# A new way of cooperative cycle detection against financial crime

## Decentralised cycle detection using cross-institutional transactions

**Ziggy Beijer**[1]
**Supervisor(s): Zeki Erkin**[1]**, Kubilay Atasu**[1]**, Lourens Touwen**[1]
[1]EEMCS, Delft University of Technology, The Netherlands

A Thesis Submitted to EEMCS Faculty Delft University of Technology,
In Partial Fulfilment of the Requirements
For the Bachelor of Computer Science and Engineering
June 22, 2025

Name of the student: Ziggy Beijer
Final project course: CSE3000 Research Project
Thesis committee: Dr. Z. Erkin, Dr. K. Atasu, L. Touwen, M. Khosla

An electronic version of this thesis is available at http://repository.tudelft.nl/.

**Abstract**

The act of masking the origin of illegal funds, to inject them into the economy in seemingly legal manners is called money laundering. Adversaries make use of money laundering to stay undetected when using illegally obtained money, from stealing, fraud, or other criminal activities. These money laundering processes often span multiple institutions or countries. To combat this, anti money laundering systems have evolved, also known as AMLs. AMLs have gone from simple rule-based approaches to using machine learning to analyse money transfer graphs. However, many money laundering operations still go undetected, particularly due to the assumption that all transaction data is centrally accessible. Yet in practice, institutions are not able to share their data with others because of privacy regulations and concerns. This restricts AMLs when deployed in a decentralised setting. This paper presents the first step towards an algorithm that allows two institutions to detect cycles between them, without these institutions exposing their own subgraphs to the other. The method uses a depth first search algorithm to find associated border vertices, then applies data reduction techniques to minimize the data shared between institutions. These border vertices are then compared to infer the presence of a cycle. While not yet deployable in real-world settings, the algorithm demonstrates improved communication and computational complexity over existing solutions and lays the groundwork for future privacy-preserving AML tools.

# 1 Introduction

Money Laundering is the act of moving illegitimate funds through multiple accounts to mask the source of money, with the aim of legalising the money by injecting it into the economy. The United Nations divided the process of money laundering into three stages: Placement, Layering, and Integration. In the placement phase, criminals place their funds into the financial system directly from the crime. In the Layering phase, criminals try to make funds untraceable, by moving them with different transactions through multiple institutions and countries. In the Integration phase, criminals try to make the money available by presenting the money as if it originated from legitimate sources. When successful, criminals can now use this money to buy assets or do investments. [10] With the increase in digitization of transactions both in real world and cryptocurrency, the ability for cybercriminals to launder money has also grown. These fraudulent transactions are hard to detect. The United Nations claim that 2 to 5 percent of the global GPD is laundered each year [10]. The world of finance can be represented in graphs, where vertices represent user accounts, and edges represent the transaction between different accounts. There are eight recognised patterns that criminals use to launder money, and one of them is the Simple Cycle pattern [1]. Financial data is very privacy sensitive, and this makes cooperation between institutions to detect adversaries much more difficult [9]. Due to regulations regarding privacy, institutions are not allowed to share any sensitive data. This results in patterns between institution being very hard to detect. The existing solutions to detect cycles in a decentralised way [8], require a lot of constant communication between institutions. This brings us to our research question:

*How can we detect simple cycles in bank transactions in a decentralised setup, while minimizing communication between institutions?*

The aim of this research is to find a solution to detect cycles between two institutions in a decentralised setup, that does not require constant communication between institutions.

In Section 2, we will explain some fundamental concepts needed to understand the paper. Section 3 will lay down related works, and earlier adjacent research done on this topic. Section 4 will state the context of the problem, and Section 5 presents our proposed solution. Section 6 will show the analysis of our solution. Section 7 will go into how responsible and ethical our research is. Section 8 talks about the limitations of the algorithm and discusses our findings, it will propose future research topics, and present a final conclusion.

# 2 Preliminaries

This section will explain all the fundamental concepts that are needed to understand this paper.

## 2.1 Simple Cycle Pattern

There are eight different recognised Anti Money Laundering Patterns [1]. One of these patterns is the Simple Cycle pattern. Given a directed graph, A cycle is defined as follows: Let $G = (V, E)$ be a graph and let $e_1, ..., e_n$ be a trail with vertex sequence $a_1, ..., a_n, a_1$. (It returns to its starting point.) The subgraph $G'$ of $G$ induced by the set of edges $e_1, ..., e_n$ is called a cycle of G. [4]. A simple cycle is then defined as being a cycle, where there are no repeated vertices other than the exception of the first and last vertex. In other words, a simple cycle does not contain any sub-cycles.

## 2.2 Decentralised setup

Multiple institutions might not want to share their data, due to privacy concerns. Because of this, institutions will not have a complete overview of all existing accounts and transactions. If this is the case, we are talking about a decentralised setup. Institutions will only be able to see their own accounts and transactions that involve at least one of their accounts.

## 2.3 Locality Sensitive Hashing (LSH)

Locality Sensitive Hashing (LSH) is a technique used to find approximate nearest neighbours in an efficient manner [7]. The core idea of LSH is to map data into different hash buckets by using random hash functions. The point of this is that these hash functions have a high probability to map similar data into the same hash buckets. There are various ways of achieving LSH, each with their own applications. We will focus on achieving LSH using MinHash signatures.

### 2.3.1 MinHash LSH

MinHash LSH is a technique for efficiently finding similar sets by approximating their Jaccard similarity. At first, each input set is transformed into a MinHash signature. The MinHash signature is computed by defining $k$ different hash functions, $h_1, h_2, \ldots, h_k$. For each hash function, we compute the hash of every element in the set, and take the minimum hash value. Repeating this process for all $k$ hash functions, we end up with a signature vector of length $k$. This vector has the property that it probabilistically preserves Jaccard similarity. Sets with similar elements will have similar MinHash signatures. The signature is then split into $b$ bands, each containing $r$ rows. Each band has its own hash table, so

each band is hashed into a bucket in this corresponding hash table. Sets that share at least one band will be placed in the same bucket in at least one of the hash tables. This way, similar sets are likely to collide in at least one bucket, making it efficient to find similar sets without comparing all sets directly.

# 3 Related Work

This chapter highlights the research that has been done on cycle detection. It is split into three sections, where each section has their own context. The first section focuses on cycle detection within a centralised context. The second section highlights a parallel cycle detection algorithm, that partitions the original graph into subgraphs, and the third section explains an algorithm that detects cycles within a decentralised context.

## 3.1 Cycle detection algorithms in centralised setup

There are many ways of detecting cycles in a centralised setup [5]. The most naive solution is to run a depth first search for each vertex on a graph with a threshold, and check if the initial vertex is visited. Tarjan proposed an algorithm [12] that divides the graph into strongly connected components. Vertices are given an index and pushed on a stack in the order the vertices were visited. Simultaneously, it tracks the lowest reachable index, also called lowlink. If the index of a vertex is equal to their lowlink, it is part of a strongly connected component. This vertex becomes the root. If the strongly connected component consists of more than one vertex, we can conclude it is in a cycle.

## 3.2 Parallel cycle detection algorithms

in 1999, Bader presented an algorithm to detect cycles in parallel [2]. By dividing the graph into subgraphs, computation can be done for each subgraph simultaneously. His algorithm describes three phases. In phase 1, each processor finds the cycles in their local subgraph. In phase 2, an express graph is constructed. The express graph consists of only border (entrance and exit) vertices. If an entrance vertex has a path to an exit vertex, an edge will be added in between them. In phase 3, express graphs are merged together, creating a new graph which you can test the presence of cycles on. Even though this solution distributes graphs to different processors, it is not completely decentralised, as the third step requires a central entity to have knowledge on the entrance and exit vertices of all subgraphs. This means that if used for financial crime detection, an entity will need to have access to parts of subgraphs of both institutions, which introduces privacy concerns.

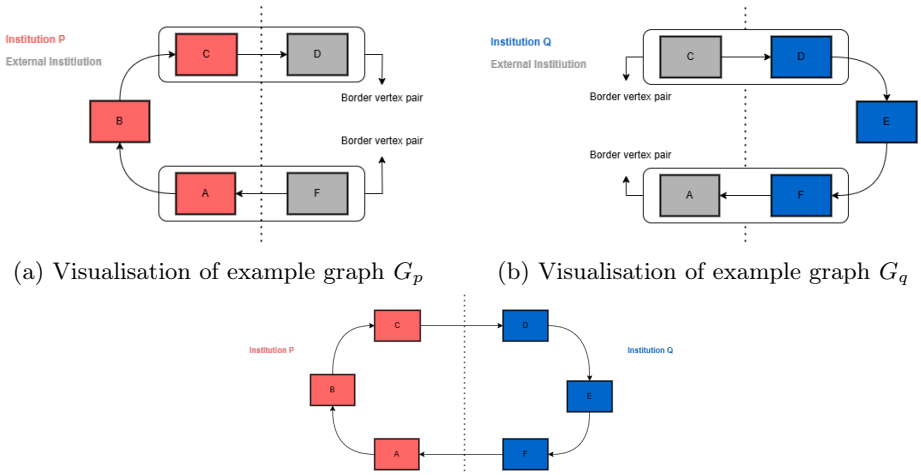
## 3.3 Cycle detection algorithms in decentralised setup

Earlier research proposes solutions to detect cycles in a decentralised setup. Jense proposed a privacy-preserving solution [8], that is able to anonymously detect cycles in a graph. That is, vertices are not able to learn any information on other parts of the graph. Each vertex independently uses the same protocol. In this protocol, a vertex can perform three actions: initiate, propagate, echo, and trace. First of all, a vertex can initiate. When initiating, the vertex sends an encrypted message to all neighbors. A vertex can also propagate. When a vertex receives a message, it propagate the message to further neighbors, but reduces the range by one. Vertices keep propagating messages, until the range equals zero, or a vertex

with no further neighbours is reached. After propagating, a vertex also sends an echo back to the source of the message. This echo will be sent back until it reaches the vertex that initiated. When the initial vertex receives an echo from itself, it can conclude its presence in a cycle. Lastly, when a cycle is detected, the initial vertex sends a trace message, to determine the path of the cycle. This solution can be deployed in a decentralised setup to detect cycles between institutions. If deployed in a decentralised setup, communications between institutions need to be very tight, as forwarding messages and echoes need to be transmitted to the other institutions. These encrypted messages add up to a lot of constant communication. Especially when an institution wants to check their entire subgraph, which could consist tens of millions of vertices. Jense states that his algorithm with threshold $l$, number of vertices $V$, and the sum of the length of all cycles $c$ has a worst-case upper bound of $O(V^l + c)$. When considering their proposed threshold of $l = 6$, this results in a communication complexity of $O(V^6 + c)$. Note that this algorithm works for more than three institutions, but there is no reduction in communication complexity when only considering two institutions.

# 4 Problem Statement

Let $G = (V, E)$ be a directed graph for financial transactions, where $V$ is the set of vertices representing different accounts, $E$ is the set of edges representing transactions made between different accounts. Each edge has two attributes, namely the amount of money transferred, and a flag that indicates if a transaction crosses between institutions. We denote this flag for vertex $i, j$ as $c_{i\to j}$. If $c_{i\to j} = 1$, the edge spans multiple institutions, if $c_{i\to j} = 0$, the edge does not span multiple institutions. In our setting, there are two institutions, $P$ and $Q$, with their own subgraphs $G_p = (V_p \subseteq V, E_p \subseteq E)$ and $G_q = (V_q \subseteq V, E_q \subseteq E)$. We assume that $G_p \cup G_q = G$ and $G_p \cap G_q \neq \varnothing$, that is to say there is overlap between $G_p$ and $G_q$. From this point, we refer to vertices in $G_p \cap G_q$ as border vertices. Because they create a "border" between the two graphs. From the perspective of one institution, there are two types of border vertices: internal and external border vertices. Internal border vertices are vertices internal to the institution. That is, an internal border vertex is an account managed by that institution, while external border vertices are vertices outside of the institution. From the perspective of one institution, a vertex might be internal, which means it will be external from the perspective of the other institution, and vice versa. Furthermore, we refer to border vertex-pairs when one internal and external border vertex are connected with a cross-institutional transaction. Moreover, we assume that the vertices in $G_p \cap G_q$ are denoted with identical identifiers. For an overview and visualisation of internal and external vertices, as well as border vertex pairs, an example of graphs $G_p$, $G_q$ and $G$ are illustrated in Figure 1a, Figure 1b, and Figure 1c respectively. A table with all terminology in this paper is found in table 1.




(a) Visualisation of example graph $G_p$

(b) Visualisation of example graph $G_q$

(c) Visualisation of example graph $G$

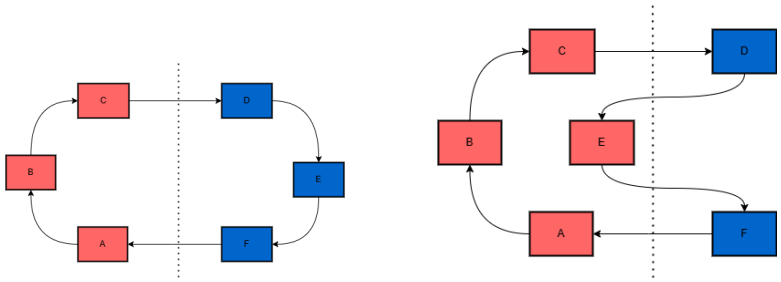
Figure 1: Example graphs

| Notation | Description |
|---|---|
| $G$ | Total transaction graph |
| $V$ | Set of all vertices in $G$ |
| $E$ | Set of all edges in $G$ |
| $c_i$ | Flag indicating cross-institutional transaction |
| $P$ | Institution $P$ |
| $Q$ | Institution $Q$ |
| $G_P$ | transaction subgraph of institution P |
| $V_P$ | Set of all vertices in $G_P$ |
| $E_P$ | Set of all edges in $G_P$ |
| $G_Q$ | transaction subgraph of institution Q |
| $V_Q$ | Set of all vertices in $G_Q$ |
| $E_Q$ | Set of all edges in $G_Q$ |
| $K$ | threshold depth |
| $O_{Pn}$ | Set of all connected outgoing external border vertices of vertex $n$ |
| $O_P$ | Set of all sets $O_{P1}$, ..., $O_{Pn}$ |
| $I_{Pn}$ | Set of all connected incoming external border vertices of vertex $n$ |
| $I_P$ | Set of all sets $I_{P1}$, ..., $I_{Pn}$ |
| $LSH_{P\text{out}}$ | Banding matrix containing all sets from $O_P$, hashes |
| $LSH_{P\text{in}}$ | Banding matrix containing all sets from $I_P$, hashed |
| $O_{Qn}$ | Set of all connected outgoing external border vertices of vertex $n$ |
| $O_Q$ | Set of all sets $O_{Q1}$, ..., $O_{Qn}$ |
| $I_{Qn}$ | Set of all connected incoming external border vertices of vertex $n$ |
| $I_Q$ | Set of all sets $I_{Q1}$, ..., $I_{Qn}$ |
| $C_{Q\text{in}}$ | Set containing incoming candidates |
| $C_{Q\text{out}}$ | Set containing outgoing candidates |

Table 1: Overview of terminology

# 5 Collaborative Simple Cycle Mining

The algorithm relies on the fact that a cycle always has to return to it's original vertex. This means we can analyse the border vertices from both sides. Consider two border vertex-pairs, $r$ and $s$. If we can detect that a vertex from institution $P$ has a connection to $r$ and a connection from $s$, while a vertex from institution $Q$ has a connection to $s$ and a connection from $r$, we can conclude these vertices are in the same cycle. To illustrate this, the graph in Figure 2a has two border vertex-pairs, namely $c \to d$ and $f \to a$. These two pairs have a different polarity, as $c \to d$ goes from institution $P$ to institution $Q$, while $f \to a$ goes from institution $Q$ to institution $P$. Since vertex $b$ and vertex $e$ are both connected to these two border vertex-pairs, we can conclude that $b$ and $e$ are in the same cycle. My proposed solution uses this key characteristic to their advantage. Note that this characteristic does not hold when inverted. In other words, if two vertices from different institutions are in a cycle, they don't necessarily share two border vertex-pairs. This is illustrated in Figure 2b. As you can see, vertex $b$ and vertex $d$ are both in the same cycle, but they are not connected to the same two border vertex-pairs. While both vertices are connected to $c \to d$, $b$ is connected to $f \to a$, while $d$ is connected to $d \to e$




(a) Institution P on the left (red); Institution Q on the right (blue)

(b) Institution P on the left (red); Institution Q on the right (blue)

Figure 2: Example graphs

## 5.1 Phase One

Our solution is divided into different phases. In the first phase, institution $P$ runs algorithm 1 on their graph $G_P = (V_P, E_P)$. The algorithm traverses the graph for each vertex $n \in V_P$ with a threshold depth $K$ to discover all outgoing connected border vertex-pairs, and adds the found external border vertices to the set $O_{Pn}$. These sets are then stored in $O_P$.

**Algorithm 1:** Outgoing External Border Vertex Discovery

**Input:** Graph $G = (V, E)$, depth $K$
**Result:** Set $O_p$ of sets of discovered vertices

Denote $N_i = \{j \mid (i, j) \in E\}, \forall i \in V$;
Initialize set $O_P \leftarrow \emptyset$;
**for** $n \in V$ **do**
    Initialize set $O_{Pn} \leftarrow \emptyset$;
    Initialize set $X \leftarrow \{n\}$;
    Initialize set $Q \leftarrow \emptyset$;
    **for** $k \leftarrow 1$ **to** $K$ **do**
        Initialize set $N \leftarrow \emptyset$;
        **foreach** $v \in Q$ **do**
            **if** $v \notin X$ **then**
                Append $v$ to $X$;
            **foreach** $j \in N_v$ **do**
                **if** $c_{i \to j} = 1$ **then**
                    Append $j$ to $O_{Pn}$;
                **else if** $j \notin X$ **then**
                    Append $j$ to $N$;
        **if** $N = \emptyset$ **then**
            **break**;
        **else**
            $Q \leftarrow N$;
    $O_P$.append($O_{Pn}$);
**return** $O_P$

After this, institution $P$ runs algorithm 2, but on their inverted graph $G'_P = (V'_P, E'_P)$. This algorithm traverses the inverted graph for each vertex $n \in V'_P$ with a threshold depth $K$ to discover all incoming connected border vertex-pairs, and adds the found external border vertices to the set $I_{Pn}$. These sets are then stored in $I_P$.

**Algorithm 2:** Incoming External Border Vertex Discovery

**Input:** Graph $G' = (V', E')$, depth $K$
**Result:** Set $I_p$ of sets of discovered vertices

Denote $N_i = \{j \mid (i,j) \in E\}, \forall i \in V'$;
Initialize set $I_p \leftarrow \emptyset$;
**for** $n \in V'$ **do**
    Initialize set $I_{Pn} \leftarrow \emptyset$;
    Initialize set $X \leftarrow \{n\}$;
    Initialize set $Q \leftarrow \emptyset$;
    **for** $k \leftarrow 1$ **to** $K$ **do**
        Initialize set $N \leftarrow \emptyset$;
        **foreach** $v \in Q$ **do**
            **if** $v \notin X$ **then**
                Append $v$ to $X$;
            **foreach** $j \in N_v$ **do**
                **if** $c_{i \to j} = 1$ **then**
                    Append $j$ to $I_{Pn}$;
                **else if** $j \notin X$ **then**
                    Append $j$ to $N$;
        **if** $N = \emptyset$ **then**
            **break**;
        **else**
            $Q \leftarrow N$;
    $I_P$.append($I_{Pn}$);

## 5.2 Phase Two

In the second phase, institution $P$ computes MinHash signatures for all sets in $O_P$ and $I_P$ computed in phase one, $O_{P1}, O_{P2}, \ldots, O_{Pn}$ and $I_{P1}, I_{P2}, \ldots, I_{Pn}$, respectively. We create two banding matrices, $LSH_{P\text{out}}$ and $LSH_{\text{Pin}}$. The MinHash signatures from $O_{P1}, O_{P2}, \ldots, O_{Pn}$ are hashed into $LSH_{\text{Pout}}$, and the MinHash signatures from $I_{P1}, I_{P2}, \ldots, I_{Pn}$ are hashed into $LSH_{\text{Pin}}$. After the banding matrices are constructed, institution $P$ sends both banding matrices over to institution $Q$.

## 5.3 Phase Three

The third phase happens after institution $Q$ receives the banding matrices from institution $P$. Institution $Q$ runs algorithm 3 on their graph $G_Q = (V_Q, E_Q)$. The algorithm traverses the graph for each vertex $n \in V_Q$ with a threshold depth $K$ to discover all outgoing connected border vertex-pairs, and adds the found internal border vertices to the set $O_{Qn}$. These sets are then stored in $O_Q$.

**Algorithm 3:** Outgoing Internal Border Vertex Discovery

**Input:** Graph $G = (V, E)$, depth $K$
**Result:** Set $O_Q$ of sets of discovered vertices

```
Denote N_i = {j | (i, j) ∈ E}, ∀i ∈ V;
Initialize set O_Q ← ∅;
for n ∈ V do
    Initialize set O_Qn ← ∅;
    Initialize set X ← {n};
    Initialize set Q ← ∅;
    for k ← 1 to K do
        Initialize set N ← ∅;
        foreach v ∈ Q do
            if v ∉ X then
                Append v to X;
            foreach j ∈ N_v do
                if c_{i→j} = 1 then
                    Append i to O_Qn;
                else if j ∉ X then
                    Append j to N;
        if N = ∅ then
            break;
        else
            Q ← N;
    O_Q.append(O_Qn);
return O_Q
```

After that, institution $Q$ runs algorithm 3 on their inverted graph $G'_Q = (V'_Q, E'_Q)$. The algorithm traverses the graph for each vertex $n \in V'_Q$ with a threshold depth $K$ to discover all outgoing connected border vertex-pairs, and adds the found internal border vertices to the set $I_{Qn}$. These sets are then stored in $I_Q$.

**Algorithm 4:** Incoming Internal Border Vertex Discovery

**Input:** Graph $G' = (V', E')$, depth $K$
**Result:** Set $I_Q$ of sets of discovered vertices

```
Denote N_i = {j | (i, j) ∈ E}, ∀i ∈ V';
Initialize set I_Q ← ∅;
for n ∈ V' do
    Initialize set I_Qn ← ∅;
    Initialize set X ← {n};
    Initialize set Q ← ∅;
    for k ← 1 to K do
        Initialize set N ← ∅;
        foreach v ∈ Q do
            if v ∉ X then
                Append v to X;
            foreach j ∈ N_v do
                if c_{i→j} = 1 then
                    Append i to I_Qn;
                else if j ∉ X then
                    Append j to N;
        if N = ∅ then
            break;
        else
            Q ← N;
    I_Q.append(I_Qn);
return I_Q
```

### 5.4 Phase Four

In the fourth phase, institution $Q$ computes MinHash signatures for all sets in $O_Q$ and $I_Q$ computed in phase three, $O_{Q1}, O_{Q2}, \ldots, O_{Qn}$ and $I_{Q1}, I_{Q2}, \ldots, I_{Qn}$, respectively. Now, for each vertex $n \in V_Q$, we query $LSH_{P\text{out}}$ with the hash signature of $I_Q n$, which returns a set $C_{Q\text{in}}$ of all vertices that have a connection to $n$. We also query $LSH_{P\text{in}}$ and $O_Q n$ to get a set $C_{Q\text{out}}$ of all vertices that have a connection from $n$. We compute the intersection between $C_{Q\text{in}}$ and $C_{Q\text{out}}$. If the intersection is not empty, we can conclude $n$ is part of a cycle.

## 6 Analysis

### 6.1 Complexity Analysis

#### 6.1.1 Communication Analysis

To find the worst-case upper bound in terms of the communication between the two institutes, we only have to look at the size of our banding matrix, as this is the only thing shared between the institutions. MinHash signatures of size $s$ will be computed for every vertex

$V$, and these are divided into bands $b$, which are hashed into buckets for each band. $s$ is a constant, so the final worst-case upper bound size of these hash tables is $O(bV)$.

#### 6.1.2 Computational Analysis

To find the worst-case upper bound in terms of our computational complexity, we first have to look at the subgraph of institution $p$, $G_P(V_p, E_p)$. Phase one deploys an algorithm that performs a depth first search from each vertex $V_p$ with a threshold $k$. Therefore, the worst-case upper bound in terms of time complexity of phase one is $O(V_p^2 + V_p E_p)$. In phase two, we create minhash signatures for the incoming and outgoing border-vertex sets of each vertex. In the worst case, every vertex is also a border-vertex within our threshold, which means that the time complexity results to $O(V_p^2)$. Deriving from the communication analysis, we know that constructing and sharing the banding matrices costs $O(bV_p)$, where $b$ is the amount of bands. Now that the banding matrix has been shared, and received by institution $q$, we have to look at the subgraph of $q$, $G_q(V_q, E_q)$. In the third phase, we run a similar algorithm to phase one, where we perform a depth first search from each vertex. After that we query the banding matrix and compute an intersection, which are both linear in terms of time complexity. This means the worst-case upper bound for phase three is $O(V_q^2 + V_q E_q)$. This means that for the whole algorithm, when adding all phases together, we get to $O(V_p^2 + V_p E_p + V_p^2 + bV_p + V_q^2 + V_q E_q)$. After simplifying, the worst-case upper bound in terms of computational complexity will evaluate to $O(V_p^2 + V_p E_p + bV_p + V_q^2 + V_q E_q)$. Considering the global graph $G = (V, E)$, we can make an assumption that $V_p \cup V_q \approx V$, and $E_p \cup E_q \approx E$. This allows us to simplify further by combining terms, resulting in the final worst-case upper bound in terms of computational complexity of $O(V^2 + VE + bV)$, where $b$ is the number of bands used for signature banding.

### 6.2 Performance evaluation

To evaluate our algorithm, we used the *HI-Small_trans* dataset from AMLWorld [1][11]. However, this dataset consists of many different institutions. Because we only consider two institutions, we filter out all but two institutions from the dataset and construct two graphs for them. We add an additional flag to each edge, which indicates if that edge crosses between institutions. After we construct the two graphs, we are able to run our algorithm on these. To evaluate our runtime performance, we chose the two biggest institutions in the dataset. These are institutions "070" and "012", with 39,735 and 12,180 vertices respectively. The tests of running the algorithm on these two institutions are performed on a single machine (AMD Ryzen 7 6800H, 16GB RAM) with a threshold $K$ of 6. Over 5 runs, the algorithm took of 3 hours and 54 minutes to complete on average.

## 7 Responsible Research

When doing research on financial topics, especially money laundering, it is important to keep in mind which impact your results might have on adversaries. The research might be very helpful for adversaries to improve their money laundering evasion tactics. We believe our research helps in the fight against adversaries, by presenting a new way for institutions to detect cycles in a decentralised setup. Furthermore, we believe this research minimally aids adversaries in the money laundering process, because this research does not present any new concepts that could be used by adversaries to evade detection algorithms. To test our

solution, we use the synthetic dataset AMLWorld [1][11]. Because this data is generated, there was no risk of leaking privacy sensitive information while doing this research.

An important thing to mention is that our algorithm is not fully privacy-preserving. This needs to be taken into account when deploying this algorithm in real life contexts, to make sure no privacy sensitive data is leaked. Before this algorithm is implemented in a real life context, analysis needs to be done to confirm there is no possibility for personal data to be leaked, and if there is a possibility of privacy sensitive data leakage, measures need to be taken to exclude this possibility. Ultimately, the risk of leaking data when implementing this algorithm lies with the person responsible for implementing the algorithm, so it is very important to be aware of this risk. That being said, an example of how to implement the algorithm has been made publicly available on GitHub, at https://github.com/ziggybeijer/Collaborative-Simple-Cycle-Mining-research-project [3]

# 8 Discussion

## 8.1 Practical Applicability

At this stage, this solution is not ready for deployment in real-world scenarios. There are two primary reasons for that. First, the algorithm at this time is not fully secure. We use MinHash signatures together with a banding matrix, to reduce data shared and find similar sets. These signatures are hashed into hash tables, which reduces redundancy. Unfortunately, this does not inherently provide data security. A paper by Turati et al. [13] shows that locality sensitive hashing does not guarantee privacy. They demonstrate that they are able to obtain sensitive data from LSH structures using Sybil attacks [6]. As a result, our algorithm cannot currently be considered fully privacy-preserving. Second, we were unable to properly evaluate the algorithm's effectiveness due to the lack of a publicly available dataset that is well-suited to our specific use case. We will elaborate on this in the limitations section. Only after these challenges are sufficiently addressed should the solution be considered for real-world application.

## 8.2 Algorithmic Limitations

The proposed algorithm has several technical limitations. It is limited to detecting cycles between exactly two institution; cycles that involve more than two institutions are not detected. Furthermore, within two institutions not every possible cycle can be identified. In particular, cycles that loop multiple times between two institutions will also remain undetected.

## 8.3 Evaluation Dataset Challenges

We evaluated our algorithm using the AMLWorld dataset, which was the best publicly available option. However, this dataset presents significant limitations for our context. The dataset consists of thousands of institutions. This results in connections between two single institutions to be very sparse. Most institution pairs do not share a single edge, and those that do often share very few. This extreme sparsity makes it difficult to assess the algorithm's performance, as the dataset does not accurately reflect realistic interaction patterns between two collaborating institutions. Unfortunately, at this point in time, there does not exist a dataset designed for two institutions specifically, which is also publicly available.

| | Our proposed solution | Jense's proposed solution [8] |
|---|---|---|
| Communication Complexity | $O(bV)$ | $O(V^6 + c)$ |
| Computational Complexity | $O(V^2 + VE + bV)$ | $O(V^6 + c)$ |

Table 2: Communication and Computational complexity of our proposed solution and Jense's solution [8].

### 8.4 Comparison to existing solution

When institutions aim to evaluate their entire graph, our solution offers improved communication and computational complexity compared to the method proposed by Jense [8]. This can be shown by comparing the worst case upper bound communication and computational complexities, presented in Table 2. However, Jense's method is more suitable for use cases involving queries on specific vertices, since our algorithm processes the entire graph in a single run.

## 9 Conclusions and Future Work

### 9.1 Conclusion

This paper, presents the first step towards detecting cycles between two institutions while minimizing communication costs. Although not yet ready for deployment in practical settings, the algorithm offers a promising foundation to combat money laundering in the future. The algorithm is useful when two institutions need to analyze their entire transaction graphs under constrained communication bandwidth. The communication and computational complexity of this algorithm are better than previous solutions. However, if institutions already possess a shortlist of suspicious vertices, or if bandwidth and computational power are not an issue, other solutions are more applicable, such as the algorithm proposed by Jense [8].

### 9.2 Future Work

Future work on this topic should first of all focus on researching how to make the proposed algorithm fully privacy-preserving. For example, research can be done to see if combining homomorphic encryption with locality sensitive hashing results in a fully privacy-preserving algorithm that still has a reasonable communication and computational complexity. Research could also be done to see if oblivious hashing could be used in combination with locality sensitive hashing, to design a fully privacy-preserving algorithm. Having a fully privacy-preserving algorithm would make our solution more practical to deploy in a real life setting. Other future research could focus on the possibility of modifying the algorithm to support more than two institutions, so our solution would be more practical to use in settings with numerous different institutions.

# A Appendix

# References


[1] Erik Altman, Jovan BlanuÅ¡a, Luc von NiederhÃ€usern, Beni Egressy, Andreea Anghel, and Kubilay Atasu. Realistic Synthetic Financial Transactions for Anti-Money Laundering Models. *Advances in Neural Information Processing Systems*, 36:29851–29874, December 2023.

[2] D A Bader. A Practical Parallel Algorithm for Cycle Detection in Partitioned Digraphs. January 1999.

[3] Ziggy Beijer. Github repository, June 2025. https://github.com/ziggybeijer/Collaborative-Simple-Cycle-Mining-research-project.

[4] Edward A Bender and S Gill Williamson. *Lists, Decisions and Graphs.* the University of California, San Diego, 2010. 164–165.

[5] Thomas H. Cormen, editor. *Introduction to algorithms.* MIT Press, Cambridge, Mass, 3rd ed edition, 2009. OCLC: ocn311310321.

[6] John R. Douceur. The sybil attack. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, page 251â260, Berlin, Heidelberg, 2002. Springer-Verlag.

[7] Omid Jafari, Preeti Maurya, Parth Nagarkar, Khandker Mushfiqul Islam, and Chidambaram Crushev. A Survey on Locality Sensitive Hashing Algorithms and their Applications, February 2021. arXiv:2102.08942 [cs].

[8] Juno Jense. Finding Bounded-Length Cycles in Decentralised Networks under Privacy Constraints. 2024.

[9] Foivi Mouzakiti. Cooperation between Financial Intelligence Units in the European Union: Stuck in the middle between the General Data Protection Regulation and the Police Data Protection Directive. *New Journal of European Criminal Law*, 11(3):351–374, September 2020. Publisher: SAGE Publications Ltd STM.

[10] United Nations. *Money Laundering*, 2020. https://www.unodc.org/unodc/en/money-laundering/overview.html.

[11] IBM Research. IBM Transactions for Anti Money Laundering (AML), 2023. https://www.kaggle.com/datasets/ealtman2019/ibm-transactions-for-anti-money-laundering-aml.

[12] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.

[13] Florian Turati, Karel Kubicek, and Carlos Cotrini. Locality-Sensitive Hashing Does Not Guarantee Privacy!Attacks on Google's FLoC and the MinHash Hierarchy System. *Proceedings on Privacy Enhancing Technologies.*