graph G_T involves extending G to G_T, which has a complexity of $\mathcal{O}(|E|)$. The time taken to find the dominating set using greedy heuristic is $\mathcal{O}(|V|)$ and the time taken for verification is again $\mathcal{O}(|V|)$. Thus, for estimating the time taken for adding the block, we take the product of all the components and figure out the scaling factor. We will explain the details for estimating time complexity in Section 4.3.

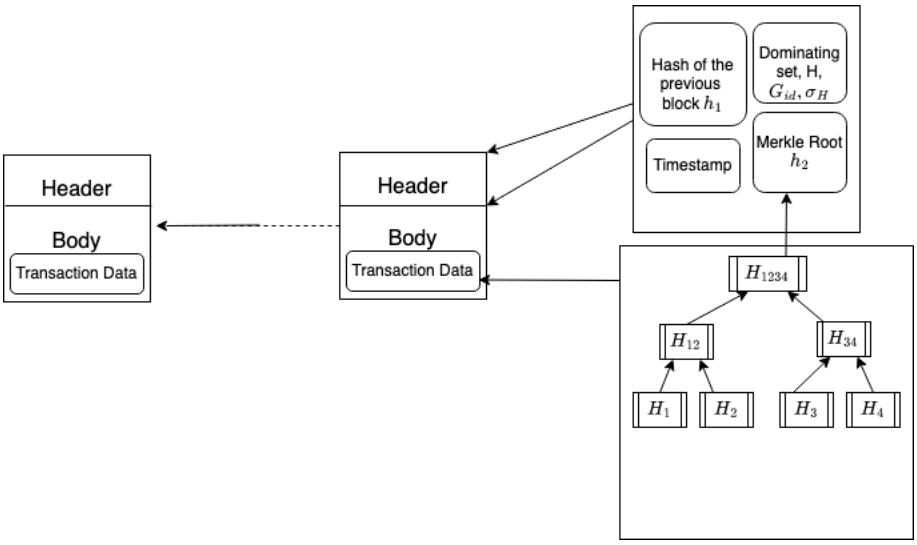

Figure 2: Block Structure when *Chrisimos* is used for generating Blockchain

The structure of a block added to the blockchain is shown in Fig. 2. It is similar to the one used for hash-based PoW except for the nonce part. Instead of the nonce, it now contains id of the graph instance, the hash of the graph instance and a valid signature, the minimum degree of the graph, and the dominating set of the extended graph.

Hardness of the graph instance. It is quite possible that a miner is able to generate a minimal dominating set for a particular graph instance in no time. This does not break the security of the Bitcoin network. The hardness of the problem cannot be pre-decided and a no single entity has control over what sort of instance arrives in the network. Thus we do the complexity analysis of finding a minimal dominating set for any graph instance in general. It is quite possible that generating a minimal dominating set for a particular instance is easy but there is no way to verify whether that is the minimum dominating set. Thus we do a probabilistic analysis and provide proof of the existence of a dominating set. Theorem 1 shows that there exists a dominating set for the given graph. Now a miner can definitely come up with a dominating set of cardinality lower than this bound. Since no one knows the exact solution or what could be termed as *minimum dominating set* for a graph, the probabilistic analysis provides insight into the average case instead of focusing on the corner cases.

4 Analysis of *Chrisimos*

4.1 Correctness of Extend Function

Lemma 4 *The expected number of vertices of G_T covered by the dominating set $DS(G)$ is $\frac{3n}{2}$.*

Proof 4 *The given graph $G(V, E)$ has n vertices namely $v_1, v_2, \ldots, v_n$ and we expand G to $G_T(V_T, E_T)$ using our construction technique where we add n new vertices namely $w_1, w_2, \ldots, w_n$. As per our construction, w_1 is connected to each neighbor of v_1 with probability $\frac{1}{2}$. Now if a vertex v_i has an edge with v_j in G, then w_i has an edge with v_j in G_T with probability $\frac{1}{2}$. So if a vertex v_i is connected to d_i number of vertices of V then the expected number of neighbors of v_i in $V_T \setminus V$ will be $\frac{d_i}{2}$. Using the linearity property of expectation it can be said that the number of vertices in $V_T \setminus V$ covered by a dominating set $DS(G)$ of graph G will be $\frac{n}{2}$. So the expected number of vertices of G_T covered by $DS(G)$ will be $(n + \frac{n}{2}) = \frac{3n}{2}$.*

After considering the dominating set of G, the expected number of vertices in $V_T \setminus V$ left to be covered will be $\frac{n}{2}$. The next theorem gives an upper bound on the size of the dominating set for these $\frac{n}{2}$ vertices.

Lemma 5 *If the probability of an edge in the induced sub-graph of G_T on all the vertices w_i for $i \in \{1, 2, \ldots, n\}$ is p then the expected size of the dominating set of these $\frac{n}{2}$ vertices in this induced graph will be k where k is the minimum integer for which $\frac{n}{2}(1-p)^k < 1$.*

Proof 5 *If we select a random vertex among this $\frac{n}{2}$ vertices for inclusion in its dominating set, then the expected number of vertices it will cover is $(\frac{n}{2} - 1)p + 1 \approx \frac{np}{2}$. So the remaining vertices to cover will be $(\frac{n}{2} - \frac{np}{2}) = \frac{n}{2}(1-p)$. So selecting a vertex in the dominating set will reduce the size of the set of vertices to cover by a factor of $(1-p)$. Similarly after selecting k such vertices in the dominating set the remaining number of vertices to cover will be $\frac{n}{2}(1-p)^k$. When $\frac{n}{2}(1-p)^k < 1$ then there will be no vertex left to cover. From this inequality, we get the minimum k which is also the cardinality of the dominating set of these $\frac{n}{2}$ vertices.*

Lemma 6 *If the vertices in $DS(G)$ covers $\frac{3n}{2}$ vertices of G_T, where $|DS(G)| \leq \frac{n(1+\ln(1+\delta))}{1+\delta}$, and the remaining $\frac{n}{2}$ vertices in $G_T - G$ is covered by a dominating set of size $k : \frac{n}{2}(1-p)^k < 1$, then the lower bound for the probability of an edge connecting vertices in $G_T - G$ is $\frac{\delta}{n}$.*

Proof 6 *We need a dominating set $DS(G)$ of G such that $|DS(G)| \leq \frac{n(1+\ln(1+\delta))}{1+\delta}$, covering $\frac{3n}{2}$ vertices of G_T. Given the expected size of the dominating set of the remaining $\frac{n}{2}$ vertices of G_T is k and $DS(G_T)$ is the dominating set of G_T, $|DS(G_T)|$ must be less than $(|DS(G)| + k)$ for getting the dominating set within the bound stated in Theorem 1. Therefore,*

$$|DS(G_T)| \leq |DS(G)| + k \implies k \geq |DS(G_T)| - |DS(G)| \tag{6}$$

To construct G_T, we extend G to $G_T(V_T, E_T)$ where $V_T = 2n$ and $V \subset V_T, E \subset E_T$. For edges in $G_T - G$, an edge exists between a given pair of vertices w_i and $w_j : w_i, w_j \in V_T \setminus V$ with probability p. It is observed that due to the extension procedure, the expected degree of a vertex $v_i \in G$ increases from d_i to $\frac{3d_i}{2}$ and $DS(G)$ covers $\frac{n}{2}$ new vertices of G_T. Also, the expected degree of a vertex $w_i \in V_T \setminus V$ becomes $\frac{d_i}{2} + p(n-1)$. The size of our desired dominating set depends on the minimum degree of the graph. Finding the dominating set of the graph G is our main goal and to integrate it in the Blockchain structure we extend G to G_T. So we expect all the important factors of our system to be components of G. This creates a necessity of making some vertex of G as the vertex with a minimum degree in G_T also after extension. So it is expected that the following inequality holds true and gives us a choice for the value of p.

$$\frac{\delta}{2} + p(n-1) \geq \frac{3}{2}\delta \implies p \geq \frac{\delta}{n-1} \approx \frac{\delta}{n} (\because n >> \delta) \tag{7}$$

From the above equation, we get the lower bound on p that decides the probability with which vertices in set $V_T \setminus V$ must be interconnected. The miners will try to find a dominating set $DS(G_T)$ of G_T where $|DS(G_T)| \leq \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}}$. Additionally, we need a dominating set $DS(G)$ of G where $|DS(G)| \leq \frac{n(1+ln(1+\delta))}{1+\delta}$. From Lemma 5, we have,

$$\begin{aligned} &\frac{n}{2}(1-p)^k < 1 \implies (1-p)^k < \frac{2}{n} \implies k \ln(1-p) < \ln\frac{2}{n} \implies k \ln\frac{1}{1-p} > \ln\frac{n}{2} \\ &\implies k > \frac{\ln\frac{n}{2}}{\ln\frac{1}{1-p}} = \frac{\ln\frac{n}{2}}{\ln(1+p+p^2+\ldots)} > \frac{\ln\frac{n}{2}}{p+p^2+p^3+\ldots} = \frac{\ln\frac{n}{2}}{p(1+p+p^2+\ldots)} \end{aligned} \tag{8}$$

Now k must be greater than $|DS(G_T)| - |DS(G)|$. So,

$$|DS(G_T)| - |DS(G)| \leq \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}} - \frac{n(1+ln(1+\delta))}{1+\delta} \leq k \tag{9}$$

Given Eq.8, and substituting $k = \frac{\ln\frac{n}{2}}{p(1+p+p^2+\ldots)}$, it is sufficient to show,

$$\frac{\ln\frac{n}{2}}{p(1+p+p^2+\ldots)} \geq \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}} - \frac{n(1+ln(1+\delta))}{1+\delta} \tag{10}$$

The L.H.S of the Eq. 10 decreases with an increment in p. So we need to check upon substituting minimum p derived from Eq.7, whether the above inequality hold or not. Using the inequality $\forall x \geq 1, \frac{1+\ln(1+x)}{1+x} < \frac{1+\ln x}{x}$*, we get*

$$\frac{(1-p)\ln\frac{n}{2}}{p} \geq (\frac{n}{\delta}-1)(\ln\frac{n}{2}) > \frac{n(1+\ln(\delta))}{\delta} > \frac{2n(1+\ln(1+\delta))}{1+\delta} - \frac{n(1+ln(1+\delta))}{1+\delta} > \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}} - \frac{n(1+ln(1+\delta))}{1+\delta} \tag{11}$$

which is true as $\frac{n}{\delta} - 1 \approx \frac{n}{\delta}$ *and* $\delta < \frac{n}{2e}$*. It is fair enough to consider* $\delta < \frac{n}{2e}$*. If the minimum degree is greater than this, then it becomes easier to find dominating sets for graphs and that would not be provided as input instances to the Blockchain for mining.*

4.2 Correctness of Output

Lemma 7 *The dominating set for* $G_T(V_T, E_T)$ *is within the bound stated in Theorem 1.*

Proof 7 *To verify the PoW solutions one peer need to check whether the solution provided is the dominating set of* G_T*. The peers don't need to be sure that the dominating set found by a miner is the minimum dominating set in the graph. She can only choose the smallest one she has received in the epoch and can accept the corresponding block. An honest miner will check whether the dominating set for* G_T *follows the bound mentioned in Theorem 1. Additionally, a rational miner will use her maximum computation power to ensure that the solution is not just satisfying the bound but has a low cardinality. It is highly unlikely that miners collude and submit a bad solution (i.e. exceeding the bound). Even if there exists one miner that provides a better solution than the rest, he gets rewarded.*

Lemma 8 *If the cardinality of dominating set for* $G_T(V_T, E_T)$ *is within the bound stated in Theorem 1 then in expectation, the cardinality dominating set for* $G(V, E)$ *is within this bound as well.*

Proof 8 *When G is extended to* G_T*, then the expected degree of each vertex in G increases by factor* $\frac{1}{2}$*. This implies the minimum degree of* G_T *increases to* $\frac{3\delta}{2}$*. If a miner starts with finding the dominating set for G, then as per Lemma 4,* $\frac{n}{2}$ *vertices of* $G_T \setminus G$ *are already covered in expectation. From Lemma 5 and Lemma 6, we infer that upon setting the probability of edge formation in* $G_T \setminus G$ *to* $\frac{\delta}{n}$*, we get a dominating set covering the remaining* $\frac{n}{2}$ *vertices of* $G_T \setminus G$ *that has the value at least* $\frac{2n(1+\ln(1+1.5\delta)}{1+1.5\delta} - \frac{n(1+\ln(1+\delta)}{1+\delta}$*. From Corollary 7, it follows that dominating set of* G_T *is within the bound stated in Theorem 1. If we consider that the expected bound is* $\frac{2n(1+\ln(1+1.5\delta)}{1+1.5\delta}$*, then in expectation, G had a dominating set of size at most* $\frac{n(1+\ln(1+\delta)}{1+\delta}$*.*

On the other hand, if a miner uses the following strategy: choose the dominating set of $G_T \setminus G$ *first and then the dominating set for G, then the cardinality may exceed* $\frac{2n(1+\ln(1+1.5\delta)}{1+1.5\delta} - \frac{n(1+\ln(1+\delta)}{1+\delta}$*. But in this case,* G_T*'s dominating set still lies within the bound implying the dominating set of G has a cardinality lower than* $\frac{n(1+\ln(1+\delta)}{1+\delta}$*. The above analysis proves that any strategy applied by the miner to find the dominating set for* G_T *does not degrade the solution quality of the dominating set for graph G in expectation.*

4.3 Time Complexity Analysis

Lemma 9 *The time taken for the block generation is* $\mathcal{O}(|E|)$

Proof 9 *The miner needs to extend the graph G to* G_T*. By analyzing* Extend *function, we observe that the output of* $K(h_1, h_2)$ *is* $2|E|$*. The miner needs to read this bit string and connect vertices in G with vertices in* $G_T \setminus G$*. This is* $\mathcal{O}(|E|)$*. In the next step, to insert edges with probability* $\frac{\hat{\delta}}{n}$ *in* $G_T \setminus G$*, the miner keeps tracks of* $\hat{\delta}\frac{n-1}{2}$ *indices. This is* $\mathcal{O}(|V|)$*. Since* $|V| << |E|$ *so the time complexity for constructing* G_T *is* $\mathcal{O}(|E|)$*.*

The miner now finds a dominating set in a graph instance $G_T(V_T, E_T)$ *where* $|V| = n, E \subseteq V \times V$ *using greedy heuristic, defined in Algorithm 5. In each iteration, we pick up the vertex with the maximum degree, check its neighbor, add the vertex to the dominating set and delete it from the graph, and mark its neighbor as visited. Next, we select another unvisited vertex having the highest degree in the residual graph. This continues till no unvisited vertex remains in the graph. The number of iterations in the worst case is the sum of the degree of the vertices in the dominating set. But this just involves exploring all the vertices and thus it is* $\mathcal{O}(|V|)$*. This is again dominated by the time complexity of extending the graph. Thus the time complexity of Algorithm 2 is* $\mathcal{O}(|E|)$*.*

Lemma 10 *The time taken for the block verification is $\mathcal{O}(|V|)$.*

Proof 10 *Upon analyzing Algorithm 3, the verifier just needs to check the neighbors of the vertices in dominating set. It need not fully generate the graph G_T, just the information regarding connections of vertices in dominating set is required. The time complexity is the summation of the degrees of vertices in dominating set, without any double counting of a vertex. Thus, the time complexity is $\mathcal{O}(|V|)$.*

5 Chain Section Rule in *Chrisimos*

It may happen that two or more miners solved their instances at about the same time and published their blocks, creating the situation that is known as a *fork in blockchain systems*. Forks are usually resolved in the synchronization phase using the rule specified for the particular Blockchain. Only one of the blocks will pass both the verification and synchronization phases. We define a rule whereby the maximum work done in a block will win the race. The work done in a block must be proportional to the size of the graph as well as the effort put by the miner in finding a dominating set of lower cardinality. Thus, we define it as the product of the number of edge and the number of vertices in the extended graph, scaled by the ratio of the expected permissible size of dominating set for the extended graph and the size of the dominating set returned as a result. The mathematical expression for work done in block B that has found dominating set $DS(G_T)$ for graph G_T as follows:

$$\text{Work-done}_B = \left(2|E| + \frac{\hat{\delta}(|V|-1)}{2}\right) \times |V| \times \frac{\frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}}}{|DS(G_T)|} \tag{12}$$

where $|E_T| = \left(2|E| + \frac{\hat{\delta}(|V|-1)}{2}\right)$ and expected permissible size $DS(G_T) \leq \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}}$ (deduced from Lemma 6 as degree $d(v)$ of $v \in V$ increases to $\frac{3d(v)}{2}$ in expectation). If there is a fork at block B', then the miner chooses the chain, starting from B', having the highest work done, i.e. if $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ be m such forks from block B', then choose the chain $C_i \in \mathcal{C}$ such that $\sum_{B \in C_i : C_i \in \mathcal{C}} \text{Work-done}_B$ is maximum.

In our protocol, we consider that each graph instance is provided sequentially after a block interval time elapses. So we maintain an order on the graph instances that get included in the future blocks. In block generation and verification, the miner needs to check whether the graph id is strictly increasing. It is not possible for an adversarial miner to insert any graph instance of his or her choice and mine blocks. All graph instances must come from the given public address and the authenticity of the graph is checked by the miners. No stale graph instance with a lower graph id will be accepted. An honest miner considers a block B committed if B is buried at least f blocks deep in its adopted chain. Thus, we summarize the rules for checking whether a chain is valid or not as follows:
(i) Verifier does not accept a block that has mined a graph instance whose graph id G_{id} is either less than or equal to the graph id of the instance mined in the previous block of the main chain.
(ii) *A block is said to be checkpointed if it has received at least f confirmations*: All blocks before the last checkpointed block is also considered checkpointed. Additionally, we assume all honest miners reach a consensus over a single chain having the last checkpointed block. This implies that within the next f blocks, the view of the network will be the same for all the miners till the last checkpointed block but can differ after that.
(iii) *If the miners observe another sub-chain with higher work done*: In this situation majority of the honest miners have a consensus on the view of the main chain till the last checkpointed block (i.e. the block that has f confirmations). Now the miners encounter another sub-chain that has induced a fork. The following cases can happen:

- If any block in the other sub-chain violates a rule (i), discard the chain.
- If the fork starts from a block before the last checkpointed block, then the chain is discarded.
- If the fork starts from or after the last checkpointed block, then there could be two possible cases:
  (a) If the fork occurs from the checkpointed block, then all sub-chains having a length less than f will be discarded. For the rest of the sub-chains with a length of at least f, follow the rule (b) mentioned below.
  (b) If fork occurs after the checkpointed block, then select the sub-chain with the highest work done. If there is a tie, randomly choose one of the chains.

In the next section, we discuss the security of our proposed scheme.

6 Security of *Chrisimos*

When a miner adopts a new highest-work-done chain, either through mining or by receiving from other miners, it broadcasts and mines on top of the highest-work-done chain. Our goal is to prove that the Nakamoto consensus using our proposed PoW mechanism guarantees *safety* and *liveness* Ren (2019). The properties are defined as follows:

- *Safety*: Honest miners do not commit different blocks at the same height.
- *Liveness*: If all honest miners in the system attempt to include a certain input block then, after a few rounds, all miners report the input block as stable.

We show that in the proposed scheme, the estimated time for block addition is sufficient for a graph instance to be solved guaranteeing progress. Since all the graph instance is solved and added sequentially into the blockchain, we argue the safety property in terms of selfish-mining attacks and double-spending attacks. The property of safety and liveness holds in the synchronous model. In an asynchronous setting, it is tricky to argue whether all graph instances announced in the network get added to the blocks of the blockchain. We leave this analysis as a part of the future work. We state certain lemmas that justify the safety and liveness of *Chrisimos*. An informal proof of these lemmas has been discussed in Section 6. We do not quantify the value of f as that would be correlated to the winning chance of an adversary to mine a valid block. The analysis would require a different mathematical model and we keep it as part of future work.

Lemma 11 *For a graph instance G, the block time interval T^G_{max} is sufficient for adding the block to the Blockchain.*

We had shown during the estimation of block interval time that it takes into account the block generation as well as verification time. If the block generation time is τ (if the graph instance is already present in the lookup table), we set T^G_{max} to $l\tau : l > 1$. If the graph instance is not present in the lookup table, the time is estimated by scaling it based on the edge count and vertex count of the graph instance. Since the lookup table is prepared by using a greedy heuristic, a rational miner will definitely get a solution by at least using the greedy heuristic.

Our proof works since we assumed a synchronous model. In an asynchronous setting, it is tricky to argue whether all graph instances announced in the network get added to the blocks of the blockchain. We leave this analysis as a part of the future work.

Lemma 12 *The chance of a double spending attack on any block before the last checkpointed block in the main chain is negligible.*

This follows from point (iii) of chain selection rule where the honest majority has consensus till the last checkpointed block. Any fork after this will lead to the selection of the sub-chain having the highest work done.

Lemma 13 *In the presence of an honest majority, the probability of any malicious miner pulling off a selfish mining attack is negligible.*

We provide a proof sketch for the following lemma based on the chain rule defined in Section 5. The adversary has to start mining the private chain after the last checkpointed block else as per rule (iii), its privately mined chain will anyway get discarded. We do consider the two cases where the miner will start privately mining from the checkpointed block or after that:
(a) If the malicious miner induces a fork from the last checkpointed block B_l, the malicious private subchain must be of length $\geq f$ as per rule (iii)(a) mentioned in Section 5. The chain of the adversary remains the same till block B_l and starts differing from here, so we label these blocks as $B'_{l+i}, 1 \leq i \leq z$ where $z \geq f$. From Lemma 11, any block $B'_{l+t}, 1 \leq t \leq f$ mined by the adversary must have fetched the dominating set on the same graph instance as that in block B_{l+t}. For the subchain having B'_{l+i} blocks to be selected, $\sum_{i=1}^{z}$ Work-done$_{B'_{l+i}} > \sum_{i=1}^{f}$ Work-done$_{B_{l+i}}$. From Lemma 11, z cannot exceed $f + 1$. This is because each graph instance is provided sequentially after elapse of block interval time. If the malicious miner luckily finds a solution quite earlier than the rest of the miners for the $(l + f + 1)^{th}$ instance, it will have a higher chance of winning due to the impact of an additional block B_{l+f+1}. If $z = f$, then the cumulative work done in all the blocks of the sub-chain must be higher. This would require the adversary to be lucky in at least one of the blocks where it had managed to find a better solution and the rest of the blocks in the sub-chain must provide a solution as good as the one provided by an honest miner.
(b) If the adversary starts selfish mining after the checkpointed block then any sub-chain of length less than f would do but it needs to follow the criteria (iii) (b) of the chain selection rule to win the mining game.

In both cases, if the adversary finds that at i^{th} block, $1 \leq i \leq f$, the cumulative work done in his private sub-chain is less than the cumulative work done in the main chain then it is highly likely he will abandon selfish mining and try to

add the new block on the main chain. To continue mining on his sub-chain, he has to calculate the expected probability that he wins the mining game by adding the next block and this is conditioned on the fact that others must return a result far worse than what the adversary does.

Apart from this, we also prove that even if an adversary knows a graph G apriori, it has a negligible advantage over any honest miner in finding a dominating set for extended graph G_T.

Lemma 14 *If a P.P.T adversary knows the graph $G(V, E)$, $n = |V|$ and hash function $\mathcal{H}$ is collusion resistant, then the adversary has a negligible advantage over an honest miner who needs to find the dominating set $DS(G_T)$ for graph G_T without having any prior knowledge of G.*

Proof 11 *If an adversary $\mathcal{A}$ with polynomial time computation power knows the graph G before it is even broadcasted to other miners, $\mathcal{A}$ randomly samples $x \leftarrow \{0,1\}^{2\lambda}$, generate $\mathcal{H}(x)$ and uses the rule mentioned in function K to generate the bit string of length $2|E|$ and precompute G_T. When the graph G is provided as a problem instance, $\mathcal{A}$ forms the block with a set of transactions. The probability that now the output of K using the hash of the previous block, h_1, and Merkle root of the transaction set, h_2, is the same as the output predicted by $\mathcal{A}$ using random string x is negligible in λ. This follows from the fact that $\mathcal{H}$ is a collision-resistant hash function Ishai et al. (2005).*

$$Pr[x \leftarrow \{0,1\}^{2\lambda} : \mathcal{H}(h_1, h_2) = \mathcal{H}(x) \wedge (h_1||h_2) \neq x] \leq negl(\lambda) \tag{13}$$

Thus, with negligible advantage, $\mathcal{A}$ will manage to precompute the dominating set $DS(G_T)$ before any other miner.

Since the probability of success in the above case is negligible, $\mathcal{A}$ will try the other strategy of precomputing the dominating set $DS(G)$ for graph G and using it to compute the dominating set of $DS(G_T)$. The adversary checks the number of vertices in $G_T \setminus G$ that are covered by $DS(G)$. For the rest of the uncovered vertices, $\mathcal{A}$ needs to find the dominating set. We had shown in Lemma 5, $DS(G)$ will cover $\frac{n}{2}$ vertices of $G_T \setminus G$, in expectation. So, in average cases, $\mathcal{A}$ has to find a dominating set for the rest of the $\frac{n}{2}$ vertices. However, this strategy does not guarantee $\mathcal{A}$ will get a better solution than the rest of the honest miners. The adversary may end up selecting more vertices to cover G_T as it first explores G and then $G_T \setminus G$ instead of the entire graph G_T.

7 Experimental Analysis

All the miner needs is the capability to store the graph, do the transformation and then find the dominating set. After a graph instance is part of the chain, a miner can just delete the graph. We do not need any ASIC or high-end machines to run heuristic. If any miner invests more computational power they win the game. This is equivalent to what happens in hash-based PoW as well. The winning probability is proportional to the computation power.

7.1 Setup

For our experiments, we use Python 3.10.0, and NetworkX, version 3.1 - a Python package for analyzing complex networks. System configuration used is Intel Core i5-8250U CPU, Operating System: macOS 12.4, Memory: 8 GB of RAM. We use synthetically generated graph instances (based on Barabási-Albert Model Albert and Barabási (2002) and Erdős-Rényi Model Seshadhri et al. (2012)). Social networks and other utility networks follow the power-law model where few vertices are central to the graph instance. We choose appropriate parameters to generate the synthetic graph instances such that they simulate real-life networks. The order of the graph varies between 1000 to 200000, and the average degree of the graph varied between 10 to 50. For a given number of vertices, we took the average over all the instances with varied edge counts. We run Algorithm 2 to estimate the block proposer time. We use the greedy heuristic in the module `FindDominatingSet` but the miner is free to choose any algorithm. For estimating the time taken to verify the block, we run Algorithm 3.

7.2 Observation

The plot in Fig. 3 shows the impact of increasing the size of the graph instance (increase in number of vertices) on the block proposer time (or generation time) and verification time. For small-order graphs, it is of the order of seconds but for graphs having vertex count more than 75000, the block proposer time goes up to 7 mins. The time taken by the verifier shows a slow and steady increase and it is less than a minute even for a graph of size 200000. Note that in our experiment, we extend the graph using the *Extend* function and that results in a doubling of vertex count. So if we report the result for vertex count 200000, it actually denotes the execution time of finding and verifying dominating set on a graph of size 400000. The block generation time is approximately 5 times that of the block verification time.

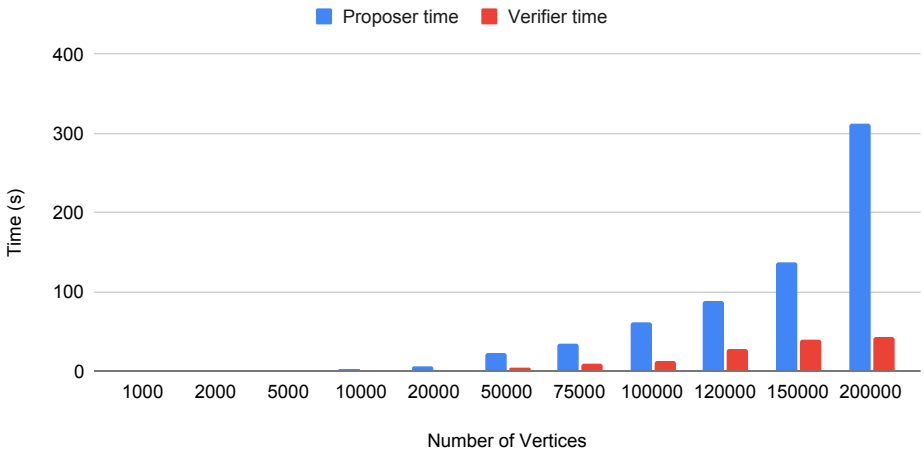

Figure 3: Plot showing impact of the increasing number of vertices on block proposer time and block verification time

7.3 Discussion

We observe that the block generation time increases almost linearly with an increase in the number of vertices in the graph instance. This is because the block generation time is dependent on exploring all the edges in the given graph instance. We consider moderately connected graph instances. Finding dominating set in too sparse or too dense graph becomes easier so no utility company would provide such graph instances. On the contrary, the verification time increases slowly (but linearly) compared to the block generation time with an increase in the size of the input graph. The reason is that the verifier just needs to check whether the vertices in the dominating set cover the entire graph. The time complexity is bounded by the number of vertices in the graph and it is less than the number of edges in the graph. This is fair enough as block verification must be done faster than block generation in any blockchain to allow more miners to join the Bitcoin network without too much spend on verification.

Our results show that if a graph instance of an order as large as 200000 is provided for mining, the block interval time will be around 10 min to 12 min on average. This is similar to that of hash-based PoW. The block interval time might increase if the graph size increases but then the throughput of the Blockchain will reduce and hence a decision has to be made whether graph instance larger than 200000 must be allowed at the cost of reduced throughput.

8 Related Works

The proof of useful work was first introduced in Ball et al. (2017) correlates consensus protocol hardness to solving NP-complete problems of practical interest. A mathematical formalization of the PoUW and its properties has been specified as well. Further constructions for PoUW mining were given by Loe et al. Loe and Quaglia (2018), and Dotan et al. Dotan and Tochner (2020). In all these previous approaches the security of the system was not rigorously analyzed and in many cases, the attacker can manipulate the system by providing easy instances to solve.

A hybrid scheme for PoUW based on the traveling salesman problem (TSP) has been proposed in Loe and Quaglia (2018); Syafruddin et al. (2019) but the utility of the approach has not been discussed. PoUW based on machine-learning problems was suggested by Baldominos and Saez (2019) but these systems could be easily attacked. Another hybrid approach Philippopoulos et al. (2020) to mining combines hash value calculations with difficulty-based incentives for problem-solving. The miners get an incentive for solving real-life problems for scientific purposes. However, the protocol is susceptible to an attack where miners find multiple solutions to a real-life instance, and instead of using the best one, they mine several blocks in a row using all the solutions. Another PoUW protocol named proof-of-search Shibata (2019) is used as a tool to find approximate solutions to any optimization problem. However, the complexity of the overall proof-of-search protocol is too large, and the system is too restrictive in terms of providing a solution. Another PoUW protocol based on the doubly parallel local search (DPLS) algorithm named Ofelimos has been proposed in Fitzi et al. (2022). DPLS represents a general-purpose stochastic local-search algorithm having a component called the exploration algorithm. The latter is used to produce a set of points in a solution space based on the given input parameters preventing miners from picking easy exploration steps and reducing the complexity of mining. However, they do not take into account the difficulty of NP-hard problem instances which might lead to unfairness while block

generation. A recent work called CO Consensus Protocol (COCP) Todorović et al. (2022) proposes efficient utilization of computing resources by providing valid solutions for the real-life instances of any combinatorial optimization problem. An unsolved problem present in the instance pool is selected and miners start solving for the given instance. A miner upon finding the solution sends it to a solution pool. The role of the solution pool is to keep all valid solutions for each considered instance and to enable the identification and rewarding of the miner who found the best solution. This is a major drawback because the authors assume that a centralized entity will check the solution pool and return the best solution among all. This assumption may lead to problems if the entity running the solution retrieval module on this pool is malicious and tends to favor a solution from a particular miner.

Our protocol overcomes the drawbacks observed in the state-of-the-art useful Proof-of-Work. Unlike in Fitzi et al. (2022), we do not consider pool mining in our solution. Introducing a pool of miners will make our system fair as miners will be rewarded for their contribution. As a part of future work, we would explore pool mining for *Chrisimos*.

9 Conclusion

We propose *Chrisimos*, a useful Proof-of-Work that solves a problem having real-life utility instead of wasting resources in generating nonce for hash-based PoW. Miners are asked to find a minimal dominating set on a real-life graph instance. Finding a minimum dominating set of size less than a positive integer is an NP-complete problem so miners use a heuristic of their choice. We estimate the block generation time for each graph instance. Verifiers collect all the solutions that arrive within this time and select the one with the lowest cardinality. Miners earn an additional reward that incentivizes them to put in their best effort. We also mention a new chain rule that resolves disputes in the event of a fork. Through experimental analysis and formal proof, we show that our scheme is secure and resolves problem of wastage of energy faced for hash-based PoW by replacing it with a graph theoretic problem.

As a part of future work, we would propose a mathematical model that would allow us to quantify the computation power of the adversary and figure out its chance of winning the mining game over an honest miner. Based on the mathematical model, we would analyze the various attacks observed on the Bitcoin blockchain. We would also propose a generic model that would allow miners to accept any NP-complete problem and provide a solution to that. Additionally, we want to utilize pool mining to scale the system where each miner in the pool would solve the problem partly and then aggregate the partial solutions in generating the final solution.

Acknowledgment

{Prabal: Few of the references formatting needs to be corrected.}

References

Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of modern physics* 74, 1 (2002), 47.

Noga Alon and Joel H Spencer. 2016. *The probabilistic method*. John Wiley & Sons.

Andreas M Antonopoulos. 2014. *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.".

Razieh Asgarnezhad and Javad Akbari Torkestani. 2011. Connected dominating set problem and its application to wireless sensor networks. In *The First International Conference on Advanced Communications and Computation, INFOCOMP*. 46–51.

Adam Back et al. 2002. Hashcash-a denial of service counter-measure. (2002).

Alejandro Baldominos and Yago Saez. 2019. Coin. AI: A proof-of-useful-work scheme for blockchain-based distributed deep learning. *Entropy* 21, 8 (2019), 723.

Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. 2017. Proofs of useful work. *Cryptology ePrint Archive* (2017).

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

Miroslav Chlebík and Janka Chlebíková. 2004. Approximation Hardness of Dominating Set Problems. In *Algorithms – ESA 2004*, Susanne Albers and Tomasz Radzik (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 192–203.

Louis de Jong. 2023. Exploring Proof of Capacity and Proof of Spacetime: The Exciting Future of Blockchain Consensus Mechanisms. `https://onxrp.com/proof-of-capacity-and-proof-of-spacetime/`.

Maya Dotan and Saar Tochner. 2020. Proofs of Useless Work–Positive and Negative Results for Wasteless Mining Systems. *arXiv preprint arXiv:2007.01046* (2020).

John R Douceur. 2002. The sybil attack. In *Peer-to-Peer Systems: First InternationalWorkshop, IPTPS 2002 Cambridge, MA, USA, March 7–8, 2002 Revised Papers 1*. Springer, 251–260.

Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. 2015. Proofs of space. In *Advances in Cryptology–CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*. Springer, 585–605.

William Farrington. 2022. Confronting consensus: Will bitcoin ever switch to proof-of-stake? `https://currency.com/will-bitcoin-ever-switch-to-proof-of-stake`.

Matthias Fitzi, Aggelos Kiayias, Giorgos Panagiotakos, and Alexander Russell. 2022. Ofelimos: Combinatorial Optimization via Proof-of-Useful-Work: A Provably Secure Blockchain Protocol. In *Advances in Cryptology–CRYPTO 2022: 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15–18, 2022, Proceedings, Part II*. Springer, 339–369.

Fedor V Fomin, Fabrizio Grandoni, and Dieter Kratsch. 2009. A measure & conquer approach for the analysis of exact algorithms. *Journal of the ACM (JACM)* 56, 5 (2009), 1–32.

Michael R Garey and David S Johnson. 1979. Computers and intractability. *A Guide to the* (1979).

Samuel Huestis. 2023. Cryptocurrency's Energy Consumption Problem. `https://rmi.org/cryptocurrencys-energy-consumption-problem/`.

Pavel Khahulin Igor Barinov, Viktor Baranov. 2018. POA Network Whitepaper. `https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper`.

Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. 2005. Sufficient conditions for collision-resistant hashing. In *Theory of Cryptography: Second Theory of Cryptography Conference, TCC 2005, Cambridge, MA, USA, February 10-12, 2005. Proceedings 2*. Springer, 445–456.

Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2020a. Proof-of-burn. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 523–540.

Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2020b. Proof-of-burn. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers 24*. Springer, 523–540.

Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. 2017. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology–CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part I*. Springer, 357–388.

Sunny King. 2013. Primecoin: Cryptocurrency with prime number proof-of-work. *July 7th* 1, 6 (2013).

Sunny King. 2014. What is Gapcoin? `https://gapcoin.org`.

Sunny King and Scott Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August* 19, 1 (2012).

Angelique Faye Loe and Elizabeth A Quaglia. 2018. Conquering generals: an np-hard proof of useful work. In *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*. 54–59.

Tal Moran and Ilan Orlov. 2019. Simple proofs of space-time and rational proofs of storage. In *Advances in Cryptology–CRYPTO 2019: 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2019, Proceedings, Part I 39*. Springer, 381–409.

Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review* (2008), 21260.

Carlos G Oliver, Alessandro Ricottone, and Pericles Philippopoulos. 2017. Proposal for a fully decentralized blockchain and proof-of-work algorithm for solving NP-complete problems. *arXiv preprint arXiv:1708.09419* (2017).

Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gaži, Joël Alwen, and Krzysztof Pietrzak. 2018. Spacemint: A cryptocurrency based on proofs of space. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*. Springer, 480–499.

Pericles Philippopoulos, Alessandro Ricottone, and Carlos G Oliver. 2020. Difficulty Scaling in Proof of Work for Decentralized Problem Solving. *Ledger* 5 (2020).

Moritz Platt, Johannes Sedlmeir, Daniel Platt, Jiahua Xu, Paolo Tasca, Nikhil Vadgama, and Juan Ignacio Ibañez. 2021. The Energy Footprint of Blockchain Consensus Mechanisms Beyond Proof-of-Work. In *2021 IEEE 21st International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. 1135–1144. https://doi.org/10.1109/QRS-C55045.2021.00168

Nader Jafari Rad. 2019. New probabilistic upper bounds on the domination number of a graph. *The Electronic Journal of Combinatorics* (2019), P3–28.

Yongsheng Rao, Saeed Kosari, Zehui Shao, Ruiqi Cai, and Liu Xinyue. 2020. A Study on Domination in Vague Incidence Graph and Its Application in Medical Sciences. *Symmetry* 12, 11 (2020). https://doi.org/10.3390/sym12111885

Ling Ren. 2019. Analysis of nakamoto consensus. *Cryptology ePrint Archive* (2019).

Comandur Seshadhri, Tamara G Kolda, and Ali Pinar. 2012. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E* 85, 5 (2012), 056109.

Naoki Shibata. 2019. Proof-of-search: combining blockchain consensus formation with solving optimization problems. *IEEE Access* 7 (2019), 172994–173006.

Willa Ariela Syafruddin, Sajjad Dadkhah, and Mario Köppen. 2019. Blockchain scheme based on evolutionary proof of work. In *2019 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 771–776.

Paolo Tasca and Claudio J Tessone. 2017. Taxonomy of blockchain technologies. Principles of identification and classification. *arXiv preprint arXiv:1708.04872* (2017).

Milan Todorović, Luka Matijević, Dušan Ramljak, Tatjana Davidović, Dragan Urošević, Tatjana Jakšić Krüger, and Đorđe Jovanović. 2022. Proof-of-Useful-Work: BlockChain Mining by Solving Real-Life Optimization Problems. *Symmetry* 14, 9 (2022), 1831.

John Tromp. 2015. Cuckoo cycle: A memory bound graph-theoretic proof-of-work. In *Financial Cryptography and Data Security: FC 2015 International Workshops, BITCOIN, WAHC, and Wearable, San Juan, Puerto Rico, January 30, 2015, Revised Selected Papers*. Springer, 49–62.

Guangyuan Wang. 2014. *Domination problems in social networks*. Ph. D. Dissertation. University of Southern Queensland.

Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. 2020. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials* 22, 2 (2020), 1432–1465.