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

August 8, 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 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 security of the scheme. 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, Dominating set, NP-complete problem.

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 Blockchains are prone to Sybil attacks. An attacker creates an artificially large number of fake identities Douceur (2002) with the intention of controlling the network. 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. This makes PoW-based mining 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. Alternate consensus protocols overcome this problem by quantifying work done based on the utilization of other resources that have no adverse impact on nature. However, an alternate consensus has its own problems. Proof-of-Stake King and Nadal (2012) and Proof-of-Authority Igor Barinov (2018) eliminate democracy and move the power to the "richest in the room". Proof-of-Space Park et al. (2018) needs more storage space when more miners are added to the network. Proof-of-Spacetime Moran and Orlov (2019) requires users to lock up a certain amount of coins in order to store data on the Blockchain, which can create an

entry-level barrier for new users. Proof-of-Burn Karantias et al. (2020) wastes coins as mining power is proportional to the amount of money a participant is willing to burn.

Previous works on useful PoW. Instead of changing the underlying consensus mechanism, we analyze the literature of Proof-of-Useful-Work (PoUW) Ball et al. (2017). PoUW 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. 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 artificial instance of the traveling salesman problem (TSP) has been proposed in Syafruddin et al. (2019), where the utility of artificial TSP is questionable. Another hybrid approach to mining Philippopoulos et al. (2020) combines hash value calculations with difficulty-based incentives for problem-solving. However, miners can be dishonest, 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 *Ofelimos* Fitzi et al. (2022) is based on the doubly parallel local search (DPLS) algorithm. DPLS represents a general-purpose stochastic local-search algorithm having a component called the exploration algorithm. However, their approach does not take into account the difficulty of NP-hard problem instances which might lead to unfairness while block generation. A recent work called Combinatorial Optimization 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. A miner upon finding the solution to a combinatorial optimization problem sends it to a solution pool controlled by a centralized entity. This is a major drawback because the entity controlling the solution pool could be malicious and tends to favor or sabotage a particular miner.

Choosing dominating set problem. We address the drawbacks of the state-of-the-art by proposing a novel useful PoW. We do not claim to reduce energy consumption but replace the hash function with a graph-theoretic problem of finding a minimal dominating set. Dominating set of a graph has a lot of applications in real life Rao et al. (2020). In Wireless Sensor Networks (WSNs), the operator selects a certain subset of nodes called backbone nodes. Backbone nodes form a minimal dominating set of WSNs that reduces the communication overhead, increases the bandwidth efficiency, and decreases the overall energy consumption Asgarnezhad and Torkestani (2011). Online social network sites like Facebook, LinkedIn, and X are among the most popular sites on the Internet. Due to the financial limitation of the budget, it becomes important to effectively select the nodes in such a network to realize the desired goal. Dominating sets have an important role in social networks to spread ideas and information within a group. It can be used for targeted advertisements, and alleviate social problems Wang (2014).

Contributions. We summarize the contributions as follows:
(i) We propose *Chrisimos*[1], a useful PoW where miners find a minimal dominating set of a real-life graph instance instead of generating nonce for hash.
(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 table to estimate *block interval time* for any new graph instance.
(iii) Any miner returning the smallest dominating set in a given block interval time wins the mining game. 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.
(iv) We define a new chain selection rule where instead of choosing the longest chain Nakamoto (2008), we choose the chain that has the highest work done. We prove that our chain selection rule guarantees the security of the proposed scheme.
(v) 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.

1.1 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 5.1, in Section 5 we discuss the security of

[1] In Greek, Chrisimos means useful

our proposed PoW scheme, experimental analysis in Section 6 shows that *Chrisimos* adds a block to the chain within a bounded time similar to hash-based PoW, and finally we conclude the paper in Section 7.

2 Background and Notations

2.1 Notations used

We denote a graph instance as $G(V, E)$ where V is the set of vertices and $E \subseteq V \times V$ are the edges connecting the vertices in the graph G, and $n = |V|$. For any vertex $v_i \in V$, $N_i = N(v_i)$ denotes set of neighbors of the vertex v_i where $(v_i, w) \in E \iff w \in N(v_i)$. We denote $\delta = min_{v_i \in V}(N(v_i))$ as the minimum degree of G and $\gamma = max_{v_i \in V}(N(v_i))$ as the maximum degree of G. $DS(G)$ denotes the dominating set of the graph G and we define k as the permissible limit of the dominating set as defined in Alon and Spencer (2016). The terms related to Blockchain such as *block* $\mathcal{B}$, *transaction, transaction fee, coinbase transaction*, *Merkle root* h_{MR}, *miner* M, *block reward* and *fork* are defined as follows:

(a) *Block* $\mathcal{B}$: A grouping of transactions, marked with a timestamp, and a fingerprint of the previous block. The block header is hashed to find a proof-of-work, thereby validating the transactions. Valid blocks are added to the main blockchain by network consensus.

(b) *Transaction*: A transaction is a message that informs the Bitcoin network that a transfer of ownership of bitcoins has taken place, allowing the recipient to spend them and preventing the sender from spending them again once the transaction becomes confirmed.

(d) *Transaction Fee*: The amount of Bitcoins (BTC) included in transactions as encouragement for miners to add the transactions to blocks and have them recorded on the blockchain.

(b) *Coinbase transaction*: The first transaction in each Bitcoin block, which distributes the subsidy earned when a miner successfully validates it as well as the cumulative fees for all transactions included in the block. The coinbase transaction effectively creates new bitcoin.

(c) *Merkle root* h_{MR}: The Merkle root is the final node in a Merkle tree. It is a hash which includes all other hashes in the tree. If a single hash is altered within a tree, this change will ripple upwards, changing the Merkle root completely. Merkle roots are used in Bitcoin as efficient commitments to large data sets. The Merkle root of all transaction IDs (txids) in a block is included in the block header, so that if the Merkle root changes, the Proof-of-Work is rendered invalid. This setup ensures that once a block is published, no transaction within the block can be altered

(d) *Miner* M: A miner is someone who tries to create blocks to add to the blockchain (the term also refers to a piece of software that does this). Miners are rewarded for their work by the Bitcoin protocol, which automatically assigns new bitcoins to the miner who creates a valid block. This is how all bitcoins come into existence.

(e) *Block reward*: The reward, including the mining subsidy and transaction fee, that miners receive for successfully mining a Bitcoin block.

(f) *Fork*: A change to the Bitcoin protocol that can come as a codebase fork, blockchain fork, hard fork, or soft fork, with one version of the protocol “forking"" off from another. Forks can also include codebase forks and blockchain forks, though these are not necessarily protocol changes.

A graph G transformed using transaction set $\mathcal{T}$ is denoted as $G_T(V_T, E_T)$ where $|V_T| = 2|V|$, $|V|$ denotes cardinality of set V .

2.2 Hash-based PoW

Bitcoin P2P network uses Adam Back’s Hashcash Back et al. (2002) for PoW. A miner constructs a block that has a transaction and a block header. A difficulty target is specified in the block header that defines the required PoW 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 to be added, and the 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 nonce 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.

2.3 Dominating set problem

A dominating set $DS(G)$ 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 $DS(G)$ is minimal dominating set of the graph G if it does not contain any other dominating set as a proper subset. A dominating set of G of the lowest 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)

Since dominating set problem is an NP-complete problem, we provide proof of the existence of a dominating set within a feasible bound for any graph. This proof ensures that any miner will be able to find the solution within the given bound.

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

Proof. 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 $\frac{n'(1+\ln(1+\delta'))}{1+\delta'}$. □

Choosing a small bound k 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.

3 Our Proposed Solution

3.1 System and adversarial model

There are three types of participants in our protocol: (i) any utility company supplying a graph instance, (ii) a committee selecting the graph instance, and (iii) 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 its profit as remuneration with the miner who solves the dominating set for the given graph instance. Any utility company submits its graph instances to a common public platform and the identity of the utility company is not revealed. The common public platform is managed by a committee and we assume that the majority of the members are honest. A secret key sk_C is shared among the committee members using distributed key generation protocol Gennaro et al. (2007) and the corresponding public key is pk_C. These members select a graph instance, assign an identifier and sign the graph. At least `t-of-n` need to sign the graph instance where $t > \frac{n}{2}$ and the signature scheme used is universally unforgeable Boneh et al. (2006). The identifiers are assigned in increasing order. So the latest graph will have an identifier higher than the previous instances. A representative of the public platform announces the graph instance to the Bitcoin network. Instead of storing the graph instance, miners can download the graph instances from that platform.

We assume that any adversary has bounded computation power. We consider a synchronous communication model, i.e., if a message is sent at time t, it reaches the designated party by $t + \Delta$, where Δ is the upper bound within which a transaction is confirmed in the blockchain and becomes visible to others.

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, the coinbase transaction and reward transaction set by the utility company to create set $\mathcal{T}$.
(ii) *Preprocessing Phase*: The committee members decide on a graph instance G, assigns an id id_G, generates hash of the graph $\mathcal{H}(G)$ and signs it using pk_C. The network receives a graph G, along with the tuple having graph id id_G, hash of the graph $\mathcal{H}(G)$, minimum degree of the graph δ, block interval time T^G_{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 is supplied from the legitimate public platform.

a. *Verification of Input*: A miner M 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 id_G is higher than the graph id of the previous instance solved and added in the last block $\text{prev}_{\mathcal{B}}$.
b. *Input Transformation*: Given the graph G, M transforms it to $G_T(V_T, E_T)$. The transformation is dependent on $\mathcal{T}$ and the hash of $\text{prev}_{\mathcal{B}}$.

(iii) *Mining of Block*: M 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 verifier 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, the verifier updates the last stored block to $\mathcal{B}$ provided the dominating set of G_T is valid. Once T^G_{max} expires, 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 propose a rule for the graph transformation such that it is dependent on the transaction set of the block. Since all miners work on the same instance in a block interval time, if a miner has fetched the solution for G, any other miner can steal this solution and add it to its block during block propagation. Our proposed transformation ensures no miner can manipulate the given graph instance or earn a reward like a free rider. No two transaction sets of a block are the same due to differences in the coinbase transaction. If a miner M extends graph G to G_T using $\mathcal{T}$, another miner M' will have a different transaction set $\mathcal{T}'$. Transforming G would lead to $G'_T \neq G_T$ and M' has to find a valid dominating set on G'_T.

The transformation involves extending the graph instance G by adding another $|V|$ number of vertices. The new vertices are connected to the graph G based on a rule that depends on the Merkle root formed using $\mathcal{T}$ and the hash of the previous block $\text{prev}_{\mathcal{B}}$. The *Extend* function is defined as follows and mentioned in Algorithm 1:

(a) Miner gets hashes $h_{\text{prev\_}\mathcal{B}}$ and h_{MR}, where $h_{\text{prev\_}\mathcal{B}}$ is the hash of the previous block and h_{MR} is Merkle root formed using the transaction set $\mathcal{T}$. We propose a function K that uses $h_{\text{prev\_}\mathcal{B}}$ and h_{MR} and extends the graph G to G_T. The function K is defined as follows: $K : \{0,1\}^\lambda \times \{0,1\}^\lambda \rightarrow \{0,1\}^{|2E|}$. The output has an equal number of 0's and 1's, and we state the steps for generating the output:
  (i) If $2|E| \leq \lambda$, then first λ bits of $\mathcal{H}(h_{\text{prev\_}\mathcal{B}}, h_{MR})$ is the output of $K(h_{\text{prev\_}\mathcal{B}}, h_{MR})$.
  (ii) If $2\lambda \geq 2|E| > \lambda$, the output of $K(h_{\text{prev\_}\mathcal{B}}, h_{MR})$ is defined as follows: first λ bits is $\mathcal{H}(h_{\text{prev\_}\mathcal{B}}, h_{MR})$, denoted as L_1, and rest of the $min(2|E| - \lambda, \lambda)$ bits are the $1's$ complement of L_1.
  (iii) If $2|E| > 2\lambda$, we generate the output of $K(h_{\text{prev\_}\mathcal{B}}, h_{MR})$ 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 $|E|$.

(b) 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 $|N_i| \geq |N_j|, i < j$ 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.

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 $G_T(V_T, E_T)$

(c) 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 $V_T \setminus V$.

(d) The miner connects vertices in V and $V_T \setminus V$, where connections of w_i with neighbours of $v_i, 1 \leq i \leq |V|$ is 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*.*

(e) *Procedure GetList(*$|V|, \hat{\delta}$*)*: To connect the vertices w in $V_T \setminus V$, probability that 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 8. 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 minimum information needed to generate graph G_T. We define 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. The rule ensures that the number of 1's in the bits-sequence must be at least $p\binom{n}{2}$. We discuss the steps by which M can generate the connections between vertices in $G_T \setminus G$ as follows.

- The entire bit sequence for connections is divided into chunks, each of the size $\lfloor \frac{n}{\hat{\delta}} \rfloor$ bits. For selecting the first chunk, the rule is :
  - If $\hat{\delta}$ is odd, set first $\lfloor \frac{n}{\hat{\delta}} \rfloor - 1$ bits as 0s and last bit 1, or $x = (\lfloor \frac{n}{\hat{\delta}} \rfloor - 1)'s0|1$.
  - If $\hat{\delta}$ is even, set first bit as 1 and rest $\lfloor \frac{n}{\hat{\delta}} \rfloor - 1$ bits as 0s , or $x = 1|(\lfloor \frac{n}{\hat{\delta}} \rfloor - 1)'s0$.

  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.
- 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. Thus 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 $|E_T| = 2|E| + p\binom{n}{2} = 2|E| + \frac{\hat{\delta}(n-1)}{2}$. We also illustrate the steps of input transformation via an example.

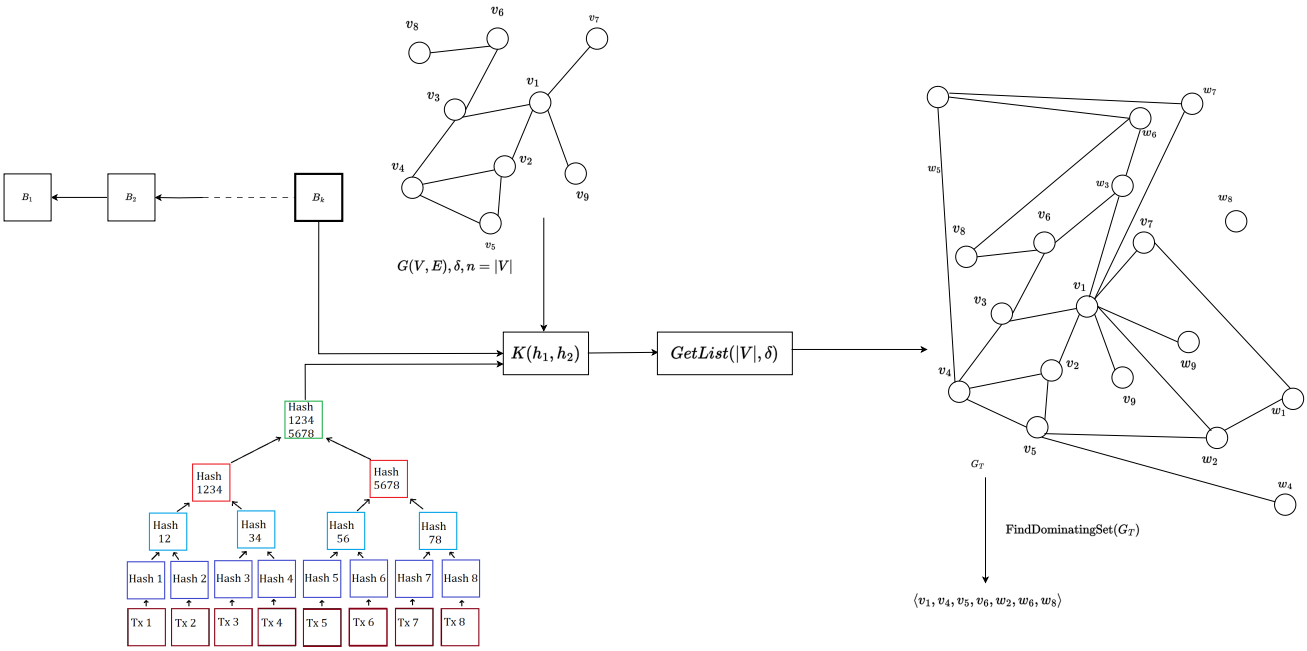

Figure 1: An instance of *Chrisimos* for adding a new block

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 at most the permissible size k within a given block interval time T^G_{max}, pre-specified in the lookup table. The function ComputeBound on G_T returns k, as defined in Theorem 1. The pseudocode is defined in Algorithm 2.

Algorithm 2: Block Generation

Input: $G(V,E), id_G, H, \sigma_H, pk_C, \mathcal{T}, h_{\text{prev\_}\mathcal{B}}$
if $H \neq \mathcal{H}(G)$ *or* $SigVerify(\sigma_H, H, pk_C) \neq 1$ *or* $id_G \leq GetPrevBlockGraphid()$ **then**
  abort
end
Compute $h_{MR} = MerkleRoot(\mathcal{T})$
Compute $G_T \leftarrow Extend(G, h_{\text{prev\_}\mathcal{B}}, h_{MR})$
$k = ComputeBound(G_T)$
$DS' = V$
while $|DS'| > k$ *and time elapsed* $< T^G_{max}$ **do**
  $DS(G_T) \leftarrow$ FindDominatingSet(G_T)
  if $|DS(G_T)| \leq k$ **then**
    $DS' \leftarrow DS(G_T)$
  end
end
if *time elapsed* $< T^G_{max}$ *and* $DS' \leq k$ **then**
  block_header $= HeaderGen(h_{\text{prev\_}\mathcal{B}}, h_{MR}, DS')$
  $\mathcal{B} \leftarrow BlockGen($block_header$, \mathcal{T})$
  return $\mathcal{B}$
end
else
  abort
end

It broadcasts a block $\mathcal{B}$ containing the transaction set $\mathcal{T}$, the hash of the previous block $h_{\text{prev\_}\mathcal{B}}$, Merkle root of the transaction set $\mathcal{T}$ denoted as h_{MR}, address for fetching the graph G along with the id_G, $\mathcal{H}(G)$, signature on $\mathcal{H}(G)$, degree $\hat{\delta}$, and the dominating set of G_T.

Block Verification

The verifier initializes the variable $past\_size_{DS} = 2|V|$. When the miner receives the first block, it checks if the block has a valid dominating set. Cardinality of this dominating set is assigned to $past\_size_{DS}$. Now the verifier will cache this block until it gets a block with dominating set of sizes lower than $past\_size_{DS}$. The verifier will continue to check for new blocks till the T^G_{max} expires. After this, it will accept the last block it had stored. Any solution appearing after T^G_{max} is 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.

When the verifier receives $\mathcal{B}$, it scans for the transaction set $\mathcal{T}$. 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 id_G, hash $\mathcal{H}(G)$, signature σ_H on $\mathcal{H}(G)$, hash of the previous block $h_{\text{prev\_}\mathcal{B}}$, and Merkle root of the transaction set $\mathcal{T}$ denoted as h_{MR}. It downloads G from the public domain using id_G, checks if graph G provided is correct and whether id_G 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_{MR}. The solution $DS(G_T)$ can be written as $V_{ds} \cup W_{ds}$. This is because some vertices of graph G, denoted as V_{ds}, and some vertices in $G_T \setminus G$, denoted as W_{ds} will cover the graph.

The verifier checks whether the vertices in V_{ds} cover the rest of the vertices G. If not all vertices are covered then the verifier sets the variable *uncovered* to true. Next, the verifier calculates the indices that specify 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 $h_{\text{prev\_}\mathcal{B}}$ and h_{MR}. Let the output be S. The verifier can now infer the positions having bit 1 in the bit sequence of size $2|E|$ if he knows the positions having bit 1 in the bit sequence of size λ. We use the method $calc\_index$ on input S for calculating the indices and store it in $Indices_S$. Once the verifier gets the indices, it will know which vertices in $V_T \setminus V$ got covered. We explain the function $calc\_index$ with an example. The verifier has to generate a binary sequence of size λ and then use the rule explained in the example to get a bit string of size $2|E|$ with an equal number of 0's and 1's.

Example 2. *For ease of analysis, we consider* $\lambda = |H(h_{prev\_\mathcal{B}}, h_{MR})| = 10$. *If we have* 2|E| *of size* 200, *and* $\mathcal{H}(h_{prev\_\mathcal{B}}, h_{MR})$ *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* $\mathcal{H}(h_{prev\_\mathcal{B}}, h_{MR})$ *is* 1, *then all the bits at position* $m\lambda + i$ *will be* 1, *where* $m \mod 2 = 0$ *and*

Algorithm 3: Block Verify

Input: $\mathcal{B}, past\_size_{DS}$
Parse $\mathcal{B}$ to get $h_{\text{prev\_}\mathcal{B}}, h_{MR}, id_G, H, \sigma_H, \delta$, dominating set $DS(G_T) = V_{ds} \cup W_{ds}$ and $G \leftarrow get(id_G)$ from block header and transaction set $\mathcal{T}$
Set $visited\_set = \phi$
if $h_{MR} \neq MerkleRoot(\mathcal{T})$ *or* $H \neq \mathcal{H}(G)$ *or* $SigVerify(\sigma_H, H, pk_C) \neq 1$ *or current time* $\geq T^G_{max}$ *or* $id_G \geq GetPrevBlockGraphid()$ *or* $(past\_size_{DS} < |DS(G_T)|)$ **then**
  reject solution
end
else
  $S \leftarrow \mathcal{H}(h_{\text{prev\_}\mathcal{B}}, h_{MR})$
  $Indices_S \leftarrow calc\_index(M)$
  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 $v'' \in Indices_S(N(v))$ **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 and 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 $v'' \in Indices_S(N(v))$ **do**
        Mark v'' as visited if not visited and add v'' to $visited\_set$
      end
    end
  end
  if $|visited\_set| = 2|V|$ **then**
    $past\_size_{DS} = |DS(G_T)|$
  end
  else
    reject solution
  end
end

$m\lambda + i < 2|E|, m \in \mathbb{N}, \lambda = 10, 1 \leq i \leq \lambda$. *So bits at positions* 2, 22, . . . , 182, 4, 24,. . . , 184, 6, 26,. . . , 186, 8, 28,. . . , 188, 9, 29,. . . , 189 *will be 1. So we get a total of* 50 *indices where the bit is* 1. *We had mentioned that the bits from* $\lambda + 1$ *to* 2λ *are the complement of the first* λ *bits. Bit bit positions* 1, 3, 5, 7, 10 *are* 1 *in the output* $\mathcal{H}(h_{prev\_\mathcal{B}}, h_{MR})$. *If a bit* i *is 0 in this string, then the bit at position* $z\lambda + i$ *will be* 1 *where* $z \mod 2 = 1$. *So bits at positions* 11, 31,. . . , 191, 13, 33,. . . , 193, 15, 35,. . . , 195, 17, 37,. . . , 197, 20, 40,. . . , 200 *will be* 1. *In total, we get* 100 *indices where the bit is* 1.

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 for *block verification* is mentioned in Algorithm 3. Miners may collude and send out random solutions without doing any work. However, only a single honest miner per computational task is enough to foil the entire colluding effort. Colluding miners are not likely to cooperate especially if they know that all they need to do is a relatively small amount

of useful work to win the competition and claim the rewards. The rationale behavior will be to start competing, and return the best solution. Thus, the mining game induces competition among the miners leading the system to act like a *decentralized minimal dominating set solver*.

Block Reward: A miner who submits a dominating set of G_T having least cardinality within T^G_{max} wins the mining game and gets the block reward. This incentivizes rational miners to compete for finding the best solution. The block reward comprises fee from the transaction set in the block, 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. We expect a fair utility company to decide on the remuneration directly proportional to the hardness of the problem. In the next section, we discuss how the dominating set for G is retrieved from the dominating set of G_T.

3.3 Retrieval of solution

A miner finds the 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 observe that $DS(G_T)$ 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 one needs to 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 is that w_i is connected to the neighbor of v_i. We provide the pseudocode in Algorithm 4 and justify in Lemma 10 that the mapping results in a good solution for G in expectation.

Algorithm 4: Find $DS(G)$

Input: Block B
Parse block header of B to get $DS(G_T), id_G, h_{\text{prev\_}\mathcal{B}}, h_{MR}, \delta$ and retrieve 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
  $S \leftarrow \mathcal{H}(h_{\text{prev\_}\mathcal{B}}, h_{MR})$
  $Indices_S \leftarrow calc\_index(S)$
  for v *in* $V \setminus visited\_set$ **do**
    for $w \in Indices_S(N(v))$ **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

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*, denoted as T^G_{max}.

Estimating Block Interval Time

The block interval time comprises (a) block generation and (b) block verification. The crux of block generation is solving dominating set for the given graph instance. We describe each method and estimate the time taken.

(a) *Block Generation*: In Algorithm 2, we have introduced the function `FindDominatingSet` over the extended graph G_T without defining it. 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 $\mathcal{O}(e^{n'})$.*

Proof. 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 $e^{n'}$. □

It 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. Thus, the miners are allowed to use an algorithm of their own choice that returns a result in polynomial time. To provide an estimate of the runtime for several synthetically generated datasets, we apply a greedy heuristic Alon and Spencer (2016) in the module `FindDominatingSet` of Algorithm 2.

The greedy heuristic works as follows: select the vertex with maximum degree γ, say v_γ, as the first element of $DS(G)$, and discard all the neighbors of v_γ from V_T. Repeat the process by selecting the next vertex of maximum degree in $V_T \setminus DS(G)$ until $V_T = \phi$. The heuristic returns a dominating set $DS(G) : |S^*_G| \leq |DS(G)| \leq \ln\gamma |S^*_G|$ where S^*_G is the minimum dominating set.

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. 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.

Algorithm 5: Greedy Heuristic for Dominating Set

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

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 $z\left(1 - \frac{\delta'+1}{n'}\right)$. So we can say that at each step of selection, the number of uncovered vertices is reduced by a factor of $\left(1 - \frac{\delta'+1}{n}\right)$.

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,

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

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 $\frac{n'(1+\ln(1+\delta'))}{1+\delta'}$.

□

This bound is loose and the result returned by the greedy heuristic has a cardinality lower than this.

(b) *Block Verification*: The verifier checks if the dominating set returned by (a) is valid using Algorithm 3.

We use certain benchmark graph datasets and record the time taken to solve the minimal dominating set using the steps stated above in (a) and (b). The block interval time is pre-computed and maintained in a look-up table for all these graph instances. 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 $\mathcal{O}(|V|)$.

Lemma 4. *The time taken for the block generation is* $\mathcal{O}(|E|)$

Proof. 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 5. *The time taken for the block verification is* $\mathcal{O}(|V|)$.

Proof. 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 $\mathcal{O}(|V|)$. □

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 describe the procedure to estimate the block interval time for a new graph instance from the lookup table:
(A) We discuss the mining strategy followed by each miner: (i) Miner runs the greedy heuristic for dominating set problem 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?* (ii) 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.

Thus, if the 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 new block to be added.
(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 the maximum vertex count less than $|V''|$ is selected. Let that instance be $G^*(V^*, E^*)$, with minimum degree δ^*, where vertex count of the entry is $|V^*|$ and edge count of the entry is $|E^*|$. Time taken for proposing a new block for the given graph having edge count $|E''|$ will scale up (or scale down) by factor $\frac{(2|E''|+\delta''(|V''|-1))\times|V''|}{(2|E^*|+\delta^*(|V^*|-1))\times|V^*|}$.

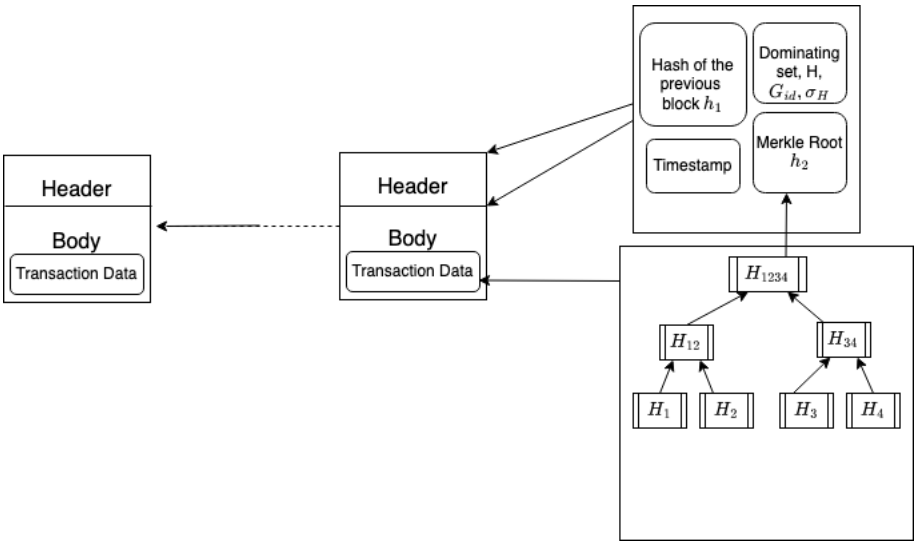

Figure 2: Block Structure when *Chrisimos* is used for generating Blockchain

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.

4 Correctness of Chrisimos

Lemma 6. *The expected number of vertices of G_T covered by the dominating set $DS(G)$ is $\frac{3n}{2}$.*

Proof. 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 $(n + \frac{n}{2}) = \frac{3n}{2}$. □

The expected number of vertices in $V_T \setminus V$ not yet covered by $DS(G)$ is $\frac{n}{2}$. The next theorem gives an upper bound on the size of the dominating set for these $\frac{n}{2}$ vertices.

Lemma 7. *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 b where b is the minimum integer for which $\frac{n}{2}(1-p)^b < 1$.*

Proof. 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 b such vertices in the dominating set the remaining number of vertices to cover will be $\frac{n}{2}(1-p)^b$. When $\frac{n}{2}(1-p)^b < 1$ then there will be no vertex left to cover. From this inequality, we get the minimum b which is also the cardinality of the dominating set of these $\frac{n}{2}$ vertices. □

Lemma 8. *If $DS(G)$ covers $\frac{3n}{2}$ vertices of G_T, where $|DS(G)| \leq \frac{n(1+\ln(1+\delta))}{1+\delta}$, and remaining $\frac{n}{2}$ vertices in $G_T - G$ is covered by a dominating set of size $b : \frac{n}{2}(1-p)^b < 1$, then the probability of an edge connecting vertices in $G_T - G$ must be greater than $\frac{\delta}{n}$.*

Proof. We have $DS(G) : |DS(G)| \leq \frac{n(1+\ln(1+\delta))}{1+\delta}$, that covers $\frac{3n}{2}$ vertices of G_T. Given the expected size of the dominating set of the remaining $\frac{n}{2}$ vertices of G_T is b and $DS(G_T)$ is the dominating set of G_T, $|DS(G_T)|$ must be less than $(|DS(G)| + b)$ for getting the dominating set within the bound stated in Theorem 1. Therefore, $|DS(G_T)| \leq |DS(G)| + b \implies b \geq |DS(G_T)| - |DS(G)|$.

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 into the Blockchain structure we extend G to G_T. We expect all the important factors of our system to be components of G and thus we set p so that the vertex with a minimum degree in G_T comes from set V. The following inequality $\frac{\delta}{2} + p(n-1) \geq \frac{3}{2}\delta \implies p \geq \frac{\delta}{n-1} \approx \frac{\delta}{n} (\because n >> \delta)$ gives the lower bound on p. This p decides the probability with which vertices in set $V_T \setminus V$ must be interconnected. The miners will 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 7, we have,

$$\begin{aligned} \frac{n}{2}(1-p)^b < 1 \implies (1-p)^b < \frac{2}{n} \implies b \ln(1-p) < \ln\frac{2}{n} \implies b \ln\frac{1}{1-p} > \ln\frac{n}{2} \implies \\ b > \frac{\ln\frac{n}{2}}{\ln\frac{1}{1-p}} = \frac{\ln\frac{n}{2}}{\ln(1+p+p^2+\ldots)} > \frac{\ln\frac{n}{2}}{p+p^2+p^3+\ldots} = \frac{\ln\frac{n}{2}}{p(1+p+p^2+\ldots)} \end{aligned} \tag{6}$$

So, $|DS(G_T)| - |DS(G)| \leq \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}} - \frac{n(1+ln(1+\delta))}{1+\delta} \leq b$. Substituting $b = \frac{\ln\frac{n}{2}}{p(1+p+p^2+\ldots)}$ in the above equation, it is sufficient to show,

$$\frac{\ln\frac{n}{2}}{p(1+p+p^2+\ldots)} \geq \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}} - \frac{n(1+ln(1+\delta))}{1+\delta} \tag{7}$$

The L.H.S of the Eq. 7 decreases with an increment in p. We need to check whether the above inequality holds upon substituting $p = \frac{\delta}{n}$.

$\forall x \geq 1, \frac{1+\ln(1+x)}{1+x} < \frac{1+\ln x}{x}$, so we have,

$$\frac{(1-p)\ln\frac{n}{2}}{p} \geq (\frac{n}{\delta}-1)(\ln\frac{n}{2}) > \frac{n(1+\ln(\delta))}{\delta} > \frac{2n(1+\ln(1+\delta))}{1+\delta} - \frac{n(1+ln(1+\delta))}{1+\delta} > \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}} - \frac{n(1+ln(1+\delta))}{1+\delta} \tag{8}$$

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. □

Lemma 9. *The dominating set for G_T is within the bound stated in Theorem 1.*

Proof. 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 10. *Given $|DS(G_T)|$ is within the bound stated in Theorem 1 then in expectation, $|DS(G)|$ follows this bound.*

Proof. When G is extended to G_T, the expected degree of each vertex in G increases by a factor $\frac{1}{2}$. The expected 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 6, $\frac{n}{2}$ vertices of $G_T \setminus G$ are already covered in expectation. From Lemma 7 and Lemma 8, we infer that upon setting the probability of edge formation in $G_T \setminus G$ to $\frac{2\delta}{n}$, we get a dominating set of cardinality at least $\frac{2n(1+\ln(1+1.5\delta)}{1+1.5\delta} - \frac{n(1+\ln(1+\delta)}{1+\delta}$, covering the remaining $\frac{n}{2}$ vertices of $G_T \setminus G$. If the expected upper bound on $|DS(G_T)|$ is $\frac{2n(1+\ln(1+1.5\delta)}{1+1.5\delta}$ as per Theorem 1, then $|DS(G)| \leq \frac{n(1+\ln(1+\delta)}{1+\delta}$ in expectation.

A miner can use the strategy of finding the dominating set of $G_T \setminus G$ and then find the dominating set for G. But there is a risk that 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 using the strategy of finding the dominating set of G_T within the bound ensures 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. □

5 Security analysis

Our goal is to prove that the Nakamoto consensus using our proposed PoW mechanism guarantees *safety* and *liveness* Ren (2019). We define the properties as follows:
(i) *Safety*: Honest miners do not commit different blocks at the same height.
(ii) *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.

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 chain selection rule whereby the chain having maximum work done is selected.

5.1 Chain selection rule in *Chrisimos*

Work done in a chain is summation of the work done in the individual blocks forming the chain. The work done in a block is 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 the work done as the product of the number of edges 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 work done in block B is defined as:

$$\text{Work-done}_B = \left(2|E| + \frac{\hat{\delta}(|V|-1)}{2}\right) \times |V| \times \frac{\frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}}}{|DS(G_T)|} \tag{9}$$

where $|E_T| = \left(2|E| + \frac{\hat{\delta}(|V|-1)}{2}\right)$ and expected permissible size $DS(G_T) \leq \frac{2n(1+\ln(1+\frac{3\delta}{2}))}{1+\frac{3\delta}{2}}$ (minimum degree of G_T increases to $\frac{3\delta}{2}$ 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. An honest miner considers a block B committed if B is buried at least f blocks deep in its adopted chain. We do not quantify the value of f as it must be correlated to the winning chance of an adversary to mine a valid block and we keep the analysis as part of future work.

We summarize the rules for checking the validity of a chain as follows:
(i) Verifier rejects a block that has a graph instance with id either less than or equal to the graph id of the instance mined in the previous block of the main chain. If the graph instance has a malformed signature (not signed by the committtee) or the hash of the graph does not match, verifier rejects the block.
(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 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 the 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.

5.2 Security analysis

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 Nicolas et al. (2019). 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 5.

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

Proof. 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. □

Lemma 12. *The probability of a double spending attack on any block before the last checkpointed block is negligible.*

This follows from rule (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. *The probability of a selfish mining attack is negligible provided the signature scheme used by committee members is universally unforgeable and the majority of miners in the blockchain network are honest.*

Proof. We provide a proof sketch for the following lemma based on the chain rule defined in Section 5.1. 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 5.1. 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.

Another possibility to launch selfish mining is by privately mining a longer chain. It is not possible for a miner to generate several graph instances and mine a longer chain. It follows from rule (i) of *chain selection rule*, where a verifier will reject any illegitimate graph instance not signed by the committee members. Since we assume that the majority of the committee members are honest, and a universally unforgeable signature scheme is used to sign the instance, miners will not be able to forge signatures for all the instances. Hence, the adversary will able to pull off the attack with negligible probability. □

6 Experimental Analysis

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. Social networks and other utility networks follow power-law model where few vertices are central to the graph instance. Thus we use synthetically generated graph instances (based on Barabási-Albert Model Albert and Barabási (2002) and Erdős-Rényi Model Seshadhri et al. (2012)) mimicking this model to generate the benchmark datasets. 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. To estimate the time taken to verify the block, we run Algorithm 3.

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. 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.

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.

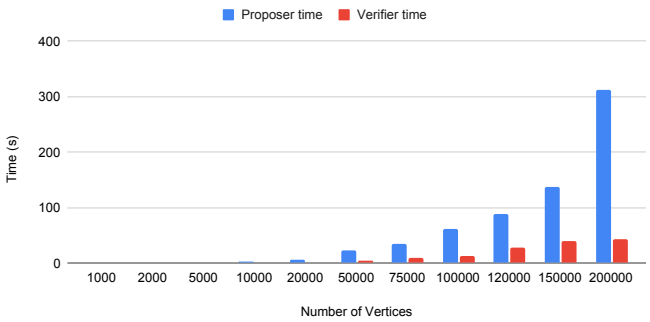

Figure 3: Plot showing impact of increasing number of vertices on block proposer time and block verification time

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.

7 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. Miners return a solution and verifiers collect all the solutions that arrive within a given block interval time. Verifier selects the solution with the lowest cardinality and this simulates a decentralized minimal dominating set solver. We also mention a new chain rule that resolves disputes in the event of a fork and ensures security of our proposed PoUW.

As a part of future work, we will 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 will 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

We thank Department of Computer Science and Information Systems, BITS Pilani, KK Birla Goa Campus, Goa, India for funding our research.

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).

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

Dan Boneh, Emily Shen, and Brent Waters. 2006. Strongly unforgeable signatures based on computational Diffie-Hellman. In *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings 9*. Springer, 229–240.

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.

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.

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).

Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology* 20 (2007), 51–83.

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`.

Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. 2020. 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.

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.

Kervins Nicolas, Yi Wang, and George C Giakos. 2019. Comprehensive overview of selfish mining and double spending attack countermeasures. In *2019 IEEE 40th Sarnoff Symposium*. IEEE, 1–6.

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`

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.

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.

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.