Computing the Heaviest Conflict-free Sub-DAG in DAG-based DLTs

Quentin Bramas

Computing the Heaviest Conflict-free Sub-DAG in DAG-based DLTs *

Quentin Bramas
ICUBE, University of Strasbourg, CNRS
Strasbourg, France
bramas@unistra.fr

Abstract

In this paper, we consider DAG-based Distributed Ledger Technologies (DLTs), *i.e.*, DLTs where each block can reference several previous blocks hence forming a Directed Acyclic Graph of Blocks (BDAG). Each block has a weight (usually a constant normalized to one) and our goal is to compute the heaviest sub-BDAG that does not contain conflicting blocks.

First, we prove that computing such a sub-BDAG is NP-complete. Then, we show that the difficulty comes from concurrent conflicts and we present an optimal algorithm that is polynomial if the number of concurrent conflicts is bounded. Finally, we evaluate the performance of our algorithm on random BDAGs against an existing algorithm called GHOSTDAG and show that, in addition to being optimal, our algorithm is also more efficient in practice.

Keywords: Blockchain Technology, Distributed Ledger, Tangle, IOTA, DAG-based DLT, BlockDAG

1 Introduction and Background

A Distributed Ledger Technology (DLT) is a distributed protocol executed by a set of nodes to maintain an append-only data structure. In Bitcoin, the data structure is a chain of blocks, each block containing transactions. Blocks are appended one after the other to form a chain. Each block requires some amount of computational power, *called weight*, to be created. Due to the delay of the network between the participants, several branches (*forks*) can be created, hence forming a tree of blocks. In Bitcoin, a new block references the last block of the heaviest branch *i.e.*, the branch that maximizes the sum of the weights of the blocks it contains. This behavior is at the core of the security of Bitcoin.

This research paper focuses on DLTs that enable blocks to reference multiple previous blocks, thereby forming a Directed Acyclic Graph known as a Block Directed Acyclic Graph (BDAG). Several existing DLTs use this data structure to store cryptocurrency transactions. For instance, the IOTA [15] protocol uses such a data structure, called the *Tangle* (see Figure 1 for instance). Each new block references several previous blocks (two in IOTA) and the selection of these previous blocks, done by Tip Selection Algorithm (TSA), has been the focus of the previous work [2, 5, 6, 10].

As in Bitcoin, the parent selection is at the core of the security of DAG-based DLTs, however, the heaviest branch rule can be generalized in several ways. For instance, one can select the blocks with the longest paths to the genesis, but this rule does not take advantage of the fact that blocks can merge several branches. This is why several other algorithms exist to select the parents of new blocks.

IOTA's Tip Selection Algorithm selects two parents[1] randomly. Random selection is at the same time used to resolve conflicts. Indeed, random selection is biased towards blocks that are confirmed by more blocks so that in case of conflicting transactions, the one that is more confirmed has a higher chance to be selected as parents (hence increasing even more its probability to be selected in the future). However, it has been shown that resolving conflicts in a randomized way is not a good solution [6].

*This work was partially funded by the ANR project BASE-BLOC, ref. ANR-22-CE25-0012

[1] the algorithm can easily be generalized to select $k \geq 2$ parents

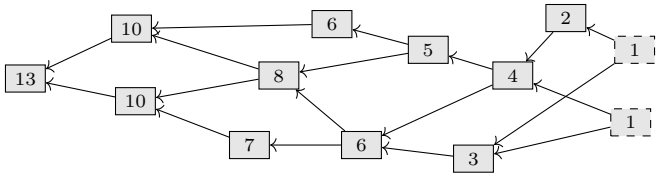

Figure 1: An example of a BDAG. In each site is written its cumulative weight. The two dashed blocks are called tips.

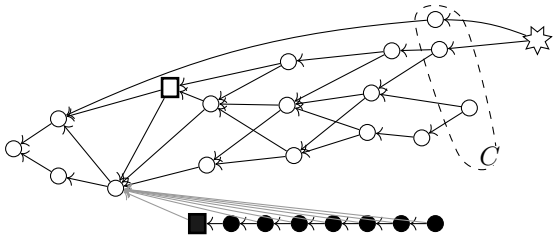

Figure 2: In this example, the white square is considered correct and the black one is discarded.

Other works focus on the selection of a sub-DAG that is considered legitimate, such as GHOSTDAG [16] and [12]. A legitimate sub-DAG should not contain conflicting transactions and have a structure that is close to what one can expect if only correct nodes are running the protocol. However, in these solutions, resolving conflicts is secondary and they are not guaranteed to obtain the heaviest sub-DAG.

In our solution, we take a different approach. We want to find a maximal conflict-free sub-DAG. Indeed, we want to use the fact that any block that is not in conflict with another block can be merged, even if it is not well-connected to the other blocks. We see no reason to discard a block based only on the DAG structure, except when choosing between two conflicting blocks. However, we show that finding a maximum (a maximal having the maximum size) conflict-free sub-DAG is NP-complete. Despite these negative results, we show that the difficulty comes from concurrent conflicts (defined formally later), and we propose an optimal algorithm that is polynomial if the number of concurrent conflicts is bounded. We evaluate our strategy against the GHOSTDAG algorithm and show that, in addition to being exact, our algorithm runs faster. Let us now present some examples that show the difficulty of the problem.

Motivations In previous work, the *cumulative weight* of a block has been defined as the sum of the weights of all its descendants (including itself). Figure 1 shows an example of a BDAG (a DAG of blocks) where the number in each block is its cumulative weight. In general, Cumulative weight is used to determine what is the current state of the Ledger. In Bitcoin, the heaviest branch rule is a way to select the branch that maximizes the cumulative weight of the genesis. In the GHOST variant, forks are resolved from the one closest to the genesis, to the farthest, by selecting the child with the highest cumulative weight (which does not necessarily results in the heaviest branch). However, in DAG-based DLTs, there is no natural way to resolve conflict using the cumulative weights of the conflicting blocks.

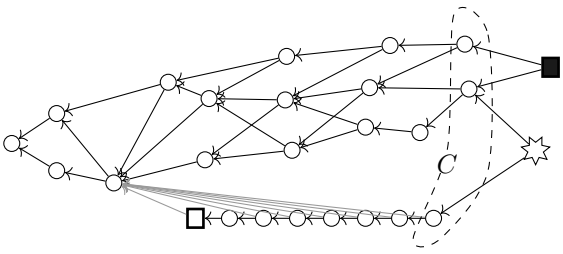

Figure 3: In this example, the new site can merge both branches.

In fact, the goal of existing solutions is usually not to manage conflicts, but more to select a plausible DAG that corresponds to what an honest set of participants would have generated. In the definition of GHOSTDAG [16], there is no explicit check for conflicting blocks. In the IOTA tip selection algorithm, the first parent is selected without worrying about conflicts.

In Figure 2, we can see an example of a BDAG with a sub-set of blocks that forms a line that is connected to the other blocks through a single block. Assuming the two square blocks are in conflict, it is natural to discard the black blocks since they seem malicious, and this is probably what happens using existing tip selection algorithms.

However, in Figure 3, if an algorithm selects the black square as parent, it directly implies that the line behind the white square is discarded. But one can notice that in this example, by discarding the black square, the obtained sub-BDAG is the heaviest conflict-free sub-BDAG, and a new block can select two parents among the tips in the set C.

In this paper, our goal is to manage conflicts when selecting the current state of the DAG-based DLT, in order to maximize the total weight of the obtained solution. In particular, if there are no conflicts, we believe the entire DAG should be used.

However, there is no natural way to resolve conflicts in a BDAG. For instance, in the BDAG shown in Figure 4, discarding the blocks with the smallest cumulative weight in the two conflicting pairs (the two squares and the two triangles) results in a very small BDAG.

In fact, cumulative weight evolves when a conflict is resolved. Here, discarding the triangle block v_2 with cumulative weight 5 changes the cumulative weight of u_1 to 1. So that applying the same rule discard u_1 and we end up with a full branch, which is intuitively a good solution.

However, as we saw, the cumulative weight can change so that the block with the smaller cumulative weight changes. But does it depend on the order in which we resolve conflicts? The answer is yes. Figure 6 shows an example of a BDAG such that, by first discarding the square block with the smallest cumulative weight u_1, the triangle block with the smallest cumulative weight is now v_2. Discarding v_2 results in a BDAG with a total weight of 9. However, by first discarding v_1, the two square blocks have the smallest cumulative weight, and discarding any of them results in a BDAG with a total weight of 8.

One can think that there maybe is a natural order in which conflict should be resolved in order to maximize the total weight of the obtained BDAG. However, in Figure 5 we can see that the heaviest conflict-free BDAG is obtained by discarding the two blocks with the smallest cumulative weight, u_1 and v_2.

Contributions Our contribution is threefold. First, we define formally the notion of conflicts in BDAG and conflict-free sub-BDAG. Second, we define the problem of Heaviest Conflict-free Sub-BDAG and prove that it is NP-complete. Third, we propose an optimal algorithm SeHeS to find a heaviest conflict-free sub-BDAG that is polynomial if the number of concurrent conflicts (defined later) is bounded. We evaluate our algorithm SeHeS by simulation and show that the running time outperforms the existing GHOSTDAG algorithm but is slower than the two greedy approaches (that are not optimal).

As a side note, when applied to a chain-based DLT (such as Bitcoin), SeHeS corresponds exactly to the heaviest branch rule, so our algorithm is a generalization of the heaviest branch rule.

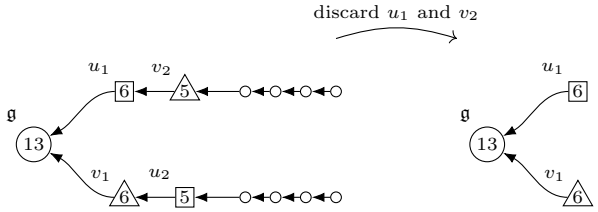

Figure 4: Resolving the conflicts by discarding the blocks with the smallest cumulative weights results in a very small sub-BDAG

2 Model

BDAG and sub-BDAG

Definition 1. *A* BDAG $G = (V, E)$ *is a Directed Acyclic Graph having a particular node* $\mathfrak{g}$ *called the* genesis *that is the ancestor of all the nodes in the BDAG.*

In a BDAG, nodes usually have at most a fixed number of parents (usually 2, such as in the IOTA protocol), so it is easier to orient the edges towards the parents to represent the fact that they are selected when the block is created, *i.e.*, edges point towards where a block is "attached" in the BDAG. If there exists a path from u to v, v is an ancestor of u and u is a descendant of v.

The set of descendants, resp. ancestors, of a block b is, denoted $Desc(b)$, resp. $Anc(b)$, inclinding b (we use the adjective *strict* when considering the set excluding the block b). Hence $Desc(\mathfrak{g}) = V$.

A sub-BDAG $G' = (V', E')$ of a BDAG $G = (V, E)$ is a BDAG induced by a subset of the nodes $V' \subset V$ that is also a BDAG with the same genesis $\mathfrak{g} \in V'$. In particular, if a block is in G' then all its ancestor in G are also in G' (otherwise the G' would not be a BDAG) We usually refer to a sub-BDAG S as a set of vertices, as the edges are induced. Similarly, we sometimes use G abusively to refer to the vertices V.

Conflicting Blocks We assume we have access to a set of *directly conflicting* sets of blocks $Conflicts(G)$. In practice a set of blocks $\{u_1, u_2, \dots u_k\} \in Conflicts(G)$ are directly conflicting if they contain transactions that update the same state, or in the UTXO model, use the same input transaction (*i.e.*try to spend the same fund). Since the descendants of a block implicitly confirm it, then if v_1, resp. v_2, is a descendant of u_1, resp. u_2, and u_1 and u_2 are directly conflicting (*i.e.*$u_1, u_2 \in C$ for some $C \in Conflicts(G)$), then v_1 and v_2 are *conflicting*.

In a BDAG, the parents of a block cannot be conflicting, otherwise, the block is not valid and is not included in the BDAG. Except for this rule, there is no restriction about the parents of a block, in particular, they can reference blocks that are far from one another (in terms of hops).

The set of conflicting sets of a sub-BDAG S of G consists in the intersections of each conflicting set in $Conflicts(G)$ with S that have a size at least 2:

$$Conflicts(S) = \{C \cap S \mid C \in Conflicts(G) \text{ s.t. } |C \cap S| \geq 2\}$$

A sub-BDAG is said to be *conflict-free* $Conflicts(S) = \emptyset$.

Weight and Cumulative Weight The weight of a block b, denoted $w(b)$, is a value that represents the confidence given to this particular block. This is a very abstract definition as it can refer to different kinds of things. For simplicity, our results only assume that this is an additive metric, for instance, it can represent the amount of computational power used to generate the block. In Bitcoin, the chain of blocks that maximizes the sum of the weights is considered to be the legitimate chain.

We define the cumulative weight of a block only for a given sub-BDAG since it can have different values depending on how we resolve potential conflicts.

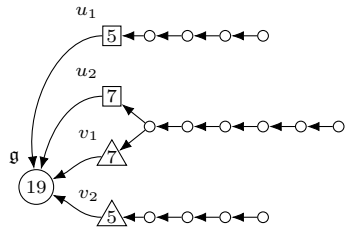

Figure 5: Example of a BDAG where the heaviest sub-BDAG is obtained by discarding the conflicting blocks with the largest cumulative weight.

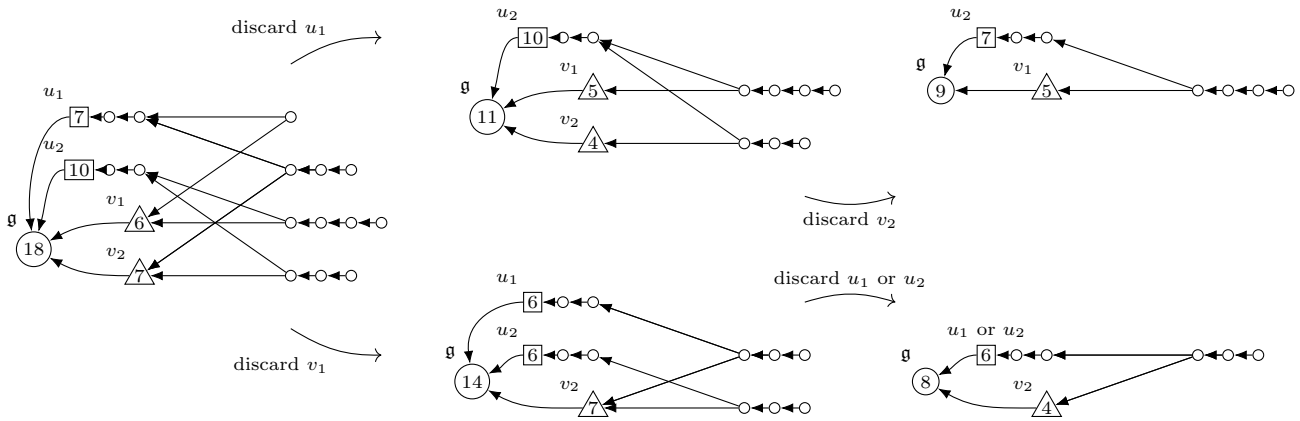

Figure 6: An example of a BDAG (left) containing two pairs of conflicting blocks, the squares and the triangles. The numbers represent the cumulative weight. The order in which we resolve the conflicts has an impact on the weight of the obtained conflict-free sub-BDAG (right).

Definition 2. *Let G be a BDAG and S a sub-BDAG of G. The* cumulative weight *of a block b in S, denoted $w_S(b)$ it the sum of the weights of b with all its descendants in S:*

$$w_S(b) = \sum_{b' \in Desc(b)} w(b')$$

The weight *of S is the cumulative weight of the genesis in S, $w(S) = w_S(\mathfrak{g})$, or equivalently, the sum of the weights of all the blocks in S.*

The Heaviest Conflict-Free Sub-BDAG Problem Given a BDAG G, with conflicts $Conflicts(G)$, the problem of finding the Heaviest Sub-BDAG Problem is to find a sub-BDAG S such that $Conflicts(S) = \emptyset$ and for all sub-BDAG S', $Conflicts(S') = \emptyset$ implies $w(S') \leq w(S)$.

3 Maximum Conflict-Free sub-BDAG is NP-complete

In this section, we prove that the problem of checking the existence of a conflict-free sub-BDAG of a given size is NP-complete[2]. In the proof of the Theorem, we use nodes with 3 parents for simplicity, but it can easily be modified to consider nodes with at most 2 parents: for each node u having 3 parents p_1, p_2, and p_3, one can add a node v with parents p_2 and p_3 and set the parents of u to be p_1 and v. After this modification, the set of nodes that are conflicting with u remains unchanged.

Theorem 1. *Checking if a BDAG G has a conflict-free sub-BDAG of size $k \in \mathbb{N}$ is NP-complete.*

Proof. First, it is clear that the problem is in NP since, given a sub-BDAG G' of G, one can easily check that it is of size k and that it is conflict-free. A naive algorithm could be to check if each pair of nodes in G' is conflicting or not (this can be checked directly or by iterating over their ancestors).

The proof NP-hardness is done by reduction from 3-SAT. Let F be a 3-SAT formula with variables $x_1, \dots x_n$ and clauses $C_1 \dots C_m$. The goal is to create a BDAG so that the formula is satisfiable if and only if there exists a sub-BDAG having a given size. From F we build the following BDAG $G = (V, E)$ where nodes are partitioned into 4 sets

$$V = \{\mathfrak{g}\} \cup V_x \cup V_C \cup V_W$$

$\mathfrak{g}$ is a single node that is the genesis of the BDAG; V_x is a set of $2n$ nodes representing the variables of the formula; V_C is a set of $7m$ nodes representing the clauses of the formula; and V_W is a set of $7mB$ nodes,

[2]As usual the NP-completeness of the optimization problem is deduced from the NP-completeness of the corresponding decision problem.

where B is a greater than $1 + 2n + 7m$, for instance, $B = 7(m + n)$. Nodes in V_W are used to give weight to each clause so that satisfying a clause is associated with the inclusion in the sub-BDAG of B nodes from V_W.

Nodes in V_x are denoted $u_{x_1}, u_{\overline{x_1}}, u_{x_2}, u_{\overline{x_2}}, \ldots u_{x_n}, u_{\overline{x_n}}$. Each node in V_x has a single parent, the genesis. For all i, $1 \leq i \leq n$, u_{x_i} and $u_{\overline{x_i}}$ are directly conflicting. The idea is that, choosing between including u_{x_i} or $u_{\overline{x_i}}$ in the sub-BDAG is equivalent to assigning a truth value to the variable x_i (true or false) in the formula F.

Nodes in V_C are partitioned into m sets of 7 nodes, $V_C = V_{C_1} \cup V_{C_2} \cup \ldots \cup V_{C_m}$. A set V_{C_i}, is associated with a clause C_i, and the goal is to be able to include one of the nodes in V_{C_i} if and only if the clause is satisfied, *i.e.*, one literal is true in the clause. To do so, let a, b, and c be the 3 literals in the clause C_i, then, let the 7 nodes in the set V_{C_i} be denoted

$$u_{C_i}(a, b, c), \quad u_{C_i}(a, b, \overline{c}),$$
$$u_{C_i}(a, \overline{b}, c), \quad u_{C_i}(a, \overline{b}, \overline{c}),$$
$$u_{C_i}(\overline{a}, b, c), \quad u_{C_i}(\overline{a}, b, \overline{c}), \text{ and}$$
$$u_{C_i}(\overline{a}, \overline{b}, c)$$

In G, node $u_{C_i}(a, b, c)$ has 3 parents corresponding to the literals a, b, and c. In more details, if a is the positive variable x_j, then $u_{C_i}(a, b, c)$ has node u_{x_j} as parent and $u_{C_i}(\overline{a}, b, c)$ has node $u_{\overline{x_j}}$ as parent, otherwise, if a is the negative variable $\overline{x_j}$, then $u_{C_i}(a, b, c)$ has node $u_{\overline{x_j}}$ as parent and node $u_{C_i}(\overline{a}, b, c)$ has node u_{x_j} as parent. We proceed similarly for the literals b and c. One can see that at most one node in V_{C_i} can be included in a conflict-free sub-BDAG because the nodes corresponding to the literal a and $\overline{a}$ are in conflict, so their descendants are also in conflict. Moreover, one can see that one node in V_{C_i} can be included in the sub-BDAG if and only if at least one literal of the clause C_i is also included in the sub-BDAG.

Finally, each node in V_C has exactly B strict descendants (such descendants do not have other parents) in V_W. As a result, V_W contains $7mB$ nodes. The obtained BDAG is illustrated in Figure 7.

Now we prove that the satisfiability of the formula F is equivalent to finding a conflict-free sub-BDAG of size at least mB. Let f be a truth-assignment of the variables such that F is satisfied ($\forall i, f(x_i) \in \{true, false\}$, $f(F) = true$), then we consider the sub-BDAG G' of G that contains the genesis, all the nodes u_{x_i} for which $f(x_i) = true$, all the nodes $u_{\overline{x_j}}$ for which $f(x_j) = false$, all all their non-conflicting descendants. For all $i \in [1..m]$, there must exist at least one node in V_{C_i} that can be included in the sub-BDAG without creating conflicts because in F at least one literal is true for the clause C_i. For instance if C_i contains a, b and c and $f(a) = true$, $f(b) = f(c) = false$, then $u_{C_i}(a, \overline{b}, \overline{c})$ can be included in G' because $u_a, u_{\overline{b}}$, and $u_{\overline{c}}$ are in G' (and not their conflicting blocks). Since a node in each V_{C_i} is included in G', we also include its B strict descendants. Since there are m clauses, G' contains at least mB nodes (in fact it contains $mB + m + n + 1$ nodes).

Conversely, consider a conflict-free sub-BDAG G' of G of size at least mB. Since B is greater than $1 + 2n + 7m$, then G' must contain nodes the descendants of at least m nodes in the set V_C. Since nodes in each V_{C_i} are pairwise conflicting, for each $i \in [1..m]$, then G' must contain the descendant from exactly one node in each set V_{C_i}, for all $i \in [1..m]$. Since all the ancestors are part of the sub-BDAG, the nodes associated with the literals that appear in a clause are also included in G'. Since G' is conflict-free, for each $i \in [1..n]$, either u_{x_i} or $u_{\overline{x_i}}$ is included in G' but not both (we know at least one is included because each variable appears in at least one clause). Let f be the truth-assignment function such that $f(x_i) = true$ if u_{x_i} is in G' and $f(x_j) = false$ otherwise (*i.e.*, if $u_{\overline{x_j}}$ is in G'). Since one node in each V_{C_i} is part of G', then at least one literal in each clause has the correct truth value, so each clause in F, and hence F, is satisfied by f. □

4 SeHeS: Sequential Heaviest Sub-BDAG Algorithm

In this section, we present a Sequential Heaviest Sub-BDAG Algorithm (SeHeS) algorithm, which computes a maximal conflict-free sub-BDAG. The algorithm has an exponential complexity in the worst case but is polynomial if the maximum number of concurrent conflicts is bounded. In other words, if conflicts are mostly sequential (*i.e.*, not concurrent), then our algorithm is very efficient.

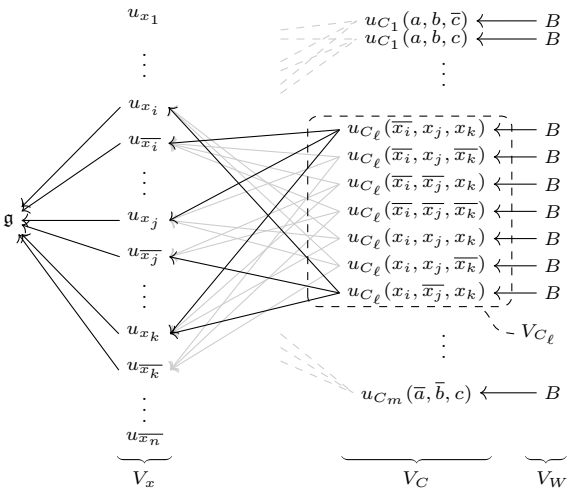

Figure 7: BDAG G associated with a 3-SAT formula F containing a clause $C_\ell = \overline{x_i} \vee x_j \vee x_k$. For all $i \in [1..n]$ nodes u_{x_i} and $u_{\overline{x_i}}$ are conflicting. Nodes in the set V_{C_ℓ} are pairwise conflicting because they have conflicting parents.

4.1 Description of the Algorithm

We first need to define the notion of concurrent conflicts and resolver sets, before presenting the algorithm.

Definition 3. *A set of conflicting sets $\mathcal{C}$ is* concurrent *with a conflicting set C if there exists two tips t_1 and t_2 of G such that*

(i) t_1, respectively t_2, is a descendant of a block in $\mathcal{C}$

(ii) t_1 is a descendant of a block in $b \in C$

(iii) t_2 is not a descendant of b (either it has no ancestor in C, or it is descendant of different block in C),

A set closed for the concurrent relation is called a closed-concurrent *set of conflicts* i.e.*, all the concurrent conflicts are part of the set.*

Definition 4. *Let G be a BDAG with conflicts $Conflicts(G)$. A* conflict resolver *R is a set of blocks such that*

- *Each block in R is in a conflicting set: $\forall b \in R, \exists C \in Conflicts(G), b \in C$*
- *Each conflicting set contains at most one block in R: $\forall C \in Conflicts(G), |R \cap C| \leq 1$*

The set $\overline{R}$ denotes the complement of R in the union of all the conflicting sets, which corresponds to the discarded blocks*:*

$$\overline{R} = \bigcap_{C \in Conflicts(G)} C \setminus R$$

The SeHeS algorithm works by first computing the smallest closed-concurrent set $\mathcal{C}$ of conflicts using Algorithm 1. Then, it computes the heaviest sub-BDAG that removes the conflicts in $\mathcal{C}$ using the procedure `BestResolve`. This is done by trying all the possible ways the conflicts can be resolved, *i.e.*, keeping one block in each conflicting set in $\mathcal{C}$ and discarding all the descendants of the other blocks. The complexity is the number of possible ways we can select one block in each set $\mathcal{C}$, *i.e.*, $\prod_{C \in \mathcal{C}} |C|$ iterations. Then we recursively call the SeHeS algorithm on the obtained sub-BDAG. We choose a recursive algorithm to help the explanation, but it can easily be replaced by an iterative one. The pseudo-code of the algorithm is given in Algorithm 2.

Algorithm 1: `ConcurrentConflicts`(C_0): return all the conflicting sets that are concurrent with a given conflicting set C_0

```
C ← {C_0}
while true do
    Common ←the common ancestors of blocks in C
    forall tip t of blocks in C do
        C ← C ∪ the set of conflicting sets impacting t
    end
    C ← C \ Common
    if C has not changed then return C ;
end
```

Algorithm 2: `SeHeS`(G): compute a heaviest conflict-free sub-BDAG of G

```
Algorithm SeHeS(G)
    if there exists no conflicting set in G then
        return G
    end
    C ← smallest set in {ConcurrentConflicts(C) | ∀C ∈ Conflicts}
    return SeHeS(BestResolve(G, C));

Procedure BestResolve(G, C)
    return heaviest BDAG in {Resolve(G, R) | ∀ resolver R }
```

4.2 Proof of Correctness

In this section, we prove the correctness of our algorithm. We first define what is the conflict resolver associated with a conflict-free sub-BDAG. In fact, it is clear that a conflict-free sub-BDAG is obtained by resolving the conflicts present in the original BDAG. In the remaining, G denotes a BDAG and $\mathcal{C}$ is a fixed subset of the conflicting sets of G: $\mathcal{C} \subset \mathit{Conflicts}(G)$. We are interested in how a sub-BDAG S of G resolves the conflicts in $\mathcal{C}$.

Definition 5. *The conflict resolver of $\mathcal{C}$ associated with S, denoted $CR(\mathcal{C}, S)$ is defined by $CR(\mathcal{C}, S) = \bigcup_{C \in \mathcal{C}} (S \cap C)$.*

Definition 6. *Let R be a conflict resolver. The* conflict-free descendants *of $\mathcal{R}$ in G, denoted $CF\text{-}Desc_G(R)$ is the set of descendants of R in G without the descendants of the complement $\overline{R}$: $Desc_G(R) \setminus Desc_G(\overline{R})$.*

Definition 7. *We denote by V_{safe} the set of blocks that are not descendants of any block that is part of a conflict:*

$$V_{safe} = V \setminus \mathit{Desc}(G, \mathit{Conflicts}(G))$$

The next lemma proves that a heaviest sub-BDAG can be built back from its associated conflict resolver set.

Lemma 1. *A heaviest conflict-free sub-BDAG S of a BDAG G verifies*

$$S = CF\text{-}Desc_G(CR(\mathit{Conflicts}(G), S)) \cup V_{safe}$$

where the union is disjoint.

Proof. For each block $b \in S$, either b is a descendant of R or not. In the former case b cannot be a descendant of $\overline{R}$ (because S is conflict-free), so $b \in CF\text{-}Desc_G(R)$. In the latter, $b \in V \setminus \mathit{Desc}(G, \mathit{Conflicts}(G)) = V_{safe}$. So we have that

$$S \subset CF\text{-}Desc_G(CR(S)) \cup V_{safe}$$

It remains to show that the right side is conflict-free, and the Lemma follows since S is a heaviest conflict-free sub-BDAG.

$R = CR(Conflicts(G), S)$ is the conflict resolver associated with S for all the conflicts of G. Hence $CF\text{-}Desc_G(CR(Conflicts(G), S))$ does not contain conflicting blocks. Indeed, assume for the sake of contradiction that u_1 and u_2 are conflicting, then they are respectively descendants of two directly conflicting blocks v_1 and v_2 in $C \in Conflicts(G)$. But at most one element from C is in R, so the other must be in $\overline{R}$, so either u_1 or u_2 is not in $CF\text{-}Desc_G(CR(Conflicts(G), S))$. Then blocks in V_{safe} cannot be in conflict with another block so $CF\text{-}Desc_G(CR(Conflicts(G), S)) \cup V_{safe}$ is conflict-free. □

The next lemma is the main argument of the proof of the correctness of our algorithm. It establishes that when $\mathcal{C}$ is a closed-concurrent set of conflicts, a sub-BDAG is concretely generated from three mutually independent sets and that changing the way we resolve the conflicts in $\mathcal{C}$ only impacts the first set.

Lemma 2. *Let $\mathcal{C}$ be a closed-concurrent set of conflicting sets and S a heaviest conflict-free sub-BDAG of G.*
Let $R = CR(\mathcal{C}, S)$ and $R' = CR(Conflicts(G) \setminus \mathcal{C}, S)$. Then,

$$S = CF\text{-}Desc_G(R) \cup CF\text{-}Desc_G(R') \cup V_{safe}$$

and the union is disjoint.

Proof. Using the previous Lemma, it is sufficient to show that (observe that $CR(Conflicts(G), S) = R \cup R'$)

$$CF\text{-}Desc_G(R \cup R') = CF\text{-}Desc_G(R) \cup CF\text{-}Desc_G(R') \tag{1}$$

The remaining proof is condensed due to space constraints. The $\supset$ inclusion is straightforward (and also true if $\mathcal{C}$ is not closed-concurrent). Indeed, we have

$$\begin{aligned} &Desc(\overline{R \cup R'}) \subset Desc(\overline{R}) && \text{and} \\ &Desc(R) \subset Desc(R \cup R') && \text{so} \end{aligned}$$

$$CF\text{-}Desc_G(R) = Desc(R) \setminus Desc(\overline{R}) \subset CF\text{-}Desc_G(R \cup R')$$

The same thing is true for R'.

For $\subset$ inclusion, let $b \in CF\text{-}Desc_G(R \cup R')$. Assume by contradiction that

$$b \notin CF\text{-}Desc_G(R) \cup CF\text{-}Desc_G(R').$$

If $b \in Desc(R)$, then $b \in Desc(\overline{R})$, so b has a ancestor in $\mathcal{C}$ and in $Conflicts(G) \setminus \mathcal{C}$, say c. Since $\mathcal{C}$ is closed-concurrent, by the property (iii) of the definition, all the descendants of $\mathcal{C}$ are descendants of c. We obtain the same result if $b \in Desc(R')$.

If $c \in R'$, then

$$Desc(R \cup R') \subset Desc(R') \text{ and } Desc(\overline{R'}) = Desc(\overline{R \cup R'}),$$

which make Equation (1) true, a contradiction.

Otherwise, if $c \notin R'$ (recall that $c \notin \mathcal{C}$), then

$$Desc(R) \subset Desc(\overline{R \cup R'}) \text{ and } Desc(\overline{R'}) = Desc(\overline{R \cup R'}),$$

which also make Equation (1) true, a contradiction. □

We are now ready to prove the correctness of our algorithm with the following theorem.

Theorem 2. *Algorithm 2 returns a heaviest conflict-free sub-BDAG of the input BDAG G.*

Proof. We prove this Theorem by induction on the size of *Conflicts*. If *Conflicts* is empty, then G itself, returned by our algorithm, is conflict-free, hence it is the heaviest sub-BDAG.

Let $\mathcal{C}$ be a smallest closed concurrent set of conflicting sets, chosen by our Algorithm, such that $S = SeHeS(G) = SeHeS(BestResolve(G, \mathcal{C}))$. Let S^* be a heaviest sub-BDAG of G. We want to show that $w(S) = w(S^*)$ and we assume that the result is true for a smaller set of conflicts, in particular, S is a

heaviest sub-BDAG of $BestResolve(G, \mathcal{C})$ (because conflicts in $\mathcal{C}$ are not part of the conflicts of the BDAG $BestResolve(G, \mathcal{C})$).

Recall that the BestResolve procedure maximizes the weight of the sub-BDAG among all the resolver sets of $\mathcal{C}$, so

$$w(\mathit{CF\text{-}Desc}_G(\mathit{CR}(\mathcal{C}, S))) \geq w(\mathit{CF\text{-}Desc}_G(\mathit{CR}(\mathcal{C}, S^*)))$$

By defining S' as follows

$$\begin{aligned} S' = &\mathit{CF\text{-}Desc}_G(\mathit{CR}(\mathcal{C}, S)) \cup \\ &\mathit{CF\text{-}Desc}_G(\mathit{CR}(\mathit{Conflicts}(G) \setminus \mathcal{C}, S^*)) \cup V_{safe} \end{aligned}$$

we obtain $w(S') \geq w(S^*)$ using Lemma 2 (the union is disjoint). Moreover, S' and S resolve the conflict in $\mathcal{C}$ in the same way so S' is contained in $G_1 = BestResolve(G, \mathcal{C})$. By induction, S is a heaviest sub-BDAG of G_1 so $w(S) \geq w(S')$, hence $w(S) \geq w(S^*)$. □

5 Evaluation

We evaluate our algorithm on random BDAG and compare it with the GHOSTDAG Algorithm executed with a parameter $k = \infty$ to maximize the weight of the solution (and this also simplifies the algorithm). We have also executed two greedy methods, where each conflicting set is resolved one by one, by keeping the block with the greatest cumulative weight and discarding the directly conflicting blocks (hence recomputing the cumulative weight). The order in which the conflicts are resolved is either in the decreasing (`greedy_max`) or increasing (`greedy_min`) order based on their cumulative weight. The source code for the evaluation and the generation of the figures is available online [7].

We generate the BDAG randomly, round by round, by adding blocks at each discrete round following a Poisson distribution with a parameter $\lambda = 10$. At each round, there is a 0.5 probability that, among the new blocks, the first two are directly conflicting. Each block selects uniformly at random its parents among the previous tips of the BDAG, making sure that its parents are not conflicting (the second parent is changed randomly until it does not conflict with the first one). For each directly conflicting pair of blocks, we simulate that some conflicts (with a 0.2 probability) can only be referenced by the next 100 following blocks, after which they are not referenced. This helps create random BDAGs with some heterogeneity. Observe that the generated BDAG contains a lot of conflicts and we do this to stress the algorithms we evaluate.

It is important to understand that the results we show here are very dependent on the chosen topology. One can easily create a particular BDAG such that a given algorithm performs very badly. Our topology is random enough not to be too disadvantageous against a particular algorithm and still shows some interesting behavior. This evaluation is meant to be a first approach for evaluating such algorithms and opens several questions that should be further analyzed in future work.

For each size between 100 to 600, with increments of 100, we run the algorithms on 200 random BDAGs and retrieve the running time and the weight ratio of the solution (*i.e.*, the ratio between the size of the output BDAG with the optimal solution given by our algorithm SeHeS). The figures show the average and the 95% confidence interval.

We can see in Figure 8 that the weights of the solutions are quite similar as the weight ratio is very close to 1. Surprisingly, the greedy_min algorithm outputs almost the optimal solution.

Regarding the running time, one can see in Figure 9 that our algorithm SeHeS outperforms GHOSTDAG, despite the fact that SeHeS is optimal. Indeed, we know that there exist instances where our algorithm would take an exponential time, however, in BDAG where conflicts are resolved after some time, our algorithm is very efficient, even when dealing with a lot of conflicts. However, it is outperformed by the greedy algorithms that take less than 10ms on a BDAG of size 600. An interesting question for future work can be to see if one can create an hybrid algorithm that detects the advantageous topology to benefit from the good performance of the greedy approach.

6 Related Work

The GHOST [17] protocol proposes to take into account the blocks outside the main chain in the chain selection rule, but each block still references a single previous block. Lewenberg et al. [12] defined the Inclusive

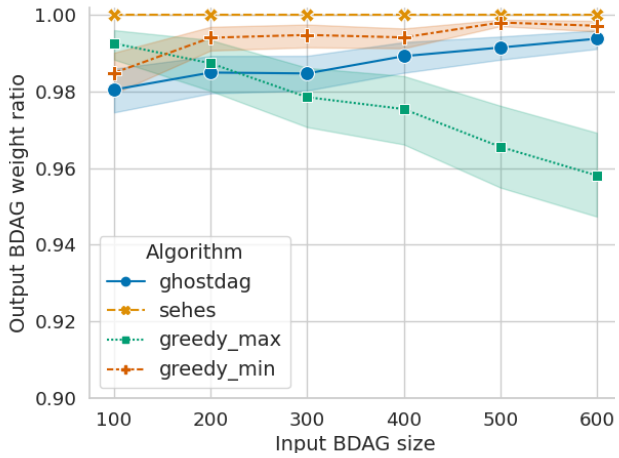

Figure 8: Weight ratio compared to the optimal solution of the sub-BDAG found by our algorithm SeHeS (ratio 1), GHOSTDAG, and two greedy algorithms, depending on the size of the input BDAG.

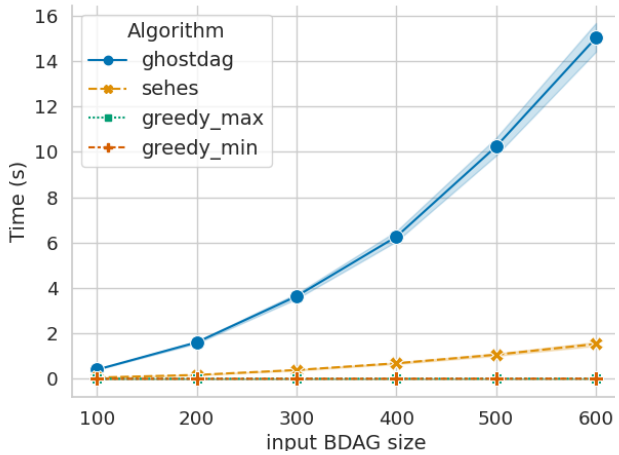

Figure 9: Running time of our algorithm SeHeS, GHOSTDAG, and two greedy algorithms, depending on the size of input BDAG.

protocol where each block references a main previous block (like in Bitcoin) and in addition references all the other unconfirmed blocks. The blocks hence form a DAG called a *blockDAG*. The protocol is thus very similar to Bitcoin but with a higher throughput since transactions included in blocks that are not on the main chain are still confirmed (and ordered) when they are referenced. Further, the authors present the random selection of transactions inside each block in order to analyze how rational participants could deviate from this rule to maximize their rewards. Prism [3] is a protocol where blocks form a DAG but where the structure is more constrained. Blocks have different types depending on their role: proposer blocks, transaction blocks, and voter blocks. Transaction selection in DAG-based DLTs has been further analyzed using game theory by Perešìn et al [14]. They showed that many existing protocols were vulnerable to the greedy strategy where a participant always selects the transaction that maximizes its reward. Birmpas et al. [4] proposed a new theoretical framework that captures a large family of DAG-based ledger implementations and showed how fairness and efficiency suffer from high transaction rates despite all agents behaving honestly in a given DAG-based ledger. The performance of DAG-based DLTs has also been studied by Cao et al. [9].

Sycomore [1, 11] is another approach to extend Bitcoin by allowing blocks to form a DAG structure. In Sycomore the structure is controlled such that multiple parallel branches are created when the transaction load is high and branches are merged when the load decreases.

Analyzing the structure of the DAG to identify the honest blocks has been done by Wang et al [18] and their proposed MaxCord algorithm where honest block identification is modeled as a generalized maximum independent set problem.

Among the existing studies in the field, the work by Sompolinsky et al [16] represents the closest approach to our research objectives. Their goal is to compute a heaviest sub-BDAG that is a k-cluster. Since the problem is NP-complete they proposed a greedy algorithm called GHOSTDAG to obtain a maximal, but not maximum, solution.

The main difference with our solution is we do not require the solution to be a k-cluster (but still the problem remains NP-complete because of the conflicting blocks), and our proposed algorithm is optimal. Our main contribution is that it is still possible to find an efficient algorithm when the number of concurrent conflicts is bounded.

In the IOTA cryptocurrency, blocks also refer to several previous blocks, forming a DAG called the *Tangle* [15] (defined by Serguei Popov). The tange faces similar challenges but an important difference is that blocks contain a single transaction so the DAG is expected to be much more wide than other DAG-based DLTs. In such a context, the problem of selecting a set of tips is even more challenging and the security associated with the tip selection algorithm has been widely analyzed [2,5,6,8,10,13]. We believe our algorithm is well suited for the IOTA tangle.

7 Conclusion

We defined formally the problem of Heaviest Conflict-free Sub-BDAG and prove that it is NP-complete. We have proposed an optimal algorithm called SeHeS that finds an exact solution in polynomial time if the number of concurrent conflicts is bounded. We have evaluated our algorithm SeHeS by simulation and shown that the running time outperforms the GHOSTDAG algorithm. An interesting future work would be to see how to make an incremental version of our algorithm in order to recompute a solution after every new block insertion. In another line of research, it can be interesting to see how to incentivize participants to use our algorithm to maximize the weight of the BDAG in the long run.

References

[1] Emmanuelle Anceaume, Antoine Guellier, Romaric Ludinard, and Bruno Sericola. Sycomore: A permissionless distributed ledger that self-adapts to transactions demand. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)*, pages 1–8. IEEE, 2018.

[2] Vidal Attias and Quentin Bramas. How to choose its parents in the tangle. In *Networked Systems: 7th International Conference, NETYS 2019, Marrakech, Morocco, June 19–21, 2019, Revised Selected Papers 7*, pages 275–280. Springer, 2019.

[3] Vivek Bagaria, Sreeram Kannan, David Tse, Giulia Fanti, and Pramod Viswanath. Prism: Deconstructing the blockchain to approach physical limits. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 585–602, 2019.

[4] Georgios Birmpas, Elias Koutsoupias, Philip Lazos, and Francisco J Marmolejo-Cossío. Fairness and efficiency in dag-based cryptocurrencies. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers*, pages 79–96. Springer, 2020.

[5] Quentin Bramas. The stability and the security of the tangle. In *International Conference on Blockchain Economics, Security and Protocols (Tokenomics 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.

[6] Quentin Bramas. Efficient and secure tsa for the tangle. In *Networked Systems: 9th International Conference, NETYS 2021, Virtual Event, May 19–21, 2021, Proceedings*, pages 161–166. Springer, 2021.

[7] Quentin Bramas. Computing the Heaviest Conflict-free Sub-DAG in DAG-based DLTs: Simulation Material, June 2023.

[8] Gewu Bu, Wassim Hana, and Maria Potop-Butucaru. E-iota: an efficient and fast metamorphism for iota. In *2020 2nd Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 9–16. IEEE, 2020.

[9] Bin Cao, Zhenghui Zhang, Daquan Feng, Shengli Zhang, Lei Zhang, Mugen Peng, and Yun Li. Performance analysis and comparison of pow, pos and dag based blockchains. *Digital Communications and Networks*, 6(4):480–485, 2020.

[10] Mauro Conti, Gulshan Kumar, Pranav Nerurkar, Rahul Saha, and Luigi Vigneri. A survey on security challenges and solutions in the iota. *Journal of Network and Computer Applications*, page 103383, 2022.

[11] Aimen Djari, Emmanuelle Anceaume, and Sara Tucci-Piergiovanni. Simulation study of sycomore++, a self-adapting graph-based permissionless distributed ledger. In *2022 4th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, pages 103–110. IEEE, 2022.

[12] Yoad Lewenberg, Yonatan Sompolinsky, and Aviv Zohar. 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*, pages 528–547. Springer, 2015.

[13] Nahid Alimohammadi Madenouei. Exploring the scalability, throughput and security characteristics of the tangle distributed ledger technology through simulation analysis, 2020.

[14] Martin Perešíni, Ivan Homoliak, Federico Matteo Benčić, Martin Hrubỳ, and Kamil Malinka. Incentive attacks on dag-based blockchains with random transaction selection. *arXiv preprint arXiv:2305.16757*, 2023.

[15] Serguei Popov. The tangle. *White paper*, 1(3):30, 2018.

[16] Yonatan Sompolinsky, Shai Wyborski, and Aviv Zohar. Phantom ghostdag: a scalable generalization of nakamoto consensus: September 2, 2021. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 57–70, 2021.

[17] Yonatan Sompolinsky and Aviv Zohar. 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*, pages 507–527. Springer, 2015.

[18] Xu Wang, Guohua Gan, and Ling-Yun Wu. Framework and algorithms for identifying honest blocks in blockchain. *Plos one*, 15(1):e0227531, 2020.