Twisting Lattice and Graph Techniques to Compress Transactional Ledgers

Rémi Géraud, David Naccache, Răzvan Roşie

Twisting Lattice and Graph Techniques to Compress Transactional Ledgers

Rémi Géraud[1], David Naccache[1], and Răzvan Roşie[1]

ENS, CNRS, INRIA and PSL Research University, Paris, France,
given_name.family_name@ens.fr

Abstract. Keeping track of financial transactions (e.g., in banks and blockchains) means keeping track of an ever-increasing list of exchanges between accounts. In fact, many of these transactions can be safely "forgotten", in the sense that purging a set of them that compensate each other does not impact the network's semantic meaning (e.g., the accounts' balances). We call *nilcatenation* a collection of transactions having no effect on a network's semantics. Such exchanges may be archived and removed, yielding a smaller, but equivalent ledger. Motivated by the computational and analytic benefits obtained from more compact representations of numerical data, we formalize the problem of finding nilcatenations, and propose detection methods based on graph and lattice-reduction techniques. Atop interesting applications of this work (e.g., decoupling of centralized and distributed databases), we also discuss the original idea of a "community-serving proof of work": finding nilcatenations constitutes a proof of useful work, as the periodic removal of nilcatenations reduces the transactional graph's size.

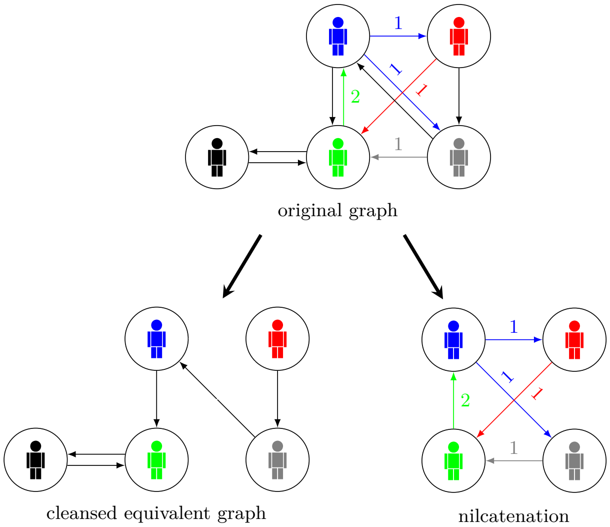
Key words: nilcatenation, subset-sum problem, lattices, LLL

1 Introduction

Transactional ledgers are a staple of modern technology — whether it is data, value or goods being tracked, concrete implementations require strong consistency guarantees and efficient data structures. Furthermore, it may be useful to perform sanity checks on data, such as in bank ledgers for instance, to ensure that an account's balance is legitimate (i.e., the amount can be explained as an inflow of money, whose source can be tracked). In another setting, namely centralized DBMS, it is typical to undergo high volumes of concurrent queries; auditing data causes extra pressure on the various locking strategies. Moreover, in some blockchains/distributed ledgers, the ledger keeps track over time of *all* individual transactions, and these transactions are *atomic*: they cannot be merged or split. As time passes, the number of transactions grows, and so do the storage requirements.

To give an intuition of the network and storage requirements, the full Bitcoin blockchain claimed (as of June 2017) more than 120 GB [1]. Hopefully, most users do not need to archive the full database, and Bitcoin proposes a form of

Fig. 1. Decoupling a transactional multigraph into the cleansed part (left) and the nilcatenation (right).



compressed partial storage.[1] That being said, even with Bitcoin's Merkle-tree-based mechanism, there is considerable stress on the network, in particular when users need to access historic data.

Nevertheless, we think it is important to look for generic solutions beyond this particular case, and these motivational examples highlight the realisation that storage requirements will only grow. This calls for a research into how information can be efficiently stored or cleansed, i.e., *represented*. Such a representation should be *semantically preserving*, at least to the extent that the effect of no transaction is lost in the process.[2] In many cases, some details might become irrelevant (e.g., the precise number of successive transactions between two parties) and it might be possible then to clump some events together into a more compact form. Finding efficient representations of transactional graphs is the main purpose of this work.

The Nilcatenation Problem. Throughout the following sections, we will consider a set of accounts, and the transactions between them represented as labeled edges between nodes in a graph G. More precisely, G is a *multigraph*, as multiple transactions are allowed between users. See Appendix A for precise definitions of these standard notions.

The "nilcatenation problem" (NCP) on G consists in constructing a transaction graph G' which is smaller than G, but provably semantically equivalent. That the new and old information coincide should be easy to verify, and the shorter ("purged") graph makes duplication and checking easier. Concretely, this consists in identifying a subgraph that can be removed without affecting any

[1] So do a few other cryptocurrencies, such as Ethereum.

[2] Which is very different from "that no transaction is lost in the process".

account's balance (see Figure 1). The notion of nilcatenation is generic, in that it applies to any graph labeled with numbers[3], and therefore bears applications in many situations.

Applications. In the context of distributed (anonymous) cryptocurrencies and distributed ledgers, we point out that identifying nilcatenations can be seen as a service to the community, and therefore be rewarded... in coins, just as any other proof of work. In that respect, the perspective of a cryptocurrency allowing users to mine by nilcatenation is not unrealistic, and we discuss it in Section 5.

Alternatives. The problem of bookkeeping is certainly not new, and many solutions have been proposed to address storage requirements. In the traditional (centralized) setting, complete archiving is the *de facto* solution.

As we mentioned above, trying to avoid the perpetual duplication of history was an early concern of cryptocurrencies, starting with Bitcoin's Merkle-based fast transaction checking [13]. With this scheme, it is possible to verify that a transaction has been accepted by the network by downloading just the corresponding block headers and the Merkle tree. Nodes that do not maintain a full blockchain, called *simplified payment verification* (SPV) nodes, use Merkle paths to verify transactions without downloading full blocks. Other cryptocurrencies use forgetful mechanisms (e.g. Ethereum and [5, 6]). Such approaches prevent auditability, insofar as the origin of old enough transactions is lost.

Related work. In general, constructing a *useful* proof-of-work (PoW) is a hard problem. One direction, mentioned in [2], is to use the PoW mechanism to solve specific problems having a well-investigated computational complexity (e.g. orthogonal vectors). Some cryptocurrencies, such as Primecoin, propose PoWs challenging the workers to find chains of prime numbers. A different working thread relies on proofs-of-storage, where a prover needs to demonstrate to a verifier that it stores a specific file. Filecoin is a recent, incipient proposal where the miners get rewarded by the amount of data they store. To the best of our knowledge, no prior work attempted to introduce of PoW for finding nilcatenations in a multigraphs, aiming to compress data.

1.1 Contributions

- We introduce and formalise the nilcatenation problem, phrased in terms of weighted multigraphs. We show — via a reduction to the (multi-dimensional) subset-sum problem — that NCP is **NP**-complete (Theorem 1).
- Our main contribution is the introduction of efficient algorithms to find nilcatenations (Section 4), which is optimal when the underlying subset-sum problem has a low enough density (Theorem 3). This is expected to be realistic, assuming maximal transactions are in the order of billions of economical units. Our approach is based on a combination of graph-theoretical and lattice reduction techniques, as explained in Section 3.3.

[3] It may be possible to extend our work to some more general algebraic structures.

- As a complement, we explore the possibility of using NCPs as proofs of work, to be used in cryptocurrency-like settings (Section 5). Reward models are presented and the practical precautions needed to correctly fine-tune the resulting incentive are also discussed. We analyse cheating strategies and provide countermeasures. Along the way, we point out several interesting questions raised by the analysis of this problem.

2 Preliminaries

Notations. We will make use of the following standard notations: $[n]$ denotes the set $\{1, \ldots, n\}$. For a set S, we denote by $s \leftarrow_\$ S$ the action of sampling s uniformly at random from S, and by $|S|$ the cardinality of S. PPT stands for a “probabilistic polynomial time". Polynomial-time reductions are written as $\leq_P$. We use standard notations for (multi)graphs, which are detailed in Appendix A.

2.1 The subset-sum problem

We recall the well-known definition of the *subset-sum problem* (SSP, [10]):

Definition 1 (Subset-Sum Problem). *Given a finite set $A \subset \mathbb{Z}$, and a target value $t \in \mathbb{Z}$, find a subset $S \subseteq A$ such that $\sum_{s \in S} s = t$.*

We denote by the size of the instance the cardinality of A. The SSP is known to be **NP**-complete [9]. The multi-dimensional case considers p “parallel” SSP instances under the constraint that an index-set solution to one problem remains a solution to the other $p - 1$. The *density* of a particular SSP instance of size n is defined [11] as: $d = n/\big(\max_{a \in A} \log a\big)$. While generic SSP instances are hard to solve, low-density instances can be solved efficiently using approximation techniques or lattice reduction [11, 7]. We also quickly consider:

Definition 2 (0-target Subset-Sum Problem, or 0TSSP). *Given a vector $A \in \mathbb{Z}^n$, find a vector $\epsilon \in \{0, 1\}^n$, $\epsilon \neq \mathbf{0}$, such that $\langle A, \epsilon \rangle = 0$, where $\langle \cdot, \cdot \rangle$ denotes the inner product.*

Proposition 1 (SSP $\leq_{\mathbf{P}}$ 0TSSP). *Let $\mathcal{O}$ be a 0TSSP oracle. There exists a PPT algorithm $\mathcal{A}$ that solves an instance of an SSP problem within n calls to $\mathcal{O}$, where n denotes the size of the instance.*

Proof (Intuition). Let an SSP problem be defined by $A = \{a_1, \ldots, a_n\}$ and target sum t. We assume $a_i \neq 0, \forall i \in [n]$ and $t > 0$.[4] If all $a_i > 0$ (or $a_i < 0$), we create a new 0TSSP instance of size $n + 1$, $A' = \{a_1, \ldots, a_n, -t\}$, and query $\mathcal{O}$: a solution for A' trivially provides a solution to the original SSP instance (A, t).

When some $a_i < 0$, we set $d \leftarrow \sum_{a_i > 0} a_i$, $e \leftarrow \sum_{a_i < 0} a_i$ and construct n new 0TSSP instances:

[4] If $t < 0$, we obtain an equivalent problem by changing the sign for each element a_i and for the target t.

$$B_i = \{b_1, \dots, b_n, -t - i \cdot f\}, \quad t''_i = 0, \ \forall i \in [n]$$

where $b_i \leftarrow a_i + f$ and $f \leftarrow |d| + |e| + t$. Observe that $b_i > 0, \forall i \in [n]$.

If the original problem has a subset sum t, then one of the new 0TSSP instances will have a solution. On the other hand, if one of these n 0TSSPs has a solution, the original SSP has a solution as well.

Let S be a solution to the i-th 0TSSP B_i, then $\sum_{j \in S} b_j = t + i \cdot f$. Equivalently, $j \cdot f + \sum_{j \in S} a_j = t + i \cdot f$, which is $\sum_{j \in S} a_j = t + (i - j) \cdot f$.

- If $i > j$, then we get that $\sum_{j \in S} a_j > d + e$, which cannot be true.
- If $i < j$, then we get that $\sum_{j \in S} a_j \leq -d - e$, which again cannot be true, since we assumed that all a_i cannot be negative.

Thus, we are only left with the possibility that $i = j$, and thus $\sum_{j \in S} a_j = t$. □

Remark 1. In fact, the polynomial reduction shown in the proof of Proposition 1 shows that SSP is equivalent with its 0 target version, both being **NP**-complete. Indeed, we trivially have 0TSSP $\leq_{\mathrm{P}}$ SSP.

3 Formalising the NCP

3.1 A First Definition

In all that follows, the history of transactions is assumed to form a multigraph $G = (V, E, \phi)$, where the vertices V correspond to accounts, and a labeled edge $e = a \overset{u}{\to} b$ corresponds to a transaction from a to b of amount u, denoted as $\phi(e) = u$.

The *balance* $b(v)$ of an individual account v is given by the difference between incoming transactions and outgoing transactions, i.e., $b(v) = \sum_{e:\bullet \to v} \phi(e) - \sum_{f:v \to \bullet} \phi(f)$, where $(\bullet \to v)$ denotes all incoming edges, i.e. all the elements in E of the form (w, v) for some $w \in V$; similarly $(v \to \bullet)$ denotes all outgoing edges. Let $b(G)$ denote the vector $\{b(v) : v \in V\}$, which we refer to as the graph's *semantics*.

Definition 3 (Nilcatenation Problem, NCP). *Given a weighted multigraph $G = (V, E, \phi)$, find $\widetilde{E} \subseteq E, \widetilde{E} \neq \emptyset$, such that $b(G) = b(G - \widetilde{G})$, where $\widetilde{G} = (V, \widetilde{E}, \phi)$. We call $(\widetilde{G}, G - \widetilde{G})$ the* nilcatenation *of G.*

Remark 2. In other terms, finding a nilcatenation consists in finding edges that can be removed without impacting anyone's balance — i.e., that preserve the graph's semantics. By definition, for every vertex $\tilde{v} \in \widetilde{G}$, we thus have $b(\tilde{v}) = 0$.

3.2 NCP and SSP

Definition 4 (NCP, alternative definition). *Let $G = (V, E, \phi)$ be a weighted multigraph. Write $V = \{v_1, ..., v_n\}$, and represent an edge $e : v_i \overset{r}{\to} v_j$ as the* *vector* $r \cdot e_{ij} \in \mathbb{Z}^n$ *where* e_{ij} *is the vector of* $\mathbb{Z}^n$ *with* 1 *in position* j*,* -1 *in position* i *and* 0 *in the remaining components. This defines a bijection between* E *and* G*'s adjacency matrix* $\mathbf{E}$*. The matrix* $\mathbf{E}$ *is a list of* m *such vectors* $\mathbf{E} = (e_1, \ldots, e_m)$*. The* nilcatenation problem *consists in finding a non-zero* $\epsilon \in \{0,1\}^m$ *such that*

$$\sum_{i=1}^{m} \epsilon_i e_i = \langle \mathbf{E}, \epsilon \rangle = 0 \ ,$$

where we have extended the notation $\langle \cdot, \cdot \rangle$ *in the obvious way. The* nilcatenation *of* G *is then defined as* $(\widetilde{G}, G - \widetilde{G})$*, where* $\widetilde{G} = (V, \widetilde{E}, \phi)$ *and* $\widetilde{E} = \{e_i \in E, \epsilon_i = 1\}$*.*

Remark 3. In many cases, we are chiefly interested in the largest ϵ (in terms of Hamming weight), because these result in the largest nilcatenations.

Figure 2 illustrates on a toy example the matrix $\mathbf{E}$ and identifies a nilcatenation.

Remark 4. Definition 4 makes clear the parallel between the NCP and the multi-dimensional version of the SSP (Definition 2). For $n = 2$, the NCP problem consists of a $2 \times m$ matrix with no 0 entries, the goal being to find an index subset for the columns that sum up to the all-zero column. Thus, the NCP and 0TSSP are exactly the same problem when $n = 2$.

In fact, more is true: NCP can be seen as a *multi-dimensional* variant of the subset-sum problem with zero as target, where the entries belong to $\mathbb{Z}^{|V|}$ instead of $\mathbb{Z}$. Note however that NCP is a remarkably *sparse* special case of that multi-dimensional SSP.

Theorem 1 (NCP $\equiv_P$ 0TSSP).

Proof (Intuition). By the above discussion, a 0TSSP oracle provides a solution to any NCP instance. Vectors of $\mathbb{Z}^{|V|}$ can be described as **integers** using a base $|V|$ encoding. Therefore we have a reduction from 0TSSP to NCP.

Conversely, assume an NCP oracle, then we can construct an NCP instance with all zeros except in two columns (in effect, this is an $n = 2$ instance). Then, by the remark made above, the NCP oracle solves a 0TSSP instance. □

Corollary 1. ***NCP*** *is* ***NP****-complete.*

Proof. This follows from the fact that SSP is **NP**-complete, then SSP $\equiv_P$ 0TSSP by Proposition 1, and NCP $\equiv_P$ 0TSSP by Theorem 1. □

$$\mathbf{E}^\top = \begin{pmatrix} -10 & 10 & 0 & 0 \\ -5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 40 & 0 & 0 & -40 \\ 5 & 0 & 0 & -5 \\ 0 & -20 & 20 & 0 \\ 0 & -5 & 5 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -30 & 30 \\ 0 & 0 & -5 & 5 \end{pmatrix}$$

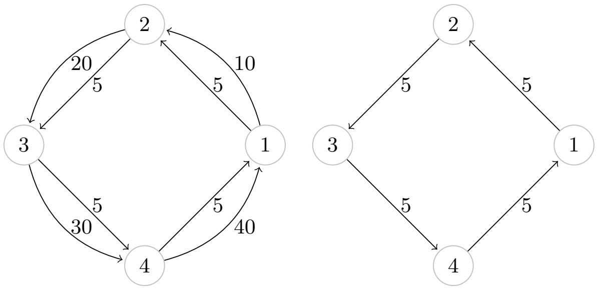


Fig. 2. A simple support example, depicting a multigraph with a nilcatenable subgraph. $\mathbf{E}^\top$ is the transpose of the adjacency matrix introduced in Definition 4, corresponding to the multigraph (middle); the multigraph on right stands for the nilcatenation.

3.3 Solving a Generic NCP Instance

Following the previous observation, one may be tempted to leverage known SSP solving techniques to tackle the NCP. However, the reduction from NCP to SSP is not very interesting from a computational standpoint: coefficients become very large, of the order of Bb^n, where B is the upper bound of the representation of E, and b is the chosen basis. This encoding can be somewhat improved if we know the bounds $B_i^{\pm}$ for each column, because we can use better representations. However, in practice it becomes quickly prohibitive; even brute-forcing the original NCP is less computationally demanding — the subset-sum problem can be solved exactly (classically) in worst-case time $\mathcal{O}(2^m)$ by brute-forcing all combinations , and even state-of-the-art algorithms only have marginally better complexity, namely $\mathcal{O}(2^{m \cdot 0.291\ldots})$ [8, 3].

If we wish to tackle the NCP directly, for $n > 2$, the meet-in-the-middle approaches inherited from subset-sum solvers do not apply, as in that case there is no total order on $\mathbb{Z}^n$. Instead we will leverage the famous LLL lattice reduction algorithm [12]. Given as input an integer d-dimensional lattice basis whose vectors have norm less than B, LLL outputs a reduced basis in time $\mathcal{O}(d^2 n(d + \log B) \log Bf)$[14], where f stands for the cost of d-bit multiplication.

To see why lattice reduction would solve the problem, first note that E can be represented as an $n \times m$ matrix with rational (or integer) coefficients. It is a sparse matrix, having (at most) two non-zero entries per column, i.e. (at most) $2m$ non-zero entries out of nm. Let I_n be the $n \times n$ identity matrix and let $\mathcal{E} = (I_n|E)$ be the result of concatenating the two blocks: $\mathcal{E}$ is an $n \times (n+m)$ matrix, having at most $n + 2m$ non-zero elements out of $n(n+m)$.

Now if there is a solution to the NCP, then it belongs to the lattice generated by $\mathcal{E}$. In particular this is a short vector: if this is the shortest vector, then LLL[5] will find it with overwhelming probability. The question of solving the NCP from a solution to the shortest-vector problem (SVP) depends on the density, topology and weights' probabilistic distribution of the multigraph. A proof of optimality for some graph families (denoted “hub graphs") is worked out in Section 3.4.

In practice, however, using this technique directly is impractical. The main reason is that LLL's complexity on a large graph is dominated by m^3, and real-world ledgers handle many transactions, with m being of the order of 10^8 *per day*. Therefore this intuition needs improvements to become practical, as we discuss in Section 4.

3.4 Solving NCP using a single SVP oracle query

The algorithm we propose in Section 4 relies on LLL as an SVP-oracle, to find a short vector and solve the given NCP instance. In other terms, we claim that *specific* NCP instances can be solved, with overwhelming probability, using a single query to an SVP-oracle.

[5] Or BKZ [17], or one of their variants. We use these algorithms here as SVP oracles.

We also extend the work of [7, 11, 15] where similar proofs are laid out for SSP and multi-dimensional SSP (henceforth MDSSP) instances with uniformly sampled entries. As a starting point, we recall the following result:

Theorem 2 (Pan and Zhang [15]). *Given a positive integer A, let a_{ji} where $j \in [n]$, $i \in [m]$ be independently uniformly sampled random integers between 1 and A, $e = (e_1, e_2, \ldots, e_m)$ be an arbitrary non-zero vector in $\{0,1\}^m$ and $s_j = \sum_{i=1}^{m} a_{ji} e_i$, where $j \in [n]$.*

If the density $d < 0.9408\ldots$ then with overwhelming probability the multi-dimensional subset sum problem (MDSSP) defined by a_{ji} and $s_1, \ldots, s_n$ can be solved in polynomial time with a single call to an SVP oracle.

One can attempt to reduce the NCP instance to an MDSSP one; however, the impeding issue is the distribution of a_{ji}, which is not uniform in general, so the above result does not apply directly. However, we may hope to get a useful result, based on the crux point that we are working with sparse subsets of a_{ji} (as defined by the edge multiset E of our multigraph). Here, by a sparse subset, we mean one where at least half of the elements are 0.

We use the following notion: call a "hub multigraph", one that contains a vertex directly connected to all the other nodes (we call this vertex a "hub vertex"). For such graphs, we can prove the following:

Theorem 3 (NCP for Hub Multigraphs). *Given $A \in \mathbb{N}$, let a_{ji} where $j \in [n]$, $i \in [m]$ be a sparse set of independently sampled (not necessary uniform) integers between 0 and A, $e = (e_1, e_2, \ldots, e_m)$ be an arbitrary non-zero vector in $\{-1,0,1\}^m$ and $s_j = \sum_{i=1}^{m} a_{ji} e_i = 0$, where $j \in [n]$.*

If the density $d < 0.488\ldots$ and if there exists i such that $\forall j \in [n], a_{ji} \neq 0$, then the MDSSP defined by a_{ji} and $s_1, \ldots, s_n$ can be solved in polynomial time with a single call to an SVP oracle.

Proof. We follow closely the proof of [15], and diverge in the last part of their demonstration, i.e., in obtaining the probability for an SVP-oracle to return an accurate answer, given a uniform, low density instance of the MDSSP.

The multigraph is finite, therefore at a given point in time, the maximum number of arcs connecting one vertex with another one is bounded by M, thus $\forall i : \deg^+(v_i) \leq M \wedge \deg^-(v_i) \leq M$. Let $m = M \cdot n \cdot (n-1)/2$ and let $\mathbf{e}$ be the solution to the SSP problem.

We begin by defining an appropriate basis for a lattice. The idea is to write the basis as

$$\mathbf{B} = \begin{pmatrix} \mathbf{I_m} | N \cdot \mathbf{E^t} \\ \mathbf{b_{m+1}} \end{pmatrix}$$

where $\mathbf{I_m}$ stands for the identity matrix, $\mathbf{E}$ is the multigraph's adjacency matrix (as described in Definition 4) and $N > \sqrt{(m+1)/4}$. The last component of the basis, namely $\mathbf{b_{m+1}}$, will be a special vector of the form:

$$\mathbf{b_{m+1}} = (\tfrac{1}{2}, \tfrac{1}{2}, \ldots, \tfrac{1}{2}, \tfrac{1}{2}, 0, 0, 0, \ldots, 0, 0)$$

Let L be the lattice generated by $\mathbf{b_1}, \mathbf{b_2}, \dots, \mathbf{b_{m+1}}$. We can observe that $\mathbf{e} = (e_1 - \frac{1}{2}, e_2 - \frac{1}{2}, \dots, e_m - \frac{1}{2}, 0, 0, \dots, 0) \in L$. We define X as the set of vectors different by $\pm\mathbf{e}$ (with a smaller norm) that belong to L:

$$X = \{\mathbf{v} \in L : \|\mathbf{v}\| \leq \|\mathbf{e}\| \text{ and } \mathbf{v} \notin \{\pm\mathbf{e}, \mathbf{0}\}\}$$

Roughly, we want to prove that with overwhelming probability, the problem has a unique solution, which is given by $\pm\mathbf{e}$. We make two remarks and prove the second one:

1. If $X = \emptyset$, then $\pm\mathbf{e}$ are the only short non-zero vectors in L.
2. $X = \emptyset$ with probability exponentially close to 1.

The crux part in the proof is bounding the value of $\Pr[X = \emptyset]$. Let $\mathbf{v} \in X$ such that $\mathbf{v} = \sum_{i=1}^{m+1}(z_i \cdot \mathbf{b}_i)$, having $z_i \in \{-1, 0, 1\}$. Since $N > \sqrt{(m+1)/4}$, the last n elements in $\mathbf{v}$ must be 0. Hence, we set up $\mathbf{v}$ as follows:

$$\begin{aligned} v_i &= z_i + \frac{1}{2}z_{m+1}, \text{ for } i \in [m] \\ v_{m+1} &= \frac{1}{2}z_{m+1}, \\ v_{m+1+j} &= N \cdot \left(\sum_{i=1}^{m} z_i \cdot a_{ji}\right) = 0 \text{ for } j \in [n] \end{aligned}$$

By using the previous notations, we now rewrite the condition as:

$$\sum_{i=1}^{m} a_{ji}(v_i - v_{m+1}) = \sum_{i=1}^{m} a_{ji} z_i = 0, \forall j \in [n].$$

Then, following the same technique as in [15], we let

$$D = \left\{\mathbf{v} \in \mathbb{Z}^{n+1} \middle| \exists (z_1, \dots, z_{m+1}) \in \mathbb{Z}^{n+1} \text{ s.t. } v_i = z_i + \frac{1}{2}z_{m+1} \wedge v_{m+1} = \frac{1}{2}z_{m+1}\right\}$$

and bound the required probability:

$$\begin{aligned} \Pr[X \neq \emptyset] \leq \Pr &\left[\sum_{i=1}^{m} a_{ji}(v_i - v_{m+1}) = 0 : j \in [n] \wedge \mathbf{v} \notin \{\mathbf{0}, \pm\mathbf{e}\}\right] \\ &\times |\{\mathbf{v} \in D \mid \|\mathbf{v}\| \leq \|\mathbf{e}\|\}| \end{aligned} \tag{1}$$

- If z_{m+1} is even, then $\|\mathbf{v}\| = \sqrt{\frac{m+1}{4}}$, implying $|\{\mathbf{v} \in D \mid \|\mathbf{v}\| \leq \|\mathbf{e}\|\}| = 2^m$.
- If z_{m+1} is odd, the cardinality of the second expression in Equation 1 corresponds to the number of points with integer coordinates in the $m+1$ dimensional ball centered at the origin and having radius $\sqrt{\frac{m+1}{4}}$, which is bounded by $2^{1+(m+1)c}$, where c is a constant described in [11] ($c = 2.047\dots$).

Thus we get that $|\{\mathbf{v} \in D \mid \|\mathbf{v}\| \leq \|\mathbf{e}\|\}| \leq 2^c$. All that remains is to approximate the first term in Equation 1.

We now **diverge** from the original proof and investigate what happens if the a_{ji} are not sampled uniformly at random, but rather form a sparse set, following some unknown distribution. This observation is related to the way in which a_{ji} are induced by the multigraph in the blockchain we described. Thus:

$$\begin{aligned}\Pr\left[\sum_{i=1}^{m} a_{ji}(v_i - v_{m+1}) = 0 : j \in [n] \wedge \mathbf{v} \notin \{\mathbf{0}, \pm\mathbf{e}\}\right] &\leq \Pr\left[\sum_{i=1}^{m} a_{ji} z_i = 0 : j \in [n]\right] \\ &= \prod_{j=1}^{n} \Pr\left[\sum_{i=1}^{m} a_{ji} z_i = 0\right]\end{aligned} \tag{2}$$

We stress that the form of z_i obtained in [15] differ from the form of z_i we use, due to the fact that in our version of the problem, the target sum in the MDSSP is 0. As an observation, the previous probability bound we obtain in Equation 2 can be equivalently stated: $\Pr[\sum_{i=1}^{m} z_i \cdot a_{ji} = 0] \iff \Pr[\mathbf{z}^t \cdot \mathbf{a_j} = 0]$, where $\mathbf{a_j} = (a_{j1}, \ldots, a_{jm})$.

Let $\mathbf{E^t}$ be the matrix defined by a_{ji} (example given in Figure 2). The condition $\Pr[\mathbf{z}^t \cdot \mathbf{a_j} = 0]$ states that $\mathbf{z}$ is in the left nullspace of the matrix $\mathbf{E^t}$ (which is sparse, given that the a_{ji} form a sparse set). Because $\mathbf{e}$ is already in the left null-space ($\mathbf{e}^t \cdot \mathbf{E^t} = \mathbf{0}^t$), the problem to solve becomes now to find the probability that $\mathbf{z}$ exists and that it is shorter than $\mathbf{e}$.

If the matrix $\mathbf{E^t}$ has rank $n-1$ then the dimension of the left nullspace is 1 (following from the Rank-Nullity theorem); hence $\mathbf{z}$ is an integer multiple of $\mathbf{e}$, thus failing to have a shorter norm than $\pm\mathbf{e}$. Finally, we estimate the probability of $\text{rank}(\mathbf{E^t}) < n-1$. Observe the form of $\mathbf{E^t}$, as the matrix associated to a random "hub" multigraph ($\exists i$ such that $\forall j, a_{ji} \neq 0$). If there exists a row j for which $a_{ji} \neq 0$, then we can apply elementary matrix operations, such that $\mathbf{E^t}$ will have a sub-matrix of size $n-1$ which is diagonal.

Hence, we used the hypothesis to prove that $\Pr[\mathbf{z}^T \cdot \mathbf{a_j} = 0] = 0$, which is equivalent to the claim that there is no shorter vector than $\pm\mathbf{e}$ in L, when $L = (\mathbf{I_m}|\mathbf{E}^t)$, with $\mathbf{E}$ being the matrix of a "hub" graph. As shown above, for such graphs, $\Pr[X = \emptyset] = 1$, which completes the proof. □

4 Faster NCP Solving

While the lattice reduction approach discussed in Section 3.3 cannot be efficiently applied directly on a large multigraph to find a solution to the NCP, it can work on small multigraphs. We describe in this section a simple pruning algorithm that reduces the problem size dramatically. This algorithm breaks down the NCP instance into many smaller NCP sub-instances, which can be tackled by LLL. Furthermore, each instance can be dealt with independently, which makes our approach parallelizable. Importantly, nothing is lost in this divide-and-conquer strategy: any solution in the original instance can be found in the pruned instance(s).

In other terms, we first leverage the particular form NCP — namely the graph-related properties — to conservatively reduce problem size. This is possible thanks to the following two observations:

1. We only need to consider strongly connected components. Indeed, if $v, w \in V$ belong to two different strongly connected components of G, then by definition there is no path going from v to w and back. Therefore any amount taken from v cannot be returned, so that the balance of v cannot be conserved. Thus, all the edges of $\widetilde{E}$ are contained in a SCC of G.
2. Let H be a nilcatenation of G. Then H must satisfy a "local flow conservation" property: the flow (Definition 6) through any cut of H is zero; equivalently, the input of each vertex equates the output. Subgraphs failing to satisfy this property are dubbed *obstructions* and can be safely removed.

Definition 5 (First-Order Obstruction). *Let $G = (V, E, \phi)$ be a weighted multigraph. A vertex $v \in V$ is a* first-order obstruction *if the following conditions hold:*

- *The in-degree and out-degree of v are both equal to 1.*
- *The weights of the incoming and the outgoing edges are different.*

We may define accordingly "zeroth-order" obstructions, where the minimum of the in- and out-degree of v is zero (but such vertices do not exist in a strongly connected component), and *higher-order* obstructions, where the in- or out-degree of v is larger than 1, still satisfying the local-flow conservation property:

Definition 6 (Local conservation SSP). *Let $v \in V$, let E_I the multiset of v's in-edges, and E_O the multiset of v's out-edges. The* local conservation SSP *is the problem of finding $S_I \subseteq E_I, S_O \subseteq E_O$ such that $\sum_{e \in S_I} \phi(e) = \sum_{f \in S_O} \phi(f)$.*

4.1 Strongly Connected Components.

It is straightforward to see that a partition of G into k strongly connected components corresponds to a partition of E into $(k+1)$ multisets: each strongly connected component (SCC) with its edges, and a remainder of edges that do not belong to SCCs. As explained above, this remainder does not belong to $\widetilde{E}$.

The partition of a graph into strongly connected components can be determined exactly in linear time using for instance Tarjan's algorithm [18]. To each component, we can associate a descriptor (for instance a binary vector defining a subset of E), and either process them in parallel or sequentially, independently.

This corresponds to reordering V so that E is a block diagonal matrix, and working on each block independently.

Fig. 3. The pruning algorithm, used to split components which fail to satisfy the local flow conservation property.

```
Data: Multigraph $G = (V, E, \phi)$
Result: Multigraphs $\{G_i = (V_i, E_i, \phi_i)\}$, having no simple obstruction
Function Pruning(G):
    $\{G_1, \dots, G_\ell\} \leftarrow$ Tarjan($G$)
    foreach $G_k = (V_k, E_k, \phi_k)$ do
        foreach $v \in V_k$ do
            if $\min(d_v^+, d_v^-) = 0$ then
                remove all edges connected to $v$ in $E_i$
            else if $d_v^+ = d_v^- = 1$ then
                denote $e_{\text{in}}$ the incoming edge
                denote $e_{\text{out}}$ the outgoing edge
                if $\phi_k(e_{in}) \neq \phi_k(e_{out})$ then
                    delete $e_{\text{in}}$ and $e_{\text{out}}$ from $E_k$
                end
            end
        end
    end
    return $\{G_1, \dots, G_\ell\}$
end
```

4.2 The Pruning Algorithm

We can now describe the pruning algorithm (Figure 3), that leverages the observations of this section.

The algorithm works as follows: (1) decomposes the graph into its SCCs; then (2) removes first-order obstructions[6] in each component. Removing obstructions may split a strongly connected component in twain (we can keep track of this using a partition refinement data structure), so we may repeat steps (1) and (2) until convergence, i.e., until no obstruction is found or no new SCC is identified. This gives the obvious recursive algorithm `RecursivePruning`.

Complexity analysis. The average-time complexity of this algorithm depends a priori on the graph being considered, and in particular on how many SCCs we may expect, how probable it is that an obstruction creates new SCCs, how frequent obstructions are, etc. If we turn our attention to the worst-case behaviour, we can in fact consider a multigraph for which this algorithm would take the most time to run.

Tarjan's algorithm has time complexity $\mathcal{O}(n + m)$, and first-order obstruction removal has time complexity $\mathcal{O}(n)$. Thus the complete pruning's complexity is determined by the number of iterations until convergence. The worst graph would thus have one obstruction, which upon removal splits its SCC in two; each

[6] Higher-order obstructions can also be removed, although there is a trade-off to consider, see Remark 5.

sub-SCC would have one obstruction, which upon removal splits the sub-SCC in two, etc. Assuming that this behaviour is maintained all the way down, until only isolated nodes remain, we see that there cannot be more than $\log_2 n$ iterations.

Each iteration creates two NCP instances, each having $n/2$ vertices and around $m/2$ edges. Thus the complete pruning algorithm has worst-case complexity $\mathcal{O}((m+n)\log n)$. [7]

Remark 5. If we now extend the pruning algorithm to also detect higher-order obstructions, say up to a fixed order d, then the obstruction removal step costs $\mathcal{O}(2^d n) = \mathcal{O}(n)$ since 2^d is a constant. Thus the asymptotic worst-case complexity is not impacted. However the constant term might in practice be a limiting factor, especially since higher-order obstructions may be rare. Making this a precise statement requires a model of random multigraphs (see the open questions in Section 6). To compensate for the extra cost of detecting them, order-d obstructions should be frequent enough: we *conjecture* in an informal manner that this is not the case, and that there is no gain in going beyond the first order in most practical scenarios.

4.3 Fast NCP solving

We can now describe in full the fast NCP solving algorithm in Figure 4. It consists in first using the pruning algorithm of Section 4.2, which outputs many small NCP instances, and then solving each instance using an SVP oracle (in practice, a lattice reduction algorithm) as described in Section 3.3.

```
Data: Multigraph $G = (V, E, \phi)$
Result: Nilcatenations $\{\widetilde{G}_i\}$
Function FindNilcatenations(G):
    $\{G_1, \ldots, G_\ell\} \leftarrow$ RecursivePruning($G$)
    foreach $G_k = (V_k, E_k, \phi_k)$ do
        $\widetilde{E}_k \leftarrow \mathrm{SVP}(I|E_k)$
        $\widetilde{G}_k \leftarrow (V_k, \widetilde{E}_k, \phi_k)$
    end
    return $\{\widetilde{G}_1, \ldots, \widetilde{G}_\ell\}$
end
```

Fig. 4. The complete fast NCP solving algorithm.

Remark 6. For completeness, we mention that the algorithm in Figure 4 is theoretically *guaranteed* to return a result if the density of each problem defined by G_k and used to feed the SVP oracle is small, and G_k defines a hub-graph.

If we are only interested in the largest connected nilcatenation, as will be the case in the following section, then only the largest subgraph needs to be returned.

5 NCP-solving as a Proof of Work

A proof of work is a computational problem whose solution required (extensive) computation. Such constructions were first introduced to fight against e-mail spam,

[7] We ignore the fact that each subproblem can be worked on independently in parallel.

but they are increasingly popular at the heart of distributed cryptocurrencies, since the inception of the Bitcoin blockchain [13].

In almost all cases however, computing a proof of work requires operations that, as such, are useless. We think that this waste of energy is unnecessary, and that to a certain extent it is possible to use alternative mechanisms to achieve "community-serving" proofs of work.

The idea in what follows is to recognise as a valid proof of work the result of ledger nilcatenations. As we discussed above, the NCP is hard, and intuitively, larger nilcatenations would require more work to be found. Rewarding nilcatenations would encourage users to look for them and publish them (in the form, maybe, of "nilcatenation blocks", NCB); as a result, all users would benefit from a more compact representation. We stress here that this is **only** a possibility, and that there are implementation details to be accounted for, if this idea is integrated in any existing blockchain.

To give a flavour, we distinguish between **unpermissioned** and **permission-based** blockchains. In the former case, a typical scenario consists of an anonymous user owning multiple public/private key pairs for the transactions in which he/she is involved. Suppose the execution of a transaction involves sending an amount to an address identified through the hash of a fresh public-key; then the addresses (accounts) are not repeated multiple times. In such a case, the multigraph representation of the transactional ledger contains no loops, resulting in trivial, empty nilcatenations. In the latter case — **permission-based** blockchains — the accounts represented via addresses can be reused and therefore a representation of the set of transactions via a multigraph is possible. This enables a PoW implementation based on the NCP problem.

Theorem 4 (Proof of Work). *Let $\mathcal{B} = [\mathcal{B}_1, \dots, \mathcal{B}_n]$ stand for a transactional blockchain, blocks $\mathcal{B}_i$ being generated by a (deterministic) function $f_n : \mathcal{X}_1 \times \cdots \times \mathcal{X}_n \to \mathcal{X}_{n+1}$ sampled from a family $\{\mathcal{F}_n\}$, for $n \geq 1$. Let $G = (V, E, \phi)$ be the multigraph representation of the transactions and $(\widetilde{G}, G - \widetilde{G})$ a nilcatenation. There exists a blockchain $\mathcal{B}'$ for the multigraph $(G - \widetilde{G}, E - \widetilde{E}, \phi)$ and a blockchain $\mathcal{B}''$ for $(\widetilde{G}, \widetilde{E}, \phi)$, both obtained through $\{\mathcal{F}\}$.*

Proof (Intuition). The proof is straightforward. If the transactional multigraph can be decoupled into its nilcatenation and cleansed multigraph, two ledgers $\mathcal{B}', \mathcal{B}''$ can be generated (through the means of f) for each of these components, and the union of their transactions in $\mathcal{B}', \mathcal{B}''$ can be used to check the validity against $\mathcal{B}$. If the number of transaction in a block is fixed, dummy exchanges with a value of 0 can be artificially added. □

Concretely, an NCB is similar to "standard" blocks, but checked in a different way. Instead of containing only transactions, NCBs also contain a description of the nilcatenation. Users responsible for publishing NCBs get rewarded as a function of the size of their nilcatenations. Users receiving nilcatenation blocks would check them and accept them only if they are valid.[8]

[8] Note that NCBs need not be removed from the blockchain.

Cheating strategies. Before NCBs can be used as a proof of work, however, we must consider cheating strategies and fine-tune the incentives, so that honest computation and quick dissemination are the rational choices for miners. We identify two cheating strategies, dubbed ghost cycles and welding, for which we suggest countermeasures. We then discuss the question of how to reward NCBs.

The following subsections clarify these points; as a summary, to use NCP as a proof of work, one should: (1) require that nilcatenations obey the ghostbusting rules of Section 5.1, i.e., belong to a randomly-sampled subgraph of a snapshot of the transaction multigraph; (2) only accept connected nilcatenations as explained in Section 5.2; (3) be rewarded linearly in the size of the nilcatenation, as described in Section 5.3.

5.1 Ghost cycles

Ghost cycle creation. One attack is the following: a cartel of users may create many transactions with the sole intent to make nilcatenation easier. They may create cycles or cliques of transactions, then reap and share the reward for "finding" this removable set. In fact, they merely need to graft their transactions to an existing, large enough sequence of transactions. Such a strategy could take the following form:

1. Find the longest path of identical transactions that point to the controlled node: write them $v_i \xrightarrow{r} v_{i+1}$, with $i = 0, \ldots, n$ and v_{n+1} being the nodes under adversarial control. Note that r is fixed. Searching for such a cycle can be done by starting from v_{n+1}, and performing a depth-first search on the transaction graph.
2. Compute the expected gain of a nilcatenation-based proof of work that removes $(n+1)$ transactions: call it G_{n+1}. Such a quantity would be publicly known, and we may assume for simplicity that $G_n > G_m$ whenever $n > m$.
3. If $G_{n+1} > r$, make a new transaction $v_{n+1} \xrightarrow{r} v_0$; then send the nilcatenable cycle $\{v_0, \ldots, v_{n+1}\}$ as a "proof of work".

By using several accounts, artificially-long chains can be created by a user, only to be immediately "found" and removed. We dub these "ghost cycles" (this includes cliques and other structures as well), and this form of cheating is of course highly undesirable.

Ghostbusting. There are two (complementary) ways to combat ghosts. An economical approach, discussed in Section 5.3, consists in making ghosts unprofitable. A technical countermeasure, called *ghostbusting* is described in Appendix B, ensures that ghosts cannot be leveraged, except perhaps with negligible probability. The rationale is to ask for miners to solve the NCP on a *randomized* subset of the transaction graph, where it is very unlikely that they have enough accounts to construct ghost cycles.

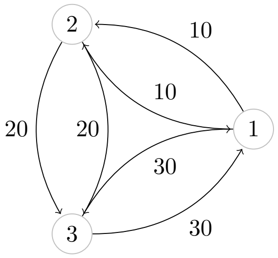


Fig. 5. Concatenation of three independent nilcatenations.

5.2 Welding nilcatenations

Another interesting question, motivated by the increased number of cryptocurrency miners who parallelize their work, is to measure how much parallel computation helps in solving the NCP. As described previously (see Section 4), the pruning algorithm generates many small graphs that can be dealt with independently.

In our scenario, after gathering enough nilcatenations published by peers, a user could assemble them into a single, larger instance and claim the rewards for it. From a theoretical standpoint, a large, disjoint nilcatenation satisfies Definition 4.

However the incentive there would be to produce quickly many small nilcatenations. This is, again, highly undesirable.

As a first countermeasure, users reject disconnected nilcatenations (this is easy to check), i.e., only accept connected ones. This encourages miners to look for larger nilcatenations, and also limits the efficiency of miner pools.

Such an approach does not prevent, in theory, users from joining together partial nilcatenations into a larger one. Consider for instance the graph of Figure 5, where user 1 finds a nilcatenation 10-10, user 2 finds 20-20, and user 3 finds 30-30. Then they may collude to generate a larger, connected nilcatenation.

However, we *conjecture* that it is a hard problem in general to assemble nilcatenations that are not disjoint into a larger one; or at the very least, that this is as expensive as computing them from scratch. Furthermore, the ghostbusting constraints reduce the possibilities of collusion by preventing adversarially-controlled nodes from participating in the nilcatenation graph.

5.3 Determining the reward

Using the NCP as a proof of work, we reward users that computed a valid nilcatenation. The exact reward should be finely tuned to provide the correct incentives. Note that this depends on whether or not the cryptocurrency applies transaction fees.

Transaction fees. If such fees apply, then creating a ghost is a costly operation from an adversarial point of view. The system should set the reward for a nilcatenation with m edges, denoted $\mathrm{reward}(m)$, to be lower than or equal to the cost of creating a ghost of size m, which we may assume is $m \cdot c$ where c is the transaction fee. We may settle for $\mathrm{reward}(m) = m \cdot c$. Similar techniques may apply where a larger spectrum of transaction fees are available.

Note that using a sub-linear reward function is counter-productive, as it encourages producing many small nilcatenations, rather than a large unique one. Conversely, using a super-linear reward function, while encouraging larger nilcatenations, also makes ghosts profitable above a certain size.

No transaction fees. If there are no transaction fees, then the aforementioned method does not apply (since $c = 0$). For cryptocurrencies that do not use transaction fees, ghostbusting (Section 5.1) limits the creation of ghost cycles. In such cases, the reward function may be an arbitrary affine function in the size of the nilcatenation.

6 Conclusion and open questions

We initiate the problem of nilcatenation, a soon-to-be pressing question for transactional graphs and distributed ledgers of appreciable size. This problem, dubbed NCP, is formalised and shown to be **NP**-complete. We introduce an algorithm, based on a combination of graph and lattice reduction techniques, that finds nilcatenations on a given transactional graph in most practical settings (and approximations thereof in other cases). Since nilcatenations are hard to find, easy to check, and useful, we suggest using them as community-serving proofs of work. We discuss the precautions and incentives of doing so, and discuss how nilcatenation blocks may complement the incentives of cryptocurrencies.

To the best of our knowledge this is the first community-serving proof of work w.r.t. ledger compression to be described and analysed in the literature.

Future research directions. As regards future research directions, this work opens many interesting questions in both the theoretical and practical fields:

- What are the graph-theoretic properties of transaction ledgers? Only very few studies address this question [16]. In particular, what would be a realistic "random labeled multigraph" model? Can anything be said about its strongly connected components?
- What is the typical size of an SCC after having run the pruning algorithm?
- How frequent are higher-order obstructions, and what is the most efficient way to detect them?
- Given measurable properties of a transaction ledger (density, degree distribution, etc.), what is the probability that our algorithm returns the optimal result? In other terms, how can the results of Section 3.4 be extended to more general settings?
- Are there profitable cheating strategies that work in spite of the proposed countermeasures?

Acknowledgements. The authors want to thank the anonymous reviewers for valuable comments. Roşie was supported by EU Horizon 2020 research and innovation programme under grant agreement No H2020-MSCA-ITN-2014-643161 ECRYPT-NET.

References

1. Bitcoin's blockchain size. https://blockchain.info/charts/blocks-size, 2016.
2. Marshall Ball, Alon Rosen, Manuel Sabin, and Prashant Nalini Vasudevan. Proofs of useful work. 2017.
3. Anja Becker, Jean-Sébastien Coron, and Antoine Joux. Improved generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 364–385. Springer, 2011.

4. Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.
5. J. D. Bruce. Purely P2P crypto-currency with finite mini-blockchain, 2013. URL: https://pdfs.semanticscholar.org/3f64/123ce97a0079f8bea66d3f760dbb3e6b40d5.pdf.
6. J. D. Bruce. The mini-blockchain scheme, 2014. URL: http://cryptonite.info/.
7. Matthijs J. Coster, Antoine Joux, Brian A. LaMacchia, Andrew M. Odlyzko, Claus-Peter Schnorr, and Jacques Stern. Improved low-density subset sum algorithms. *Computational Complexity*, 2:111–128, 1992.
8. Nick Howgrave-Graham and Antoine Joux. New generic algorithms for hard knapsacks. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 235–256. Springer, 2010.
9. Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.
10. Richard M. Karp. Computational complexity of combinatorial and graph-theoretic problems. In *Theoretical Computer Science*, pages 97–184. Springer, 2011.
11. Jeffrey C. Lagarias and Andrew M. Odlyzko. Solving low-density subset sum problems. *Journal of the ACM (JACM)*, 32(1):229–246, 1985.
12. Arjen Klaas Lenstra, Hendrik Willem Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261(4):515–534, 1982.
13. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
14. Phong Q. Nguyen and Damien Stehlé. An LLL algorithm with quadratic complexity. *SIAM J. Comput.*, 39(3):874–903, 2009.
15. Yanbin Pan and Feng Zhang. Solving low-density multiple subset sum problems with SVP oracle. *J. Systems Science & Complexity*, 29(1):228–242, 2016.
16. Dorit Ron and Adi Shamir. Quantitative analysis of the full bitcoin transaction graph. In *International Conference on Financial Cryptography and Data Security*, pages 6–24. Springer, 2013.
17. Claus Peter Schnorr. *Block Korkin-Zolotarev bases and successive minima*. International Computer Science Institute, 1992.
18. Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM journal on computing*, 1(2):146–160, 1972.

A Graphs and Multigraphs

We will make use of the following standard definitions: A *graph* $G = (V, E)$ is the data of a finite set V and $E \subseteq V \times V$, called respectively the vertices and edges of G. A sequence of edges $(s_1, t_1), \ldots, (s_n, t_n)$ such that $t_i = s_{i+1}$ for all $1 \leq i < n$ is called a *directed path* from s_1 to t_n. The *degree* of a vertex $v \in V$ is the number of edges connected to v. The *in-degree* (resp. *out-degree*) of v is the number of edges ending at (resp. starting at) v. In this work we consider an extension of graphs where edges can be repeated and are labeled: A *labeled multigraph* is denoted by $G = (V, E, \phi)$ where now E is a multiset of couples from $V \times V$ (not just a set), and $\phi : E \to \mathbb{Z}$ gives the label associated to each edge[9].

[9] We may equivalently replace $\mathbb{Z}$ by $\mathbb{Q}$. Since we know that, in practice, transactions have a finite precision, we may always think of them as integers.

We will use the following notation: If $e = (a, b) \in E$, and $r = \phi(e)$, we represent the edge e by writing $a \xrightarrow{r} b$. The definition of strongly connected components is given below and it naturally extends to multigraphs. In particular, any strongly connected component is connected; the converse does not hold.

Definition 7. *If $G = (V, E)$ is a graph, then a* strongly connected component *(or* SCC*) of G is a maximal subgraph of G such that for each two distinct nodes x and y, there exists a directed path from x to y, and a directed path from y to x.*

B Ghostbusting

A natural idea to fight ghost cycles could be to restrict which part of the transaction graph can be nilcatenated. It could be restricted in “time”, or in “space”, but straightforward approaches are not satisfactory:

- For instance, if B_t denotes the blockchain at a given time t, we may only consider a threshold time T, and only accept nilcatenations for B_s, where $t - s > T$. However this does not prevent an adversary from creating ghost cycles over a longer period of time.
- Alternatively, observe that since the transaction that “closes the cycle” originates from the cheater, we may require that the nilcatenation doesn’t contain this node. This countermeasure is easily bypassed by creating a new account whose sole purpose is to claim the rewards from the associated proof of work.

What the above remarks highlight is the need that nilcatenations are computed on a graph that is *not under the adversary’s control.*

Using the procedure described in Figure 6, we can sample a subgraph SG uniformly in the transaction graph. This procedure relies on the idea that a block on the chain depends on its ancestors, because it carries digests from all the preceding blocks (as per the blockchain mechanism). The principle of ghostbusting is that only nilcatenations among the nodes of SG should be accepted.

```
Ghostbusting($t, B_t$):
  1. Let $b_t$ be the block at time $t$
  2. seed ← $\mathbf{H}(b_t)$
  3. $SG$ ← SubGraphGen(seed)
  4. return $SG$
```

Fig. 6. The ghostbusting procedure creates a subgraph SG by hashing the defining block b_t. Miners are required to find nilcatenations in SG.

Note that the sampling procedure must be deterministic, so that verifiers can ensure that the nilcatenation indeed belongs to the authorised subgraph, and so that miners all address the same task.

Here we use a pseudorandom function $\mathbf{H}$ for which computing preimages is difficult, i.e. given y it should be hard to find x such that $\mathbf{H}(x) = y$. Most standard cryptographic hash functions are believed to satisfy this property — however we should refrain from specifically using SHA-256 itself, because Bitcoin’s proof of work results in blocks whose SHA-256 hash has a large known prefix.

A simple workaround is to use for $\mathbf{H}$ a function different from standard SHA-256, e.g. $\mathbf{H}(x) = \text{SHA-256}(0\|x)$.

The subgraph SG is obtained via `SubGraphGen` by selecting *nodes* (i.e. accounts, which may be under adversarial control), and all edges between these nodes. To be accepted, a nilcatenation should only contain nodes from this subgraph.

Proposition 2. *Assuming that the adversary has control over k out of n nodes, and that the sampled subgraph contains ℓ nodes, with $k < n/2$, the probability that at least $m \leq \ell$ of these nodes are under adversarial control is*

$$\frac{1}{2k-n} \cdot \frac{k^m}{n^\ell}\left(k^{\ell+1-m} - (n-k)^{\ell+1-m}\right).$$

In the limit that $k \ll n$, this probability is approximately $(k/n)^m$, which does not depend on the choice of ℓ.

Proof. We assume that $\mathbf{H}$ is a random oracle [4]. Thus SG is sampled perfectly uniformly in G. Thus, a given node will have probability k/n to be controlled by an adversary. There are ℓ nodes in SG, hence the probability of choosing at least m adversarial nodes is 0 if $m > \ell$ and $\Pr[C_{\geq m}] = \Pr[C_m] + \Pr[C_{m+1}] + \cdots + \Pr[C_\ell]$ otherwise, where C_p is the event where exactly p chosen nodes are under adversarial control. Since the nodes are picked uniformly at random,

$$\Pr[C_p] = \left(\frac{k}{n}\right)^p \left(1 - \frac{k}{n}\right)^{\ell-p} .$$

Therefore,

$$\begin{aligned}\Pr[C_{\geq m}] = \Pr[C_m] + \cdots + \Pr[C_\ell] &= \sum_{p=m}^{\ell} \left(\frac{k}{n}\right)^p \left(1 - \frac{k}{n}\right)^{\ell-p} \\ &= \frac{1}{2k-n}\left(k\left(\frac{k^m}{n^m}\left(1-\frac{k}{n}\right)^{\ell-m} + \frac{k^\ell}{n^\ell}\right) - \frac{k^m}{n^{m-1}}\left(1-\frac{k}{n}\right)^{\ell-m}\right) \\ &= \frac{1}{2k-n} \cdot \frac{k^m}{n^\ell}\left(k^{\ell+1-m} - (n-k)^{\ell+1-m}\right)\end{aligned}$$

Assuming $k \ll n$, we can use a series expansion in k/n of the above to get:

$$\Pr[C_{\geq m}] = \left(\frac{k}{n}\right)^m \left(1 + \frac{k}{n}(m - \ell + 1) + O((k/n)^2)\right),$$

and in particular the result follows. □

Hence, the probability that an adversary succeeds in creating a large ghost cycle when the ghostbusting procedure is used gets exponentially small.

As regards how the "seed block" b_t should be chosen, we only require that all miners and verifiers agree on a deterministic procedure to decide whether b_t is acceptable. For simplicity, we suggest to instantiate b_t as the last block in the blockchain.