# Insane in the (block)Chain: Solving ILPs for Fun and Profit

JORDAN BARKIN, LAVANYA SINGH, and LUKE KENWORTHY

Additional Key Words and Phrases: blockchain, NP, integer linear programming, distributed systems

## 1 INTRODUCTION

Despite its obvious and significant shortcomings, the popularity of blockchain shows no immediate signs of waning. The cryptocurrency Bitcoin, the largest scale use of blockchain technology in 2021, has a market capitalization exceeding $1T, and its chain executes up to 400,000 transactions per day [1]. The verification of these transactions, so-called Bitcoin "mining," consumes an estimated 121.36 terawatt-hours per year, or more electricity than all of Argentina [2]. Energy inefficiency is a significant roadblock to scaling blockchain technology and a cause for ethical and environmental concern.

A bitcoin miner's workload consists of randomly guessing 4 byte "nonces" until a single correct value is found. These computations have no utility other than providing a computationally difficult proof-of-work routine. Guessing random nonces accomplishes no productive work. This problem, however, is not inherent to blockchain technology. The proof of work must be an easy-to-verify witness that demonstrates that a significant amount of computational work has been performed. Consequently, any NP-complete problem will suffice as proof-of-work. By redesigning cryptocurrency proof-of-work algorithms to perform productive NP-complete work, we can mitigate blockchain's energy waste. Integer linear programming (ILP), an NP-complete problem with many industry applications, is a compelling candidate for a useful proof-of-work. We propose ILPcoin, a novel implementation of a cryptocurrency taking advantage of this new proof-of-work model. [1].

## 2 SYSTEM OVERVIEW

The Bitcoin network uses hashing as its proof of work. Miners search for nonces that, when embedded in a particular block, cause the block to hash to a value containing an algorithmically-defined number of leading zeroes. This number of leading zeroes is Bitcoin's "hardness factor," which measures the difficulty of the hash puzzle. While this random number generation and hashing is wasted work, it does provide the benefit of randomness. With the hashing scheme, it is possible for someone with minimal computational power to get lucky and guess a nonce that solves the hash puzzle. This property informally gives Bitcoin its anti-dominance guarantee: with high probability, as long as honest Bitcoin miners control more than half of the computation power on the network, all transactions on the network will behave normally. Miners are probabilistically able to mine blocks in proportion to their computational power.

With integer linear programming, such luckiness is less likely. Because integer linear programs are so important, even though there is no known polynomial time algorithm to solve an integer linear program, there exist highly-optimized solvers that can solve practical ILP instances efficiently. All of these solvers use similar methods to reduce computational time, such as the branch-and-bound technique. Thus, in practice, all miners on the network

[1] The ILPcoin source code is available here: http://github.com/lsingh123/ilpcoin

will take similar paths to solve an ILP. If the network solely uses ILPs as proof-of-work, the miner with the most computational resources available will dominate the network. For examine, consider a network with 5 computers. Computer 1 calculates ILP solutions using a solver 3 times faster than Computers 2-5. Computer 1 will deterministically mine every block first and control the blockchain, despite not controlling a majority of the computation resources on the network. This does not provide the anti-dominance guarantee.

To solve this issue, we take a hybrid approach to proof of work, which is the main contribution of our system. Miners pop an integer linear programming problem off a centralized queue and compute its solution. The miner must then generate a nonce that, when hashed with the previous block AND the solution to the ILP, produces a hash with a certain number of leading zeroes. In this case, miners gain a greater “head start" the quicker they solve the ILP problem, but there is some randomness in the system. Miners are incentivized to both perform work and verify transactions by the inclusion of a mining reward.

The nonce and the ILP solution serve as the certificate. Once a miner finds both, it broadcasts the certificate to its peers. Peers verify the proof of work by ensuring that the ILP solution is correct and that the hash of the previous block, ILP solution, and the nonce meet the leading zeroes specification. Once a peer verifies the certificate, they add the block to the blockchain.

# 3 HYBRID PROOF OF WORK THEORY

## 3.1 Definition of ILP

NP is the class of decision problems verifiable in polynomial-time, so, rigorously, the decision problem variant of Integer Linear Programming is NP-complete. In order for the proof-of-work to be easy to verify, we use the decision variant of ILP.

*Definition 3.1 (Decision Integer Linear Programming).* The problem Decision Integer Linear Programming takes as input a set of linear constraints $C = \{c_1, c_2, \ldots\}$, a linear objective function $d$, and a value $k$. A solution to Decision Integer Linear Programming returns an assignment $\phi_i$ to each variable such that $d(\phi_1, \ldots, \phi_n) \geq k$ and $\forall c \in C, c(\phi_1, \ldots \phi_n) = 1$.

The definition of “Minimize Decision Integer Linear Programming" is identical to that above, except the value of the objective function on assignment $\phi$ must be less than $k$. In this paper, we use Decision Integer Linear Programming and Minimize Decision Integer Linear Programming interchangeable, and refer to both as ILP.

## 3.2 Theoretical Results

We will briefly discuss a theoretical model of the hybrid proof of work system to examine the anti-dominance properties of this approach. We assume there are two types of computers, $\alpha$ and $\beta$. In our model network, there is one $\alpha$ computer that is $d$ times faster (in terms of ILP solving time and hash rate) than all other computers on the network. We assume $d > 1$. The network also contains $n$ other $\beta$ computers. Each computer tries to find a nonce by repeatedly sampling random nonces with replacement. The hash puzzle has a difficulty of $h$, where miners have a $\frac{1}{h}$ chance of guessing a valid number each time they hash a new one.

Theorem 3.2 (Bitcoin Advantage Factor). *In an $n - \beta$ computer Bitcoin network, the $\alpha$ computer has a $\frac{d}{n+d}$ chance of mining any given block.*

Proof. Let $r$ denote the hashrate of the $\beta$ computers in hashes per second. The total hash rate of the network is $nr + dr$, of which the $\alpha$ computer controls $dr$. Thus, the $\alpha$ computer has a $\frac{d}{n+d}$ chance of mining any given block. □

In the Bitcoin network, the $\alpha$'s probability of mining a particular block is proportional to its computing advantage. In the case of the hybrid proof of work system, the $\alpha$ computer gets a head start on this hash puzzle because it finishes the ILP first. Let $b$ represent the time it takes for the $\beta$ computer to solve the ILP, expressed as the number of hashes it could produce in the time it takes to solve the puzzle. We express difficulty in this way based on our assumption that an $\alpha$ computer can both solve an ILP and produce hashes $d$ times faster than a $\beta$ computer.

Theorem 3.3 (ILPcoin Advantage Factor). *In an $n - \beta$ computer ILPcoin network, the $\alpha$ computer mines a block with probability*

$$1 - \left(\frac{n}{n+d}\right)\left(1 - \frac{1}{h}\right)^{b(d-1)}$$

*where $b$ denotes the number of hashes that a $\beta$ computer could compute in the time taken to solve one ILP.*

Proof. The $\alpha$ computer requires $\frac{b}{dr}$ seconds to solve this puzzle, while the $\beta$ computer requires $\frac{b}{r}$, giving the $\alpha$ computer $\frac{b}{r} - \frac{b}{dr}$ seconds to guess hashes without any $\beta$ computers guessing. In that time, it generates $rd(\frac{b}{r} - \frac{b}{dr}) = b(d-1)$ hashes, giving it a $1 - (1 - \frac{1}{h})^{b(d-1)}$ chance of generating the right hash during its head start. If it does not guess the right hash during the head start, then, as in the Bitcoin case, it has a $\frac{d}{n+d}$ chance of mining the right block while the $\beta$ computers each have a $\frac{1}{n+d}$ chance. Using the law of total probability, the $\alpha$ computer has a $1 - (1 - \frac{1}{h})^{b(d-1)} + (1 - \frac{1}{h})^{b(d-1)} \frac{d}{n+d}$ probability of solving the hash puzzle for a given block, while the $\beta$ computers have a $(1 - \frac{1}{h})^{b(d-1)} \frac{n}{n+d}$ total probability of mining a block.

□

This expression gives us some key insights. Let $f = 1 - (1 - \frac{1}{h})^{b(d-1)}$ denote the "head start" factor. As $f \longrightarrow 1$, the probability of the $\alpha$ computer mining a given block approaches 1. As $f \longrightarrow 0$, the probability of the $\alpha$ computer solving the puzzle approaches $\frac{d}{n+d}$, the same as in the bitcoin case. There are three factors that impact $f$: $h$ decreases $f$, while $b$ and $d$ increase $f$. Thus, a more difficult hash puzzle causes a network equalizing effect, while a more difficult ILP and a higher performance of the $\alpha$ computer relative to the $\beta$ computers cause the $\alpha$ computer to dominate the network.

## 3.3 Application

To apply these results, consider a network of 10,000 miners on the network ($n = 9,999$), where the $\alpha$ computer is 5x faster than each $\beta$ computer ($d = 5$). Each $\beta$ computer has a hashrate of 50 million hashes per second ($r = 50,000,000$) and a $\beta$ computer solves and ILP in 120 seconds($b = 120 * r = 6,000,000,000$ hashes). Each random number has a $1/100,000,000,000$ chance of being the right nonce ($h = 100,000,000,000$). In the bitcoin case, the $\alpha$ computer would have a .03998% chance of mining the block. It would need to be 9,999x faster than each $\beta$ computer to control the whole network (have over a 50% chance of mining any given block). In the hybrid model, the $\alpha$ computer would have an 21.37% chance of solving the hash puzzle. This gives the $\alpha$ computer a significant advantage relative to its share of the overall network hashrate, which is only .03998%. It would only need to be roughly 13 times faster than each $\beta$ computer to control the network. Note that, if we make the hash puzzle 10 times more difficult, the $\alpha$ computer's advantage decreases to 2.42%, requiring roughly 115 times as much computer speed to control the network.

These theoretical results show that, depending on tunable ILPcoin parameters, a faster ILP solver can provide a significant advantage. This incentivizes miners to invest in faster computer machinery and reduce any unequal access to this computing power. At the same time, these results also show the impact of a computing advantage in the hybrid model. A small subset of speedy, adversarial miners can team up and realistically dominate the network. Fortunately, these results also show that the computing advantage can be thwarted by increasing the difficulty of the hash puzzle, $h$. Thus, an adaptive algorithm can set the hash rate in such a way that it would be highly unlikely for an adversarial group of miners to control the blockchain. For the sake of simplicity, we do not implement real-time hardness tuning.

## 4 IMPLEMENTATION DETAILS

An ILPcoin network consists of four types of participants, who communicate over the network:

(1) a centralized queue, which provides the ILPs to serve as proof-of-work;
(2) miners that solve ILPs and compute nonces;
(3) verifiers, that maintain the state of the chain and verify blocks; and
(4) customers, who provide ILPs that they need solved to the queue.

These standalone components rely on two shared libraries: an ILP library which handles representation and solving of ILPs and a Blockchain library that handles representation of the blockchain. The rest of this section presents the implementation details for each of these six components.

### 4.1 ILP Library

All components of the blockchain need to represent ILPs and their solutions in a consistent way. Additionally, miners need to solve ILPs, and verifiers need to check their solutions. We provide all of this functionality in a common library in `src/ilpcoin/common/ilp.py`. The library exposes two classes, `Ilp` and `IlpSolution`, and has two primary goals: to abstract away the mechanics of representation, solving, and checking ILPs, and to consistently handle serialization and deserialization for communicating ILPs between components on the blockchain.

The `Ilp` class is backed by Python-MIP [3], a Python library for the modeling and solving of ILPs. A Python-MIP *model* is constructed from the definition of an ILP, which consists of an objective function, variables, and constraints. The module exposes a `solve` method that sets an internal `solution` attribute to a representation of the ILP's solution. Some feature of raw MIP models are troublesome for our use case: MIP models are not serializable, the solution is tied to the local instance of the model that solved it, and they represent general purpose ILPs, instead of the decision problems that Ilpcoin deals with. These issues made the model class alone unsuitable for use as our representation of ILPs. To address these issues, our `Ilp` and `IlpSolution` class introduce additional state. For each ILP, we store the following:

(1) The underlying MIP model;
(2) a globally unique UID;
(3) a value `k`, the threshold above (if a maximization problem) or below (if minimization) that the objective function must evaluate to for a set of values for each variable to be considered a solution to this ILP; and
(4) a flag `maximize`, which is set if the ILP represents a maximization problem and false otherwise.

The `Ilp` class exposes functions for serialization and deserialization, both to hexidecimal strings and bytes; a `solve` function with a timeout that the miner uses to solve and produce an `IlpSolution` object; and a `check` function, which checks if an `IlpSolution` object represents a correct solution to the Ilp.

`IlpSolution` objects represent the solutions to Ilps and are produced by miners. Each `IlpSolution` either contains the set of values for each variable that satisfies the Ilp or has a `no_solution` flag set. Like `Ilp` objects, `IlpSolution` objects are serializable to both hexidecimal strings and bytes.

Together, these two classes abstract all of the Ilp-related implementation details away from the other components of ILPcoin.

### 4.2 Centralized Queue

As the only centralized component of ILPcoin, the queue fulfills three roles:

(1) It maintains a queue of ILPs provided by *customers* for the blockchain to use as proof of work. The top of the queue represents the current problem being solved by the chain. When enough verifiers (more than `VERIFIERS_REQUIRED`) have informed the queue of a correct solution to the ILP, the top of the queue is popped, and the chain begins working on the next problem.
(2) It acts as a DNS for other nodes (miners and verifiers) on the network by exposing a `/get_neighbors(n)` endpoint that returns a list of *n* verifier addresses, randomly sampled from the set of all verifiers that have registered with the queue. If fewer than n verifiers are registered, the endpoint returns all verifiers. Verifiers use this endpoint to learn about sets of neighbors to whom they can gossip newly verified blocks.
(3) It acts as the single centralized entrypoint for *customers*. Customers submit ILPs that they want solved to the queue using the `/add_ilp` endpoint, and ask the queue for the solution to any ILP that has been historically solved on the blockchain using the `/get_solution_by_id/<uid>` endpoint. The ILP forwards these requests to a subset of verifiers, which iterate over the chain to find the desired solution.

### 4.3 Sample Customer

Our implementation includes a prototype of an ILPcoin customer in `src/ilpcoin/sample_customer/__main__`. Recall that customers are third parties who have integer linear programs that they wish to solve. Customers need to add ILPs to the queue and retrieve a solution once the chain has found one. We create a sample customer that adds integer linear programs to the queue, sleeps for a set period of time, and then queries the queue for a solution to the ILPs. We create instances of the travelling salesperson problem and the knapsack problem used for testing, demos, and the sample customer. The customer posts a request to the endpoint `/add_ilp` with the serialized ILP in the request data. Serialization is performed using the ILP library described in Section 4.1. To retrieve a solution, the customer makes a `GET` request to the queue endpoint `/get_solution_by_id/<id>`. The queue will either respond with an error code or a serialized ILP solution, which the customer can deserialize using the ILP library. The complete customer workflow can be implemented in fewer than 10 lines of code.

## 4.4 Blockchain Library

ILPcoin includes a Blockchain library in `src/ilpcoin/common` that contains all of the functionality necessary to represent and store a blockchain. Miners and verifiers use this library to verify, serialize, deserialize, and hash blocks. The system's modular design makes testing easier and reduces the occurrence of bugs due to inconsistencies in hash functions or wire protocols.

*4.4.1 Transaction.* The smallest unit of a blockchain is an individual transaction. Transactions consist of a sender, receiver, and amount, all of which are represented as fields in the `Transaction` class. The class includes a custom hash function. The hash function concatenates the sender, receiver, and amount and returns the hexadecimal representation of the sha256 hash of the concatenated byte-string. All parties in the system use `Transaction.hash` to hash a transaction, minimize hashing errors.

*4.4.2 Block.* A block consists of a nonce, the hash of the previous block, an ILP problem, an ILP solution, and a list of transactions. A nonce is an integer value that the miner can modify until the hash of the block satisfies the hardness property. The hash of the previous block is computed using `Block.hash`, which computes the sha256 hash of the concatenation of each component in the block, as well as the hash of each transaction in the transaction list. The ILP solution is a list of variable assignments. The transaction list contains `BLOCKSIZE` transactions, where `BLOCKSIZE` is a constant global to the entire blockchain.

The ILP problem is an integer representing the unique identifier for this particular ILP, as assigned by the queue. We made this design choice to reduce storage redundancy, since the queue already stores the full ILP. While this makes blocks lighter, it also adds additional communication overhead because verifiers must ask the queue for the full ILP before verifying that the solution is correct. Additionally, the queue becomes a single point of failure that all verifiers rely on. Further optimizations, such as caching a portion of the queue at each verifier, can alleviate these problems.

*4.4.3 Blockchain.* A blockchain is a list of Blocks. The `Blockchain` class contains a method to verify transactions as described below:

```
# this method confirms that transaction at position trans_index in block block_index is valid
# a valid transaction is defined as one that does not double spend
def verify_transaction(transaction: Transaction, block_index: int, trans_index: int) -> bool:
```

This method is called once to verify individual transactions or repeatedly to verify entire blocks.

*4.4.4 Wire Protocol.* Verifiers and miners send transactions, blocks, and blockchains over the network to each other. This requires a wire protocol. We choose to use Python's pickle function as our wire protocol. This simplifies the serialization and deserialization logic greatly and saved programmer time and program complexity. However, pickle is known to have devastating security vulnerabilities, and unsanitized input should never be unpickled. If this system is deployed, this vulnerability should be patched.

All serialization and deserialization occurs within the Blockchain library using the `serialize` and `deserialize` functions for each class described above. This simplifies client code and makes testing easier. It also reduces the occurance of errors due to serialization and deserialization.

## 4.5 Verifier

A verifier is the only component of the system that maintains a copy of the entire blockchain. The blockchain is represented using the blockchain library described in Section 4.4. Verifiers are responsible for sharing blockchains with each other during initialization, verifying blocks that miners send them, advertising blocks to each other, and exposing records of solved ILPs and user wallets to third-parties. In the following sections, we examine each of these responsibilities in detail.

The code for a verifier is in the directory `src/ilpcoin/verifier`. `server.py` contains a `Server` class that runs the Flask app and defines the endpoints that each verifier accepts requests on. This class is a thin shim layer that handles request packaging and unwrapping. The core functionality is in `verifier.py`, which has a `Verifier` class that extends the `Server` class to handle the responsibilities described below.

*4.5.1 Initialization.* The first verifier to come online has, by convention, an id of 1. This first verifier makes the “genesis block," which is the first block in the chain and contains a valid proof of work with no transactions. This bootstrapping procedure is similar to that of Bitcoin [4]. No other verifier should make a genesis block.

Every verifier registers its existence with the queue as soon as it comes online. It then asks the queue for a list of $n$ neighboring verifiers, for some global constant $n$, since the queue acts as the DNS server for the system. These neighbors are encoded as a list of unique id's, but a large-scale implementation will include IP addresses for each verifier. Once a verifier has neighbors, it has to get a copy of the blockchain from them. It asks all neighbors for the length of their blockchain using the `/get_length` endpoint. It then asks the neighbor with the longest blockchain to send the entire blockchain over the wire, and adopts this as its view of the chain. If its neighbors all advertise a length of 0, it sleeps and tries again.

*4.5.2 Verification.* A verifier exposes a `/send_block/<id>` endpoint that other verifiers and miners use to submit blocks for verification. The `id` field in this path is the id of the sender, and this will be useful during block advertising. Once a verifier receives a block, it processes the block and responds with either `SUCCESS` or an error code that indicates where verification failed, such as `INVALID_TRANSACTION` and `INVALID_NONCE_OR_POW`.

Verification itself has four components. First, the verifier checks that the first transaction in the block is the mining reward, of some fixed size. Second, the verifier iterates through the rest of the transactions in the block to check for double spending. The current algorithm to do this is naive and replays the entire chain to determine the sender's wallet size. This can be optimized as the system scales, perhaps by snapshotting user wallet values within a block or using a Merkel tree [5] and Bitcoin-style transactions [6]. Third, the verifier must validate the proof of work. It checks that the hash of the block (calculated using the shared blockchain library) satisfies the hardness property. Additionally, it verifies the solution to the ILP. It asks the queue for the top of the queue to confirm block freshness. Without this check, a malicious miner could reuse solutions to stale ILPs to mine blocks without doing the computational work necessary. It then grabs the full ILP from the queue using the `/get_ilp_by_id/<id>` endpoint and verifies that the given solution is correct using the shared ILP library described in Section 4.1.

Fourth, the verifier validates that the hash of the previous block bundled within the block is indeed the hash of the block currently at the top of its blockchain. This ensures that the blockchain is an immutable, totally ordered list. Without this property, a malicious miner or verifier could swap the order of blocks, validating invalid transactions or invalidating valid transactions.

*4.5.3 Advertising.* Verifiers use a gossip algorithm [7] to notify each other of successfully verified blocks. Once a verifier verifies a block successfully, it adds that block to a queue of blocks to verify. A separate thread pops blocks off this queue and sends them to all of this verifier's neighbors. Advertising must be performed concurrently to block verification to avoid system-wide deadlock. If advertising and verification happened in a single thread, deadlock could occur as follows: Verifier $A$ sends a block to verifier $B$ for verification. Verifier $B$ tries to advertise the block to verifier $A$, but verifier $A$ is blocking on the request to verifier $B$. Verifier $B$ blocks on the request to verifier $A$ so the system grinds to a halt. The gossip algorithm also runs the danger of recursing forever if verifier $A$ advertises a block to verifier $B$, who advertises it back to verifier $A$ and so on. To mitigate this problem, the `/send_block` endpoint also requires the id of the sender. This allows the receiving verifier to avoid broadcasting a block to the originating party.

Verifiers use to this gossip algorithm to reach consensus on the state of the blockchain. This consensus algorithm is similar to the one used in Bitcoin. This advertising is also necessary to advance the state of the queue, because the queue will only pop an ILP once a certain number of verifiers have registered a solution to it. The gossip algorithm can also help a verifier notice an out-of-date chain. If upon block advertisement, all neighbors indicate that the previous hash is incorrect, this means that the current verifier must have a stale view of the blockchain.

## 4.6 Miner

The miner's responsibility is to provide the proof of work for the block chain and to create the blocks. During initialization, a miner asks the queue for addresses of neighboring verifiers. It then requests the most recent block in the chain from the verifier and the most recent ILP from the top of the queue. Armed with the previous block and an ILP, the miner assembles a block. It first adds its transactions to the block, including the mining reward, which is a transaction of a fixed amount that the miner sends to itself. It then completes the proof-of-work by solving the ILP and hashing random numbers until it finds one that satisfies the hash puzzle. Once it finds a solution, it packages up the block and broadcasts it to neighboring verifiers.

## 5 EXPERIMENTAL RESULTS

Compared to a single machine solving an ILP, ILPcoin faces significant overhead because of verification time. An ILP is not considered “solved" until a certain number of verifiers have learned of and verified its solution. Intuitively, the more verifiers in the system, the longer it takes for ILPcoin to solve an ILP. To test the effect of the number of verifiers on the efficiency of mining blocks, we ran our system with varying numbers of verifiers to see how many blocks it could validate in one minute. The results are shown in Table 1. We found that, as more verifiers are added to the network, it takes more time for blocks to propagate. The addition of verifiers to the network requires more communication across the network to propagate a correct block and additional verification overhead. The case with one verifier is significantly faster because no synchronization has to occur across the network. This case forms a degenerate network, where the queue, miner, and verifier coordinate to solve an ILP without any decentralization.

| Number of Verifiers | Number of blocks validated in 60 sec |
|---|---|
| 1 | 49 |
| 2 | 15 |
| 3 | 9 |
| 4 | 6 |
| 5 | 5 |
| 6 | 4 |

Table 1. Number of verifiers vs blocks validated.

## 6 DISCUSSION

**Proof of Work** We chose Integer Linear Programming as a basis for proof-of-work because of its broad applicability in industry. Reductions to Integer Linear Programming tend to be more straightforward than reductions to any other NP-complete problem and many scheduling and supply chain coordination problems reduce naturally to Integer Linear Programming. Because ILP is in NP, verifying the existence of a solution is easy but verifying the impossibility of a solution may be difficult (unless NP = coNP). To avoid this problem, we allow an `nosolution` flag in the `IlpSolution` class in the ILP Library. If a miner is unable to find a solution before a timeout elapses, then it marks the ILP as unsolveable. This system may allow for false negatives, where an ILP with a difficult-to-find solution is marked as impossible.

One drawback of this choice of problem is that there are known, effective approximation algorithms for Integer Linear Programming, unlike other NP-complete problems (e.g. the Closest Vector Problem [8]). ILPcoin verifiers require exact solutions, but the existence of good approximation algorithms can be leverage to create efficient (but not polynomial-time) ILP solvers. ILP is a well-studied problem, so such algorithms exist and are widely available [3]. The ideal choice of problem would have no algorithm better than random guessing, but the existence of efficient algorithms for ILP and the ease of reduction to ILP is evidence that this is impossible.

Another known difficulty with using NP-complete problems as proof-of-work is the difficulty of generating and tuning problems. Problems must be generated continuously to keep the chain running. A hash puzzle is automatically generated by the creation of block, but generating tunably difficult instances of NP-complete problems is known to be difficult [9, 10]. ILPcoin solves this problem by randomly generating knapsack problems to keep the chain running. The difficulty of the proof-of-work should also be tunable to allow a "system adminstrator" to control the rate of mining to accommodate factors like inflation, increasing computational power and the creation of efficient ILP solvers. Such tuning is easy for a hash puzzle because it simply requires changing the number of zeroes required to solve the puzzle. Tuning is not so straightforward for an ILP. One option is to require that customers submit ILPs with a certain number of variable and constraints at a given time. Another option is to pad easy ILPs with randomly generated variables and constraints that are extraneous to the solution that the customer desires. Lastly, because ILPcoin requires that miners solve a hash puzzle in addition to solving an ILP, problem difficulty can be tuned by tuning the difficulty of the hash puzzle. That being said, the harder the hash puzzle, the more computational work that the chain collectively "wastes" in solving useless hash puzzles. More complete discussion of the trade-off between hash puzzle difficulty and ILP difficulty is expressed in Theorem 3.3.

**Centralization** The usage of a single centralized queue that distributes ILP problems appears to contradict the decentralization of blockchain. A certain amount of centralization is inevitable in every system. Even Bitcoin requires a bootstrapping protocol to discover other nodes on the chain. At the very least, users of a blockchain must trust the developers who wrote the trusted computing base or developed the blockchain protocol. We envision ILPcoin as a decentralized currency maintained by a single administrator with out-of-band accountability constraints. In Bitcoin, the administrator is the Bitcoin development community, who face little accountability in the real world. ILPcoin is designed to be administrated by a business, likely one that needs ILPs to be solved, that can be held accountable by out-of-band parties like the government.

In practice, few systems are perfectly decentralized, but with care, the impacts of centralization can be mitigated. ILPcoin seeks to mitigate these impacts by limiting the functionality of the queue. In particular, the queue is *not* involved in the verification or assignment of ILPs. The queue simply acts as a centralized storage system. It can cause failstop faults for the entire system, but not byzantine faults. The queue can stop blocks from being added to the chain (by popping ILPs off of the queue immediately) but it cannot add bad blocks to the chain without controlling a certain amount of verifiers or miners, as explained in Section 3. Those committed to decentralization can replace the centralized queue with a decentralized queue that exposes the same interface.

The centralization of the queue does open ILPcoin to attacks that traditional blockchains do not suffer. In a production system, the queue should authenticate customers to prevent malicious miners from submitting easy ILPs to the queue to control the chain. The queue is also the natural choice to enforce "hardness" requirements on ILPs described above. The queue can require that clients submit ILPs satisfying some hardness criteria, or it can pad ILPs with extra variables and constraints to make problems more difficult. While this makes the proof-of-work hardness tunable, additional queue functionality creates a bigger attack surface for the queue. Because the queue is a single point of failure and part of the trusted computing base for ILPcoin, we keep it as thin as possible.

**Useful Work** While Integer Linear Programming is a useful problem with many applications, our blockchain itself can only solve one Integer Linear Program at a time. In order for the chain to be fair, each miner must be mining the same ILP at the same time. If miners each solve different ILPs to mine the same block, then solution-finding will occur in "waves." Miners with similar computational power will find solutions in a similar amount of time, resulting in sleepy periods, where no blocks are mined, and active periods, where many blocks are added to the chain at once. In a “hoarding attack", miners can solve a bunch of ILPs and hoard solutions to add many blocks to the chain at once, allowing them to dominate a portion of the chain. If blocks aren't assigned specific ILP problems, then a malicious miner can reuse solutions to old or easier ILPs to mine blocks more quickly than peers. Hashing-based proof-of-work schemes do not suffer this problem because hashes are inherently tied to their corresponding blocks.

One solution to the problems outlined above is for the queue to randomly or deterministically assign ILPs to each miner. We avoid this solution because it allows a malicious queue to add arbitrary blocks to the chain. A malicious miner who control the queue can repeatedly assign itself easy ILPs. In ILPcoin, the failure of the centralized queue can, at most, cause the system to grind to a halt. In the proposed model, the failure of the centralized queue compromises the integrity of the blockchain itself, defeating the purpose of a decentralized currency system. This model begins to veer away from blockchain and towards a single centralized node that pays worker nodes to solve ILPs.

## 7 RELATED WORK

The design of ILPcoin is closely based on the design of Bitcoin, as codified in Satoshi Nakamoto's original whitepaper [6] and in the Bitcoin wiki [11]. There has been much previous work on useful, less wasteful proof-of-X protocols to replace the hash puzzle. Some systems use proof-of-stake [12, 13], where mining power is proportional to the stake, as measured in currency, that the miner has in the system. This system has known vulnerabilities, including the nothing-at-stake attack, where miners extend every fork because doing so is computationally cheap [14]. Alternatively, a system can use proof-of-capacity, where miners must store a portion of a large file in order to mine blocks [15, 16]. Intel SGX's [17] trusted hardware also enables proof-of-time protocols [18], where a signed timestamp is used to guarantee that some time passes between block mining.

ILPcoin focuses on a problem that is hard to solve but easy to verify, where hard and easy are defined in terms of computational work. There are other blockchain variants inspired by this specific idea. FoldingCoin and Gridcoin [19, 20] require miners to solve a portion of a protein folding problem in order to mine a block. Deep learning problems as proof of work is another active area of research [21, 22], much of which is theoretical in nature. Complexity theorists have also contributed blockchain systems that use provably hard problems as proof of work [9, 10], but generating such problems randomly and on-the-fly is an open question. As far we know, ILPcoin is the first blockchain powered by Integer Linear Programming. Whether that is a step forward or backward for humanity is up for debate.

## 8 CONCLUSION

As a prototype of a blockchain with a useful proof-of-work scheme, ILPcoin demonstrates some of the obstacles to utilizing the massive amount of computational power wasted in traditional proof-of-work systems. Any problem for which a solution can be found in a better-than-random fashion will give some outsized advantage to faster computers relative to the miner's share of the overall network computing power. Unfortunately, most useful problems do have such efficient but not polynomial time algorithms. While a hybrid proof-of-work system mitigates this effect, it does not eliminate the advantage and it introduces some wasted work. We demonstrated this delicate balance with our theoretical results in Section 3.

Our implementation of this system and discussion demonstrates some of the issues with having centralized components in a decentralized blockchain, as well as measures for mitigating these effects. For example, a centralized queue is important for keeping the solving of ILPs synchronized across the network, but it should not be involved in verification lest it become a single point of failure in recording the blockchain. Our primary contributions include the implementation of a decentralized useful proof-of-work blockchain, as well as our hybrid proof-of-work theoretical results.

## REFERENCES

[1] Raynor de Best. Daily bitcoin transactions 2017-2021, Apr 2021.

[2] Cristina Criddle. Bitcoin consumes 'more electricity than argentina', Feb 2021.

[3] Túlio A. M. Toffolo and Haroldo G. Santos. Python-mip.

[4] Genesis block. *Bitcoin Wiki*, Mar 2021.

[5] Ralph C. Merkle. A digital signature based on a conventional encryption function. In *A Conference on the Theory and Applications of Cryptographic Techniques on Advances in Cryptology*, CRYPTO '87, page 369–378, Berlin, Heidelberg, 1987. Springer-Verlag.

[6] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Cryptography Mailing list at https://metzdowd.com*, 03 2009.

[7] Alan Demers, Dan Greene, Carl Hauser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing*, PODC '87, page 1–12, New York, NY, USA, 1987. Association for Computing Machinery.

[8] I. Dinur, G. Kindler, R. Raz, and S. Safra. Approximating cvp to within almost-polynomial factors is np-hard. *Combinatorica*, 23(2):205–243, April 2003.

[9] Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Vasudevan. *Proofs of Work From Worst-Case Assumptions: 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19–23, 2018, Proceedings, Part I*, pages 789–819. 01 2018.

[10] Marshall Ball, Dana Dachman-Soled, Mukul Kulkarni, and Tal Malkin. Non-malleable codes from average-case hardness: Ac0, decision trees, and streaming space-bounded tampering. Cryptology ePrint Archive, Report 2017/1061, 2017. https://eprint.iacr.org/2017/1061.

[11] Bitcoin wiki. *Bitcoin Wiki*, Feb 2021.

[12] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. Cryptology ePrint Archive, Report 2016/889, 2016. https://eprint.iacr.org/2016/889.

[13] Phil Daian, Rafael Pass, and Elaine Shi. *Snow White: Robustly Reconfigurable Consensus and Applications to Provably Secure Proof of Stake*, pages 23–41. 09 2019.

[14] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains, 2017.

[15] Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy*, pages 475–490, 2014.

[16] Sunoo Park, Albert Kwon, Georg Fuchsbauer, Peter Gaži, Joël Alwen, and Krzysztof Pietrzak. *SpaceMint: A Cryptocurrency Based on Proofs of Space*, pages 480–499. 12 2018.

[17] Victor Costan and S. Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016:86, 2016.

[18] Hyperledger sawtooth. 2020.

[19] FoldingCoin Inc. Waste not, want not: Your computer could help cure cancer. Jan 2018.

[20] Gridcoin white paper: The computation power of a blockchain driving science  data analysis.

[21] Andrei Lihu, Jincheng Du, Igor Barjaktarevic, Patrick Gerzanics, and Mark Harvilla. A proof of useful work for artificial intelligence on the blockchain, 2020.

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