# MAD-DAG: Protecting Blockchain Consensus from MEV

Roi Bar-Zur
Technion
Israel

Aviv Tamar
Technion
Israel

Ittay Eyal
Technion
Israel

## Abstract

Blockchain security is threatened by *selfish mining*, where a *miner* (operator) deviates from the protocol to increase their revenue. Selfish mining is exacerbated by adverse conditions: *rushing* (network propagation advantage for the selfish miner), *varying block rewards* due to block contents, called *miner extractable value* (MEV), and *petty-compliant* miners who accept bribes from the selfish miner.

The state-of-the-art selfish-mining-resistant blockchain protocol, *Colordag*, does not treat these adverse conditions and was proven secure only when its latency is impractically high.

We present *MAD-DAG*, Mutual-Assured-Destruction Directed-Acyclic-Graph, the first practical protocol to counter selfish mining under adverse conditions. MAD-DAG achieves this thanks to its novel ledger function, which discards the contents of equal-length chains competing to be the longest.

We analyze selfish mining in both Colordag and MAD-DAG by modeling a rational miner using a Markov Decision Process (MDP). We obtain a tractable model for both by developing conservative reward rules that favor the selfish miner to yield an upper bound on selfish mining revenue. To the best of our knowledge, this is the first tractable model of selfish mining in a practical DAG-based blockchain. This enables us to obtain a lower bound on the security threshold, the minimum fraction of computational power a miner needs in order to profit from selfish mining.

MAD-DAG withstands adverse conditions under which Colordag and Bitcoin fail, while otherwise maintaining comparable security. For example, with petty-compliant miners and high levels of block reward variability, MAD-DAG's security threshold ranges from 11% to 31%, whereas both Colordag and Bitcoin achieve 0% for all levels.

## 1 Introduction

Cryptocurrencies such as Bitcoin [41] have become increasingly popular, with a market capitalization surpassing $3.5 trillion as of 2025 [2]. At their core lie *blockchains*, decentralized protocols for maintaining an append-only ledger of blocks, each containing the hash of the previous block and a list of *transactions*, transfers of assets among users and more elaborate logic.

In *Proof of Work* (PoW) blockchains such as Bitcoin [41], Kaspa [53], Litecoin [3], and Bitcoin Cash [1], *miners* add new blocks by expending computational power to solve cryptographic puzzles [21] and broadcasting the solutions. In *Nakamoto Consensus* (NC), the protocol underlying Bitcoin [41], miners extend the *longest chain*[1] they observe. For each block in the longest chain, miners receive two rewards: a *subsidy* of newly created coins and the *transaction fees* paid by users whose transactions are included.

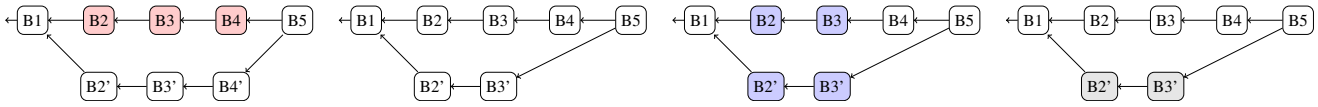
Previous work (§2) has identified that NC is vulnerable to *selfish mining* [8, 23, 28, 30, 42, 46, 64], where a miner controlling sufficient computational power can withhold blocks to build a private chain, then release it while it is longest, causing other miners' blocks to be discarded due to no longer being in the longest chain. By doing so, the selfish miner increases their revenue. The attack can be strengthened due to *rushing*, where a well-connected miner propagates blocks faster to win ties, and due to *petty-compliant miners* [9, 14], who can be bribed to *mine on* (extend) the selfish miner's chain during ties. These adverse conditions reduce the *security threshold* (the minimum power needed for profitable selfish mining) from 25% [46, 64] to as low as 0% [9, 23, 46].

Even without rushing or petty compliance, security degrades under *varying block rewards*. These arise from *whale transactions* with exceptionally high fees [8, 38, 47] or *Maximal Extractable Value (MEV)*, where miners reorder or insert transactions to capture opportunities such as arbitrage [18, 44]. For Bitcoin in particular, as block subsidies halve every 4 years, miner revenue increasingly depends on varying fees [41]. In addition, projects like BitVM [6, 35] that aim to enable smart contracts on Bitcoin will further increase reward variability.

FruitChains [43] and Colordag [4] mitigate selfish mining under idealized assumptions: limited rushing, no petty-compliant miners, and constant block rewards. Neither protocol was analyzed under adverse conditions that break these assumptions. But even under idealized assumptions, FruitChains suffers from a selfish mining strategy that is always slightly profitable; and Colordag's analysis showed its security threshold approaches 50%, but only under a parameter choice that also yields impractically high *protocol latency*, that is, the delay before a block is unlikely to be discarded or denied subsidy (Colordag employs a mechanism that denies the subsidy of a subset of blocks to disincentivize selfish mining). This leaves an open challenge to design a protocol that strictly disincentivizes selfish mining under adverse conditions and with practical latency.

To this end, we introduce MAD-DAG (§3), a DAG-based blockchain protocol with practical latency that, compared to both Colordag and NC, better resists selfish mining, especially under the aforementioned adverse conditions. MAD-DAG achieves this thanks to its novel *ledger function*. A blockchain's ledger function determines the content of the *ledger* (the transactions that are considered valid) given all the observed blocks and their contents. The *MAD (Mutual Assured Destruction)* ledger function takes a novel approach by completely discarding the content of blocks on the longest chain that have another chain with the same length (red blocks, Figure 1a). This ensures that the miner does not receive any transactions fees or MEV. This approach is in contrast to both NC and Colordag, which simply take the content of all blocks in the longest chain. The MAD ledger function severely hinders a selfish miner's ability to create an alternative chain, since they would need it to be strictly longer, as ties yield contentless blocks. This negates any advantage gained from rushing or from bribing petty-compliant miners.

[1] The chain that most work was expended on.




**(a) MAD ledger function, equal length chains** **(b) MAD ledger function, different length chains** **(c) Colordag's reward mechanism, $N_\ell \geq 5$** **(d) Colordag's reward mechanism, $N_\ell < 5$**

**Figure 1: Colordag's reward mechanism and MAD-DAG's ledger function. Red blocks are *destructed*. Blue blocks are *contested*. Gray blocks are *unacceptable*.**

To disincentivize selfish mining due to transaction fees, it could have sufficed to burn transactions fees and still consider the transactions themselves as valid. But, this approach does not work for MEV, which is transparent to the consensus protocol, and a stronger measure is necessary, motivating our choice of the MAD ledger function.

Our MAD ledger function is also incentive compatible to petty-compliant miners: If a selfish miner attempts to bribe a petty-compliant miner to mine on their chain that has equal length to the longest chain, the petty-compliant miner can point to both chains, causing their contents to be discarded and enabling the petty-compliant miner to use the content in their own block.

To evaluate MAD-DAG's security, we conduct the first analysis of selfish mining in DAG-based blockchains under adverse conditions (§4). We analyze both Colordag and MAD-DAG by modeling the mining process from the viewpoint of a single rational miner maximizing their revenue using a *Markov Decision Process (MDP)*. We solve each MDP using the Probabilistic Termination Optimization (PTO) method [64] with dynamic programming [11].

To obtain a tractable model of the protocols, we upper-bound the selfish miner's revenue by slightly modifying the protocols' subsidy mechanisms. Colordag's subsidy mechanism, which we use in MAD-DAG as well, rewards only blocks on the longest chain that are *uncontested*, meaning there are no *competing blocks* at identical heights (distance from root) on other *acceptable* chains (non-blue blocks, Figure 1c). To determine which chains are considered *acceptable*, Colordag measures their symmetric difference from the longest chain (the set of blocks unique to each chain). Chains differing by more than a parameter $N_\ell$ are considered not acceptable and are discarded before determining uncontested blocks (gray blocks, Figure 1d). For analysis, we use a modified subsidy mechanism where the selfish miner's blocks are always acceptable, and others' blocks are acceptable only under a stricter condition. This simplification leads to many states being equivalent, resulting in a significantly smaller state space.

Our simplified approach allows us to analyze both Colordag and MAD-DAG under practical latency by setting $N_\ell$ to low values. This is the first time Colordag's security is analyzed under practical latency, whereas previous analysis [4] showed analytically that there exists a choice of $N_\ell$ that yields a high security threshold, but this choice quickly becomes impractically high.

We first focus on the effect of the *difficulty-adjustment mechanism* on the security of the protocols (§5). The difficulty-adjustment mechanism in PoW blockchains periodically adjusts the difficulty of the PoW puzzle to maintain a stable block creation rate, even as the total mining power on the network fluctuates. MAD-DAG's difficulty-adjustment mechanism is identical to NC's; the difficulty is adjusted once a fixed number of blocks have been added to the longest chain. In contrast, Colordag employs a similar mechanism that only considers the blocks that were rewarded to ensure that the rate of subsidy is stable. This choice, however, slightly reduces the impact of the subsidy penalty, as unrewarded blocks are also not counted by the difficulty-adjustment mechanism, resulting in a slight decrease in difficulty.

MAD-DAG's difficulty-adjustment mechanism increases the security threshold from 25% in NC and 34% in Colordag to 39% in MAD-DAG, in the presence of a rushing adversary and in a practical setting where $N_\ell = 25$. We further observe that when the subsidy is significantly lower than transaction fees, Colordag's security threshold drops significantly lower than NC's, while MAD-DAG's drops but to a level that is still higher than NC's.

Next, we compare the impact of the ledger function on the security of the protocols (§6). While both Colordag and MAD-DAG's security thresholds decrease as the level of block reward variability is present, MAD-DAG's ledger function is more robust. When petty-compliant miners are present, Colordag's security threshold drops to 0% at a mild level of block reward variability while MAD-DAG's security threshold remains unaffected by their presence and remains above 11% even with extreme block reward variability.

In conclusion (§7), our contributions are as follows:

- MAD-DAG, a protocol with a novel ledger function that discards the content of equal-length chains competing to be the longest chain;
- MDP of selfish mining in Colordag and MAD-DAG;
- conservative reward rules that upper-bound selfish mining revenue in Colordag and MAD-DAG, leading to the first tractable MDP of selfish mining in DAG-based blockchains; and
- analysis of Colordag and MAD-DAG with practical latency and adverse conditions, showing MAD-DAG is the most secure protocol to date against selfish mining.

As Bitcoin continues to evolve, it becomes increasingly likely that its block rewards will become more volatile, underscoring the need for a more secure protocol that can withstand adverse conditions. MAD-DAG is directly applicable to address this looming threat.

## 2 Related Work

We begin by describing why we focus on PoW blockchains (§2.1), and then present previous work on selfish mining and on mitigations for it under various conditions (§2.2). We then describe how

analyzing selfish mining is done (§2.3) and, last, survey the use of DAG structures in other protocols (§2.4).

## 2.1 Proof-of-Work Blockchains

While PoW has been criticized for energy consumption [20], leading to alternatives like Proof of Stake [10, 34] and Proof of Space [16, 22], Bitcoin's continued market dominance [2] demonstrates PoW's enduring relevance. There is ongoing research to make PoW more efficient [36, 40, 59]. We focus on addressing its vulnerability to selfish mining.

## 2.2 Selfish Mining and Mitigations

*2.2.1 Without Adverse Conditions.* Previous work analyzed selfish mining strategies in Nakamoto consensus in simplified settings (such as constant block rewards) [23, 24, 28, 29, 42, 46]. Keller [32] also analyzes selfish mining in DAG-based PoW consensus protocols in simplified settings. In contrast, we consider realistic adverse conditions, namely rushing, varying block rewards and petty-compliant miners.

Several works propose mitigations for selfish mining. FruitChains [43] provides upper bounds on selfish mining revenue but suffers from zero-risk slightly profitable deviations. Other proposals like Bobtail [12] and StrongChain [56] that modify PoW mechanisms, or the DAG-based Tailstorm [33], lack comprehensive analysis of general selfish mining strategies. Sarenche et al. [49] suggested changing the difficulty-adjustment mechanism to consider blocks that are not in the longest chain, but this does not address petty-compliant miners who will then be incentivized to ignore these blocks to negate the effect. Colordag [4] was theoretically proven to resist selfish mining when its latency is impractically high and there are no adverse conditions. In contrast to all these works, MAD-DAG mitigates selfish mining under adverse conditions and with practical latency.

*2.2.2 Rushing.* Well-connected miners can perform rushing attacks by quickly propagating their blocks in the network, e.g., through relay networks [17, 27]. With perfect connectivity, the security threshold of NC drops to 0%.

Random tie-breaking was proposed [23] to mitigate rushing's impact in NC. While this reduces the advantage of a well-connected selfish miner, it slightly lowers the threshold against other adversaries [46]. MAD-DAG's ledger function provides inherent resistance to rushing without compromising security against weakly connected miners.

*2.2.3 Reward Variability.* Block rewards vary significantly due to transaction fees and MEV [18, 44]. Carlsten et al. [14] and Tsabary and Eyal [57] showed that when there are only transaction fees, deviating from the protocol becomes profitable for any miner regardless of their mining power. WeRLman [8, 47] analyzed a middle ground with whale transactions alongside subsidy, showing that reward variability increases selfish mining profitability and decreases the security threshold. These motivate our design of MAD-DAG, which disincentivizes selfish mining even when rewards vary.

Fee smoothing mechanisms [13, 43] redistribute fees across multiple blocks to reduce variability. However, these cannot address MEV, which is not generally detectable, and they incentivize alternative payment channels [58] to bypass the mechanism. In contrast, MAD-DAG's ledger function addresses fee variability and MEV without incentivizing alternative payment channels.

Unlike fee smoothing mechanisms, alternative transaction fee mechanisms [15, 45] address incentive compatibility in the context of a single block; they do not consider their impact on selfish mining strategies.

*2.2.4 Petty-Compliant Miners.* Carlsten et al. [14] introduced petty-compliant miners who make undetectable deviations when profitable. For instance, petty-compliant miners facing ties in the longest chain would choose the chain that leaves out the transactions with the highest fees. Such behavior opens an avenue of implicit bribing, where a selfish miner deliberately leaves out transactions to incentivize petty-compliant miners to mine on their chain. Carlsten et al. [14] showed that when there is no subsidy, petty-compliant miners cause the security threshold to drop to 0%. Later work [9, 48] extended the analysis to include the impact of subsidy, and showed that while the security threshold is positive, petty-compliant miners significantly lower it.

Yaish et al. [62] discovered strategic timestamp manipulation in Ethereum, corroborating the possibility of petty-compliant miners.

To our knowledge, MAD-DAG is the first protocol to address vulnerability to selfish mining due to petty-compliant miners.

## 2.3 Analyzing Selfish Mining

Sapirshtein et al. [46] developed a method to find the optimal selfish mining strategy by solving a sequence of related MDPs. Bar-Zur et al. [64] developed an alternative method, Probabilistic Termination Optimization (PTO), which requires solving a single MDP. Both used *dynamic programming* to optimize the MDP.

Dynamic programming is intractable for large models, leading subsequent works to use *deep reinforcement learning* [8, 9, 30]. However, deep reinforcement learning does not guarantee convergence to the optimal policy, providing only a lower bound on revenue, leading to an upper bound on the security threshold.

In this paper, we bound the selfish miner's revenue from above to obtain a tractable model that can be solved using dynamic programming, guaranteeing optimality and yielding a lower bound on the security threshold. This allows us to analyze both Colordag and MAD-DAG, and to prove MAD-DAG indeed resists selfish mining under adverse conditions.

## 2.4 DAG-Based Protocols

DAG structures have been extensively explored in consensus protocols [61]. Previous works consider a PoW blockchain in a setting where the rate of block creation is high relative to network delay, causing frequent *forks* (two blocks pointing to the same previous block). These suggested using a DAG structure to protect the blockchain from *double-spending* (reversing confirmed transactions) [7, 25, 54, 63]. Other works consider a similar setting and use a DAG structure to allow blocks which are not in the longest chain to also contain transactions, allowing the ledger to grow faster [37, 51–53]. While these works focus on improving performance or security, they do not analyze strategic mining behavior such as selfish mining.

DAGs also appear in fundamentally different consensus protocols not based on mining. There are distributed protocols in which all participants are known and which are designed for Byzantine fault tolerance [19, 31, 39, 55]. A notable example is IOTA [50], which has users create blocks containing their own transactions rather than relying on miners. These protocols use DAGs as a tool to reach consensus rather than to align participants' incentives, as we do.

## 3 MAD-DAG

MAD-DAG is a PoW blockchain protocol with a DAG structure. Each block contains a list of transactions, and one or more pointers to previous blocks, making it a DAG. Each block also contains a solution to a PoW puzzle: a number (*nonce*) such that the hash of its concatenation with the block's content is smaller than a target value. This target is called the *difficulty*.

*Honest* miners follow the protocol, which prescribes the following. A miner should create a new block with transactions that are not yet included in existing blocks and pointers to all leaves in the block DAG (i.e., blocks that are not referenced by any other block). The miner should then attempt to find a valid solution to the PoW puzzle. If the miner finds a solution, they should immediately publish the new block. Upon receiving a block, honest miners should add it to their local copy of the block DAG and attempt to mine on it and on all other remaining leaves.

Transaction and block propagation occur via an underlying peer-to-peer communication network. We assume standard blockchain networking solutions and leave the network layer out of our design.

As in Colordag [4], a block that references both some other block and its ancestor at the same time is not valid.

MAD-DAG comprises three key mechanisms, which we describe next.

*Difficulty-Adjustment Mechanism.* A difficulty-adjustment mechanism determines how to update the PoW difficulty, and subsequently the block-creation rate.

MAD-DAG's difficulty-adjustment mechanism adjusts the difficulty of the Proof-of-Work puzzle every *epoch*, that is, after a fixed number of blocks is added to the longest chain. The difficulty is multiplied by the ratio between the desired epoch time and the time the last epoch took. This maintains a stable block-creation rate even as the total mining power on the network fluctuates.

This mechanism is also used in NC [41]. In Bitcoin, on average, every 2 weeks, 2016 blocks are added to the longest chain, for example.

*Subsidy Mechanism.* A subsidy mechanism determines which blocks are rewarded with subsidy.

We follow Colordag's subsidy mechanism, which rewards only a subset of blocks. The *canonical chain* is the longest chain in the DAG. In case of a tie, a protocol-defined tie-breaking rule determines the canonical chain. We consider 2 variants of MAD-DAG with different tie-breaking rules: miners either have to pick the first chain they see or pick one randomly.

A block is considered *acceptable* if it is in a chain with a symmetric difference from the canonical chain of at most $N_\ell$ blocks (non-gray blocks, Figure 1c and Figure 1d). We term this threshold the *fork sensitivity*. A block is considered *uncontested* if it is the only block with its height (maximum distance from the root) that is acceptable (white blocks are uncontested and blue blocks and contested). MAD-DAG's subsidy mechanism rewards only uncontested blocks on the canonical chain.

*Ledger Function.* A ledger function determines which blocks compose the ledger, namely, the ordered list of transactions that are considered valid.

MAD-DAG uses the novel MAD ledger function. A block is considered *destructed* if there are two chains with equal length to the canonical chain where one chain contains the block and the other does not. The MAD ledger function considers the contents of only non-destructed blocks in the canonical chain.

When two competing chains following a fork have equal length, a new block pointing to both chains may be created, causing all previous blocks since the fork to become destructed (the new block is not destructed). Since these blocks are destructed, the all transactions within these blocks are discarded, rolling back the state of ledger to its state before these blocks. Note that transactions that were previously added to the destructed blocks may once again be added to the ledger starting from the new block.

## 4 Analyzing Selfish Mining

We begin with preliminaries on Markov Decision Processes, selfish mining in an MDP and in NC (§4.1). We then present selfish mining models for both Colordag and MAD-DAG, starting with the full model (§4.2), followed by a simplified model that bounds the selfish mining revenue from above (§4.3), and afterward we describe how we solve the MDP (§4.4).

### 4.1 Preliminaries

*4.1.1 Markov Decision Processes.* A *Markov Decision Process (MDP)* is an iterative process where starting from state $s_0$, in each step $t$ an *agent* takes an *action* $a_t$ in a *state* $s_t$, receives a *reward* $R_t$ and causes the state to stochastically transition to a new state $s_{t+1}$. Formally, an MDP is a tuple $(S, A, P, R)$ where:

(1) $S$ is a set of states $s \in S$;
(2) $A$ is the set of actions $a \in A$;
(3) $P : S \times A \times S \to [0, 1]$ is the transition probability function from a state $s$ to a state $s'$ given an action $a$: $P(s, a, s') \in [0, 1]$ such that $\sum_{s' \in S} P(s, a, s') = 1$; and
(4) $R : S \times A \times S \to \mathbb{R}$ is the reward function for a transition from state $s$ to a state $s'$ given an action $a$: $R(s, a, s') \in \mathbb{R}$ such that $R_t = R(s_t, a_t, s_{t+1})$.

An MDP is solved by finding a policy $\pi : S \to A$ that maximizes the expected *utility* $u(\pi)$. The *stochastic shortest path* is a special case of an MDP, where there is a terminal state $s_T \in S$ such that no further actions can cause the state to transition to another state and no rewards are received in the terminal state [11]. A common utility function in this case is the expected sum of the rewards received until termination time $T$ (the first step in which the terminal state is reached):

$$u(\pi) = \mathbb{E}\left[\sum_{t=0}^{T} R_t\right]. \tag{1}$$

Two popular approaches for solving MDPs are *dynamic programming*, which includes *value iteration* and *policy iteration* [11],

and converges to the optimal policy; and *deep reinforcement learning* [5, 8, 30], which utilizes machine learning to allow larger state and actions spaces, but is not guaranteed to converge to the optimal policy.

*4.1.2 Selfish Mining in an MDP.* Honest miners follow the protocol. For example, in NC, they mine on the last block in the longest chain they observe and reveal their blocks immediately. A selfish miner may strategically choose which block to mine on and when to reveal their blocks.

Consider the following scenario that illustrates the attack where the block reward is constant. A selfish miner mines a block and keeps it secret while attempting to mine another block extending it. If the selfish miner succeeds, they possess a secret chain of two blocks. Meanwhile, honest miners continue mining on the longest chain they know, and eventually one of them mines a new block. The selfish miner then reveals their secret two-block chain, causing the honest miner's new block to be discarded while the selfish miner claims the rewards for both blocks in their chain.

This strategy does not increase the total rewards the selfish miner receives in this instance, as they still earn rewards for two blocks just as they would have by mining honestly. The benefit arises from the wasted effort of honest miners, in combination with the difficulty-adjustment mechanism. When this scenario repeats, fewer blocks are added to the longest chain within the epoch's target time period, causing the protocol to perceive a lower network block-creation rate. Consequently, the difficulty decreases for subsequent epochs, enabling faster block creation that increases the selfish miner's revenue over time. Hence, the utility function for a selfish miner is the rate at which rewards are received.

An MDP model for selfish mining needs to account for the difficulty-adjustment mechanism [23, 46, 64]. To analyze selfish mining in an MDP, Bar-Zur et al. [64] suggest adding to the MDP another element: the *difficulty contribution* function $D : S \times A \times S \to \mathbb{R}^+$ that maps a transition from state $s$ to state $s'$ given an action $a$ to how much the transition contributed towards the difficulty-adjustment mechanism such that $D_t = D(s_t, a_t, s_{t+1})$. The utility function of the agent would then be expected ratio between the total reward and the total difficulty contribution:

$$u(\pi) = \mathbb{E}\left[\frac{\sum_{t=0}^{\infty} R_t}{\sum_{t=0}^{\infty} D_t}\right]. \tag{2}$$

While this is a non-standard utility function, Bar-Zur et al. [64] show that an MDP with such a utility function can be transformed to a stochastic shortest path MDP whose optimal policy is also with a bounded error to the optimal policy in the original MDP with the non-standard utility function. The stochastic shortest path MDP can then be solved using standard techniques.

## 4.2 Colordag and MAD-DAG Model

We now present models of selfish mining in Colordag and MAD-DAG. For brevity, we present the models in a unified manner, where different variants can be selected to represent the different protocols.

As in previous models [8, 46], we consider a single rational selfish miner and all other miners are honest. The selfish miner has a fraction $\alpha$ of the total mining power in the network, meaning that there is always a probability of $\alpha$ that the selfish miner is the one who creates a new block. The miner has a rushing factor $\gamma$, meaning that when rushing, the selfish miner's block would reach a $\gamma$-fraction of the other miners before the newly created block. Except for the case of rushing, the network delay is zero: When a miner publishes a block, all other miners instantly receive it.

Note that we only analyze the one-color version of Colordag since its multicolor mechanism is aimed to mitigate the effects of *benign* forks (due to non-zero network delay) on the revenue of honest miners, while we assume the network delay is zero.

The subsidy is constant and worth 1 unit. There occasionally appear whale transactions, each worth $F$ units, and each block can contain at most one whale transaction. Unlike previous models, we also assume that there is a constant source of transactions that can be included in every block and net a total of $f$ units. We term $f$ the *guaranteed fee*.

The selfish miner works only on a single secret chain at once, as in previous models [8, 46, 64]. The selfish miner always references their own last block, since this never harms their revenue. But they are allowed to also reference a block on the *public chain*, the chain other miners are working on. Note that they are allowed to reference up to one additional block from there as, as any additional one would result in pointing to both a block and to its ancestor, making the selfish miner's block invalid.

All other miners are honest: They reference all known blocks and immediately publish blocks they create.

The model tracks the state of the selfish miner's and other miners' chains since the *last divergence*, that is, the last block that everyone agrees on whether it is in the canonical chain and whether it is acceptable, uncontested, or neither.

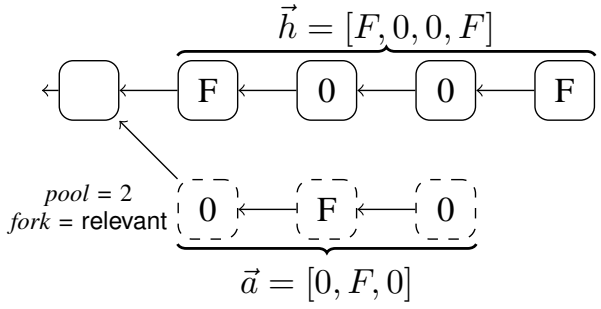
We now present the different variants of the model, followed by its utility function, its states, its actions, and its transitions and their yielded rewards and difficulty contributions (the terms in the denominator of the utility function, see §4.1.2).

*4.2.1 Modes.* To allow us to analyze the effect of the different mechanisms of MAD-DAG and Colordag and of the adverse conditions, we present different variants of the different mechanisms we use. Each combination of the following modes results with model for particular protocol.

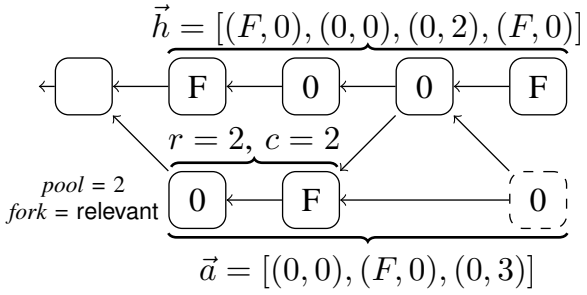
*Tie-Breaking.* The tie-breaking rule specifies how honest miners choose the canonical chain that determines subsidy rewards and transactions in the ledger. Unlike classical blockchains, that only allow blocks to specify a single parent block, determining the canonical chain exactly, a DAG allows blocks to reference multiple parents, but the order in which they are referenced can determine the canonical chain (e.g., by preferring the block referenced first).

We consider three tie-breaking rule, which were analyzed before in the context of NC [23, 46]. We describe these rules in the context of our model, in the case that a selfish miner publishes an alternative chain of equal length to the chain other miners are working on, and they honest miners have to decide which chain to pick.

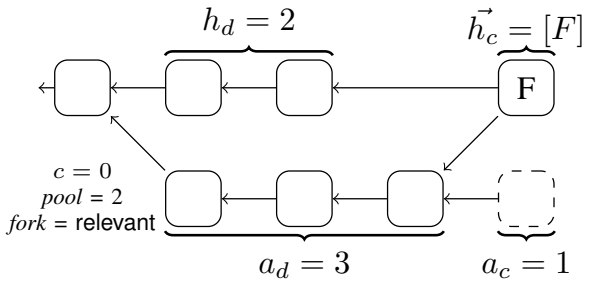
(1) First-heard: Honest miners choose the first chain they hear of. If the selfish miner had a block ready in advance, rushing is possible and the selfish miner's rushing factor determines how many honest miners choose the selfish miner's chain.



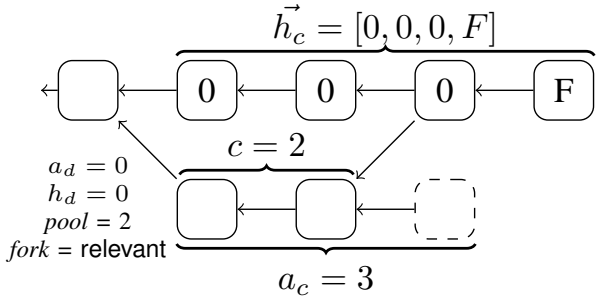

(a) NC model.




(b) Full DAG model.




(c) Upper-bound DAG model with $p$ = TRUE.




(d) Upper-bound DAG model with $p$ = FALSE.

**Figure 2: Example states in the NC and DAG models.**

(2) Random: Honest miners choose a chain uniformly at random. The selfish miner's rushing factor does not affect how many honest miners choose the selfish miner's chain.
(3) Worst-case: All miners choose the selfish miner's chain. This mode provides an upper bound on the revenue of a selfish miner in the case that the miner has extreme rushing capability or that all other miners are petty-compliant.

Note that a protocol can prescribe miners a specific tie-breaking rule like the first-heard rule or the random rule, but the worst-case rule is only used for analysis.

*Difficulty-Adjustment Mechanism.* The difficulty-adjustment mechanism determines how the block-creation rate is updated every epoch. In the context of the model, this determines the difficulty contributions.

We consider the following two difficulty-adjustment mechanisms.

(1) Uncontested: Only uncontested blocks contribute to the difficulty.
(2) Canonical: The difficulty is adjusted based on all blocks in the canonical chain.

*Ledger Function.* The ledger function determines the content of the ledger: which transactions are considered valid and net fees to the miner who included them.

We consider the two following ledger functions.

(1) Canonical: The ledger is the concatenation of all blocks in the canonical chain.
(2) MAD: The ledger is the concatenation of all non-destructed blocks in the canonical chain.

Colordag uses the uncontested difficulty-adjustment mechanism. While it doesn't specify its exact ledger function, the canonical ledger function is a natural choice. MAD-DAG uses the canonical difficulty-adjustment mechanism and the MAD ledger function.

*4.2.2 Utility Function.* Similarly to previous models [46, 64], the utility function is the expected rate of revenue of the selfish miner, the ratio of the total reward from subsidy, constant transactions, and whale transactions received to the total difficulty contribution based on the chosen difficulty-adjustment mechanism.

*4.2.3 States.* A state in the model is defined by $\left(\vec{a}, \vec{h}, \mathit{fork}, r, c, \mathit{pool}\right)$, where $\vec{a}$ is a vector of all the selfish miner's blocks since the last divergence (the last block up to which everyone agrees on the state of all mechanisms), each block contains a flag indicating whether the block contains a whale transaction and a reference (possibly null) to a block in the public chain; $\vec{h}$ is the public chain, a vector of all honest miners' blocks since the last divergence with similar flags for transactions and references; *fork* is a flag indicating whether rushing is possible (relevant) or not (irrelevant); $r$ is the number of the selfish miner blocks that have been published; $c$ is the index of the last selfish miner block in the canonical chain (only when there is ambiguity due to chains of equal length); and *pool* is number whale transactions available since the last divergence.

Figure 2b shows an example state.

*4.2.4 Actions and Transitions.* The actions in the model are as follows.

*Reveal $\ell$.* Reveal the $\ell$-th miner's block ($r \leftarrow \ell$). Only possible if $r < \ell$.

*Mine $\ell$.* The selfish miner attempts mining a block pointing to the last block in their chain and to the $\ell$-th honest block ($\ell = 0$ corresponds to null). The selfish miner succeeds with probability $\alpha$.

Honest miners attempt mining a block pointing to the last block in the public chain and the last revealed selfish miner block (unless it was referenced in a previous honest block or there are no revealed selfish miner blocks). If there are two possible candidates for the canonical chain, honest miners pick one of them based on the tie-breaking rule and the value of *fork*. If they pick the selfish miner's block, we update $c$ accordingly. The honest miners succeed with probability $(1 - \alpha)$.

If the creator of the new block is selfish (respectively, honest), the value of *fork* is updated to irrelevant (relevant).

*Merge.* The selfish miner discards all unrevealed blocks, accepts the current public chain as being the canonical one, and moves on to start a new secret chain extending it. Both chains are reset to be empty ($\vec{a} \leftarrow [\,]$ and $\vec{h} \leftarrow [\,]$). Reward and difficulty contribution are yielded based on the ledger function and the difficulty-adjustment mechanism, respectively, as follows.

First, we calculate the canonical chain based on the DAG structure of $\vec{a}$ and $\vec{h}$, breaking ambiguities due to tie-breaks with $c$. Then, we calculate which blocks are acceptable and which are uncontested, using the Colordag's algorithm [4]. After this is done, we can determine the subsidy the selfish miner receives and the difficulty contribution of the blocks in the canonical chain. Then, if the ledger function is the MAD one, we calculate which blocks are destructed, by finding all chains with equal length to the canonical chain. After this is done, we can determine the reward the selfish miner receives from constant fees for blocks in the canonical chain that the selfish miner created and from whale transactions if these blocks contain whale transactions.

For each possible transition, which results in a new block being added to the longest chain, we add a new transition where a whale transaction is added to the pool and everything else is the same. The new transition has probability $\delta$ times the probability of the original transition. Then, we normalize all the transitions' probabilities to sum to 1.

This approach is different from previous ones [8, 47]. Sarenche et al. [47] argue that simply incrementing the pool, as Bar-Zur et al. [8] do, gives the selfish miner an unfair advantage by allowing them to predict in advance whether a new whale transaction become available until the next block is created. Instead, they suggest to simultaneously add the whale transaction to the pool and to the new block that is being created. This, however, restricts the miner's ability to react to the new whale transaction in time and change the block they are trying to mine. We instead choose to interrupt the transition in the model when a new whale transaction is added to prevent the miner from knowing that a transaction will be available in advance but still give the selfish miner a chance to react to it to change their action.

*4.2.5 Bounding the State Space.* We allow the length of $\vec{a}$ and $\vec{h}$ to be up to a value of $L$. To uphold this, the action Mine is disallowed if the length of either chain is equal to $L$. While this caps the size of the state space, the model remains intractable for useful values of $L$. For example, there are roughly 100,000,000 when $L$ is 5. To overcome this, we next present a model that uses an upper bound on the revenue of the selfish miner.

*4.2.6 Utility of Honest Mining.* If the selfish miner were to mine honestly, they would get a fraction of $\alpha$ of all blocks. Each block is worth $1 + f + qF$, where $q$ is the average number of whale transactions in a block. Overall the utility of honest mining is then $\alpha(1 + f + qF)$.

In previous models with fees [8, 9, 47], we simply have $q = \delta$, as whenever a new whale transaction appears an honest miner would immediately include it. But using the new approach to update *pool*, results with a positive possibility that more than one whale transaction appears before a new block is created. Combined with the fact that transactions may overflow if there are more than *max_pool* simultaneously, it is no longer guaranteed that all whale transactions that appear will be included.

In our model, it holds that $q = \frac{\delta - \delta^{max\_pool+1}}{1-\delta^{max\_pool+1}}$. We defer the proof to Appendix A.

We corroborate our results by simulating the honest policy and comparing the simulated revenue to the one we calculate with our closed-form solution.

## 4.3 Upper-Bound Model

The upper-bound model is based on the full model presented in the previous section with several modifications that benefit the selfish miner.

First, we assume the selfish miner always includes as many whale transactions as they can, even if their blocks have been created before the transactions appeared. Thanks to this, we do not need to keep track of the content of the selfish miner's chain and can only track its length.

Second, if the ledger function is the MAD one, when the selfish miner has at least as many blocks in their secret chain as the public chain, we allow the selfish miner to cause the blocks in the public chain to be destructed without the selfish miner having to reveal a chain of equal length to the public chain. This only harms the revenue of honest miners and leaves more whale transactions to the selfish miner.

Third, we use a different notion for blocks to be acceptable: A selfish miner's block is always acceptable; an honest miner's block is acceptable if there is a path that contains it that has a symmetric difference from the canonical chain of at most $N_\ell$ blocks, and both the block's closest ancestor and closest descendant on the canonical chain were created by an honest miner. The selfish miner having more acceptable blocks reduces the number of uncontested honest miners' blocks, and consequently benefit the selfish miner when the uncontested difficulty-adjustment mechanism is used. The honest miners having less acceptable blocks increases the number of uncontested blocks the selfish miner has, and consequently benefit the selfish miner by increasing the reward they get. While this also increases the difficulty contribution when the uncontested

difficulty-adjustment mechanism is used, this overall revenue increases ($\frac{a+x}{b+x} > \frac{a}{b}$ when $b > a > 0$ and $x > 0$). Thanks to the different notion we use, there is no longer any benefit for the selfish miner to reference blocks from the public chain, unless they are in the canonical chain (to allow the selfish miner to start a new secret chain) and in that case, it does not affect the acceptability of honest miners' blocks, and cannot lead to the selfish miner's blocks being penalized. In addition, there is no longer any benefit for honest miners to reference blocks from the selfish miner's chain as any such chain would not count for the new notion of acceptability.

*4.3.1 States.* As in the previous model, states track all blocks since the last divergence. In the new model, we do so in a more structured manner (Figure 2c). The state tracks the blocks since the fork between the public chain $\vec{h_c}$ (including flags for whale transactions) and the selfish miner's secret chain $a_c$. With the canonical ledger function, the fork is also the last fork in the DAG, and these are all blocks that may end up in the canonical chain and may not. With the MAD ledger function, this is not the case, $\vec{h_c}$ and $a_c$ also track blocks for which it is known whether they are in the canonical chain or not, but it is not yet determined whether they are destructed or not. Therefore, in this model, the element $c$, the index of the last selfish miner block out of the $a_c$ blocks in the secret chain, is only relevant for the MAD ledger function.

In addition, we track blocks that have been created prior to the fork between the public chain and the selfish miner's secret chain but after the last divergence (termed *pre-fork* blocks). These are blocks that everyone agrees whether they are part of the canonical chain, but it is not yet clear whether they are acceptable, uncontested, or neither. No such block created by an honest miner can be in the canonical chain, because then it would be acceptable and everyone can determine whether it is uncontested (whether there is a selfish miner's block of the same height, as these are always acceptable), contradicting the way we selected these blocks. In fact, the only case where such blocks may exist is when the selfish miner's blocks are in the canonical chain and the honest miners' blocks are not, and it is not yet clear whether the honest miners' blocks are acceptable. In this case, they would become acceptable if and only if the current public chain would become the canonical chain.

Denote the number of pre-fork blocks that the selfish miner created by $a_d$ and the number of pre-fork blocks that the honest miners created by $h_d$. If there are more than $N_\ell$ such blocks ($a_d + h_d > N_\ell$), then the honest miners' blocks become unacceptable, as the only relevant path that contains these blocks has a symmetric difference to the canonical chain that is too large. In that case, the selfish miner's blocks are uncontested and can be rewarded. This also means that the divergence has been resolved (only partially, the canonical chain is not yet known). These blocks are no longer relevant for the state.

However, at this stage, until an honest miner's block is added to the canonical chain, all honest miners' blocks that are not part of the canonical chain would also be unacceptable. To track this, we introduce a new flag, the *honest acceptability potential* $p$, that indicates whether we are in this situation that honest miners' block out of the canonical chain have no potential to become acceptable (Figure 2d).

Summarizing, a state is defined by $\left(a_d, a_c, h_d, \vec{h_c}, \mathit{fork}, c, \mathit{pool}, p\right)$, where $a_d$ is the number of pre-fork blocks that the selfish miner created; $a_c$ is the number of blocks the selfish miner created in their secret chain; $h_d$ is the number of pre-fork blocks that the honest miners created; $\vec{h_c}$ is a vector of all blocks in the public chain, the honest miners' blocks since the fork the selfish miner created (including flags for whale transactions); *fork* is a ternary flag indicating whether rushing is possible (relevant) or not (irrelevant) or currently occurring (active); $c$ is the index of the last selfish miner block (with respect to $a_c$) that is in the canonical chain; *pool* is the number of whale transactions available since the fork between the public chain and the selfish miner's secret chain; and $p$ is a flag indicating whether there is potential for all honest miners' blocks that are not part of the canonical chain to become acceptable.

We bound the state space similarly to how we do so in the full model.

*4.3.2 Actions and Transitions.* The following actions are available.

*Adopt $\ell$.* The selfish miner discards their secret chain and adopts the first $\ell$ blocks in $\vec{h_c}$. This is only allowed if $\ell \geq a_c$ (otherwise, the selfish miner should reveal their secret chain instead).

With the MAD ledger function, if $c > 0$ and the number of blocks until the last fork in the DAG ($a_d + c + h_d + c$) is greater than $N_\ell$ then the honest miners' blocks that were discarded are not acceptable, making the selfish miner's blocks uncontested, yielding a reward of $a_d + c$. Otherwise, the honest blocks are acceptable, yielding a reward of $a_d - h_d$.

The difficulty contribution is based on the chosen difficulty-adjustment mechanism. If it is the uncontested difficulty-adjustment mechanism, a difficulty contribution of the reward plus $\ell - a_c$ is yielded. Otherwise, $a_d + \ell$ is yielded.

The public chain is shifted back by $\ell$ blocks ($\vec{h_c} \leftarrow \vec{h_c} \ll \ell$) and the transactions that were in the discarded blocks are decremented from *pool* ($\mathit{pool} \leftarrow \mathit{pool} - \#T\left(\vec{h_c}[:\ell]\right)$). The selfish miner's secret chain and all other values are reset ($a_c \leftarrow 0$, $a_d \leftarrow 0$, $h_d \leftarrow 0$, $p \leftarrow$ TRUE).

*Reveal $\ell$.* The selfish miner reveals the first $\ell$ blocks in their secret chain. This is only allowed if $\ell \geq \left|\vec{h_c}\right|$.

If the selfish miner reveals a strictly longer chain ($\ell > \left|\vec{h_c}\right|$), those blocks are shifted to be pre-fork ($a_d \leftarrow a_d + \ell$, $a_c \leftarrow a_c - \ell$), and the public chain is also shifted to be pre-fork ($h_d \leftarrow h_d + \left|\vec{h_c}\right|$, $\vec{h_c} \leftarrow [\,]$). If $p =$ FALSE (honest miners' blocks that are not part of the canonical chain cannot become acceptable) or if the number of pre-fork blocks (after the shift) is more than $N_\ell$, then the selfish miner's pre-fork blocks are uncontested, causing $p \leftarrow$ FALSE and yielding a reward and difficulty contribution of $a_d$. An additional reward based on the transactions in the revealed $\ell$ blocks is also yielded, and whale transactions that were in the revealed blocks are decremented from *pool* ($\mathit{pool} \leftarrow \mathit{pool} - \#T\left(\vec{h_c}[:\ell]\right)$). The values of $c$ and *fork* are reset to 0 and irrelevant, respectively.

If the selfish miner reveals a chain equal in length to the public chain ($\ell = \left|\vec{h_c}\right|$), one of the following cases occurs. With the canonical ledger function, if the tie-breaking rule is worst-case, then the

same as the previous case occurs. If the tie-breaking rule is different, *fork* is updated to active.

With the MAD ledger function, if the tie-breaking rule is worst-case, then the selfish miner's chain becomes the canonical chain ($c \leftarrow \left|\vec{h}_c\right|$) and the blocks in the public chain are destructed ($\#T\left(\vec{h}_c\right) \leftarrow 0$). If the tie-breaking rule is different, if there are transactions in the public chain ($\#T\left(\vec{h}_c\right) > 0$), then the blocks in the public chain are destructed ($\#T\left(\vec{h}_c\right) \leftarrow 0$) and nothing else happens (the blocks are not actually revealed); otherwise, *fork* is updated to active.

*Mine.* The selfish miner attempts mining a block pointing to the last block in their chain. Honest miner attempt mining a block pointing to the last block in the public chain, and if *fork* is active, also to the block in the selfish miner's chain with the same height.

The selfish miner succeeds ($a_c \leftarrow a_c + 1$) with probability $\alpha$. In that case, *fork* is updated to irrelevant.

If *fork* is not active, the honest miners create a new block pointing to the last block in the public chain and include a transaction if there is one available ($\vec{h_c} \leftarrow \vec{h_c}$++). They succeed with probability $1 - \alpha$.

Otherwise, if *fork* is active, there are two possible transitions. First, with probability $(1 - \gamma)(1 - \alpha)$, an honest miner creates a block pointing to the block in the public chain first, indicating they prefer it to be in the canonical chain ($\vec{h_c} \leftarrow \vec{h_c}$++).

With probability $\gamma(1 - \alpha)$, an honest miner creates a block pointing to the block in the selfish miner's block first. The selfish miner's first $\left|\vec{h}_c\right|$ blocks enter the canonical chain. If the ledger function is the canonical one, the state is updated in the exact same manner as if the miner revealed a strictly longer chain with length $\left|\vec{h}_c\right|$ (as in the Reveal $\ell$ action). If the ledger function is the MAD one, then only $c$ is updated to $\left|\vec{h}_c\right|$.

The pool is updated similarly to the full model, resulting in an identical formula for the utility of honest mining.

## 4.4 Solving the MDP

To solve all MDPs, we use PTO [64] to transform the MDP to a stochastic shortest path problem with expected horizon $10^5$, and then solve the resultant MDP with policy iteration [11] with a precision of $10^{-5}$. Using policy iteration, a dynamic programming algorithm, guarantees convergence to a near-optimal policy, allowing us to accurately calculate the security threshold.

Note that this is the first work to analyze NC with varying rewards with dynamic programming, and to obtain an accurate security threshold for NC. This was made possible by several practical improvements over previous work.

First, we trim the state space by removing infeasible states. For example, a state where the public chain has more transactions than *pool* is infeasible. We also group states that are equivalent. For example, the value of *fork* does not matter if the selfish miner has fewer blocks than the public chain.

Second, we use a more memory-efficient linear solver in each step of the policy iteration. Policy iteration is an iterative algorithm where in each step, given the current policy, we compute the future utility of starting from each state by solving a linear system of equations, and then update the policy to pick the action that maximizes the future utility. Instead of using a direct sparse linear solver as in previous work [64], we use an iterative solver, Biconjugate gradient method [26], which requires less memory, a critical difference when the state space is large. This solver sometimes does not converge to a solution. When that happens we use another variant, Bi-conjugate gradient stabilized method [60].

Third, in most cases, we restrict *max_pool* to 2. This way, the number of possible chains of a certain length grows quadratically instead of exponentially. We later verify that this restriction has little effect on the results for the cases consider ($\delta = 0.01$), but intuitively, it is simply highly unlikely that 3 whale transactions would be available at the same time.

Nevertheless, memory requirements remain significant; in all our experiments, we utilize for a server with a large amount of RAM (2 terabytes).

To find the security threshold, we simply perform a binary search to find the minimum $\alpha$ required for profitable selfish mining. In each step, we numerically calculate the utility of the output policy and compare it with the revenue of an honest miner with the current $\alpha$ value.

# 5 Difficulty-Adjustment Mechanism

We begin by analyzing the effect of the difficulty-adjustment mechanism. We compare (1) NC, (2) Colordag, which uses the uncontested difficulty-adjustment mechanism, and (3) *Canonical-DAG*, a protocol similar to Colordag that uses the canonical difficulty-adjustment mechanism instead. All of these protocols use the canonical ledger function.

Canonical-DAG serves as a stepping stone to indirectly compare Colordag and MAD-DAG while isolating the effects of the different mechanisms on the protocol. First, in this section, we compare Colordag and Canonical-DAG to show the latter is more secure due to the use of the canonical difficulty-adjustment mechanism. Then, in the next section, we will compare Canonical-DAG and MAD-DAG to show the latter is more secure due to the use of the MAD ledger function, and consequently, is also more secure compared to Colordag.

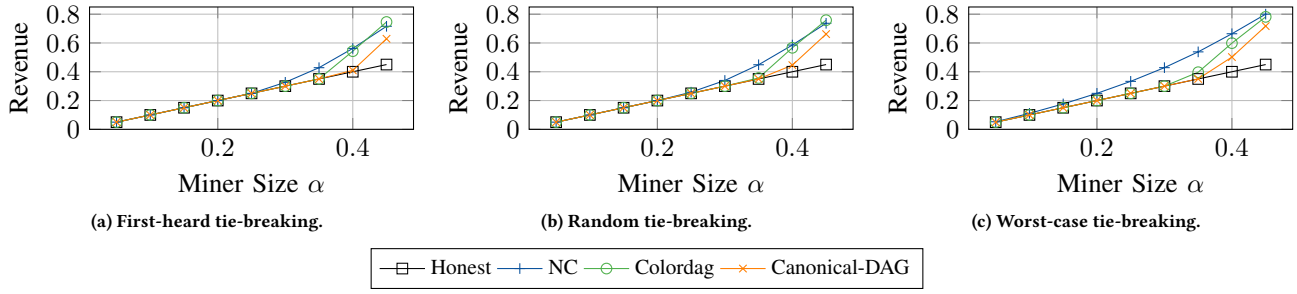
We analyze Colordag and Canonical-DAG using the upper-bound model (§4.3). We analyze NC using a model similar to that of Bar-Zur et al. [8] (Appendix B), but with the same method for updating the *pool* that we use in the new models.

While our method only allows us to compare bounds on the security threshold, it is still extremely valuable: These are the first bounds provided under adverse conditions, and a higher lower-bound on the security threshold is preferable.

Since we focus on the difficulty-adjustment mechanism in this section, we ignore whale transactions and set $\delta = 0$.

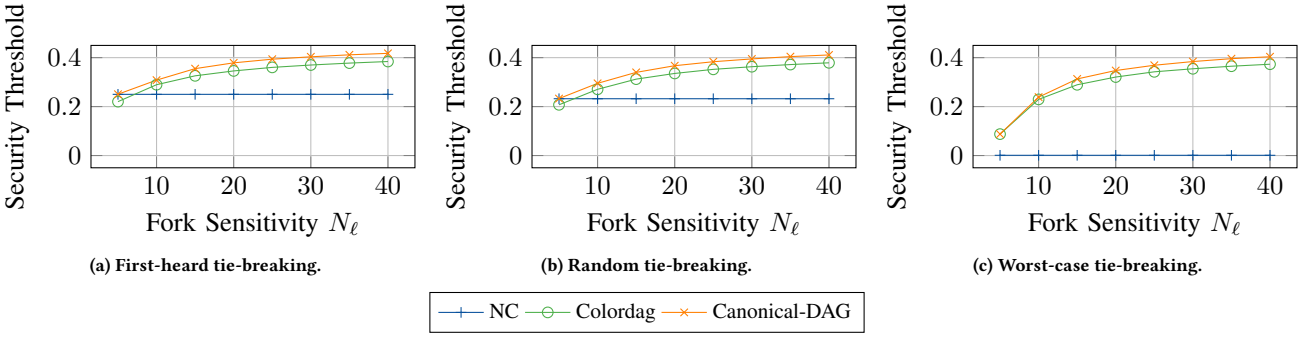
All comparisons are done for all three tie-breaking rules. For the first-heard tie-breaking rule, we use a rushing factor $\gamma$ of 0.5, as often done in previous work [8, 30, 46].

*Revenue.* We first analyze the revenue of a selfish miner in NC, Colordag, and Canonical-DAG as a function of the selfish miner's size $\alpha$ (Figure 3). We set $N_\ell$ to 15 and $L$ to 10 blocks. We plot the revenue for each of the three tie-breaking rules.



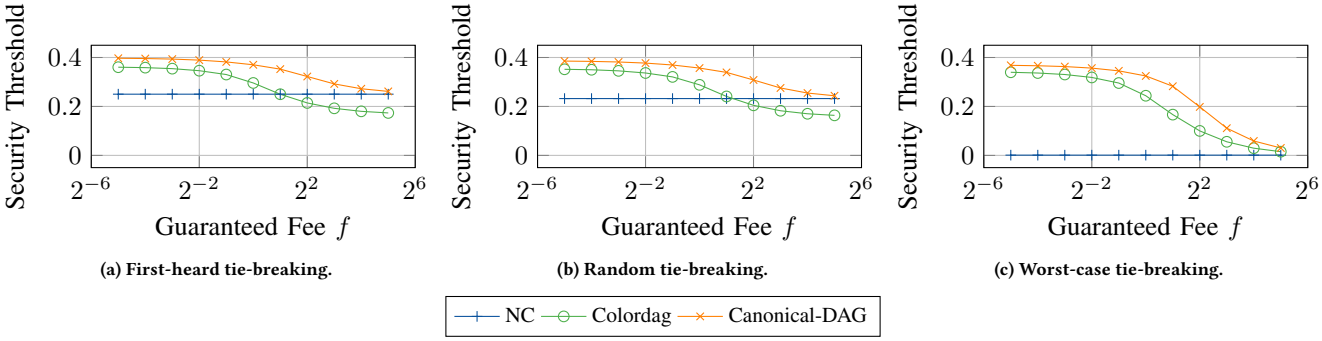

(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 3: Selfish mining revenue in NC, Colordag, and MAD-DAG.**




(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 4: Security threshold of NC, Colordag, and MAD-DAG as a function of fork sensitivity.**




(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 5: Security threshold of NC, Colordag, and MAD-DAG as a function of guaranteed fee $f$.**

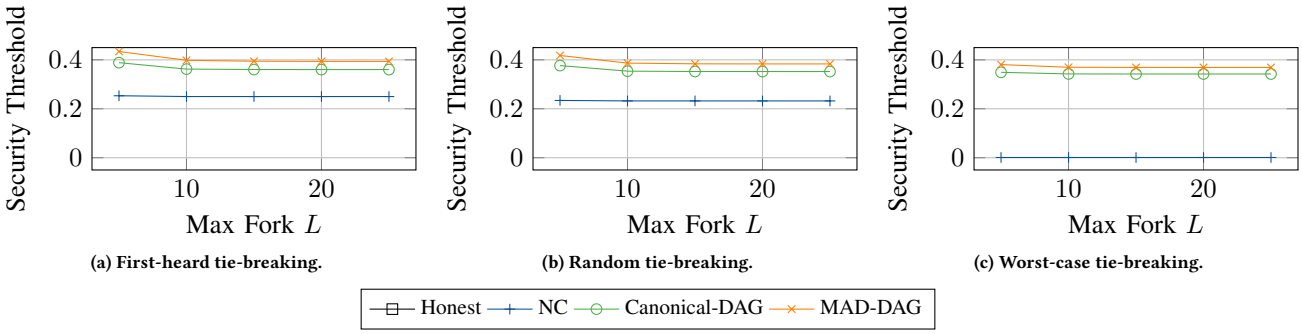
We observe that a selfish miner in Canonical-DAG obtains less revenue compared to both NC and Colordag. This experiment also demonstrates that the selfish miner's revenue is equal to their *fair share* (how much could be obtained by mining honestly) until a critical point (the security threshold), after which selfish mining yields more revenue than honest mining.

*Fork Sensitivity.* We focus next on calculating the security threshold rather than only revenue. We compare the security threshold of the three protocols as a function of $N_\ell$ (Figure 4). In this experiment, we set $L$ to 20.

The security threshold of both Colordag and Canonical-DAG increases with $N_\ell$. This is in line with the analysis of Colordag [4]. While low values of $N_\ell$ result with Colordag having a lower security threshold than NC for the first-heard and random tie-breaking rules, this is quickly reversed for higher values of $N_\ell$, as Colordag's subsidy mechanism becomes more effective as $N_\ell$ increases.

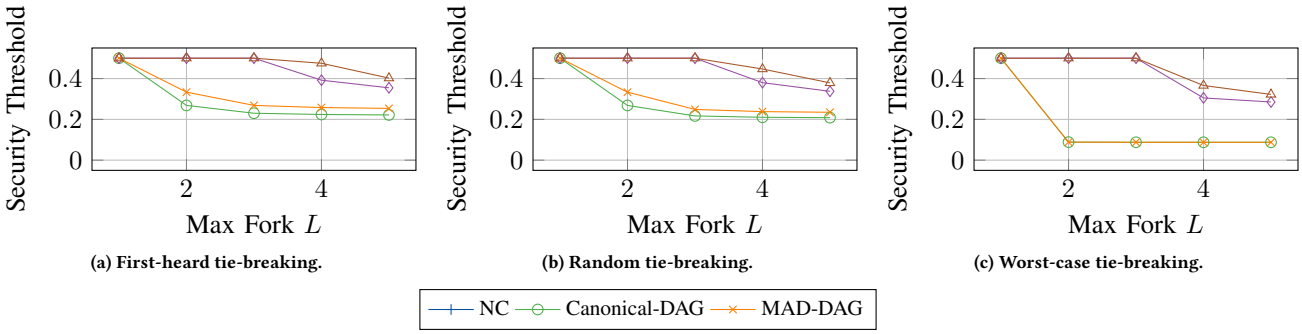
Canonical-DAG has a higher security threshold compared to NC and Colordag for all tie-breaking rules and all values of $N_\ell$.

*Guaranteed fee.* Next, we analyze how the guaranteed fee $f$ (a baseline level of fees that all blocks in the canonical chain receive)



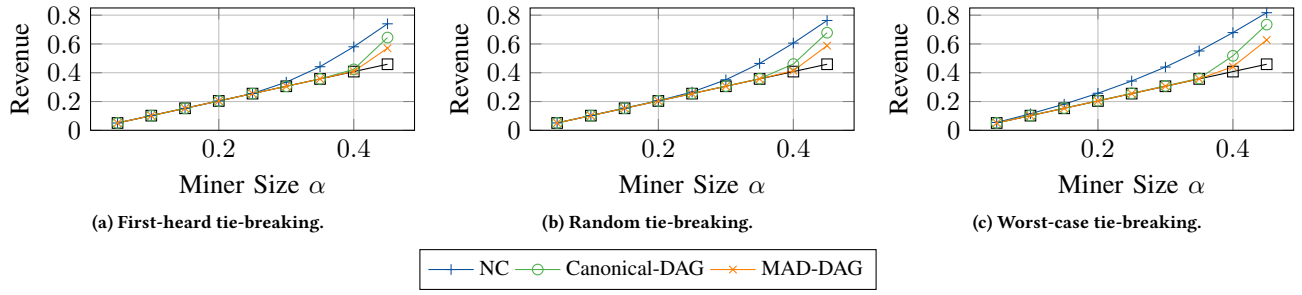

(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 6: Security threshold of NC, Colordag, and MAD-DAG as a function of maximum fork length $L$.**




(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 7: Comparison of security threshold, Colordag, and MAD-DAG in the simplified model and the full model as a function of maximum fork length $L$.**




(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 8: Selfish mining revenue in NC, Colordag, and MAD-DAG with whale transactions.**

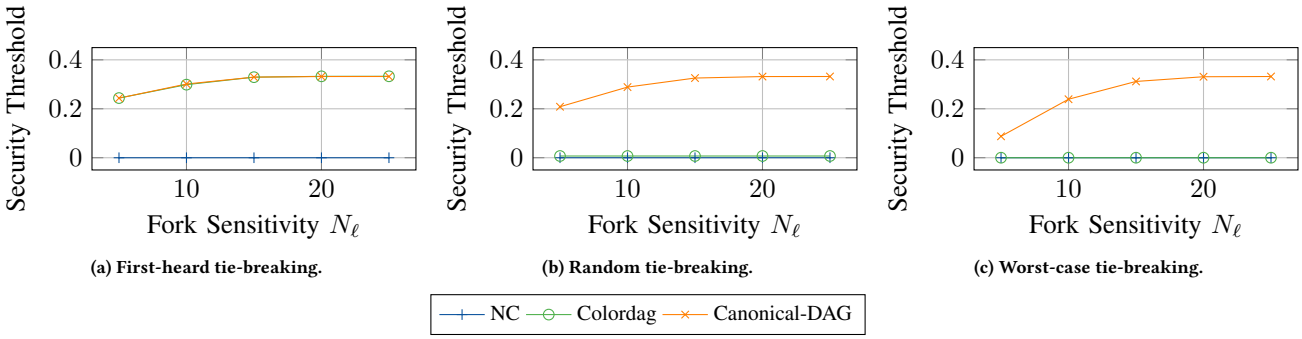
affects the security threshold (Figure 5). We set $N_\ell$ to 25 and $L$ to 10.

Again, Canonical-DAG has a higher security threshold compared to NC and Colordag in all cases.

Colordag, however, has a lower security threshold compared to NC when $f$ is high. Increasing $f$ shifts the revenue of miners from subsidy to the fees they obtain by creating blocks in the canonical chain regardless of whether they are contested or not. This results in a reward structure similar to NC. However, in Colordag, a selfish miner can still benefit from uncontested blocks due to them not contributing to the difficulty-adjustment mechanism, lowering the block creation difficulty and consequently increasing the rate of block creation.

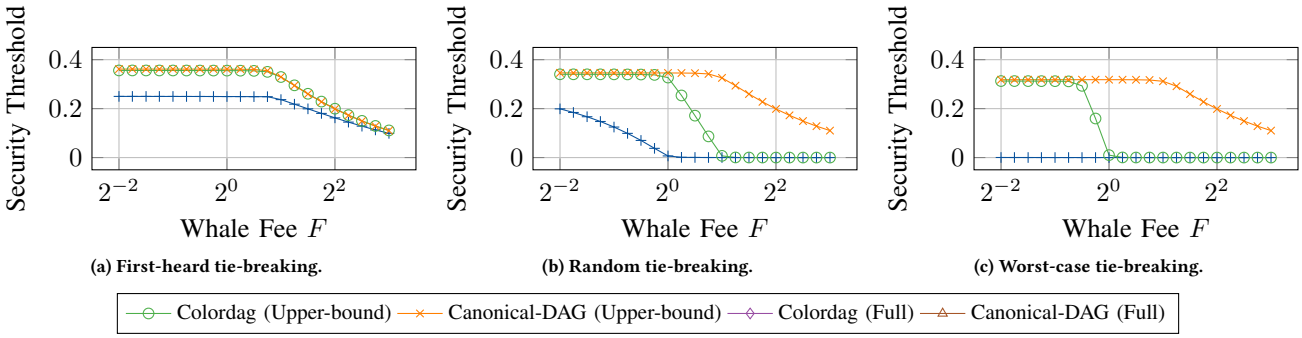
In NC, the selfish miner objective is simply the original one (the fraction of their blocks in the canonical chain) multiplied by a constant. Thus, this the guaranteed fee does not affect on the security threshold.

While the security threshold of Canonical-DAG decreases as $f$ increases, similarly to Colordag, its security threshold approaches



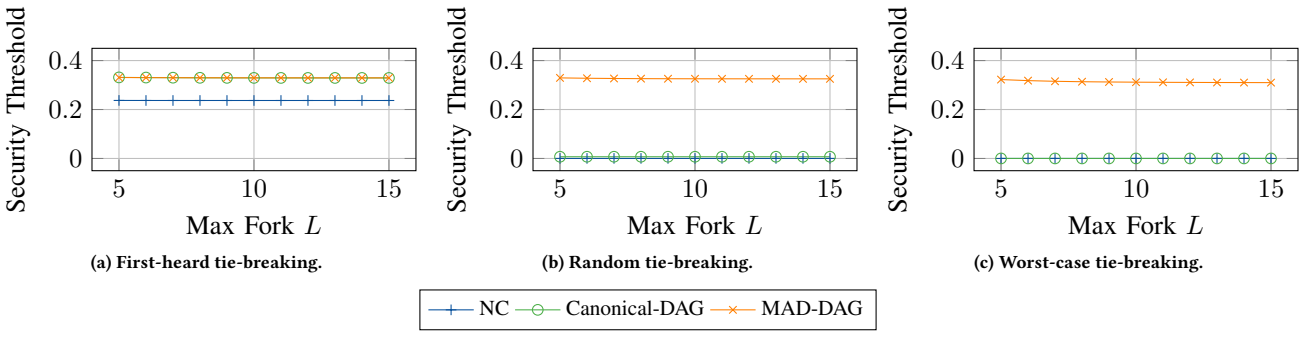

(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 9: Security threshold of NC, Canonical-DAG, and MAD-DAG as a function of fork sensitivity.**




(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 10: Security threshold of NC, Canonical-DAG, and MAD-DAG as a function of whale fee $F$.**




(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 11: Security threshold of NC, Canonical-DAG, and MAD-DAG as a function of maximum fork length $L$.**

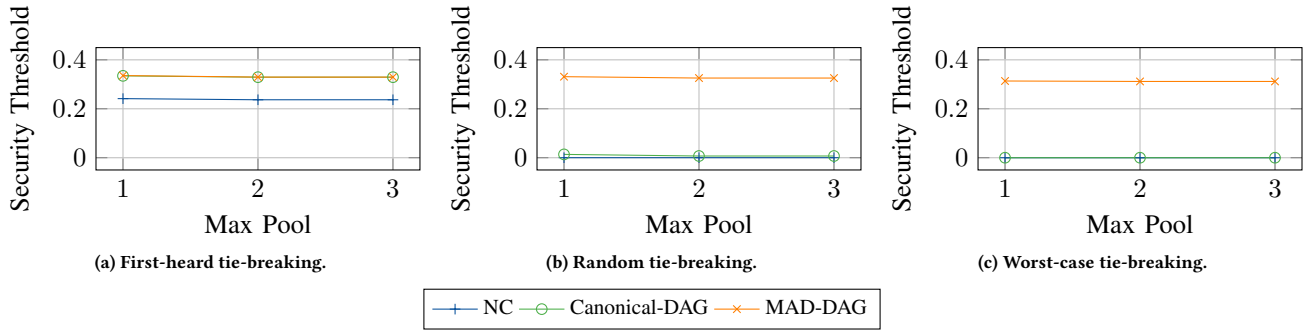
NC's as $f$ increases due to the fact that it has the same difficulty-adjustment mechanism as NC.

*State Space Restriction.* Next, we analyze how the maximum fork length $L$ affects the security threshold (Figure 6). We set $N_\ell$ to 25.

For all protocols, increasing $L$ reduces the security threshold. This effect becomes negligible when $L$ is 10 or more.

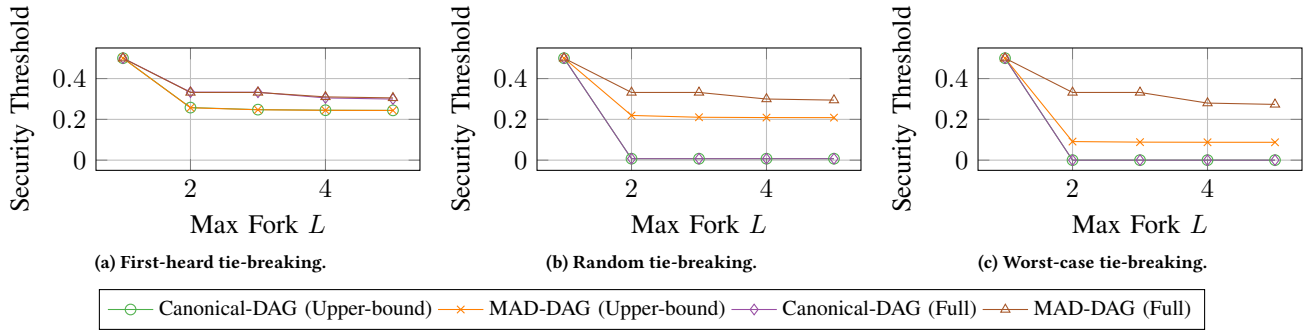
*Comparison to Full model.* Last, we compare the results between the full model and the upper-bound model for both Colordag and Canonical-DAG (Figure 7). We vary $L$ from 1 to 5, as the full model is more complex and requires too much memory for higher values. We also set $N_\ell$ to 5.

As expected, across all protocols and tie-breaking rules, increasing $L$ reduces the security threshold, as it allows more elaborate strategies for the selfish miner. In addition, the results corroborate the correctness of the upper-bound model: It has a lower security threshold compared to the full model for both protocols, which is



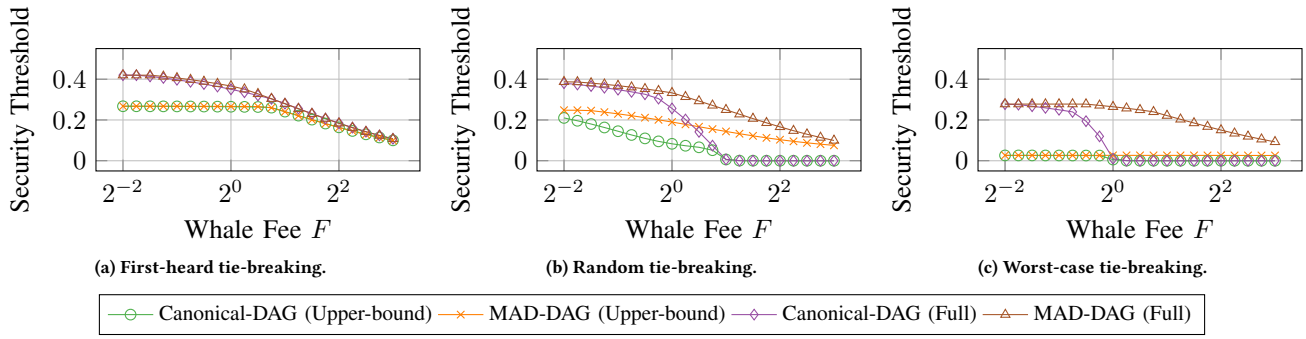

(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 12: Security threshold of NC, Canonical-DAG, and MAD-DAG as a function of maximum pool size.**




(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 13: Comparison of security threshold, Canonical-DAG, and MAD-DAG in the simplified model and the full model as a function of maximum fork length $L$.**




(a) First-heard tie-breaking. (b) Random tie-breaking. (c) Worst-case tie-breaking.

**Figure 14: Comparison of security threshold, Canonical-DAG, and MAD-DAG in the simplified model and the full model as a function of whale fee $F$.**

expected since it uses an upper bound on the revenue of the selfish miner.

The full model analysis of Canonical-DAG also yields a lower security threshold compared to Colordag, similarly to the upper-bound model.

## 6 Ledger Function

Having established that the Canonical-DAG, which uses the canonical difficulty-adjustment mechanism, is more secure than Colordag, which uses the uncontested difficulty-adjustment mechanism, we move on to analyze the impact of the ledger function. To isolate the impact of the ledger function, we compare in this section the security of NC, Canonical-DAG, and MAD-DAG. Showing that

MAD-DAG is more secure that Canonical-DAG would lead us to conclude that it is more secure than Colordag as well.

For understanding the impact of the ledger function, in this section, we consider whale transactions and set the whale frequency $\delta$ to 0.01, the whale transaction fee $F$ to 2, and the maximum number of concurrent pending whale transactions *max_pool* to 2 unless stated otherwise.

We analyze Canonical-DAG and MAD-DAG with the upper-bound model (§4.3), this time with the canonical difficulty-adjustment mechanism mode. And again, we analyze NC using a model similar to the one by Bar-Zur et al. [8] (Appendix B).

Again, we use $\gamma = 0.5$ for the first-heard tie-breaking rule.

We also fix $N_\ell = 15$ and $L = 10$ unless otherwise stated.

*Revenue.* We first analyze the revenue of a selfish miner in NC, Canonical-DAG, and MAD-DAG as a function of the selfish miner's size $\alpha$ (Figure 8).

For all miner sizes $\alpha$ and tie-breaking rules we consider, a selfish miner can get more revenue in NC compared to Canonical-DAG and more revenue in Canonical-DAG compared to MAD-DAG.

We again, move on to compare the security of the three protocols.

*Fork sensitivity.* We study the security threshold as a function of $N_\ell$ (Figure 9). In line with the previous experiment, in all cases we consider, Canonical-DAG has a higher security threshold compared to NC and MAD-DAG has a higher security threshold compared to Canonical-DAG.

With the random and worst-case tie-breaking rules, the security threshold of Canonical-DAG is near-zero for all values of $N_\ell$, while the security threshold of MAD-DAG is significantly higher, above 30% for $N_\ell \geq 15$.

*Whale transactions.* Next, we compare the security threshold of the three protocols as a function of the whale transaction fee $F$ (Figure 10). We again observe that the security threshold is highest for MAD-DAG, followed by Canonical-DAG, and lowest for NC.

In all protocols, the security threshold decreases when $F$ increases. This is in line with previous work that shows high whale fees increase selfish mining profitability [8, 9].

With the first-heard tie-breaking rule (with $\gamma = 0.5$), the security thresholds of Canonical-DAG and MAD-DAG are similar, meaning the MAD ledger function has little effect in this case. With the random and worst-case tie-breaking rules, the security threshold of Canonical-DAG drops significantly starting from $F = 1$ and $F = 0.7$, to near-0 at $F = 2$ and $F = 1$, respectively. This is because in these cases, when an honest miner creates a block with a whale transaction, it is more profitable for the selfish miner to fight for the whale transaction by attempting to create and to publish an alternative chain with a single block holding that whale transaction. MAD-DAG's security threshold is largely unaffected by the choice of the tie-breaking rule and while it decreases for high values of $F$, it remains significantly higher than the alternatives: 19.9% when $F = 4$ and 11% when $F = 8$. This is because the previously described strategy does not work when the MAD ledger function is employed.

*State Space Restriction.* We next analyze how the maximum fork length $L$ affects the security threshold (Figure 11). As before, increasing $L$ reduces the security threshold of all protocols, but the threshold remains virtually the same when $L \geq 10$.

We also analyze how the maximum pool size *max_pool* affects the security threshold (Figure 12). In all protocols, increasing *max_pool* from 1 to 2 slightly reduces the security threshold, and increasing it further from 2 to 3 has a negligible effect. This is since whale transactions are rare ($\delta = 0.01$) and are unlikely to be simultaneously available.

*Comparison to Full model.* Last, we compare the differences in security threshold between the upper-bound model and the full model for Canonical-DAG and MAD-DAG.

We first analyze the effect of the maximum fork length $L$ on the security threshold (Figure 13). Since the full model is more complex, we set $N_\ell = 5$ and vary $L$ from 1 to 5.

As in the previous comparison to the full model, the fact that for both protocols the security threshold in the upper-bound model is lower compared to the full model corroborates the correctness of the upper-bound model. The security threshold decreases again when $L$ increases, as expected.

While with the first-heard tie-breaking rule, there is little difference between the two protocols, with the random and worst-case tie-breaking rules, the security threshold of MAD-DAG is significantly higher than Canonical-DAG in both models.

The large difference between the threshold of MAD-DAG in the full model and the upper-bound model suggests that MAD-DAG may be significantly more secure than our analysis can guarantee.

Next, we analyze the effect of the whale fee $F$ on the security threshold (Figure 14). This time, we set $L = 3$ and $N_\ell = 3$.

Similar findings to the previous experiment emerge here: The security threshold decreases when $F$ increases; with the first-heard tie-breaking rule the two protocols have similar security thresholds, while with the other rules, MAD-DAG is significantly more secure; and there is a large difference between the security threshold of MAD-DAG that the upper-bound model predicts to the one the full model predicts, especially when $F$ is low, meaning that the upper-bound model is not tight and can perhaps be improved in future work.

## 7 Conclusion

We present MAD-DAG, the first practical DAG-based blockchain protocol that disincentivizes selfish mining under adverse conditions including rushing attacks, varying block rewards, and petty-compliant miners. This is achieved through the novel MAD ledger function that discards content from equal-length competing chains. We conduct the first tractable MDP-based analysis of selfish mining in DAG-based blockchains using a novel upper-bound model that conservatively favors the selfish miner.

Our analysis demonstrates that MAD-DAG's security threshold consistently matches or exceeds Colordag's and NC's. Critically, MAD-DAG retains its security against well-connected selfish miners and petty-compliant miners, while selfish mining becomes profitable in both NC and Colordag for miners of any size.

Furthermore, MAD-DAG achieves these improvements with fork sensitivity as low as 15, enabling practical deployment unlike Colordag which requires impractically high values.

As Bitcoin's block subsidies halve every four years and projects like BitVM introduce reward variability, Bitcoin faces escalating

vulnerability to selfish mining. MAD-DAG provides the most robust defense available.

## Acknowledgments

This work was supported by research grants from the Avalanche foundation, and IC3, the Initiative for CryptoCurrencies and Contracts. This work also received funding from the European Union (ERC, Bayes-RL, Project Number 101041250). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

## References

[1] [n. d.]. Bitcoin Cash - Peer-to-Peer Electronic Cash. https://bitcoincash.org/. Accessed: 2024-04-25.
[2] [n. d.]. Cryptocurrency Prices, Charts And Market Capitalizations | CoinMarketCap. https://coinmarketcap.com/. Accessed: 2025-11-11.
[3] [n. d.]. Litecoin - Open source P2P digital currency. https://litecoin.org/. Accessed: 2024-04-25.
[4] Ittai Abraham, Danny Dolev, Ittay Eyal, and Joseph Y Halpern. 2023. Colordag: An incentive-compatible blockchain. *arXiv preprint arXiv:2308.11379* (2023).
[5] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. Deep reinforcement learning: A brief survey. *IEEE signal processing magazine* 34, 6 (2017), 26–38.
[6] Lukas Aumayr, Zeta Avarikioti, Robin Linus, Matteo Maffei, Andrea Pelosi, Christos Stefo, and Alexei Zamyatin. 2024. BitVM: Quasi-Turing Complete Computation on Bitcoin. *Cryptology ePrint Archive* (2024).
[7] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. 2019. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 585–602.
[8] Roi Bar-Zur, Ameer Abu-Hanna, Ittay Eyal, and Aviv Tamar. 2022. WeRLman: to tackle whale (transactions), go deep (RL). In *Proceedings of the 15th ACM International Conference on Systems and Storage*. 148–148.
[9] Roi Bar-Zur, Danielle Dori, Sharon Vardi, Ittay Eyal, and Aviv Tamar. 2023. Deep Bribe: Predicting the Rise of Bribery in Blockchain Mining with Deep RL. *Cryptology ePrint Archive* (2023).
[10] Iddo Bentov, Ariel Gabizon, and Alex Mizrahi. 2016. Cryptocurrencies without proof of work. In *International conference on financial cryptography and data security*. Springer, 142–157.
[11] Dimitri Bertsekas. 2012. *Dynamic programming and optimal control: Volume I*. Vol. 4. Athena scientific.
[12] George Bissias and Brian Neil Levine. 2020. Bobtail: Improved Blockchain Security with Low-Variance Mining.. In *NDSS*.
[13] Rastislav Budinsky, Ivana Stančíková, and Ivan Homoliak. 2023. Mitigating Undercutting Attacks: Fee-Redistribution Smart Contracts for Transaction-Fee-Based Regime of Blockchains with the Longest Chain Rule. In *2023 IEEE International Conference on Blockchain (Blockchain)*. IEEE, 25–32.
[14] Miles Carlsten, Harry Kalodner, S Matthew Weinberg, and Arvind Narayanan. 2016. On the instability of bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 154–167.
[15] Hao Chung and Elaine Shi. 2023. Foundations of transaction fee mechanism design. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. SIAM, 3856–3899.
[16] Bram Cohen and Krzysztof Pietrzak. 2019. The chia network blockchain. *White Paper, Chia. net* 9 (2019).
[17] Matt Corallo. [n. d.]. Bitcoin relay network. https://bitcoinrelaynetwork.org/. Accessed: 2024-04-25.
[18] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. 2020. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE symposium on security and privacy (SP)*. IEEE, 910–927.
[19] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*. 34–50.
[20] Alex De Vries. 2019. Renewable energy will not solve bitcoin's sustainability problem. *Joule* 3, 4 (2019), 893–898.
[21] Cynthia Dwork and Moni Naor. 1992. Pricing via processing or combatting junk mail. In *Annual international cryptology conference*. Springer, 139–147.
[22] Stefan Dziembowski, Sebastian Faust, Vladimir Kolmogorov, and Krzysztof Pietrzak. 2015. Proofs of space. In *Annual Cryptology Conference*. Springer, 585–605.
[23] Ittay Eyal and Emin Gün Sirer. 2018. Majority is not enough: Bitcoin mining is vulnerable. *Commun. ACM* 61, 7 (2018), 95–102.
[24] Colin Finkbeiner, Mohamed E Najd, Julia Guskind, and Ghada Almashaqbeh. 2025. SoK: Time to be Selfless?! Demystifying the Landscape of Selfish Mining Strategies and Models. *Cryptology ePrint Archive* (2025).
[25] Matthias Fitzi, Peter Ga, Aggelos Kiayias, and Alexander Russell. 2018. Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition. *Cryptology ePrint Archive* (2018).
[26] Roger Fletcher. 2006. Conjugate gradient methods for indefinite systems. In *Numerical Analysis: Proceedings of the Dundee Conference on Numerical Analysis, 1975*. Springer, 73–89.
[27] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert Van Renesse, and Emin Gün Sirer. 2018. Decentralization in bitcoin and ethereum networks. In *Financial Cryptography and Data Security: 22nd International Conference, FC 2018, Nieuwpoort, Curaçao, February 26–March 2, 2018, Revised Selected Papers 22*. Springer, 439–457.
[28] Arthur Gervais, Ghassan O Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. 2016. On the security and performance of proof of work blockchains. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 3–16.
[29] Cyril Grunspan and Ricardo Pérez-Marco. 2018. On profitability of selfish mining. *arXiv preprint arXiv:1805.08281* (2018).
[30] Charlie Hou, Mingxun Zhou, Yan Ji, Phil Daian, Florian Tramer, Giulia Fanti, and Ari Juels. 2019. SquirRL: Automating attack analysis on blockchain incentive mechanisms with deep reinforcement learning. *arXiv preprint arXiv:1912.01798* (2019).
[31] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All you need is dag. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 165–175.
[32] Patrik Keller. 2025. Automated Selfish Mining Analysis for DAG-based PoW Consensus Protocols. *arXiv preprint arXiv:2501.10888* (2025).
[33] Patrik Keller, Ben Glickenhaus, George Bissias, and Gregory Griffith. 2023. Tailstorm: A secure and fair blockchain for cash transactions. *arXiv preprint arXiv:2306.12206* (2023).
[34] Sunny King and Scott Nadal. 2012. Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. *self-published paper, August* 19, 1 (2012).
[35] Victor I Kolobov, Avihu M Levy, and Moni Naor. 2025. ColliderVM: Stateful Computation on Bitcoin. *Cryptology ePrint Archive* (2025).
[36] Noureddine Lasla, Lina Al-Sahan, Mohamed Abdallah, and Mohamed Younis. 2022. Green-PoW: An energy-efficient blockchain Proof-of-Work consensus algorithm. *Computer Networks* 214 (2022), 109118.
[37] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. 2015. Inclusive block chain protocols. In *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19*. Springer, 528–547.
[38] Kevin Liao and Jonathan Katz. 2017. Incentivizing blockchain forks via whale transactions. In *Financial Cryptography and Data Security: FC 2017 International Workshops, WAHC, BITCOIN, VOTING, WTSC, and TA, Sliema, Malta, April 7, 2017, Revised Selected Papers 21*. Springer, 264–279.
[39] Dahlia Malkhi, Chrysoula Stathakopoulou, and Maofan Yin. 2023. BBCA-CHAIN: One-Message, Low Latency BFT Consensus on a DAG. *arXiv preprint arXiv:2310.06335* (2023).
[40] Michael Mirkin, Lulu Zhou, Ittay Eyal, and Fan Zhang. 2023. Sprints: Intermittent blockchain pow mining. *Cryptology ePrint Archive* (2023).
[41] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system.
[42] Kartik Nayak, Srijan Kumar, Andrew Miller, and Elaine Shi. 2016. Stubborn mining: Generalizing selfish mining and combining with an eclipse attack. In *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 305–320.
[43] Rafael Pass and Elaine Shi. 2017. FruitChains: A fair blockchain. In *Proceedings of the ACM symposium on principles of distributed computing*. 315–324.
[44] Kaihua Qin, Liyi Zhou, and Arthur Gervais. 2022. Quantifying blockchain extractable value: How dark is the forest?. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 198–214.
[45] Tim Roughgarden. 2021. Transaction fee mechanism design. *ACM SIGecom Exchanges* 19, 1 (2021), 52–55.
[46] Ayelet Sapirshtein, Yonatan Sompolinsky, and Aviv Zohar. 2017. Optimal selfish mining strategies in bitcoin. In *Financial Cryptography and Data Security: 20th International Conference, FC 2016, Christ Church, Barbados, February 22–26, 2016, Revised Selected Papers 20*. Springer, 515–532.
[47] Roozbeh Sarenche, Alireza Aghabagherloo, Svetla Nikova, and Bart Preneel. 2024. Bitcoin Under Volatile Block Rewards: How Mempool Statistics Can Influence Bitcoin Mining. *arXiv preprint arXiv:2411.11702* (2024).
[48] Roozbeh Sarenche, Svetla Nikova, and Bart Preneel. 2025. Mining Power Destruction Attacks in the Presence of Petty-Compliant Mining Pools. *arXiv preprint arXiv:2502.07410* (2025).
[49] Roozbeh Sarenche, Ren Zhang, Svetla Nikova, and Bart Preneel. 2024. Selfish mining time-averaged analysis in bitcoin: Is orphan reporting an effective

Countermeasure? *IEEE Transactions on Information Forensics and Security* (2024).
[50] Wellington Fernandes Silvano and Roderval Marcelino. 2020. Iota Tangle: A cryptocurrency to communicate Internet-of-Things data. *Future generation computer systems* 112 (2020), 307–319.
[51] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. 2016. Spectre: A fast and scalable cryptocurrency protocol. *Cryptology ePrint Archive* (2016).
[52] Yonatan Sompolinsky and Michael Sutton. 2022. The DAG KNIGHT protocol: a parameterless generalization of nakamoto consensus. *Cryptology ePrint Archive* (2022).
[53] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. 2021. PHANTOM GHOSTDAG: a scalable generalization of Nakamoto consensus: September 2, 2021. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*. 57–70.
[54] Yonatan Sompolinsky and Aviv Zohar. 2015. Secure high-rate transaction processing in bitcoin. In *Financial Cryptography and Data Security: 19th International Conference, FC 2015, San Juan, Puerto Rico, January 26-30, 2015, Revised Selected Papers 19*. Springer, 507–527.
[55] Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: Dag bft protocols made practical. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2705–2718.
[56] Pawel Szalachowski, Daniël Reijsbergen, Ivan Homoliak, and Siwei Sun. 2019. {StrongChain}: Transparent and Collaborative {Proof-of-Work} Consensus. In *28th USENIX Security Symposium (USENIX Security 19)*. 819–836.
[57] Itay Tsabary and Ittay Eyal. 2018. The gap game. In *Proceedings of the 2018 ACM SIGSAC conference on Computer and Communications Security*. 713–728.
[58] Itay Tsabary, Alex Manuskin, and Ittay Eyal. 2022. Ledgerhedger: Gas reservation for smart-contract security. *Cryptology ePrint Archive* (2022).
[59] Itay Tsabary, Alexander Spiegelman, and Ittay Eyal. 2022. Tuning PoW with Hybrid Expenditure. In *3rd International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2021)*. Schloss-Dagstuhl-Leibniz Zentrum für Informatik.
[60] Henk A Van der Vorst. 1992. Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on scientific and Statistical Computing* 13, 2 (1992), 631–644.
[61] Qin Wang, Jiangshan Yu, Shiping Chen, and Yang Xiang. 2023. Sok: Dag-based blockchain systems. *Comput. Surveys* 55, 12 (2023), 1–38.
[62] Aviv Yaish, Gilad Stern, and Aviv Zohar. 2023. Uncle maker:(time) stamping out the competition in ethereum. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*. 135–149.
[63] Haifeng Yu, Ivica Nikolić, Ruomu Hou, and Prateek Saxena. 2020. Ohie: Blockchain scaling made simple. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 90–105.
[64] Roi Bar Zur, Ittay Eyal, and Aviv Tamar. 2020. Efficient MDP analysis for selfish-mining in blockchains. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*. 113–131.

# A Calculating the Average Number of Whale Transactions per Block

Assuming all miners are honest, it means that whenever a new block is created, it includes a whale transaction if one is available and that the new block is immediately published and accepted by everyone else. Also, recall that when the number of whale transactions is *max_pool*, any excess transactions are discarded.

The number of whale transactions follows a Markov chain with states $0, 1, \ldots,$ *max_pool*. For $1 < i < max\_pool$, the probability to move right (the number of transactions increases by 1) is $\frac{\delta}{1+\delta}$ and the probability to move left (the number of transactions decreases by 1) is $\frac{1}{1+\delta}$ (this matches the ratio $\delta$ that our model stipulates). For $i = 0$, the probability to move right is $\frac{\delta}{1+\delta}$ and the probability to stay is $\frac{1}{1+\delta}$. For $i = max\_pool$, the probability to stay is $\frac{\delta}{1+\delta}$ and the probability to move left it $\frac{1}{1+\delta}$.

Calculating the steady state distribution of the chain is straightforward. We find that the probability to be in state 0 (no whale transactions are available) is exactly $p_0 = \frac{1-\delta}{1-\delta^{max\_pool+1}}$. The number of average whale transactions per block is its complement as a whale transaction will be included whenever one is available, resulting in $q = \frac{\delta-\delta^{max\_pool+1}}{1-\delta^{max\_pool+1}}$.

# B NC Selfish Mining Model

We review the model of Bar-Zur et al. [8] of NC with varying rewards. The model captures the perspective of a selfish miner who controls a fraction of $\alpha$ of the mining power while all other miners are honest.

The miner has a rushing factor $\gamma$.

The subsidy is constant and equal to 1 unit. Occasionally, a whale transaction appears, with a fee of $F$ units. Each block can contain a single transaction or none.

This model restricts the miner to mine on a single chain and does not track any blocks before the last fork. This is because an optimal selfish miner strategy only requires the miner to mine on a single chain. If at any point in time, a selfish miner with an optimal strategy switches to mine on another chain, it means it is more profitable to do so. At that point, the chance that the old chain will be rewarded only decreases, leaving it a suboptimal choice. This holds for all future steps as well, meaning the old chain should be abandoned, and the miner should focus only on the new chain.

*B.0.1 Utility Function.* The utility function takes a similar form to the model of Sapirshtein et al. [46]: the expected ratio between the total reward and the total difficulty contribution. The reward $R_t$ includes subsidy as well as transaction fees that the selfish miner receives, while $D_t$ remains the number of blocks added to the longest chain at step $t$.

*B.0.2 States.* The state space includes all states of the form $\left(\vec{a}, \vec{h}, fork, pool\right)$, where $\vec{a}$ is a vector of all blocks in the selfish miner's secret chain, including a flag for each block indicating if the block contains a whale transaction; $\vec{h}$ is a vector of all honest blocks in the public chain, the honest miners' chain starting from the fork that the selfish miner created (again, including a flag for each block indicating if the block contains a whale transaction); *fork* is a flag taking one of three values: irrelevant, relevant, or active, denoting whether rushing is possible or currently occurring; and *pool* is the number of whale transactions that have appeared since the last fork.

Figure 2a shows an example state.

*B.0.3 Actions and Transitions.* The model allows the three following types of actions. For convenience, we present these as parameterized actions that have a similar function, but in practice, each parameter value corresponds to a different action in the MDP.

*Adopt $\ell$.* The selfish miner abandons their secret chain, adopts the first $\ell$ blocks of the public chain and prepares to mine on top of the $\ell$-th block. The pool is decremented by the number of whale transactions in the public chain ($pool \leftarrow pool - \#T\left(\vec{h}\right)$, where $\#T(\cdot)$ is the number of whale transactions in the given chain). In addition, a difficulty contribution of $\ell$ is yielded.

*Reveal $\ell$.* This is only allowed when $\ell \geq |\vec{a}|$ (where $|\cdot|$ is the length of a vector) and either $\ell > \left|\vec{h}\right|$ (overriding the public chain) or $\ell = \left|\vec{h}\right|$ and $fork$ = relevant (attempting rushing).

In the first case, the selfish miner publishes exactly $\ell$ blocks, causing all honest miners to discard the public chain and prepare to mine on top of the selfish miner's revealed chain.

The public chain is reset ($\vec{h} \leftarrow [\,]$) and the selfish miner's secret chain shifts by $\ell$ blocks ($\vec{a} \leftarrow \vec{a} \ll \ell$). The pool is decremented by the number of whale transactions in the revealed blocks ($pool \leftarrow pool - \#T\,(\vec{a}[:\ell])$, where $\vec{x}[:n]$ is the prefix of $\vec{x}$ of length $n$). A reward of $\ell + F \cdot \#T\,(\vec{a}[:\ell])$ and a difficulty contribution of $\ell$ are yielded.

In the second case, $fork \leftarrow$ active.

*Wait.* The selfish miner waits until a new block is created.

Denote by $\vec{x}$++ the vector $\vec{x}$ with an appended block $x$ that contains a whale transaction if one can be added (depending on how many transactions $x$ contains in comparison to *pool*) and 0 otherwise.

If *fork* is not active, then one of two transitions may occur:

(1) The selfish miner mines a block in their secret chain ($\vec{a} \leftarrow \vec{a}$++) with probability $\alpha$, meaning rushing is currently impossible ($fork \leftarrow$ irrelevant).
(2) An honest miner mines a block in the public chain ($\vec{h} \leftarrow \vec{h}$++) with probability $1 - \alpha$, meaning rushing is currently possible ($fork \leftarrow$ relevant).

Otherwise, if *fork* is active, there is rushing currently occurring. Then, one of three transitions may occur:

(1) The selfish miner mines a block in their secret chain ($\vec{a} \leftarrow \vec{a}$++) with probability $\alpha$, meaning rushing is currently impossible ($fork \leftarrow$ irrelevant);
(2) An honest miner who received the honest miner's block first mines a block in the public chain ($\vec{h} \leftarrow \vec{h}$++) with probability $(1-\gamma)(1-\alpha)$.
(3) An honest miner who received the selfish miner's block first mines a block on the miner's chain with probability $\gamma(1-\alpha)$; all miners now agree that the published part of the selfish miner's chain is in the longest chain, discarding the previous public chain in favor of the new honest block ($\vec{h} \leftarrow [\,]$++), shortening the selfish miner's secret chain ($\vec{a} \leftarrow \vec{a} \ll \left|\vec{h}\right|$), and yielding a reward of $\left|\vec{h}\right| + F \cdot \#T\left(\vec{a}[:\left|\vec{h}\right|]\right)$ and a difficulty contribution of $\left|\vec{h}\right|$.

In the two latter cases, further rushing is still possible ($fork \leftarrow$ relevant).

Whenever the longest chain's length increases, there is a chance of $\delta$ that *pool* is incremented by 1 ($pool \leftarrow pool + 1$).

*B.0.4 Bounding the State Space.* To numerically solve the MDP, the state space needs to be bounded. The model allows the length of $\vec{a}$ and $\vec{h}$ to be up to a value of $L$. Whenever there is a possibility that a transition resulting from a certain action will violate this constraint, the action is forbidden. It is important to confirm that we never reach a state with no possible actions, since then the MDP would not be valid. But this is guaranteed, since a miner can always adopt the public chain to reduce the length of both chains.

In addition, the model also bounds the number of whale transactions in the pool by *max_pool*. When more than *max_pool* whale transactions are in the pool, the excess transactions are discarded.