Ofelimos: Combinatorial Optimization via Proof-of-Useful-Work

A Provably Secure Blockchain Protocol

Matthias Fitzi[1], Aggelos Kiayias[2], Giorgos Panagiotakos[3], and Alexander Russell[4⋆]

[1] IOHK matthias.fitzi@iohk.io
[2] University of Edinburgh & IOHK akiayias@inf.ed.ac.uk
[3] IOHK giorgos.panagiotakos@iohk.io
[4] University of Connecticut & IOHK acr@cse.uconn.edu

Abstract. Minimizing the energy cost and carbon footprint of the Bitcoin blockchain and related protocols is one of the most widely identified open questions in the cryptocurrency space. Substituting the proof-of-work (PoW) primitive in Nakamoto's longest-chain protocol with a *proof of useful work* (PoUW) has been long theorized as an ideal solution in many respects but, to this day, the concept still lacks a convincingly secure realization.
In this work we put forth *Ofelimos*, a novel PoUW-based blockchain protocol whose consensus mechanism simultaneously realizes a decentralized optimization-problem solver. Our protocol is built around a novel local search algorithm, which we call Doubly Parallel Local Search (DPLS), that is especially crafted to suit implementation as the PoUW component of our blockchain protocol. We provide a thorough security analysis of our protocol and additionally present metrics that reflect the usefulness of the system. DPLS can be used to implement variants of popular local search algorithms such as WalkSAT that are used for real world combinatorial optimization tasks. In this way, our work paves the way for safely using blockchain systems as generic optimization engines for a variety of hard optimization problems for which a publicly verifiable solution is desired.

Keywords: Blockchain, consensus, proof-of-useful-work, stochastic local search

1 Introduction

Blockchain protocols based on Proof of Work (PoW) capitalize on computational work performed by protocol participants, called miners, to ensure the security of the maintained transaction ledger. In the most prominent Proof-of-Work blockchain designs, the work performed serves no other purpose besides

⋆ This work is based upon work supported by the National Science Foundation under Grant No. 1801487.

maintaining security. These protocols also combine permissionless participation with incentives, offering rewards to miners that commit computational effort to the protocol. This has led to an increasing global commitment of energy to systems like Bitcoin as the value of the currency has grown. At the time of this writing, Bitcoin has an annualized energy expenditure on par with many small to medium countries (see, e.g., the Cambridge Bitcoin Electricity Consumption Index, `https://cbeci.org`).

This trend was identified early on as an important concern in the Bitcoin ecosystem and motivated consideration of two major avenues for potential improvement to the underlying blockchain protocol. The first is aimed at replacing the PoW mechanism with an alternate resource lottery with potentially "greener" characteristics, e.g., proof of stake [14,23,31], proof of space [17,40], proof of space-time [37], and similar mechanisms. A common challenge faced by these approaches is to ensure that the security of the resulting scheme has not been eroded by the change in the underlying primitive (from "work" to something else). The second direction—which, in principle, could entirely ameliorate the issue and is the focus of this work—is to repurpose the invested computational effort towards solving real-world problems. This direction thus posits a proof-of-*useful*-work (PoUW) design approach for blockchain protocols.

Early designs and implementation attempts such as Noocoin [13] and Primecoin [32] highlighted the fundamental issue that would plague future progress towards a satisfactory PoUW system. If the work solved is sufficiently generic, then an attacker may direct the system towards solving problem instances that are easy for them (e.g., due to precomputation or other private advantages "hidden" in the underlying instance-space structure) and hence operate with an advantage in the underlying proof-of-work mechansim, threatening security. At the same time, minimizing the attacker's ability to manipulate the system by adopting more structured "useful" work may render the system's computations useless in practice (e.g., Primecoin [32] and Gapcoin [19] compute sequences of Cunningham primes and gaps between primes respectively—both mathematical objects of dubious usefulness).

1.1 Our contributions

We propose the first PoUW-based blockchain protocol that is accompanied by a thorough security and usefulness analysis. Central to our construction is a novel general-purpose algorithm for *stochastic local search* called Doubly Parallel Local Search (DPLS). Our key technique for protocol design is to mold the whole blockchain protocol execution into a DPLS engine that demonstrably performs the steps of the algorithm in a publicly verifiable manner. The PoUW operation in our consensus protocol has the miners collectively run DPLS on instances contributed by interested clients. In more detail, our results are the following.

(I) Doubly Parallel Local Search (DPLS). We put forth a new algorithm for stochastic local search. With DPLS we achieve the following two-pronged objective: (i) The structural properties of the algorithm suitably reflect the

stochastic dynamics of the underlying permissionless blockchain operation so that a blockchain execution can be viewed as a virtual machine running the algorithm; (ii) Stochastic local search is a powerful, well-studied, and generic algorithmic paradigm for solving computationally hard optimization problems. Thus the DPLS algorithm itself can be evaluated in the context of the broad family of existing stochastic local search algorithm variants and, in particular, assessed with respect to problems of high real-world value.

DPLS is a general-purpose stochastic local search algorithm based on an underlying algorithm M, called the *exploration algorithm*: Given a set of points in the solution space, M searches for a better solution via a local exploration process that requires a modicum of computational effort. (For example, M might call for a fixed number of steps of gradient descent at each input point.) Based on M, DPLS follows a "doubly" parallel search strategy where a number of paths are pursued in parallel, and in each path a number of exploration threads via M are executed; finally, the best one according to a scoring function is selected.

(II) Moderately hard DAG computations. To consider the possibility of using a DPLS solver within a proof-of-work setting, it is essential to articulate the conditions under which running the basic exploration algorithm M exhibits *moderate hardness* (MH). This property is the necessary requirement for a computational problem to be applicable in the blockchain setting. What makes the modeling more challenging compared to, say, the case of Bitcoin's PoW algorithm is that we cannot resort to an idealized model (such as the Random Oracle Model) and must express the moderate hardness property in a way that can be suitably utilized in the security arguments of the blockchain protocol.

To capture this and at the same time reflect the parallelizable nature of DPLS, we focus on the DAG-computation abstraction which has been widely used in the modeling of parallel computations (e.g., see [39,1]). In the setting of DPLS the main computational unit is the exploration algorithm M and we are principally interested in expressing the moderate hardness of DAG computations over M. The delicate part of this modeling is to express the advantage ϵ of the adversary over the honest parties as a function of its ability to "grind" the randomness of the DAG computation as well as capitalize on any advantage obtained from observing previously published steps in the computation.

(III) The PoUW-based blockchain protocol. At a high level, our protocol calls for parties to post instances for problems of interest in the ledger, while locking funds denominated in the ledger's native token to incentivize miners to work towards solving them. Maintaining the blockchain translates to performing steps of the DPLS algorithm for the instances in the ledger and being rewarded for that—with foresight, we stress that solving such instances directly (or posting pre-solved instances) will not help an adversary in extending the ledger any faster. Problem posers can keep funding a particular DPLS computation whenever its funds are getting depleted.

The cornerstone of our protocol is its PoUW mechanism that operates in three stages. In the "pre-hash" stage, random strings are repeatedly generated and hashed until one is found that achieves a small hash (as in a standard PoW).

This string will constitute the random seed for the DPLS exploration algorithm M and will be useful for controlling "grinding attacks" in which an adversary attempts to force adoption of a random string that yields a comparatively easy computation. When the exploration stage terminates, a "post-hash" step determines with a single hash query whether the resulting value qualifies as a PoUW. This "sandwiching" of M between two small hashes is essential for security since it forces an adversarial miner to seed the computation with a randomly selected seed and learn that a successful block can be issued only after the exploration step M is complete. However, if we apply this idea naively (e.g., as a drop-in replacement to Bitcoin's PoW algorithm), there are three major disadvantages: first, a number of useful exploration steps will go to waste, since they won't lead to a block; second, adjusting the hardness of block production (which is needed for blockchain security) would impact usefulness (since miners will spend too much effort trying to find small hashes) and, finally, we do not want waste computational cycles by repeating M-computations to verify newly mined blocks.

We resolve these issues with three mechanisms. First, taking advantage of the scoring function, we have the miners publish the best value they have produced based on all their post-hash attempts; in this way, progress in the DPLS computation is not lost. Note that deviating from this strategy may only impact usefulness—the security of the protocol is maintained against any Byzantine deviation. Second, we adopt the 2-for-1 PoW mechanism of [20], which allows for the production of two types of blocks with a single hash attempt: either an "input block", in which case it is inserted in the blockchain as a transaction, or a "ranking block", which extends the blockchain and refers to any number of input blocks. Using this decoupling mechanism, we can keep the steady progress of the DPLS computation and adjust the underlying (ranking) block hardness independently. The crucial property this ensures is that as more miners join the protocol the DPLS computation is sped up proportionally; on the other hand, ranking block production can be kept steady as required for the security of the underlying blockchain protocol. In this way, the more real-world useful problem instances are submitted to the system (as evidenced by the increased funding locked with each one and the platform's native token appreciation), the more computational power will be introduced to the DPLS engine to solve them. Finally, instead of requiring verification to repeat the computations of M, block producers issue a suitable *succinct non-interactive argument of knowledge* (SNARG) (see, e.g., [24]), so that verifier complexity becomes independent of M.

We prove our protocol secure under a standard "honest majority" assumption reminiscent of the Bitcoin protocol analysis, where the distance above 1/2 depends, among other parameters, on "moderate hardness advantage" ϵ. (We note that even if $\epsilon = 1$, which is to say that we have no moderate hardness guarantees at all, our protocol remains secure with a bound close to 3/4).

As a final remark, our security treatment must additionally contend with a novel probabilistic challenge. In particular, a fundamental assumption adopted by previous proof-of-work analyses is absent: the guarantee that miners' proof-of-work victories are given by independent Poisson (or "discrete Poisson") pro-

cesses. In our setting, the process by which miners produce proofs of work is given by a non-trivial Markov chain reflecting the features described above: e.g, pre-hashing, useful work computation, post-hashing, and SNARG computation. Furthermore, adversarial miners are under no obligation to follow the Markov chain; for example, they may restart the process when they choose. (From this perspective, the classical analysis can be viewed as a chain with a single "mining" state with two transitions, one corresponding to a failed mining attempt—which simply carries the chain back to the same mining state—and one corresponding to a successful mining attempt.) This complex mining model even poses significant challenges for the analysis of honest players because honest players' states in the chain may be synchronized by various events (such as the beginning of the protocol or, depending on the details of the algorithm, delivery of a new block). Unfortunately, such synchronization is a direct threat to the production of desirable "uniquely honest" time periods, during which a unique honest miner generates a proof of work. Such uniquely honest time periods are an emblematic ingredient in the consistency of such systems (see, e.g., [30] where this is explored in detail). To manage these correlations in the model, we consider the aggregate Markov chain carried out by (all) the honest players and establish that when the parameters of the chain are under suitable control—essentially, that the "Poisson" parts of the chain "dominate" the other parts of the chain corresponding to useful work and SNARG computation—the chain converges very rapidly to the ideal distribution where each honest participant is in an independent stationary distribution. We then apply the recurrence-time properties of the stationary distribution along with standard tail bounds for independent random variables to bound the events of interest. This is then leveraged to establish the stochastic properties necessary for consistency. We remark that our techniques here are quite general, and could be applied to quite complex "mining chains," so long as they have a sufficiently substantial "Poisson part" (corresponding to standard proof-of-work discovery). In particular, the techniques can be applied to a generic mining problem—even one with very little variance in time to completion—so long as it is followed by a sufficiently difficult proof of work.

(IV) Usefulness metrics. We devise a two-pronged approach to measuring usefulness. Recall first that our blockchain protocol can be thought as a decentralized DPLS solver. The first usefulness metric asks how good is the blockchain execution as a DPLS engine. This can be done by measuring the ratio per unit of time of the number of steps that the blockchain protocol spends in DPLS computations compared to its total number of steps. We call this metric $\mathsf{U}_{\mathrm{eng}}$, as it can be thought to capture the efficiency of the blockchain protocol as an "engine" that runs DPLS. The second metric, denoted $\mathsf{U}_{\mathrm{alg}}$, reflects how useful DPLS computations are themselves. For a given instance distribution we define this metric as the ratio between the expected number of steps of the best algorithm for that instance distribution divided by the expected number of steps that DPLS takes. Note that identifying the best algorithm for a problem is typically infeasible based on current state of the art, so in this case the best algorithm can be simply substituted with the best *known* algorithm for the problem at hand.

Combining the above two metrics, we can obtain, as an overall metric of usefulness, the product $\mathsf{U}_{\text{eng}} \cdot \mathsf{U}_{\text{alg}}$. The key advantage of our two-pronged approach is that we can completely characterize U_{eng} with protocol analysis, while U_{alg} is an empirical metric that must be assessed in the context of a specific class of problems.

To conclude the discussion on usefulness metrics, we mention that for our protocol it holds that (i) $\mathsf{U}_{\text{eng}} \leq 1/2$, which stems from the fact that we balance the pre-hash probability of success to require the same effort as the worst-case time complexity of M—this enables us to prove security for any advantage ϵ in the underlying MH assumption; and (ii) U_{eng} will be close to $1/2$ if M's runtime distribution is sufficiently concentrated and the rate at which blocks are produced is sufficiently small compared to the SNARG cost. We note that the $1/2$ bound can be surpassed by taking into account the sensitivity of ϵ and adaptively setting the pre-hash difficulty, however such a direction would be only feasible if we restrict the class of exploration algorithms M to those whose hardness is well understood. Estimating U_{alg} requires some real-world baseline – as an illustrative example we choose WalkSAT [43,28], a popular local search algorithm for satisfiability problems. Given this choice, U_{alg} would result from comparing how the DPLS implementation fares with respect to running WalkSAT in isolation. Exploring this direction further goes outside the scope of the present paper but we give some insights in the full version of the paper [18]. It is worth noting that the instance distribution would be an important consideration in this analysis; for illustrative purposes in the full version of the paper, we use *Blocks World Planning*, a well known NP-hard problem in AI [26] for which there is an abundance of public data sets. Using WalkSAT as the baseline, we show that a single-thread implementation of DPLS performs reasonably well against WalkSAT, investing about twice as much computational steps, i.e., something that amounts to an estimation of $\mathsf{U}_{\text{alg}} \approx 1/2$. Similar results are obtained from additional experiments that reflect adversarial deviations and the effect of parallelization.

The above results are evidence for the non-negligible real-world usefulness of our PoUW-based blockchain protocol. We anticipate that investigating further the DPLS blockchain engine as an optimization solver will be an exciting research direction from an algorithmic perspective. There is yet another beneficial dimension of using our blockchain protocol as a DPLS solver: optimization is executed collaboratively in a publicly verifiable manner. Depending on the task, public verifiability has intrinsic usefulness and this can be seen as the price the system pays for the remaining ratio $1 - \mathsf{U}_{\text{eng}} \cdot \mathsf{U}_{\text{alg}}$. For instance, optimization tasks such as athletic-competition tournament scheduling or various matching problems (e.g., the allocation of residents to hospitals or radio frequency auctions) can benefit from public verifiability; see the full version of the paper [18] for further discussion and references.

1.2 Related work

Beyond the early work mentioned above [13,32,19], a number of other works investigated the concept. One line of research considered hybrid constructions

where the miner can choose between applying either standard PoW *or* doing some potentially useful computation [38,11,45]. Further constructions for PoUW mining were given by Loe et al. [34], Dotan et al. [16], and, closer to our work, Baldominos et al. [7] and Lihu et al. [33], who suggested to base PoUW on stochastic search and machine-learning problems. In all these previous approaches the security of the system was not rigorously analyzed and, in many cases, concrete attacks by e.g., an adversary who directly plants easy instances to solve, are feasible.

In contrast to the above, a formal security approach was adopted by [8] but the published version of the work retracted the "usefulness" dimension of the original paper. Also, their proof-of-work construction is not suited for permissionless ledgers as it does not introduce any variance in puzzle-completion time.

Finally, some alternative approaches to the problem at hand that are worth mentioning in our context are the concept of "merged mining", a technique employed in a number of cryptocurrencies where the mining effort for the blockchain has a dual use as mining Bitcoin and hence it is "useful" in this sense: Permacoin [36] where, via proofs of retrievability, the usefulness dimension is in maintaining a public file store; and useful work enforced via a trusted execution environment [44] where, in contrast to the the above solutions, full trust in a specific hardware manufacturer is required.

We stress that, to the best of our knowledge, no prior fully decentralized, PoUW-based blockchain protocol has been published along with a thorough security (or usefulness) analysis.

1.3 Organization of the paper

In Section 2 we describe the computational model and some basic notation. DPLS is presented in Section 3. Next, we expand on our notion of moderately hard DAG computations in Section 4. In Section 5 we present our blockchain protocol, whose security and usefulness we analyze in Section 6. Applications and experimental results, as well as some of the code and proofs are presented in the full version of the paper [18].

2 Preliminaries

Notation. For $k \in \mathbb{N}^+$, $[k]$ denotes the set $\{1, \ldots, k\}$. We denote sequences by $(a_i)_{i \in I}$, where I is a countable index set. For a set X, $x \leftarrow X$ denotes sampling an element from X uniformly at random. For a distribution $\mathcal{U}$ over a set X, $x \leftarrow \mathcal{U}$ denotes sampling an element of X according to $\mathcal{U}$. By $\mathcal{U}_m$ we denote the uniform distribution over $\{0,1\}^m$. We denote that some function f is negligible in λ by $f(\lambda) < negl(\lambda)$. We let λ denote the security parameter.

Security model. We adopt the computational model of [21], which is a variant of the model presented in [20]. There, the set of parties $\{P_1, \ldots, P_n\}$ running the protocol is fixed and the parties, the environment $\mathcal{Z}$, the adversary $\mathcal{A}$, and the

control program $\mathcal{C}$ coordinating the execution are all modeled as IRAMs. The adversary $\mathcal{A}$ is active and can corrupt up to t parties in order to break security.

Communication model. We follow the communication model used by most previous works [41,6] that analyze blockchain protocols in the cryptographic setting, where time is discrete and the network is (partially) synchronous. In more detail, the protocol advances in rounds and communication happens through a diffusion functionality. Honest parties can use it to send messages which may be adaptively delayed for up to Δ rounds by the adversary, but are guaranteed to be received by everyone in the network. Communication is not authenticated, in the sense that the functionality does not provide any guarantees regarding the origin of sent messages. Finally, the adversary is rushing and can additionally choose to send its own messages only to a subset of the parties.

Setup. All parties have access to a *common reference string* (CRS), sampled from a known efficiently samplable distribution, which is used to instantiate a succinct non-interactive argument (SNARG) system [24] SNARG $= (\mathsf{S}, \mathsf{P}, \mathsf{V})$. Note that there are several ways to securely establish a CRS for a SNARG in a permissionless blockchain environment. In particular, assuming the slightly stronger notion of an updatable *structured reference string* (SRS) [25,35], the construction of [29] allows to obtain a common reference string.

Random Oracle. Parties have access to a random-oracle (RO) functionality [9]. We use both RO and non-RO based moderately hard problems and, in order to argue about security, we need to be able to compare their computational costs. We thus assume that a query to RO takes c_H computational steps both for the honest parties and the adversary.

Concrete modeling. $\mathcal{A}$ and $\mathcal{Z}$ have a concrete bound of $t \cdot c_H$ steps they can take per round as well as an upper bound θ on the number of messages they can send per round.

3 Doubly Parallel Local Search

One approach to designing a PoUW blockchain for optimization problems is to: (i) first pick your favorite optimization algorithm, and then (ii) try to design a blockchain protocol around it. The disadvantage of such an approach is that any change in the target optimization problem may result in vital changes to the blockchain and consensus system, requiring new security proofs. Here, instead, we adopt a modular approach where we first build a PoUW blockchain based on a generic optimization algorithm, and later, with minimal overhead, instantiate it with the problem-specific parameters. This allows for re-using our blockchain analysis for different instantiations of the optimization algorithm.

We start by giving an overview of DPLS, the generic optimization algorithm that our blockchain protocol is implementing from a client's point of view, i.e., ignoring the internal details of the blockchain algorithm.

3.1 Overview

Clients of our protocol publish on the blockchain the optimization problems that they want miners to solve. Miners, on the other hand, run the Doubly Parallel Local Search (DPLS) algorithm to solve these problems. Solving large optimization problems may require more work than what can be computed by a miner during the mining of a single block. Thus, we design DPLS to be a *distributed algorithm* where the computation result is obtained by multiple state updates – each corresponding to a block –, some of them possibly occurring concurrently. Concurrent updates is the first source of parallelism of DPLS.

In its core, DPLS follows the well-known *stochastic local search* approach: it searches a solution space X by repeatedly exploring the neighborhood of a currently selected point, looking for a neighboring point that promises progress towards an optimal solution. More concretely, based on the description of a problem instance Λ, DPLS gradually builds a directed acyclic graph (DAG) G recording the already explored points in X. A single exploration step then consists of invoking a *generic* exploration algorithm M on G, yielding a new point in X, with the goal of extending G by a point of better quality (computed by a scoring algorithm g_Λ), thereby progressing the exploration. Note that, in a strictly sequential execution, a 'linear' graph G may be sufficient. However, maintaining a DAG of explored points allows for more general flavors of local search where multiple threads are concurrently explored by different parties.

As communication and local pre-computation are important resources in permissionless systems, we cannot afford to publish every exploration step computed by miners. Instead, each miner performs many *randomized* local exploration steps in a batch, publishing only the best one of them. To this end, the exploration algorithm M is parametrized by an *inner state* z that determines the initial state of the search in a batch, e.g., a common starting location in G to focus the batched search, and a randomness seed r ensuring that same-batch steps are independent. Batched search is the second source of parallelism of our doubly parallel algorithm.

Given the above, DPLS is parametrized by the following *problem-specific* sub-algorithms:

- *Initialization* algorithm $\mathsf{Init}(\Lambda)$: A probabilistic algorithm that takes as input an instance description Λ and outputs a DAG G.
- *Focus* algorithm $\mathsf{F}(\Lambda, G)$: A probabilistic algorithm that takes as input Λ, G and outputs an inner-state string z.
- *Exploration* algorithm $M_\Lambda(G, z, r)$: A deterministic algorithm that takes as input a DAG G, an inner state z, and a seed r, and outputs a point $x \in X$.
- *Scoring* algorithm $g_\Lambda(x)$: A deterministic algorithm that takes as input Λ and $x \in X$, and outputs the score $y \in \mathbb{R}$ of x.
- *Termination* algorithm $\mathsf{Finished}(\Lambda, G)$: A deterministic algorithm that takes as input Λ, G and outputs 1 if the algorithm has finished, and 0 otherwise.

3.2 DPLS modeled in a blockchain setting

Problem solving starts by the problem setter posting an instance description Λ together with the output of $\mathsf{Init}(\Lambda)$ in the blockchain, in the form of a special transaction. Miners work on such an instance by running the UPDATE procedure (Algorithm 1), which makes use of the sub-algorithms introduced above. The outputs produced are posted to the blockchain and are in turn used by other parties to produce additional updates. The search algorithm ends when predicate $\mathsf{Finished}(\Lambda, G)$ becomes true.

UPDATE takes as inputs the chosen instance description Λ and the party's current view of the DAG G. The inner state z is generated using algorithm $\mathsf{F}(\Lambda, G)$, while the number of invocations of M in a single batch, denoted by k, is distributed according to the geometric distribution, with the exact parameters of the distribution set by the protocol designer. The sampling of k from the geometric distribution models its integration into the useful-work mining procedure where each computation of M qualifies for block production with probability p_2—the miner must find a block to publish a state update. After k is fixed, that many seeds $(r_i)_{i\in[k]}$ are sampled independently at random, and algorithm $M(G, z, r_i)$ is invoked k times, with the best-scoring result (according to function g) being output by UPDATE.

Algorithm 1 The state update procedure.

function UPDATE(Λ, G)
- $z \leftarrow \mathsf{F}(\Lambda, G)$ ▷ Compute the inner state
- $k \leftarrow Geom(p_2)$ ▷ Sample from geometric
- $(r_i)_{i\in[k]} \leftarrow \mathcal{U}_m^k$ ▷ Sample uniformly
- $S := \{(z, r_i, x_i) | x_i := M(G, z, r_i), i \in [k]\}$ ▷ Invoke M
- $(z, r, x) := \arg\max_{(z,r,x)\in S} g(x)$ ▷ Pick best
- **return** (z, r, x)

3.3 An example

We present an instantiation of DPLS (Init,F,M,g,$\mathsf{Finished}$) for a variant of the classical WalkSAT algorithm [43,28] for the SAT problem.

First, we give a summary of the original WalkSAT algorithm. Starting from some initial configuration, at each step, WalkSAT picks a variable to flip as follows: Given the current configuration, one of the unsatisfied clauses is chosen at random. For each of the variables involved in the clause, a grade is computed which is equal to the number of clauses that are going to be broken (i.e, turn from satisfied to unsatisfied) if the chosen variable is flipped. If there exist variables that have grade 0, then one of them is selected at random and flipped. Otherwise, a variable is selected (and flipped) at random, with probability wp coming from the selected clause, and with probability $1 - wp$ coming from the variables with

the best grade. The walk continues until a solution is found, or some other condition is met, e.g., an upper bound on the total number of flips is reached. If no solution is found, the algorithm can be restarted from some other point in the solution space.

In the DPLS variant, the instance description Λ encodes the description of the SAT instance, i.e., the number of variables and the different clauses, with the solution space X being equal to the possible configurations of the SAT variables. $\mathsf{Init}(\Lambda)$, to aid parallelization, outputs a number of different initial configurations in X. In each invocation of UPDATE, miners run function F to pick at random which point/configuration in G to work on and encode this information in z. Given this configuration, exploration algorithm $M(G, z, r)$ is set to run WalkSAT for a fixed number of flips. Note, that the starting configuration is the same for the different runs of M in a single UPDATE invocation, allowing miners to focus their search. On the other hand, the randomness used by the different WalkSAT invocations comes from the respective seeds (r), leading to the exploration of different points in the solution space. To choose the best one among these points, g counts the number of satisfied clauses in the respective ending configurations. Hence, UPDATE outputs the configuration that maximizes g, which is then possibly going to be used by another miner as the starting point of another run of UPDATE. The algorithm terminates after a predefined number of updates have been posted. For the detailed code and the experimental evaluation of the performance of this algorithm, we point the reader to the full version of the paper [18].

3.4 Generality of the approach

Most well-known stochastic-local-search (SLS) algorithms [27] can be mapped to DPLS as follows: The Init function provides the initial information needed, e.g., a number of different starting locations for parallel search. Given the current location, M is set to explore a single location in its neighborhood and any randomness needed is provided by the seed. Consequently, UPDATE can be interpreted as exploring different points in the neighborhood, and then returning the one that maximizes the scoring function. This point can then serve as the next point in the search. We expect better performance when the total neighborhood size is sufficiently large, such that miners do not explore the same points due to desynchronization and the fact that the points searched are randomly determined. A subclass of SLS algorithms that has this characteristic is Very Large Scale Neighborhood search algorithms [2], where the algorithm (partially) searches a very large neighborhood before making its next step. We provide more evidence about the generality and possible real world applications of DPLS in the full version of the paper [18].

4 Moderately Hard DAG Computations

In DPLS, most of the work is spent running the exploration algorithm M. Hence, it is natural to base security on the moderate hardness of this computation. We

next describe in detail the syntax and relevant security properties required for its use in a PoUW protocol.

4.1 Syntax

As explained earlier, an important aspect of DPLS is that state updates are performed in a distributed way, and without much coordination. Based on this observation, we adopt a DAG structure for computations involving M, where each computation corresponds to a vertex on the DAG and depends on multiple previous vertices. Our notion generalizes the iterated computation paradigm [10,22], where each computation depends on a single vertex.

We note that the parameters of the computation performed will be possibly influenced by the adversary, in the sense that he may try to post a client problem to be solved, only with the purpose of subverting the underlying blockchain protocol. As the security of the blockchain depends on the hardness of individual computations of M, we must guarantee that they remain moderately hard even when parameters are chosen maliciously.

Taking into account these considerations, new vertices of the DAG are generated based on the current view, an inner-state string, and, an unpredictable seed. As explained earlier, the inner-state string allows parties to focus their work in the context of DPLS. On the other hand, the seed randomizes the computation to force the adversary to do work of average-case complexity—in contrast to possibly selecting "cheap" instances to gain an advantage in block production. Next, we formally introduce the notion of a DAG computation.

Definition 1. *(DAG computation/transcript.) A DAG computation is a sequence of instance descriptions $\mathcal{I} = (\Lambda_\lambda)_\lambda$. For every value of the security parameter $\lambda \in \mathbb{N}$, an instance description Λ specifies:*

1. *a finite, non-empty set Z (inner state);*
2. *a finite, non-empty set X (output);*
3. *a deterministic verification algorithm V; and*
4. *a deterministic exploration algorithm M.*

A transcript *of a DAG computation Λ corresponds to a labeled DAG G where each vertex in G is labeled with a tuple $(z, r, x) \in Z \times \{0,1\}^\lambda \times X$ (edges have no labels). We say that G is valid if and only if $V(G) = 1$.*

Additionally, the following conditions are satisfied:

- *(closure) if G and G' are valid, then $G \cup G'$ is also valid* [5]*;*
- *(correctness) for a valid G and $x \leftarrow M(G, z, r)$, it holds that $G \oplus (z, r, x)$ is valid, where $G \oplus (z, r, x)$ denotes the transcript resulting by adding a vertex with label (z, r, x) to G that is connected to all other vertices.*

We write $\Lambda[Z, X, V, M]$ to indicate that Λ specifies Z, X, V, M as above.

[5] Closure ensures that concurrently extending a transcript does not break validity.

We require that the instance descriptions, as well as the elements of the sets Z, X, can be uniquely encoded as bit strings of length polynomial in λ. For simplicity, we will sometimes denote by V_Λ, M_Λ the algorithms corresponding to instance description Λ.

4.2 Moderate hardness

Next, we introduce a moderate-hardness (MH) notion for DAG computations. Our notion builds on ideas found in [21,22]. On a high level, we require that the time the adversary takes to generate a given number of new vertexes in the DAG, is proportional to their number.

We proceed to describe the security experiment in more detail. Let t be equal to the *worst-case complexity* of M. The adversary has access to three oracles $\mathcal{O}, \mathcal{M}, \mathcal{V}$. Its goal is to compute m new vertices for seeds generated at random from oracle $\mathcal{O}$ in less than $(1-\epsilon)\cdot mt$ steps, where ϵ reflects the advantage of the adversary compared to M. The adversary is allowed to query oracle $\mathcal{O}$ more than m times, and possibly use oracles $\mathcal{M}$ and $\mathcal{V}$ to simulate new honestly computed vertexes and verify whether a DAG computation is valid, respectively. ϵ is parameterized by the respective rates of queries $q_\mathcal{O}/m, q_\mathcal{M}/m, q_\mathcal{V}/m$ to reflect the possible adversarial advantage. We note, that oracles $\mathcal{M}$ and $\mathcal{V}$ are provided to aid composition;[6] We require that the property holds with overwhelming probability for all m greater than some parameter k.

As we want to build a blockchain that can accommodate solving multiple optimization problems, MH is expressed w.r.t. a family of DAG computations (per security parameter level), each corresponding to a different instantiation of the DPLS algorithm.

Definition 2. *Let $\mathcal{I} = ((\Lambda_{\lambda,i})_i)_\lambda$ be a family of DAG computations. $\mathcal{I}$ is* (t, ϵ, k)-Moderately Hard *(MH) if for any PPT RAM $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, $\lambda \in \mathbb{N}$, and all polynomially large $m \geq k$, it holds that the adversary wins with probability $negl(\lambda)$ in $\mathsf{Exp}^{MH}_{\mathcal{A},\mathcal{I},\epsilon,t}(1^\lambda, m)$*

$$\left\{\begin{array}{l} st \leftarrow \mathcal{A}_1(1^\lambda); ((\Lambda_i, G_i, z_i, r_i, x_i))_{i\in[m]} \leftarrow \mathcal{A}_2^{\mathcal{O},\mathcal{V},\mathcal{M}}(st); \\ b_1 := \mathsf{Steps}_{\mathcal{A}_2^{\mathcal{O},\mathcal{M},\mathcal{V}}}(1^\lambda, st) < (1-\epsilon(\frac{q_\mathcal{O}}{m}, \frac{q_\mathcal{V}}{m}, \frac{q_\mathcal{M}}{m}))m\cdot t; \\ b_2 := \bigwedge_{i=1}^{m}((G_i, z_i, r_i) \in Q_\mathcal{O} \wedge V_{\Lambda_i}(G_i \oplus (z_i, r_i, x_i)) = 1); \\ return\ b_1 \wedge b_2 \end{array}\right\}$$

where $q_\mathcal{O}$ queries are made to oracle $\mathcal{O}(\Lambda, G, z) = \left\{r \leftarrow \{0,1\}^\lambda;\ return\ r\right\}$, $q_\mathcal{V}$ queries are made to oracle

$$\mathcal{V}(\Lambda, G, z) = \left\{ if\ V_\Lambda(G) = 0,\ then\ return\ 0,\ else\ \ return\ 1 \quad \right\},$$

[6] In the blockchain setting, the adversary sees blocks generated by other parties, simulated by oracle $\mathcal{M}$, and sends out blocks that other parties may drop or adopt depending on whether they are valid, simulated by oracle $\mathcal{V}$.

and $q_{\mathcal{M}}$ queries are made to oracle

$$\mathcal{M}(\Lambda, G, z) = \left\{ \begin{array}{l} \textit{if } V_\Lambda(G) = 0, \textit{ then return } \perp \\ \textit{else, } r \leftarrow \{0,1\}^\lambda; \textit{ return } (r, M_\Lambda(G, z, r), \mathsf{Steps}_{M_\Lambda}(G, z, r)) \end{array} \right\}.$$

Remark 1. For simplicity, in the MH experiment the adversary ($\mathcal{A}_2$) is given the power to select the DAG transcript it wants to extend (G_i). However we do not want this to be an impediment on its running time in case (a part of) G_i is already defined in the output of $\mathcal{A}_1$; for this reason we will allow $\mathcal{A}_2$ to also determine G_i implicitly by referencing the output of $\mathcal{A}_1$.

Finally, we argue that for any MH DAG computation the speed-up the adversary gets by seeing extra problem instances is bounded. Looking forward, we note that this property will be the cornerstone for the protection of our protocol against grinding attacks. The main idea is that if an attacker $\mathcal{A}$ could get a speed-up on performing DAG computations due to seeing extra problem instances, then we would be able to construct another attacker $\mathcal{A}'$ that could break MH by initially running $\mathcal{A}$ and then performing any remaining "unsolved but queried" DAG computations using M. In the language developed above, extra instances are modeled as extra queries to oracle $\mathcal{O}$, while the adversarial speed-up can be captured by the adversarial advantage difference $\epsilon(1+a, b, c)$ from $\epsilon(1, b, c)$, where a is the percentage of extra queries. We point to the full version of the paper for the formal proof of the lemma.

Lemma 1. *Let $\mathcal{I}$ be a family of DAG computations that is (t, ϵ, k)-MH. Then, $\mathcal{I}$ is also is (t, ϵ', k)-MH, where for any $a \geq 0, b, c$: $\epsilon'(1+a, b, c) := \epsilon(1, b, c) + a$*

Remark 2. It is important to note that (t, ϵ, k)-moderate hardness, for reasonable parameters, is not achievable for all families of DAG computations. To illustrate this, consider a family of DAG computations allowing for an instance to be crafted in the following way: a key pair of a trapdoor permutation is generated by the adversary, the public key is embedded in the instance, and the exploration algorithm M is designed such that it implies computing the pre-image of a random nonce. Clearly, such a DAG computation would not be moderately hard in any reasonable way.

Still, moderate hardness seems to be a reasonable assumption for a large class of computations with sufficiently simple exploration and verification algorithms, e.g., for the core randomized search computation of stochastic local search algorithms [27]. The adversary now can still craft problems trying to gain computational advantage in the DAG computation, but the unpredictability of the randomness seed can help to mitigate this effect to a large extent.

Having given an outline of the DPLS optimization algorithm as well as the necessary vocabulary for moderate hardness, we proceed to present Ofelimos, the PoUW blockchain protocol, which builds on the moderate hardness of a generic useful computation to implement both DPLS and a transaction ledger.

5 The PoUW Blockchain Protocol

5.1 Protocol description

We start, by listing a number of (informal) requirements that any protocol implementing DPLS must satisfy to qualify as a candidate protocol for useful-work mining. We then describe our protocol while motivating the design choices by these requirements. The requirements are motivated from both sides: blockchain security, as well as, efficiency of the DPLS algorithm:

1. Blockchain security:
   (a) No grinding: the adversary cannot gain mining advantage by cherry-picking DPLS exploration steps of low complexity.
   (b) Precomputation resilience: problem instances cannot be adversarily manufactured such that the adversary gains access to faster block production. Computation before seeing the head of the chain to be extended cannot contribute towards computing the respective PoUW.
   (c) Adjustable mining difficulty: The block difficulty can be adjusted to the mining power applied by the network.
2. DPLS efficiency:
   (a) Frequent updates: Results about new points explored are published (relatively) fast.
   (b) Small overhead: The computational overhead of integrating exploration algorithm M into PoUW is small (implying that honest mining performs useful work).

The high-level architecture of the protocol is similar to Bitcoin, i.e., blocks are chained together by referencing each other by hash, and, during each round, a miner selects the longest chain from his view, and tries to extend it by a block. Two modifications are applied: standard PoW is replaced by PoUW, and we apply 2-for-1 PoW [20] in order to accommodate different types of blocks for reasons explained below. See Figure 1 for further reference.

The core of the mining algorithm consists of applying the exploration algorithm M, constituting the 'useful part' of the PoUW. To defend against precomputation (Requirement 1a), the computation of M is prepended by hashing the candidate block (first $\mathcal{H}$ box in Figure 1), thereby randomizing the computation to be performed by M. Similarly to Nakamoto consensus, this 'pre-hash' of the block must lie below an initial target T_1, to antagonize grinding for parameters of M that result in lower-than-average computation complexity: resampling new parameters must be more expensive than the worst-case complexity of M.

By Requirement 1c, the mining-success probability must be reduced below the success probability of hashing against T_1—which is fully determined by the computational characteristics of the problem instance and unrelated to mining participation in the network. One possibility to address this issue would be to further lower the target T_1 to make pre-hashing as hard as required for ledger security; however, this would come against a big loss in usefulness, as miners would spend most of their time performing hashing. Instead, we have the miner

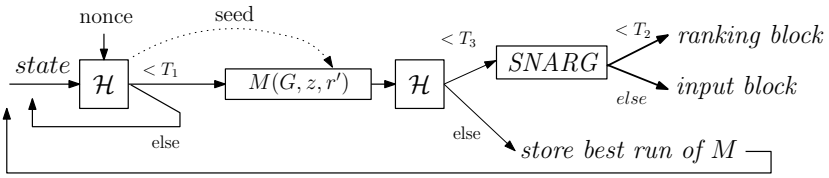


Fig. 1: Diagram of the PoUW mining procedure.

feed the output of M into one *single* round of 'post-hashing' (see second $\mathcal{H}$ box in the figure) that decides, against a threshold T_3, whether the block is eligible for publication. This second threshold adjusts the overall mining difficulty to a level required by the security analysis to guarantee good and secure blockchain characteristics. Note the additional effect of post-hashing to adapt mining difficulty: the miner only learns whether a PoUW attempt is successful *after* executing M, i.e., the computation cannot be cut short to speed up block creation.

A miner loops, many times, the computation sequence of pre-hashing (against T_1), useful work, and one post-hash, until the post-hash of a sequence lies below T_3, allowing for the block to be published. To preserve progress, the best point (by means of scoring algorithm g) from all recent computation sequences is stored for eventual inclusion in a future block to be published. Note that finding a *good* new point is decoupled from mining success, thus helping to establish Requirement 1b. Furthermore, only publishing the best one from a batch of new points, rather than greedily publishing all of them incrementally, helps to accommodate Requirement 2b.

Considering Requirement 2a under Bitcoin parameters, we cannot afford that a miner waits with his update until he mines a block. For this reason, we incorporate 2-for-1 PoW to allow for the publication of different types of blocks, so-called *ranking blocks* which are 'standard' Bitcoin blocks of high difficulty (target T_2), and so-called *input blocks* of low difficulty (target T_3, i.e., hash range $T_2 < h \leq T_3$) which are not part of the chain but are rather handled like transactions to be eventually referenced by a ranking block. A miner now includes his best point explored whenever he hits either type of a block; and by setting the input-block difficulty low enough, the update rate per miner is high enough to distribute progress in the explored points fast, while having no considerable impact on the blockchain characteristics.

A block contains two points explored using M: the 'winner' one that lead to the small post-hash, and the 'best' one that is included to progress the DPLS algorithm. In order to accommodate 2b, we minimize the cost of block verification by having the miner append a SNARG proving correctness of both exploration points contributing to the block, i.e., a SNARG proving membership to the following language: $L = \{((\Lambda, G, z, r', x'), (\Lambda_b, G_b, z_b, r'_b, x'))|V_\Lambda(G \oplus (z, r', x')) = 1 \wedge V_{\Lambda_b}(G_b \oplus (z_b, r'_b, x'_b)) = 1\}$, where $G \oplus (z, r', x')$ denotes the graph G extended with vertex (z, r', x') as defined in Section 4.

A detailed description of the PoUW procedure is given in Algorithm 2. The mining algorithm is parametrized by the longest blockchain received $\mathcal{C}$, the mes-

sage to be included in the block m, the problem instance Λ selected by the miner to work on,[7] the related transcript G extracted from $\mathcal{C}$, and the selected inner state z. The pre-hash input includes these parameters, the hash of the previous block s, and a random nonce r; and yields a unique seed r' for M. At this point, all parameters of M, Λ, G, z, and r', are fully determined based on the data initially hashed, thus establishing that each small pre-hash found by the adversary can only be used to perform one matching post-hash attempt. We note that if, in a round, a miner does not have enough steps to finish running the PoUW procedure, e.g., he only manages to find a small pre-hash, he continues the next round from the point it stopped.

Algorithm 2 The PoUW procedure is parameterized by hardness parameters $T_1, T_2, T_3 \in \mathbb{N}$, the SNARG system, hash function $H(\cdot)$, the explore algorithm M, and scoring algorithm g.

```
1: var (score_b, Λ_b, st_b, com) := (∞, ⊥, ⊥, ⊥)        ▷ Best attempt - global variables
2: var z := ⊥                                              ▷ Inner state
3:
4: function PoUW(C, m, Λ, G)
5:     s := H(head(C))
6:     if (Λ ≠ Λ_b) then z ← F(Λ, G)                       ▷ Reset inner state
7:     r ← U_λ                                             ▷ Sample nonce
8:     h := H(s, m, com, Λ, G, z, r)
9:     if (h < T_1) then                                   ▷ Pre-hash
10:        r' := H(h)                                      ▷ Seed
11:        x' := M_Λ(G, z, r')                             ▷ DAG computation
12:        h' := H(r', x')
13:        if (h' < T_3) then                              ▷ Post-hash
14:            st := (s, m, com, Λ, G, z, r, x')
15:            π := SNARG.P(Σ, (st, st_b));                ▷ Correctness proof
16:            B := ⟨st, st_b, π⟩
17:            if (h' < T_2) then C = CB                   ▷ New ranking block
18:            else Diffuse((input, B))                    ▷ New input block
19:            (score_b, Λ_b, st_b, com) := (∞, ⊥, ⊥, ⊥)   ▷ Reset best attempt
20:        else
21:            if ((Λ ≠ Λ_b) or (g_Λ(x') > score_b)) then  ▷ New best found
22:                (score_b, Λ_b, st_b, com) := (g_Λ(x'), Λ, st, H(st_b))
23:    return C
```

Moreover, ranking blocks are also treated as input blocks, and can be included in the payload of other ranking blocks. As in [20], an input block can be included in the payload of different ranking blocks in diverging chains, which ensures that

[7] Even if there are no problem instances posted by clients on the blockchain, e.g., during bootstrapping, miners can always generate a MH problem based on the hash of the block they are extending (a fixed-time hash-based PoW ([4,12]) is sufficient for this purpose). This amounts to a 'fall-back' DPLS computation.

all input blocks mined by an honest party will eventually be included in the main chain, and no progress is ever lost. The full pseudo-code of the protocol is presented in the full version of the paper [18].

Remark 3. (SNARG overhead) Note that usefulness is not necessarily substantially impacted by a large SNARG-computation overhead as each state update involves a large number of exploration steps (on average) but SNARGs for only two of the M-computations performed. This average number of exploration steps can thus be raised in a trade-off against the state-update frequency in the system, helping to establish Requirement 2b.

5.2 Deployment considerations

The following two practical aspects are of special importance when deploying our PoUW blockchain:

Multiple problem instances. The system must be able to handle multiple problem instances as the computation of a particular instance will eventually terminate. Also, multiple instances should be able to be computed concurrently to give them fair chances to progress. We achieve this concurrency by running the protocol in epochs, and interleaving different problem instances by assigning exactly one instance to each epoch. As exposed in the full version of the paper, interleaving has an additional advantage: during the epoch, unconfirmed input blocks can be immediately extended in the DAG without risking that the referrer block becomes invalid due to possible non-inclusion of the referenced block in the main chain—thus facilitating fast progress during the time slots allocated to the problem.

Incentive structure. The participation of miners in the system must be incentivized to guarantee blockchain security and progress in the useful computation. Also, since the miner is free in choosing which one of their state updates to publish in their block, choosing a good solution should be rewarded in order to expedite progress in the useful computation. In the full version, we elaborate on meeting these conditions as well as on guaranteeing reward fairness along the lines of the Fruitchains construction [42].

6 Security Analysis

Next, we formally analyze the security of our protocol. First, we show that—assuming that the underlying DAG computation is moderately hard and that honest parties control the majority of the computational power in the network—our protocol implements a robust transaction ledger. Then, we define and analyze its usefulness rate.

6.1 Ledger security

Let Π denote our blockchain protocol. The consistency analysis of the longest chain rule appearing in Π involves a number of new challenges, including: (i)

an exotic Markov chain governing the mining dynamics and the possibility of "restarts" in this chain generated by the delivery of a new block, perhaps by the adversary, and (ii) basing the hardness of generating new blocks to a problem satisfying the weak moderate hardness notion introduced in Section 4. We adapt the language of [31,5] to this setting and then develop the tools necessary for the associated probabilistic analysis. (Our treatment below does not require familiarity with these previous papers.)

For simplicity, in the main body of the paper, we discuss the case *without restarts*, which is to say that the protocol carried out by the honest parties does not restart the mining process when it learns of a longer chain, but rather completes the current computation. Intuitively, restarts *improve* the security properties of the blockchain, as they help ensure that honest parties are mining on current chains. However, the situation is somewhat complicated by the fact that restarts do permit the adversary to correlate the states of the honest parties in the Markov chain. Specifically, note that an adversary holding a chain that exceeds the length of those chains currently held by honest parties may strategically release the chain to honest players—perhaps with detailed knowledge about their current state—so as to achieve some short-term control over the distribution of honest mining successes. Despite such correlations, we show in the full version of the paper that the intuition above is correct: the adversarial advantage achieved by exposing adversarial blocks to honest miners is overshadowed by the fact that such exposures increase the length of the blockchain held by the honest recipient; in the language of the analysis below, such an exposure has an effect just as beneficial as an honest mining victory!

We adopt a discrete time model, dividing time into short "rounds" with duration c_H equal to the time taken to carry out a hash query. We reflect the essential block-generation events of an execution of the protocol with a *characteristic* string: this determines, for each round, the number of adversarial and honest ranking blocks generated. Thus our characteristic strings have the structure $w = w_1, \ldots, w_L$ where each $w_i = (h_i, a_i) \in \mathbb{N}^2$ and h_i and a_i denotes the number of honest and adversarial ranking block discoveries, respectively; here L is the lifetime of the protocol.

Ultimately, our protocol Π determines a blockchain of ranking blocks, which themselves refer to input blocks. Such a structure determines a linear order on the collection of input blocks referenced in the blockchain of ranking blocks (by ordering input blocks referenced in a particular ranking block according to the order of their references in the ranking block). Ultimately, we wish to establish the two fundamental ledger properties: *liveness* and *persistence*.

Persistence with parameter $k \in \mathbb{N}$. Once a node of the system proclaims a certain input block in the *stable* part of its ledger $\mathcal{L}$, the remaining nodes either report the input block in the same position of their ledgers, or report a stable ledger which is a prefix of $\mathcal{L}$. Here the notion of stability is a predicate that is parametrized by a security parameter k; specifically, an input block is declared *stable* if and only if it is in a (ranking) block that is more than k (ranking) blocks deep in the ledger.

Liveness with parameter $u \in \mathbb{N}$**.** If all honest nodes in the system attempt to include a certain input block then, after the passing of time corresponding to u rounds, all nodes report the input block as stable.

We establish these properties as consequences of three more elementary properties of the blockchain of ranking blocks, originally formulated in [20] (we use a slightly adapted formulation from [15]):

- **Common Prefix (CP); with parameter** $k \in \mathbb{N}$**.** The chains $\mathcal{C}_1, \mathcal{C}_2$ adopted by two honest parties at the onset of rounds $r_1 \leq r_2$ are such that $\mathcal{C}_1^{\lceil k} \prec \mathcal{C}_2$, where $\mathcal{C}_1^{\lceil k}$ denotes the chain obtained by removing the last k blocks from $\mathcal{C}_1$, and $\prec$ denotes the prefix relation.
- **Existential Chain Quality (ECQ); with parameter** $s \in \mathbb{N}$**.** Consider the chain $\mathcal{C}$ adopted by an honest party at the onset of a round and any portion of $\mathcal{C}$ spanning s prior rounds; then at least one honestly-generated block appears in this portion.
- **Chain Growth (CG); with parameters** $\tau \in (0, 1]$ **and** $s \in \mathbb{N}$**.** Consider the chain $\mathcal{C}$ possessed by an honest party at the onset of a round and any portion of $\mathcal{C}$ spanning s contiguous prior rounds; then the number of blocks appearing in this portion of the chain is at least τs. We call τ the *speed coefficient*.

One of the important conclusions of previous work is that these properties (CP, CG, and ECQ) directly imply liveness and persistence and—from an analytic perspective—can be guaranteed merely based on the characteristic string associated with a particular execution. This fact is fairly immediate for CG and ECQ, whereas identification of the properties of the characteristic string that guarantee CP is more delicate.

In the full version of the paper we give a summary of this theory and describe an extension with restarts. Fortunately, it is possible to succinctly reflect the conclusions of this theory as they relate to our needs, which is done below.

To continue, we first introduce two assumptions related to the level of moderate hardness of the underlying DAG-computation family $\mathcal{I}$ used by Π, and the complexity of the SNARG system used.

Assumption 1. *For parameters $\hat{t}, \hat{\epsilon}, \hat{k}$, we assume that the DAG computation family $\mathcal{I}$ used in Π is $(\hat{t}, \hat{\epsilon}, \hat{k})$-moderately hard.*

Assumption 2. *For parameters $c_{\mathsf{P}}, c_{\mathsf{V}}, c_{\mathsf{S}}$, we assume that there exists a SNARG system* SNARG *where running the prover (resp., verifier, setup) takes c_{P} (resp. $c_{\mathsf{V}}, c_{\mathsf{S}}$) steps.*

Let $w = w_1, \ldots, w_L$ be a characteristic string, as above. We fix a constant Γ, a time period with the following *Γ-serializing guarantee*: *if a ranking block B_2 is generated by an honest party P at least Γ rounds after the honestly-generated ranking block B_1 is diffused, then the full computation supporting B_2 (including the prehash) was carried out while P was aware of B_1.* In our setting, Γ can be set to $2 + \Delta + c_{\mathsf{P}}/c_H + \hat{t}/c_H$ (corresponding to the number of rounds taken to

produce the prehash (≤ 1), useful work ($\leq \hat{t}/c_H$), post-hash (≤ 1), and SNARG (c_{P}/c_H) for block B_2 in addition to any network delay). With this in mind, we say that t is a *Γ-isolated uniquely successful round* if the region $w_{t-\Gamma} \dots w_t \dots w_{t+\Gamma}$ satisfies $h_t = 1$ and, furthermore, that the sum $\sum h_i = 1$ over this region (recall $w_i = (a_i, h_i)$). Note that a round cannot be isolated if it is not followed by at least Γ symbols. For each t define I_t to be an indicator variable for the event that t is an isolated uniquely successful round.

The basic quantities of interest are given by two conventions for accounting for the balance of adversarial and honest successes.

Definition 3 (The barrier walk; the free walk.). *Let $x = x_1, \dots, x_n \in \mathbb{N}^*$. Define the* barrier walk $B(x)$ *by the recursive rule $B(\epsilon) = 0$ (for the empty string ϵ) and, for any $x \in \mathbb{N}^*$ and $a \in \mathbb{N}$, $B(xa) = \max(B(x) + a, 0)$. Likewise, define the* free walk $F(x) = \sum_i x_i$.

Definition 4. *For a characteristic string $w \in (\mathbb{N}^2)^L$ and $0 < t \leq L$, define the* margin effect $w_t^* = a_t - I_t \in \mathbb{N}$ *(and w^* to be the sequence of elements of $\mathbb{N}$ given by this rule). We then define $B^*(w) = B(w^*)$ and $F(w) = F(w^*)$. Finally, for a characteristic string $w = xy$ with $|x| = \ell$, we define the* ℓ-isolated margin *of w to be $\beta_\ell(w) = B(x^*) + F(y^*)$.*

The role of ℓ-isolated margin is clarified by the following, which establishes a direct connection to common prefix.

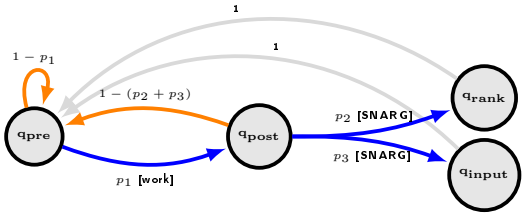
Theorem 1. *Let $w \in (\mathbb{N}^2)^L$ be the characteristic string associated with an execution satisfying the Γ-serializing guarantee. Suppose, further, that (i.) the execution satisfies $(k/s, s)$-CG, and (ii.) for any prefix xy of w for which $|y| \geq s$, we have $\beta_{|x|}(xy) < 0$. Then the execution satisfies k-CP.*

This is the major component in the following theorem; as noted, the details of this existing theory are discussed in the full version of the paper.

Theorem 2. *Let D_Π be a distribution on characteristic strings of length L (induced by a protocol Π), λ a security parameter, and $\alpha > \beta$ two constants corresponding to the rate of uniquely isolated blocks and the rate of adversarial blocks, respectively. Assume that for a constant $\delta < (\alpha - \beta)/2$, when w is drawn from D_Π, every interval of w of length poly(λ) has at least $\alpha - \delta$ uniquely isolated blocks and no more than $\beta + \delta$ adversarial blocks except with negligible probability. Then, except with negligible probability, the protocol satisfies (i.) CG with $s =$ poly(λ) and constant speed coefficient, (ii.) ECQ with $s =$ poly(λ), and (iii.) CP with parameter $k =$ poly(λ).*

Analysis of the Markov chain. In light of the description above, we are specifically interested in analyzing the sequence of (i.) adversarial mining successes and (ii.) uniquely isolated honest successes. The analysis is simplified by the fact that the time evolution of the honest parties is independent. We focus on the Markov chain pictured below, showing nodes for "pre-hash", "post-hash",

and both "ranking" and "input" block production. It is convenient for us to further decorate our transitions with delays: orange edges are traversed in a single round (or c_H time, corresponding to hash queries), the gray edges are traversed instantaneously, and the blue edges have transition times given by the distribution of useful work (upper bounded by $\hat{t}$) and SNARG times (c_{P}). (Note that the timing delays indicated in this chain could be implemented with paths of individual states connected by edges with unit delay, so this presentation can be reflected with a standard Markov chain.) While the basic security properties of the protocol depend only on the production of ranking blocks, the dynamics of the Markov chain depends on both ranking and input block production.



We begin by establishing that—despite the fact that honest parties begin the protocol synchronized (in "pre")—they quickly converge to mutually independent positions in the mining chain. Looking ahead, this mixing argument will be instrumental to establish bounds on uniquely isolated block production.

The mixing time; convergence to mutual independence. By a standard coupling argument we get the following:

Lemma 2. *Consider m particles $P_1, \ldots, P_m$ independently evolving on the Markov chain with any fixed initial states. Let $(S_1, \ldots, S_m)$ denote a random variable so that each coordinate is independent and stationary on the chain. Then letting $T = L(1 + (\hat{t} + c_{\mathsf{P}})/c_H)$,*

$$\|(P_1^T, \ldots, P_m^T) - (S_1, \ldots, S_m)\|_{t.v} \leq m(1 - p_{couple})^L \,,$$

where $\|X - Y\|_{t.v}$ denotes the distance in total variation between the random variables X and Y. Here $p_{couple} > 0$ is a constant that depends only on $c_{\mathsf{P}}/\hat{t}$.

Proof. We proceed with a standard coupling argument. Consider m particles (parties) $P_1, \ldots, P_m$, initially in the state q_{pre}, that carry out simultaneous, independent evolution according to the dynamics of the chain. We wish to show that the joint distribution of positions of all the particles quickly converges to m independent copies of the stationary distribution. For this purpose, consider m additional particles $R_1, \ldots, R_m$ on the chain, initially distributed independently according to the stationary distribution. We let P_i^t and R_i^t denote the positions of the particles at time t. We give a simple coupling C of the evolution of $P_1^t, \ldots, P_m^t$ with $R_1^t, \ldots, R_m^t$, and apply the standard "coupling lemma" which establishes convergence to the stationary distribution. The coupling C is described, at each time step, by a family of random variables U_i^t; for each

$i \in \{1, \ldots, m\}$, $U_i^t : Q \to Q$ is a function where Q is the set of states of the chain (which is in fact larger than the diagram indicates as a result of implementing the "long" transitions). The "update functions" U_i are chosen so that the full ensemble of entries $(U_i^t(q))$ (over all t, i, and $q \in Q$) are independent and each $U_i(q)$ is distributed according to the defining distribution for the state q. Then P_i and R_i are updated according to the same update: $P_i^{t+1} = U_i(P_i^t)$ and $R_i^{t+1} = U_i(R_i^t)$. Observe that the dynamics of the P_i^t are as promised, each independently evolving according to the chain; the same is true of the R_i^t, which of course continue to be independent and stationary. Observe that if $R_i^t = P_i^t$ at some time t this property will be retained by the coupling in the future (as they are subject to the same update function). Now, consider any time period of length $E = 1 + (\hat{t} + c_{\mathsf{P}})/c_H$ rounds and any pair of particles P_i and Q_i. Observe that both particles must visit the state q_{pre} during this time period (as $\hat{t}$ and c_{P} are upper bounds on the transition times of the blue transitions); it follows that if the first of the two particles to visit q_{pre} remains in that state for the remainder of the E time steps then the two particles must couple (that is, coincide during this time period and forever after). Recalling that we take $T_1 \geq \hat{t}/c_H$, we find that the probability that that first particle remains in q_{pre} when the second one arrives is at least $p_{\text{couple}} := (1-p_1)^{E/c_H} = (1-p_1)^{(\hat{t}+c_{\mathsf{P}})/c_H} = [(1-p_1)^{\hat{t}/c_H}]^{(1+c_{\mathsf{P}}/\hat{t})} \geq [(1-1/T_1)^{T_1}]^{(1+c_{\mathsf{P}}/\hat{t})} \geq (1/e - O(1/T_1))^{(1+c_{\mathsf{P}}/\hat{t})}$. Thus p_{couple} is a constant larger than zero (and can be lower bounded as a function of the constant $c_{\mathsf{P}}/\hat{t}$). Note that the events that P_i couples with R_i (for distinct i) during such an epoch are independent, and it follows that after L such epochs the probability that there is a pair (P_i, R_i) that has not coupled is no more than $n(1-p_{\text{couple}})^L$. By the standard coupling lemma (see, e.g., [3, §12]), after L epochs the distance in total variation between $(P_1, \ldots, P_m)$ and the independent stationary distribution in each coordinate is no more than $m(1-p_{\text{couple}})^L$, which tends to zero exponentially quickly in L. This proves the lemma. □

Bounds on the events of interest. Consider, as above, the population of particles (players) $P_1, \ldots, P_n$ on the Markov chain. According to an evolution of these particles, given by the random variables P_i^t, we are interested in establishing upper bounds on the rate at which the adversary produces ranking blocks, and a lower bound on the rate at which the honest players produce uniquely isolated blocks.

Lemma 3. *Consider m parties, with arbitrary initial conditions but evolving independently on the Markov chain. Let $S = (\hat{t} + c_{\mathsf{P}})/c_H + 1$ and consider any interval of R rounds, the first of which starts at least S steps after the evolution begins. Then the probability that a particular player generates at least k ranking blocks in this interval is no more than $\binom{R+S}{k}(p_1 p_2)^k \leq (R+S)^k (p_1 p_2)^k$.*

Lemma 4. *Consider m independent parties walking on the Markov chain in the stationary distribution. Let p^*_{rank} denote the stationary probability of q_{rank}, then*

$$\Pr[t \textit{ is a uniquely isolated round}] \geq m(1 - (3\Gamma)p_1 p_2)^m p^*_{rank}\,.$$

In light of Lemma 2, the following is immediate.

Lemma 5. *Consider m players evolving according to the Markov chain, where the players are initially stationary and independent. Let p_{couple} denote the coupling constant of Lemma 2. Consider two rounds $\check{s} < s$ for which $|\check{s} - s| \geq L(1 + (\hat{t} + c_{\mathsf{P}})/c_H)$. Let I_s denote the indicator random variable for the event that s is uniquely isolated. Let C denote an arbitrary event depending only on the players trajectories prior to $\check{s}$. Then $|\Pr[I_s|C] - \Pr[I_s]| \leq (1 - p_{couple})^L$.*

Lemma 6. *Consider m players evolving on the Markov chain with any fixed initial states. Let p_{iso} denote the probability that a round is uniquely isolated under the stationary distribution, bounded below by Lemma 4. Fix a parameter $\sigma > 0$ and define $L = \ln(p_{iso}\sigma/2)/\ln(1 - p_{couple})$ and $E = L(1 + (\hat{t} + c_{\mathsf{P}})/c_H)$. Let $\{R, \ldots, R + S - 1\}$ be a sequence of rounds for which $R \geq E$. Let I_s be the event that the players produce a uniquely isolated block in round s. Then*

$$\Pr\left[\sum_s I_s \leq (1-\sigma)p_{iso}S\right] \leq E\exp\left(-\frac{(1-\sigma/2)\sigma^2 p_{iso}\cdot S}{8E}\right).$$

Analysis of the adversarial successes. Next, we proceed to bound the rate of adversarial mining successes. Our analysis is going to depend on the level of moderate hardness of the underlying DAG computations family.

By Lemma 1 we argued that the speed-up the adversary gets by each extra queries to oracle $\mathcal{O}$ is bounded. In fact for a single extra query, the lemma tells us that the adversary can speed up its computation by $\hat{t}$ steps. Thus, in order to protect our protocol from grinding attacks, we *set the pre-hash hardness* parameter p_1 to $c_H/((1+\sigma)\hat{t} + 4)$, where $\sigma \in (0, 1)$ is a parameter associated with the concentration bounds we use later in our analysis. This implies that finding a small pre-hash takes on expectation $c_H/p_1 = (1+\sigma)\hat{t} + 4 > \hat{t}$ steps, i.e., it is more expensive than running M directly to compute a new PoUW; the extra steps added are related to costs occurring in our reduction later.

To aid our presentation, we define $t' := t + (2n + 4(p_2 + p_3)(nc_{\mathsf{P}} + tc_{\mathsf{V}})) \cdot p_1/c_H$ to be the increased corruption power the adversary gets, due to fact that our reduction to the MH of $\mathcal{I}$ is not tight, mainly because of the cost of generating and verifying SNARG proofs. With foresight, we let β be an estimation of the rate at which the adversary produces ranking blocks

$$\beta := p_2/\left[(1 - \hat{\epsilon}(1, 2, 2n/t')) \cdot \hat{t} + (1/((1+\sigma)p_1) + 1) \cdot (c_H - 4p_1)\right].$$

The expected number of steps to find a block, β^{-1}, is basically the number of attempts needed to find a small post-hash $(1/p_2)$, times the number of steps needed to find a small pre-hash (c_H/p_1) plus the time needed to perform the DAG computation $((1 - \hat{\epsilon}) \cdot \hat{t})$. The other constants of the formula are related to our security analysis, i.e., our reduction from an attacker against the blockchain to an attacker against MH. Finally, the parameters $0, 2, 2n/t'$ of $\hat{\epsilon}$ relate to the rate at which the adversary queries oracles $q_{\mathcal{O}}, q_{\mathcal{V}}, q_{\mathcal{M}}$ as explained in Section 4.

λ :	security parameter
n :	number of parties
t :	adversarial corruption bound
t' :	amplified adversarial corruption bound
c_H :	"mining" steps each party takes per round
$c_{\mathsf{P}}, c_{\mathsf{V}}$:	SNARG prover/verifier cost
$\hat{\epsilon}, \hat{t}, \hat{k}$:	MH DAG parameters
$T_1, p_1 = T_1/2^\lambda$:	target/success probability of prehash
$T_2, p_2 = T_2/2^\lambda$:	" of ranking block posthash
$T_3, p_3 = \frac{T_3 - T_2}{2^\lambda}$:	" of input block posthash
σ :	concentration-bound parameter
Δ, Γ :	network/serialization worst-case delay
β :	upper bound on ranking-block computation rate
δ_{MH} :	adversarial advantage in DAG computation rate
δ_{Steps} :	honest advantage in number of steps per round
δ_{tot} :	upper bound on the total block generation rate

Table 1: *The parameters of our analysis.*

Let r.v. $Z(S)$ denote the maximum number of distinct blocks computed by the adversary during S, where the pre-hash query for each of these blocks was also issued to the RO during S. We prove in the full version that the adversary cannot mine fresh ranking blocks with rate and probability better than that of breaking the moderate hardness experiment. The main proof idea is to use an adversary that creates blocks fast, to create another adversary that breaks the moderate hardness of $\mathcal{I}$. A summary of our notation is given in Table 1.

Lemma 7. *For any set of consecutive rounds S, where $|S| \geq \hat{k}(1+\sigma)p_2/(\beta \cdot t'c_H)$, it holds that $Z(S) \geq (1+\sigma)\beta \cdot t'c_H|S|$ with probability $negl(\lambda)$.*

Putting everything together. Next, we show that the probability that a uniquely successful round happens is larger than the expected adversarial mining rate per round. Towards this purpose, our next assumption ensures that the computational steps advantage of honest parties outperforms the moderate hardness advantage of the adversary, while at the same time the rate at which blocks are produced is upper bounded.

Assumption 3. *There exist constants $\delta_{\mathsf{MH}}, \delta_{\mathsf{Steps}}$ and $\delta_{tot} \in (0,1)$, such that for sufficiently large $\lambda \in \mathbb{N}$:*
- $(n-t)(1-\delta_{\mathsf{Steps}}) \geq t'$ (Steps per round gap)
- $p^*_{rank} \geq (1-\delta_{\mathsf{MH}})\beta \cdot c_H$ (Moderate hardness gap)
- $\delta_{\mathsf{Steps}} - \delta_{\mathsf{MH}} \geq \delta_{tot}$ (Steps vs. Moderate hardness gap)
- $\delta_{tot} > 3\Gamma \cdot \beta c_H(n-t)$ (Bounded block generation rate).

Based on Assumption 3, we can prove that the rate of uniquely successful rounds is bigger than the rate at which the adversary generates blocks.

Lemma 8. *It holds that $p_{iso} > (1+\delta_{tot})\beta t'c_H$.*

Together with the appropriate concentration bounds proved in Lemmas 6 and 7, Lemma 8 is sufficient to apply Theorem 2 for Π, which in turn implies that Π satisfies both Persistence and Liveness with overwhelming probability.

Corollary 1. *Given Assumptions 1,2 and 3, Ofelimos satisfies Persistence and Liveness for $k, u \in poly(\lambda)$, except with negligible probability.*

Finally, in the full version, we argue that under ideal conditions, i.e. optimal MH, small SNARG costs, etc., Ofelimos can tolerate *any dishonest minority*.

A more detailed treatment of useful work completion times. The analysis above calibrates pre-hash hardness as a function of $\hat{t}$, the worst-case completion time of useful work. In certain settings of interest, the time complexity of the useful work task may satisfy a significantly stronger bound with very high probability, in which case this reduced bound can take the place of $\hat{t}$ with only minimal changes to the development above. Specifically, if the time complexity is $\bar{t} < \hat{t}$ except with negligible probability, the value $\bar{t}$ can be uniformly substituted for $\hat{t}$ above with the addition of negligible error terms in the theorems above.

Security against multiple problem instances. As discussed in Section 5.2, our protocol can handle multiple problem instances by interleaving them. Note, that our security analysis extends to this case, since it is agnostic of the level of MH of problem instances. Instead of trying to detect the hardness level of the M computation corresponding to each submitted problem instance, our approach is to keep pre-hash hardness fixed throughout the execution of the protocol, at a level where even if the submitted computation is not MH we still retain some security guarantees.

DPLS against adversarial participation. Executing DPLS in our permissionless PoUW setting potentially implies substantial adversarial participation which can negatively influence the algorithmic performance. In particular, the adversary may not follow Algorithm 1, e.g., by publishing the result of the worst execution of M, instead of the best one.

While the presentation of DPLS is agnostic to adversarial participation, its embedding PoUW protocol is responsible to provide the respective defenses. In the full version of the paper, we present two important quality guarantees of our implementation of DPLS by our PoUW protocol as long as the adversary only controls a minority of the computational power: (i) during any sufficiently large round interval, honest parties contribute new updates at least proportionally to their relative mining power —in particular, the honest parties contribute more updates than the adversary; (ii) the adversary cannot extensively manipulate the score of its updates, as we enforce each update to additionally include the result of a "random" execution of M from the batch (the one that resulted in a small post-hash), which can be used to replace "best" execution if it had worse score in comparison.

Remark 4. (Grinding resistance amplification) As a corollary of our main hardness lemma, we can argue about the amplification of grinding resistance of a MH DAG computation achieved by the following construction: first, the new

exploration algorithm tries to find a small pre-hash, which then uses to seed the initial (potentially weakly grinding resistant) DAG computation [8]. Similarly to our PoUW, we set the hardness of finding the pre-hash to be approximately equal to the worst-case complexity of the initial DAG computation. By our lemma, it easily follows that this construction is maximally grinding resistant, i.e., the adversary gains no advantage by seeing extra problem instances, while incurring only a small loss on MH and ensuring that the (potentially useful) initial computation remains a substantial part of the exploration algorithm. In fact, we can do even better in the case where the initial DAG computation enjoys some limited form of grinding resistance, by downgrading the hardness of finding a small pre-hash proportionally to the grinding resistance parameter.

Having argued about the security of Ofelimos as a transaction ledger, we turn our attention to its usefulness as a problem solving system.

6.2 Protocol usefulness

The goal of any PoUW-based blockchain protocol is to be used to solve some, external to the blockchain, computational problem. We say that such a protocol has a high *usefulness rate* if the total computational work spent to run the blockchain and solve the external problem is not much bigger than just solving the problem with the best possible algorithm (for the setting considered), denoted by $\mathsf{A}_{\mathrm{best}}$. We study this rate for our protocol using two metrics. The first metric, $\mathsf{U}_{\mathrm{eng}}$, measures the overall ratio of computational steps, performed by honest parties, that the engine directs towards running the DPLS algorithm. Intuitively this metric captures how effective the protocol is as a DPLS engine. We generically define $\mathsf{U}_{\mathrm{eng}}$ as follows:

$$\mathsf{U}_{\mathrm{eng}} := \mathbb{E}[\text{DPLS steps per block}]/\mathbb{E}[\text{total steps per block}]$$

Next, we analyze $\mathsf{U}_{\mathrm{eng}}$ for Ofelimos. First, note that since we set pre-hash hardness based on the worst-case complexity of M, Ofelimos's $\mathsf{U}_{\mathrm{eng}}$ naturally depends on M's runtime distribution being concentrated close to $\hat{t}$. Fortunately, the core search function of local search algorithms, which M aims to model, usually boils down to iteratively evaluating candidate solutions in a neighborhood, thus making it easy for us to exactly calibrate its runtime, e.g., by counting the number of candidates evaluated. Assuming that this is indeed true, and M's running time is almost always $\hat{t}$, we show that for appropriate protocol parameters Ofelimos has a $\mathsf{U}_{\mathrm{eng}}$ close to $1/2$, i.e., half of the total work mining a new block goes to running the DPLS engine. The intuition is that as we decrease the probability of finding a new (input or ranking) block, hashing and running M costs dominate the cost of running the SNARG prover. Given now that for our scheme the cost of hashing is approximately equal to the cost of running M, the result is immediate. We formalize this in the next lemma.

[8] Our PoUW "collapses" to this construction if we set $p_2 := 1, p_3 := 0$.

Lemma 9. *Assume M has a fixed running time. Then, for any $\rho > \sigma + 4/\hat{t}$, if $p_2 + p_3 < \frac{(\rho-\sigma)\hat{t}-4}{2 \cdot c_P}$, U_{eng} is greater than $\frac{1}{2+\rho}$.*

The second metric, U_{alg}, compares the complexity of DPLS to algorithm A_{best}. Note, that for U_{alg} we only take into account the DPLS computation steps and no other steps related to the protocol, e.g, hashing, computing SNARGs.

$$\mathsf{U}_{\text{alg}} := \mathbb{E}[\text{total steps of } \mathrm{A}_{\text{best}}]/\mathbb{E}[\text{total steps of DPLS}]$$

U_{alg} cannot be studied generically as it depends on the specific external problem solved as well as the computational model we consider. For example, we expect U_{alg} to be much larger when we consider the best algorithm in a distributed setting compared to the best one in the single machine setting. Instead, in the full version of the paper, we showcase how U_{alg} can be estimated experimentally for our WalkSAT DPLS variant.

The two metrics that we introduced clearly separate costs associated with the ledger protocol (hashing and SNARGS) from costs that are induced by the specific algorithm implement. In fact, in the case where blocks are computed using the honest mining algorithm, the product of the two metrics is a good approximation of the usefulness rate.

Remark 5. (Improved U_{eng}) In the analysis of our protocol we did not make any assumptions about the grinding resistance of the underlying DAG computation $\mathcal{I}$. This had the effect of setting the pre-hash hardness (c_H/p_1) to be approximately equal to $\hat{t}$, in turn leading to U_{eng} being less than $1/2$. If $\mathcal{I}$ enjoys some non-trivial level of grinding resistance, we can take advantage of it and downgrade the pre-hash hardness, with the effect of having exactly the same security guarantees but with potentially much less work invested in hashing. In the case where $\mathcal{I}$ is maximally grinding resistant, this leads to U_{eng} being close to 1.

Acknowledgments. We thank Laurent Michel for providing us with valuable information about state-of-the-art stochastic local-search algorithms and their application to real-world problems.

References

1. A. Aggarwal, A. K. Chandra, and M. Snir. Communication complexity of prams. *Theor. Comput. Sci.*, 71(1):3–28, 1990.
2. R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002.
3. D. Aldous and J. A. Fill. Reversible markov chains and random walks on graphs, 2002. Unfinished monograph, `http://www.stat.berkeley.edu/~aldous/RWG/book.html`.
4. M. Andrychowicz and S. Dziembowski. Pow-based distributed cryptography with no trusted setup. In R. Gennaro and M. Robshaw, editors, *Advances in Cryptology - CRYPTO 2015*, volume 9216 of *Lecture Notes in Computer Science*, pages 379–399. Springer, 2015.

5. C. Badertscher, P. Gazi, A. Kiayias, A. Russell, and V. Zikas. Consensus redux: Distributed ledgers in the face of adversarial supremacy. IACR Cryptology ePrint Archive, Report 2020/1021, 2020.
6. C. Badertscher, U. Maurer, D. Tschudi, and V. Zikas. Bitcoin as a transaction ledger: A composable treatment. In J. Katz and H. Shacham, editors, *Advances in Cryptology – CRYPTO 2017*, pages 324–356. Springer, 2017.
7. A. Baldominos and Y. Saez. Coin. ai: A proof-of-useful-work scheme for blockchain-based distributed deep learning. *Entropy*, 21(8):723, 2019.
8. M. Ball, A. Rosen, M. Sabin, and P. N. Vasudevan. Proofs of work from worst-case assumptions. In *Advances in Cryptology – CRYPTO 2018*. Springer, 2018.
9. M. Bellare and P. Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Fairfax, Virginia, USA*, pages 62–73, 1993.
10. D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In H. Shacham and A. Boldyreva, editors, *Advances in Cryptology - CRYPTO 2018*, volume 10991 of *Lecture Notes in Computer Science*. Springer, 2018.
11. K. Chatterjee, A. K. Goharshady, and A. Pourdamghani. Hybrid mining: exploiting blockchain's computational power for distributed problem solving. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, 2019.
12. F. Coelho. An (almost) constant-effort solution-verification proof-of-work protocol based on merkle trees. Cryptology ePrint Archive, Report 2007/433, 2007.
13. A. Coventry. Nooshare: A decentralized ledger of shared computational resources. `https://web.archive.org/web/20220620105201/http://web.mit.edu/alex_c/www/nooshare.pdf`, 2012.
14. P. Daian, R. Pass, and E. Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In *International Conference on Financial Cryptography and Data Security*, pages 23–41. Springer, 2019.
15. B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Advances in Cryptology - EUROCRYPT 2018*, 2018.
16. M. Dotan and S. Tochner. Proofs of useless work–positive and negative results for wasteless mining systems. *arXiv preprint arXiv:2007.01046*, 2020.
17. S. Dziembowski, S. Faust, V. Kolmogorov, and K. Pietrzak. Proofs of space. In *Advances in Cryptology – CRYPTO 2015*, 2015.
18. M. Fitzi, A. Kiayias, G. Panagiotakos, and A. Russell. Ofelimos: Combinatorial optimization via proof-of-useful-work—a provably secure blockchain protocol. Cryptology ePrint Archive, Paper 2021/1379, 2021.
19. Gapcoin. Gapcoin, 2014. `https://gapcoin.org/`.
20. J. A. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT (2)*, pages 281–310. Springer, 2015.
21. J. A. Garay, A. Kiayias, and G. Panagiotakos. Consensus from signatures of work. In *Topics in Cryptology – CT-RSA 2020*, 2017.
22. J. A. Garay, A. Kiayias, and G. Panagiotakos. Blockchains from non-idealized hash functions. In *Theory of Cryptography*, 2020.
23. Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, pages 51–68, 2017.
24. J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, 2016.
25. J. Groth, M. Kohlweiss, M. Maller, S. Meiklejohn, and I. Miers. Updatable and universal common reference strings with applications to zk-snarks. In *Advances in Cryptology – CRYPTO 2018*, 2018.

26. N. Gupta and D. S. Nau. On the complexity of blocks-world planning. *Artif. Intell.*, 56(2-3):223–254, 1992.
27. H. H. Hoos and T. Stützle. *Stochastic local search: Foundations and applications*. Elsevier, 2004.
28. H. Kautz, B. Selman, and D. McAllester. Walksat in the 2004 sat competition. In *Proceedings of the International Conference on Theory and Applications of Satisfiability Testing*, 2004.
29. T. Kerber, A. Kiayias, and M. Kohlweiss. Mining for privacy: How to bootstrap a snarky blockchain. Cryptology ePrint Archive, Report 2020/401, 2020.
30. A. Kiayias, S. Quader, and A. Russell. Consistency of proof-of-stake blockchains with concurrent honest slot leaders. IACR Cryptology ePrint Archive, Report 2020/041, 2020.
31. A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Advances in Cryptology – CRYPTO 2017*, pages 357–388. Springer, 2017.
32. S. King. Primecoin: Cryptocurrency with prime number proof-of-work, 2013.
33. A. Lihu, J. Du, I. Barjaktarevic, P. Gerzanics, and M. Harvilla. A proof of useful work for artificial intelligence on the blockchain. *arXiv:2001.09244 preprint*, 2020.
34. A. F. Loe and E. A. Quaglia. Conquering generals: an np-hard proof of useful work. In *Proceedings of the 1st Workshop on Cryptocurrencies and Blockchains for Distributed Systems*, pages 54–59, 2018.
35. M. Maller, S. Bowe, M. Kohlweiss, and S. Meiklejohn. Sonic: Zero-knowledge snarks from linear-size universal and updatable structured reference strings. In *ACM CCS '19, London, UK*, pages 2111–2128, 2019.
36. A. Miller, A. Juels, E. Shi, B. Parno, and J. Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE S&P*, pages 475–490. IEEE, 2014.
37. T. Moran and I. Orlov. Simple proofs of space-time and rational proofs of storage. In *Advances in Cryptology – CRYPTO 2019*, 2019.
38. C. G. Oliver, A. Ricottone, and P. Philippopoulos. Proposal for a fully decentralized blockchain and proof-of-work algorithm for solving np-complete problems. *arXiv preprint arXiv:1708.09419*, 2017.
39. C. H. Papadimitriou and J. D. Ullman. A communication-time tradeoff. *SIAM J. Comput.*, 16(4):639–646, 1987.
40. S. Park, A. Kwon, G. Fuchsbauer, P. Gaži, J. Alwen, and K. Pietrzak. Spacemint: A cryptocurrency based on proofs of space. In *International Conference on Financial Cryptography and Data Security*, 2018.
41. R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology - EUROCRYPT 2017*, pages 643–673, 2017.
42. R. Pass and E. Shi. Fruitchains: A fair blockchain. In E. M. Schiller and A. A. Schwarzmann, editors, *ACM PODC 017, Washington, DC, USA, July 25-27, 2017*, pages 315–324. ACM, 2017.
43. B. Selman, H. A. Kautz, and B. Cohen. Noise strategies for improving local search. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, page 337–343, USA, 1994.
44. F. Zhang, I. Eyal, R. Escriva, A. Juels, and R. Van Renesse. REM: Resource-efficient mining for blockchains. In *26th USENIX Security Symposium USENIX Security 17)*, pages 1427–1444, 2017.
45. W. Zheng, X. Chen, Z. Zheng, X. Luo, and J. Cui. Axechain: A secure and decentralized blockchain for solving easily-verifiable problems. *arXiv preprint arXiv:2003.13999*, 2020.