An algorithmic approach to handle circular trading in commercial taxing system

Jithin Mathews
Department of Computer Science
Indian Institute of Technology
India
Email: cs15resch11004@iith.ac.in

Priya Mehta
Department of Computer Science
Indian Institute of Technology Hyderabad
India
Email: cs15resch11007@iith.ac.in

S.V. Kasi Visweswara Rao
Department of Commercial Taxes
Government of Telangana
India
Email: svkasivrao@gmail.com

Ch. Sobhan Babu
Department of Computer Science
Indian Institute of Technology Hyderabad
India
Email: sobhan@iith.ac.in

***Abstract*—Tax manipulation comes in a variety of forms with different motivations and of varying complexities. In this paper, we deal with a specific technique used by tax-evaders known as circular trading. In particular, we define algorithms for the detection and analysis of circular trade. To achieve this, we have modelled the whole system as a directed graph with the actors being vertices and the transactions among them as directed edges. We illustrate the results obtained after running the proposed algorithm on the commercial tax dataset of the government of Telangana, India, which contains the transaction details of a set of participants involved in a known circular trade.**

***Index Terms*—data mining, bigdata analytics, social network analysis, circular trading, forensic accounting, value added tax.**

I. Introduction

Fraudulent activity, unfortunately, is inherent in our society from time immemorial. It is primarily motivated by the unscrupulous desire of people to make personal benefits by exploiting the loopholes in the existing laws in a system. Certain types of fraudulent activities are easier to identify and scrutinize. On the other hand, there are fraudulent methods that are extremely difficult to track down due to the complexity of the processes involved in handling them. In [1], Van Vlasselaer et al. gives a formal, concise and complete definition of 'fraud':

"Fraud is an uncommon, well-considered, imperceptibly concealed, time-evolving and often carefully organized crime which appears in many types of forms."

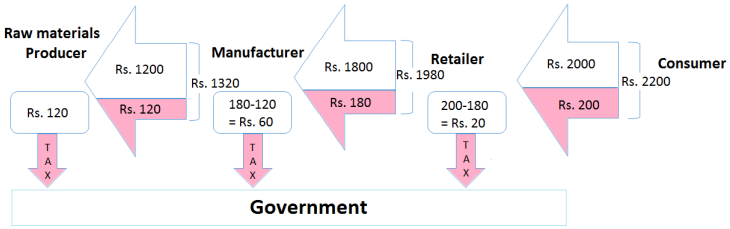
In this paper, we propose a systematic technique using social network analysis to handle a complicated type of financial fraud that is widely rampant in the commercial taxing system, known as *circular trading*. It is committed by business entities with the intention of evading tax which they are liable to pay to the government. *Circular trading* is a theft of Value Added Tax (or VAT) from the government by a business entity by creating fictitious business firms and diligently organizes with them to manipulate the financial information submitted in their commercial tax return filing. A detailed explanation with motivation and an illustration of *circular trading* is given in the coming paragraphs. It is similar to the infamous *carousal fraud* [2] which is a comparatively less sophisticated method used by fraudsters for tax evasion. *Bill trading* [3] is another technique used in tax evasion where a dealer sells some goods to another dealer without raising an invoice, but collects the tax from him. The former dealer then issues fake invoice to a third dealer, who uses it to minimize his tax liability. Note that, for conducting the proposed research work we have used the commercial tax data set shared by the Telangana state government, India.

In VAT system, when a business dealer, say dealer B, purchases some goods from another dealer, say dealer A, dealer B is liable to pay a certain amount of tax on the purchased goods to dealer A and let us call it as the $input\,tax$ paid by dealer B to dealer A on the business transaction. Similarly, when dealer B sells these goods to another dealer, say dealer C, dealer B will receive a certain amount of tax on the sold goods from dealer C and let us call it as the $output\,tax$ received by dealer B from dealer C on the business transaction. In this case, the amount of tax received by the government from dealer B is equal to the difference between the $output\,tax$ received by B and the $input\,tax$ paid by B. In other words, $tax\,payable = (output\,tax\,received - input\,tax\,paid)$.

This formula is universal for any business dealer. However, when this difference becomes a negative value, $i.e.$, when the $input\,tax$ paid becomes greater than the $output\,tax$ received, the dealer will receive Credit Carry Forward (or CCF) [4], which (s)he can claim from the government or can use it against paying tax in the future. In Figure 1, we pictorially illustrate the flow of money in a value added taxing system. Note that through out the paper cash is represented in Indian currency "$Rupees$" ($Rs.$ or ₹). Here, the producer, who makes raw materials, sells them to a manufacturer for ₹ 1200 imposing 10% of tax and thereby collecting ₹ 120 in tax. Since producer does not have any input tax, the $tax\,payable$ is ₹ 120 and he pays it to the government. The manufacturer processes

the raw materials, makes it into a product and sells it to a retailer for a higher price. Here he collects a tax of ₹ 180 from the retailer. The amount of tax that the manufacturer needs to pay to the government as a result of the previously mentioned value addition is, $tax\,payable = (180 - 120) =$ ₹ 60. Finally, the retailer adds more value to the product, like, the packing of the product, and sells it to a consumer for a higher price by collecting a tax of ₹ 200. In this case, $tax\,payable$ by the retailer to the government is $(200 - 180) =$ ₹ 20. Hence a total of ₹ 200 (= ₹ 120 + ₹ 60 + ₹ 20) is collected by the government from different stages of this transaction.

Fig. 1: Flow of money in VAT system



A. *Circular trading*

The primary motivation for *circular trading* is to hide malicious sales and(or) purchases information from the tax enforcement officers, and this is done by superimposing those transactions by carefully fabricated transactions. The classical theme in such an evasion is described in the following steps:

Step 1. Dealer A would purposefully omit some of his/her sales and purchases information in the tax returns. These malicious tax-return information will result in the reduction of the dealer's $tax\,payable$ and he/she ends up paying less tax to the government. However, this cannot continue for longtime since the dealer's financial growth may not be in proportion to the amount of tax (s)he pays and consequently becomes more likely to get caught.

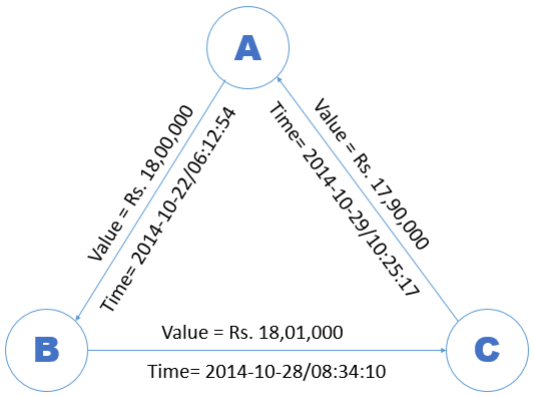
Step 2. Guided with the intention to hide the manipulation in his/her tax returns, dealer A will create a few fictitious dealers using the personal identification details of his/her trusted acquaintances.

Step 3. At this stage, dealer A will fabricate numerous sales and purchases information between himself and the fictitious dealers by making sure that the fabricated sales and purchases information are liable to a negligible amount of tax. The tax payable on these fictitious transactions is almost zero since they amount to almost zero value addition.

Here, dealer A ingeniously manages to camouflage into the nexus of fictitious dealers that (s)he has created. In fact, this helps the dealer to successfully suppress his/her sales and purchases information without getting into the hands of tax enforcement officers.

Despite of the carefully orchestrated manipulations, the dealer engaged in *circular trading* cannot avoid giving rise to undesired patterns in the flow of transactions. In this paper, we exploit this facet of the manipulated tax returns. One can easily observe that the manipulation, as defined in the last three steps, will result in the formation of flow of goods in a circular manner. For example, in *Step* 3, which is illustrated in Figure 2, dealer A seems to sell some goods to another dealer, say to dealer B, and dealer B seems to sell the same kind of goods to dealer C, and finally dealer A purchases the same kind of goods from dealer C, hence completing the cycle. Note that the $Value$ of goods transferred is almost the same in all the three transactions that create the cycle. Generally, this is not a desired pattern for the flow of goods if the transactions are authentic. These cycles become much complicated to analyze with the involvement of more than 3 dealers.

Fig. 2: Circular trading



The main difficulties in identifying malicious sales transactions are the large size of the dataset, complex sequences of the fictitious information and the large number of traders involved in *circular trading*. In this paper, we propose an algorithm to remove the fictitious transactions which are superimposed on the malicious sales transactions. This allows tax authorities to identify malicious transactions in an easy manner.

The three steps detailed in this section makes the central theme for *circular trading*. Dealers who commit this fraud often adds up more complexity to the problem by exploiting the way VAT system works in a multi-jurisdictional trading. However, the concept of goods circling around in a cycle or a circular fashion remains the same. In [5], [6] and [7], the authors have investigated on *circular trading* and other related collusion techniques used in stock market trading.

II. Problem Definition

In this section, we define the problem formally using graph theoretic terminologies and give a brief overview on the methodology used for handling the same. A thorough description of the algorithm along with its correctness and time complexity is discussed in the next section.

Table 1 shows a snapshot of the dataset used. 'ID' is the unique identity number of a dealer. 'Seller's ID' and 'Buyer's

ID' shows the direction of the flow of goods, 'Time' gives the exact time of the transaction including the date, and the variable 'Value' is the amount of tax paid by the buyer to seller. For example, the second row in Table 1 can be interpreted as a dealer with ID a selling goods to a dealer with ID b on January 14^{th} of 2015 at local time 1:01:54 pm and the buyer, dealer with ID b, gives a tax of ₹ $15,000$ to the seller.

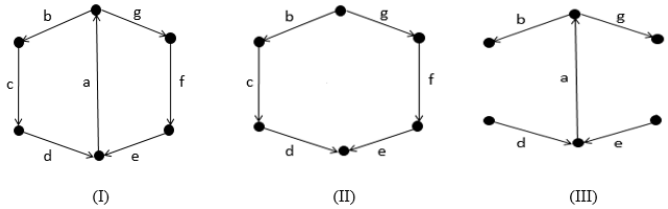
Now we define the data-structure used in storing the above mentioned dataset. The system of all transactions among all the dealers is denoted using a weighted directed graph $G = (V, E)$. Here V, which is the vertex set, is a set containing the ID's of all dealers in the transactions. A transaction is defined using a weighted directed edge, and the set of all these edges are denoted by E. The weight on any edge is a 2-tuple of its corresponding 'Value' and 'Time' attribute values, $(Value, Time)$. So the second row in Table 1 can be translated as a directed edge $\vec{ba}$ with weight $(15000, 2015/01/14/13{:}01{:}54)$. Note that graph G may contain multiple edges but no self loops. All multiple edges can be uniquely identified using the 'Time' attribute in its weight since we assume that no two transactions occur exactly at the same time.

TABLE I: Sales transactions dataset

Serial.No.	Seller's ID	Buyer's ID	Time	Value in ₹
1	m	n	2015/01/14/10:30:44	10000
2	a	b	2015/01/14/13:01:54	15000
3	x	y	2015/01/15/09:02:52	12000
4	y	m	2015/01/15/10:09:11	14000
5	b	k	2015/01/16/10:10:10	10000

As mentioned in the last section, *circular trading* results in the formation of undesired flow of goods in a circular fashion, which we call as cycles in graph theoretic terms. The problem of removing these cycles is important as the tax authorities can easily detect malicious transactions once the cycles are removed. Note that deleting an edge from a cycle results in the absence of that cycle from the graph. The order in which we delete cycles is significant since different order of edge deletion produces different directed acyclic graphs (DAGs) at the end. This is due to the simple fact that different cycles may share one or more edges among each other.

Fig. 3: Cycle deletion



For example, as illustrated in Figure 3, if a graph (given in (I)) contains two cycles that share a common edge a, $viz.$ (a, b, c, d, a) and (a, g, f, e, a), deleting edge a results in the formation of a different DAG (as given in (II)) from the DAG formed by deleting one edge each from each cycle that is not edge e, as given in (III). Hence, we chose an ordering technique for edge deletions following the guidelines given by the taxation authorities. It is described in $Observation\,1$.

Observation 1. *In circular trading a dealer fabricates sales and purchases information between himself and the fictitious dealers such that the input tax and the output tax due to the fictitious transactions are almost the same, (i.e., tax payable on the fictitious transactions are nullified).*

The $Value$ parameter of the three transactions shown in Figure 2 of Section 1 illustrates $Observation\,1$. A careful study of this observation naturally results in deleting cycles in the following particular order:

'Delete cycles in such a way that the difference between the tax values of the highest-tax-valued-edge in the cycle, (where, 'Tax value' is the second element in the 2-tuple denoting the weight of an edge), and the lowest-tax-valued-edge in the cycle is minimized.'

Using this technique, we force our algorithm to prioritize the deletion of cycles with all its edges having almost the same flow before deleting other cycles. In the next section, we define the complete algorithm and its proof of correctness along with analyzing the time complexity of the algorithm.

III. Design and analysis of the algorithm to delete cycles from a weighted directed graph in a particular order

The entire technique of deleting cycles is covered in algorithms 1, 2 and 3. Algorithm 1 invokes a function defined in algorithm 2, which in turn invokes a function defined in algorithm 3. We give the complete algorithm, a brief overview of the same, along with its proof of correctness and time complexity analysis in this section. But first, let us define few terminologies:

If there exist multiple edges from vertex x to vertex y, then $max(e_{xy})$ denotes the edge with the maximum $Value$ among all edges directed from x to y.

Critical edge of a path P or a cycle C in a graph is an edge in the corresponding path or the cycle with the minimum $Value$. We denote it by γ_P or γ_C, respectively.

Maxflow path from a vertex x to a vertex y in a graph is the path with the $Value$ of its *Critical edge* being the maximum among all the paths from vertex x to vertex y. We denote it by μ_{xy}. Note that vertices x and y cannot be the same, in which case we have a cycle and not a path. Hence, in such cases where $x = y$, we consider the "path", say μ_{xx} (or μ_{yy}), to be an unreachable path with the $Value$ of its *Critical edge* equals $+\infty$, $i.e.$, $Value(\gamma_{\mu_{xx}}) = +\infty$.

Flow value of a path P or a cycle C in a graph is the difference between the $Value$ of the maximum-valued-edge

and the $Value$ of the minimum-valued-edge (minimum-valued-edge is same as *Critical edge*) in the path or the cycle. We denote it by ϕ_P or ϕ_C, respectively.

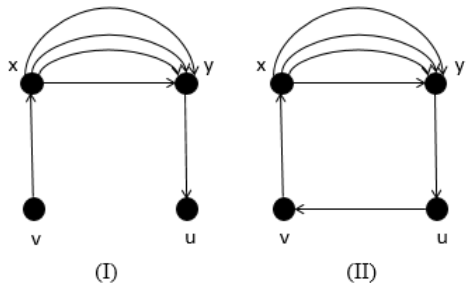
A. Algorithm overview

- In algorithm 3, we find the *Maxflow path* between two vertices, say from vertex v to vertex u in the input graph $G^{''}$, *i.e.* the path μ_{vu}, and returns it to algorithm 2.

  Here we describe the main steps involved in algorithm 3. We start by initializing two vectors, $viz.$, dist[] and parent[]. Vector parent[] is initialized to null for all the vertices, while vector dist[] is initialized to $-\infty$ for all vertices except for the source vertex v, dist[v] $= +\infty$. Then all vertices in the vertex set of the input graph $G^{''}$ is inserted into a priority queue (max-heap) based on their dist[] values. We delete the vertex in the queue with the highest dist[] value, and during each such deletion the dist[] and parent[] vectors of the deleted vertex's outgoing-neighbors (vertex n is an outgoing-neighbor of a vertex x if the graph contains an edge directed from vertex x to vertex n) are updated as follows.

  Let us call the deleted vertex as vertex ver. The dist[] and parent[] values of the outgoing-neighbors of vertex ver are updated if we find a better path from the source vertex v to the corresponding outgoing-neighbor. In other words, both vectors of an outgoing-neighbor of vertex ver are updated if the $Value$ of the *Critical edge* in the new path from vertex v to the outgoing-neighbor via vertex ver is greater than the dist[] value of the outgoing-neighbor. Note that vector dist[] is updated with the $Value$ of the *Critical edge* in the new path and vector parent[] is updated with the edge between ver and its outgoing-neighbor. The process of deleting vertices from the queue will continue until the queue becomes empty. Once all the vertices are deleted from the queue, for any vertex x belonging to $G^{''}$, other than the source vertex v, dist[x] represents the $Value$ of the *Critical edge* in the *Maxflow path* μ_{vx} in graph $G^{''}$. The *Maxflow path* μ_{vu} is returned to algorithm 2 by backtracking from the vertices present in vector parent[u] to vertex v.

Fig. 4: Cycle formation



- Algorithm 2 takes graph $G^{'}$ and an edge e as input, where edge e is directed from vertex u to vertex v. This algorithm removes a cycle C from graph $G^{'}$, which contains edge e such that its *Flow value* ϕ_C is the minimum among the *Flow values* of all the cycles containing edge e.

  It is important to note that the addition of an edge can give rise to several cycles as the graph may contain multiple edges. For example, in Figure 4, the addition of an edge from vertex u to vertex v in the graph given in (I) will create 4 different cycles as shown in (II). Hence, we need to decide which cycle is to be deleted before the other. In algorithm 2, we delete a cycle according to the order described in the last section, $i.e.$, *'Delete cycles in such a way that the difference between the tax values of the highest-tax-valued-edge and the lowest-tax-valued-edge in the cycle is minimized.'*

  In this algorithm, we invoke algorithm 3 using graph $G^{''}$, where $G^{''}$ is a copy of the input graph $G^{'}$, and the vertices u and v as parameters. Recall that algorithm 3 returns the *Maxflow path* μ_{vu} of $G^{''}$, and we store it as path P. After adding path P to a set, say set S, we delete all the edges from graph $G^{''}$ whose $Value$ is greater than or equal to the $Value$ of the maximum-valued-edge in the path P. We continue this process until no cycles are left in $G^{''}$. At this point, set S contains different paths from vertex v to vertex u. We add the edge e, $i.e.$, the edge from vertex u to vertex v, to each of the paths stored in set S. It is easy to see that after the addition of edge e set S now contains only cycles in it. Then we find the cycle, say cycle P_{min}, in set S whose *Flow value* $\phi_{P_{min}}$ is the minimum among all the cycles present in S. Finally we remove the cycle P_{min}, by subtracting a value equal to $\phi_{P_{min}}$ from each of the edges in P_{min}, and the resultant graph is then returned to algorithm 1.
- In algorithm 1, the input graph has all its edges sorted in the increasing order of time. It invokes algorithm 2, which deletes a cycle in the proposed order, until no cycles are left in the graph and the resulting directed acyclic graph is the desired output graph.

  The $Time$ and $Value$ parameters for each edge can be retrieved from its corresponding $2-$tuple $(Value, Time)$. We consider the edge set of the graph as a queue and starts deleting elements from it. Note that, always the least recent edge is deleted from the edge set as all its edges are arranged in chronological order. The deleted edges are then inserted into a new graph, say graph $G^{'}$, in the same order as they are deleted and whenever a cycle is detected in the new graph due to the addition of an edge, the function defined in algorithm 2 is invoked with the graph $G^{'}$ and the most recently inserted edge as parameters. Algorithm 2 will delete the cycle in a specific order as mentioned in the previous point, and this process

will continue until there exists no cycle in the graph. As a result, graph $G^{'}$ gets transformed into a directed acyclic graph.

B. Correctness of the algorithm

In this section, we prove the correctness of the proposed algorithm.

Let vertex v be the vertex deleted in some iteration of Step 5 in algorithm 3. One can easily verify that after the execution of Step 7, value of the vector dist[] for any outgoing-neighbor n of vertex v can be defined by the following formula:

$$dist[n] = MAX\Big(dist[n], MIN\Big(dist[v], Value\Big(max(e_{vn})\Big)\Big)\Big) \tag{1}$$

where $MAX()$ and $MIN()$ functions return the maximum and minimum values, respectively.

Lemma 3.1: After the execution of algorithm 3, for each $m \in V^{''}$, dist[m] = $Value(\gamma_{\mu_{vm}})$, where $\gamma_{\mu_{vm}}$ represents the *Critical edge* in the *Maxflow path* μ_{vm}.

Proof Assume that set O contains all vertices deleted from the Queue Q in Step 5 of algorithm 3. We use induction on $|O|$ to prove $Lemma\,3.1$.

$Base\,case$ ($|O| = 1$) : Here we prove the lemma for the first vertex inserted into set O, *i.e.*, the first vertex which is deleted from the queue Q. Initially, since dist[v] = $+\infty$ and dist[w] = $-\infty$, $\forall w \in V^{''} \setminus v$, dist[$v$] > dist[$w$]. Therefore, v is the first vertex deleted from Q in Step 5 of algorithm 3. Clearly, $Lemma\,3.1$ holds in this case as the *Maxflow path* μ_{vv}, where the source vertex and the destination vertex are the same, does not exists as per our definition of a *Maxflow path* given in the beginning of this section. Hence, for the $Base\,case$, $Lemma\,3.1$ holds and dist[v] = $+\infty = Value(\gamma_{\mu_{vv}})$.

$Inductive\,hypothesis$: Let x be the last vertex added to set O, and assume $O' = O \cup \{x\}$. In this case, the $Inductive\,hypothesis$ states that $\forall y \in O$, dist[y] = $Value(\gamma_{\mu_{vy}})$.

$Inductive\,step$: The inductive proof is complete if we show that dist[x] = $Value(\gamma_{\mu_{vx}})$.

Let $v, a_1, a_2, \cdots a_k, m, n, c_1, c_2, \cdots c_l, x$ be the *Maxflow path* μ_{vx} from vertex v to vertex x. Assume that $v, a_1, a_2, \cdots a_k, m \subseteq O$ and $n \in V'' - O$. By $Inductive\,hypothesis$, dist[m] = $Value(\gamma_{\mu_{vm}})$. Let P and P' denote the sub-paths $v, a_1, a_2, \cdots a_k, m$ and $v, a_1, a_2, \cdots a_k, m, n$ of the *Maxflow path* μ_{vx}, respectively. Clearly, $Value$ of the *Critical edge* of the sub-path P' is equal to:

$$Value(\gamma_{P'}) = MIN\Big(Value\big(\gamma_P\big), Value\big(max(e_{mn})\big)\Big) \tag{2}$$

$$\leq MIN\Big(Value\big(\gamma_{\mu_{vm}}\big), Value\big(max(e_{mn})\big)\Big) \tag{3}$$

$$\implies Value(\gamma_{P'}) \leq MIN\Big(dist[m], Value\big(max(e_{mn})\big)\Big) \tag{4}$$

Since vertex n is an outgoing-neighbor of vertex m, after the deletion of vertex m from the queue, dist[n] gets updated with the following result as derived from $Formula\,1$:

$$dist[n] = MAX\Bigg(dist[n], MIN\Big(dist[m], Value\big(max(e_{mn})\big)\Big)\Bigg)$$

Using $Inequality\,4$, the previous result can be rewritten as:

$$dist[n] \geq MAX\Big(dist[n], Value\big(\gamma_{P'}\big)\Big)$$

Consequently, dist[n] $\geq Value(\gamma_{P'})$. Since P' is a sub-path of the *Maxflow path* μ_{vx}, $Value(\gamma_{\mu_{vx}}) \leq Value(\gamma_{P'})$. Hence, dist[$n$] $\geq Value(\gamma_{\mu_{vx}})$. In Step 5 of algorithm 3, we delete the vertex with the highest dist[] value and as vertex x is deleted from the queue before vertex n, dist[x] $\geq$ dist[n] which implies dist[x] $\geq Value(\gamma_{\mu_{vx}})$.

If dist[x] $> Value(\gamma_{\mu_{vx}})$, then, the $Value$ of the *Critical edge* in the path formed by following the sequence of vertices deleted from the queue starting at vertex v and ending at vertex x, is greater than $Value(\gamma_{\mu_{vx}})$. This means that path μ_{vx} is not a *Maxflow path* from vertex v to vertex x which contradicts our assumption. Therefore, dist[x] $\not> Value(\gamma_{\mu_{vx}})$ which implies dist[x] = $Value(\gamma_{\mu_{vx}})$. Hence $Lemma\,3.1$ holds true for vertex x.

This completes the proof of $Lemma\,3.1$.

Corollary 3.2: The MAX_MIN() function defined in algorithm 3 finds the *Maxflow path* from the source vertex v to any vertex m in the input graph $G^{''}$ which can be retrieved by backtracking from the vector parent[m] to the vertex v.

Proof The above corollary is true since the vector parent[] is updated in Step 7 of algorithm 3 iff dist[] is updated, and according to $Lemma\,3.1$ for each $m \in V^{''}$, dist[m] = $Value(\gamma_{\mu_{vm}})$.

Lemma 3.3: Let $C_1, C_2, C_3, \cdots, C_{k-1}, C_k$ be the cycles present in set S ordered in the increasing order of their maximum-$Valued$-edges (we denote this ordering as $\vec{O}$) after the execution of algorithm 2. Then, the order in which these cycles are deleted from graph $G^{''}$ in Step 7 of algorithm 2 is in the reverse order of ordering $\vec{O}$, *i.e.*, cycle C_k is deleted at first, then cycle C_{k-1}, $\cdots$, C_3, C_2 and at last cycle C_1.

Proof Note that no two cycles in $\vec{O}$ can have the same maximum-$Valued$-edge. This is due to the fact that in Step 7 of algorithm 2 all edges from graph $G^{''}$ are deleted whose $Value \geq Value(M(P))$, where P is the *Maxflow path* from vertex v to vertex u in $G^{''}$ (note that, later in Step

9, the operation $P = P \cup \{e\}$ causes this *Maxflow path* P to become a cycle). Following the definition of $\vec{O}$, $Value(M(C_1)) < Value(M(C_2)) < Value(M(C_3)) < \cdots < Value(M(C_{k-1})) < Value(M(C_k))$, where $M(C)$ denotes the maximum-$Valued$-edge in a given cycle C. Suppose that $Lemma\,3.3$ is wrong, then, there exists two cycles C_i and C_l such that cycle C_i comes before cycle C_l in $\vec{O}$ and C_i is deleted before C_l from graph $G^{''}$ in Step 7 of algorithm 2. Since C_i is deleted before C_l, $Value(M(C_l)) < Value(M(C_i))$. Also, since C_i comes before C_l in $\vec{O}$, $Value(M(C_i)) < Value(M(C_l))$ and hence we reach a contradiction. Therefore, $Lemma\,3.3$ holds true.

Lemma 3.4: Algorithm 2 deletes a cycle from graph $G^{'}$ with the minimum *Flow value* among all the cycles in $G^{'}$.

Proof Let C_{min} denote the cycle with the minimum *Flow value* among all cycles in the graph $G^{'}$ given in algorithm 2. Note that $G^{'}$ is copied into graph $G^{''}$ in Step 3. In addition, note that set S contains a set of cycles belonging to $G^{''}$ from which the cycle with the minimum *Flow value* is found and deleted in steps 10 and 11, respectively. In order to prove $Lemma\,3.4$, assume the contradiction that algorithm 2 does not delete cycle C_{min} from the input graph $G^{'}$ which implies $C_{min} \notin S$. Let $C_1, C_2, C_3, \cdots, C_{k-1}, C_k$ be the cycles in set S ordered in the increasing order of their maximum-$Valued$-edges, *i.e.*, $Value(M(C_1)) < Value(M(C_2)) < Value(M(C_3)) < \cdots < Value(M(C_{k-1})) < Value(M(C_k))$, where $M(C)$ denotes the maximum-$Valued$-edge in a given cycle C. Note that every cycle in graph $G^{''}$ contains edge e, which is directed from vertex u to vertex v, since algorithm 2 is invoked by algorithm 1 in Step 7 when the addition of edge e created a cycle in graph $G^{'}$. Now let us complete the proof of $Lemma\,3.4$ using the following 3 exhaustive cases.

- $\mathbf{Case\,1}: \mathbf{Value\big(M(C_k)\big) < Value\big(M(C_{min})\big)}$
  According to $Lemma\,3.3$, C_k is the first cycle to be removed from graph $G^{''}$ in Step 7 of algorithm 2. By definition, *Flow value* of any cycle C = $\phi_C = \big(Value(M(C)) - Value(\gamma_C)\big)$. Therefore, in $Case\,1$ where $Value\big(M(C_{min})\big) > Value\big(M(C_k)\big)$, since $\phi_{C_{min}} < \phi_{C_k}$, $Value(\gamma_{C_{min}}) > Value(\gamma_{C_k})$. If we remove the common edge e (which is directed from vertex u to vertex v in graph $G^{''}$) from both the cycles, $C_{min} - \{e\}$ and $C_k - \{e\}$ are now two paths directed from vertex v to u such that $Value(\gamma_{C_{min}-\{e\}}) > Value(\gamma_{C_k-\{e\}})$. As $G^{''}$ is the input graph given to invoke algorithm 3, according to $Corollary\,3.2$ the path $C_k - \{e\}$ found by algorithm 3 should be a *Maxflow path* from v to u. However, this is not the case as $Value(\gamma_{C_{min}-\{e\}}) > Value(\gamma_{C_k-\{e\}})$. Hence *Case 1* is not valid.
- $\mathbf{Case\,2}: \mathbf{Value\big(M(C_{min})\big) < Value\big(M(C_1)\big)}$
  After the removal of cycle C_1, cycle C_{min} will be present in $G^{''}$, because, in Step 7, when all edges in $G^{''}$ whose $Value \geq Value\big(M(C_1)\big)$ are deleted, none of the edges in C_{min} are deleted as $Value\big(M(C_{min})\big) < Value\big(M(C_1)\big)$. The presence of cycle C_{min} implies the presence of the path $C_{min} - \{e\}$ that starts from vertex v and ends in vertex u. According to $Lemma\,3.3$, C_1 is the last cycle to be found in Step 7 of algorithm 2 and this contradicts $Corollary\,3.2$ as there exist the path $C_{min} - \{e\}$. Hence *Case 2* is also invalid.
- $\mathbf{Case\,3}: \mathbf{Value\big(M(C_i)\big) < Value\big(M(C_{min})\big) < Value\big(M(C_{i+1})\big)}$
  This case can easily be proved by using the arguments given in $Case\,1$ and $Case\,2$. After the removal of cycle C_{i+1} from graph $G^{''}$ in Step 7 of algorithm 2, cycle C_{min} will still be present in $G^{''}$ since in Step 7 when all edges in $G^{''}$ whose $Value \geq Value\big(M(C_{i+1})\big)$ are deleted, none of the edges in cycle C_{min} got deleted as $Value\big(M(C_{min})\big) < Value\big(M(C_{i+1})\big)$. Now recall that for any cycle C, $\phi_C = \big(Value(M(C)) - Value(\gamma_C)\big)$. Therefore, in this case, $Value\big(M(C_{min})\big) > Value\big(M(C_i)\big)$ and $\phi_{C_{min}} < \phi_{C_i}$ implies $Value(\gamma_{C_{min}}) > Value(\gamma_{C_i})$. If we remove the common edge e (which is directed from vertex u to vertex v) from both the cycles, $C_{min} - \{e\}$ and $C_i - \{e\}$ are now two paths directed from vertex v to u such that $Value(\gamma_{C_{min}-\{e\}}) > Value(\gamma_{C_i-\{e\}})$. According to $Lemma\,3.3$, C_i is the cycle found by Step 7 of algorithm 2 after the deletion of cycle C_{i+1}, but this contradicts $Corollary\,3.2$ as the path $C_i - \{e\}$ found by algorithm 3 is not a *Maxflow path* from vertex v to vertex u since $Value(\gamma_{C_{min}-\{e\}}) > Value(\gamma_{C_i-\{e\}})$. Hence *Case 3* is not valid.

This completes the proof of $Lemma\,3.4$.

Theorem 3.5: Algorithm 1 produces a directed acyclic graph by deleting all cycles from the graph $G^{'}$ in which the cycle with the minimum *Flow value* gets deleted before the other cycles.

Proof In $Lemma\,3.4$, we have already proved that algorithm 2 deletes a cycle with the minimum *Flow value* among all the cycles in graph $G^{'}$. In addition, the while loop defined in Step 6 of algorithm 1 invokes algorithm 2 (in Step 7) until $G^{'}$ has no cycles left in it. This proves the theorem.

C. Algorithm analysis

Theorem 3.6: If n is the number of vertices and m is the number of edges in the input graph $\vec{G}$ given to algorithm 1, then, algorithm 1 runs in $\mathcal{O}\big((m+n) \cdot m^2 \cdot log(n)\big)$ time in the worst case.

Proof In algorithm 3, if we are using a max heap for deleting the vertices with the largest dist[] value in Q, then, in the worst case it runs in $\mathcal{O}\big((m+n) \cdot log(n)\big)$ time. In algorithm 2, as one can easily observe, the while loop from steps $4-8$ takes the maximum amount of time. In the worst case, it may run Step 5 for $\mathcal{O}(m)$ time. Hence, algorithm 2, in the worst case runs in $\mathcal{O}\big((m+n) \cdot m \cdot log(n)\big)$ time. Finally, in algorithm 1, the while loop in steps $3-8$ may run in $\mathcal{O}(m)$ time in the worst case scenario were the addition of edges in Step 5 creates a

cycle in almost all cases. Hence, in the worst case scenario, the total time taken by algorithm 1 is $\mathcal{O}\big((m+n)\cdot m^2\cdot log(n)\big)$.

Algorithm 1 Weighted Cycle Deletion

procedure WCD($\vec{G} = (V, \vec{E})$)
▷ $\vec{G}$ is a weighted directed graph with multiple-edges and no self-loops
▷ Weight on each edge is a tuple with $Value$ and $Time$, $(Value, Time)$
▷ Edges of graph $\vec{G}$ are stored in edge-set $\vec{E}$ in their chronological order

  Initialize $G^{'} = \emptyset$
  ▷ $G^{'} = (V^{'}, E^{'})$, hence, $(G^{'} = \emptyset) \implies (V^{'} = E^{'} = \emptyset)$

  while ($\vec{E} \neq \emptyset$) **do**
    $e =$ DEQUEUE($\vec{E}$)
  ▷ Edge e is the least recent edge

    $E^{'} = E^{'} \cup e$
  ▷ Note that $V^{'}$ also gets updated in the process

    while ($G^{'}$ has a cycle) **do** ▷ DFS is used here
      $G^{'}$ = **function** DELETE_CYCLE($G^{'}, e$)
    end while

  end while
  ▷ Graph $G^{'}$ now contains the desired DAG

end procedure

IV. Case Study

We analyzed a case in which eight dealers are doing intensive *circular trading* among themselves. Figure 5 shows the details of the same in the form of a directed graph with vertices denoting the dealers, and directed edges denoting the direction of transactions along with the total amount of tax paid (in lakh of ₹, 1 lakh = ₹ 1,00,000) to the seller by the buyer.

In their monthly tax return statements, all the eight dealers show huge purchases from outside the state. Legally, they should have paid heavy taxes on all these purchases. The following points illustrate a brief overview of the transactions among them.

- The eight dealers did total purchases of ₹ 798 crores, out of which non-creditable purchases (purchases from outside the state or international imports) are ₹ 622 crores.
- They should have paid a total tax of ₹ 31.10 crores, but they paid only ₹ 4.47 crores as VAT & interstate sales tax (also known as CST).
- Hence, they evaded the payment of about 85% of tax.

They have done this by using the following ways:-

Algorithm 2 Function definition of DELETE_CYCLE()

function DELETE_CYCLE($G^{'}, e$)
▷ edge e is the most recently added edge in $G^{'}$ that formed the cycle
▷ $Value(e)$ gives the $Value$ of edge e from its ordered $2-$tuple $(Value, Time)$

  Let vertex-tuple (u, v) define the directed edge e
  ▷ *i.e.* edge e is directed from vertex u to vertex v

  Initialize set $S = \emptyset$ and $G^{''} = G^{'}$
  ▷ $G^{''} = (V^{''}, E^{''})$, hence, $(G^{''} = G^{'}) \implies (V^{''} = V^{'}$ and $E^{''} = E^{'})$

  while ($G^{''}$ has a cycle) **do** ▷ DFS is used here
    P = **function** MAX_MIN($G^{''}, u, v$)
  ▷ P denotes a path from vertex v to vertex u

    $S = (S \cup P)$
  ▷ S contains a set of ordered tuples, where each tuple denotes a path from v to u

    Delete edge $e^{''} \in E^{''}$, where, $Value(e^{''}) \geq Value(e_{max})$
  ▷ e_{max} is the edge with the largest $Value$ in P
  end while

  $\forall P^{'} \in S$, update $P^{'} = (P^{'} \cup \{e\})$
  ▷ Add edge e to each of the ordered tuple in S

  Find $P_{min} \in S$ that minimizes the difference between the $Value$ of its maximum-valued-edge and minimum-valued-edge

  ▷ e_{min} be the minimum-valued-edge in P_{min}
  Delete a flow of $Value(e_{min})$ from all the edges of $P_{min} \in G^{'}$ ▷ *i.e.*, $\forall e \in P_{min} \in G^{'}$, $Value(e) = (Value(e) - Value(e_{min}))$

  Return graph $G^{'}$

end function

- Most of the dealers have shown branch transfers (branches located in other states) which amounts to a total of ₹ 230 crores on which no tax is required to be paid.
- They have shown questionable amount of exports totalling to ₹ 105 crores on which no tax is required to be paid.
- They have shown questionable amount of interstate(CST) sales totalling to ₹ 111 crores on which a much lesser rate of tax (@2%) is applicable.

Algorithm 3 Function definition of MAX_MIN()

1: **function** MAX_MIN($(G^{''}, u, v)$)
▷ Here we use two vectors mapped to each of the vertices in $V^{''}$, $viz.$, dist[] and parent[v]
▷ $Value(e)$ gives the $Value$ of edge e from its ordered $2-$tuple $(Value, Time)$

2: $\forall w \in V^{''} \setminus v$, **Initialize** dist[w]= $-\infty$, parent[w] = $\emptyset$, dist[v] = $+\infty$, parent[v] = $\emptyset$

3: Insert all vertices in $V^{''}$ to Queue Q in decreasing order of their dist[] values
▷ $\forall x \in V^{''}$ ENQUEUE(x, Q) in decreasing order of dist[x]

4: **while** ($Q \neq \emptyset$) **do**

5: ver = DEQUEUE(Q) ▷ Delete ver from Q, where ver is the vertex with the largest dist[] value in Q

6: Let set N contains all outgoing-neighbors of ver
▷ outgoing-neighbors of a vertex v are all vertices to which v has an outward directed edge

7: $\forall n \in N$,
val = minimum(dist[ver], $Value(e_n)$)
▷ e_n is the edge with the highest $Value$ among all the edges directed from vertex ver to vertex n
If dist[n] $<$ val **then**
dist[n] = val, parent[n] = e_n

8: **end while**

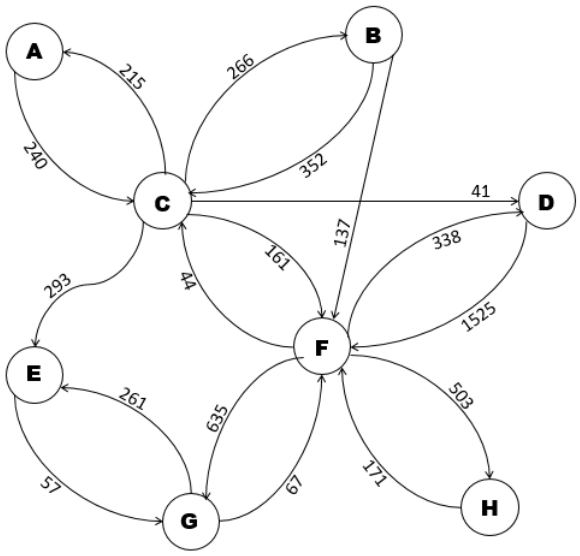
9: **Return** the path P from vertex v to vertex u
▷ Path P can be found by backtracking from the vertices present in parent[u] to vertex v

10: **end function**

- They have also shown local VAT sales of ₹ 233 crores in total on which the output tax is ₹ 11.65 crores, but have paid only ₹ 2.47 crores to the government. They could do this by raising invoices among the group members and showing Input Tax Credit (ITC). This is where *circular trading* comes into picture.

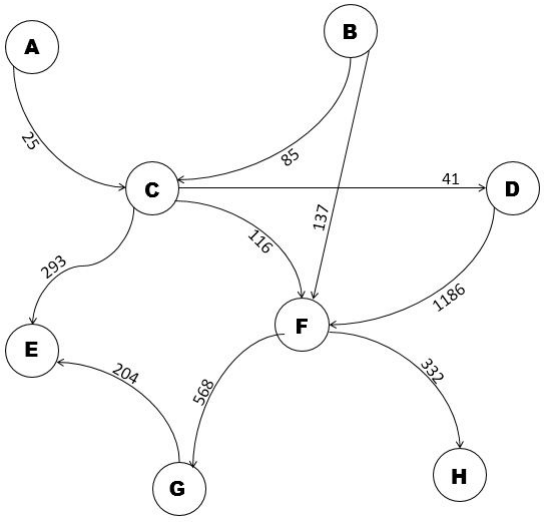
Figure 6 shows the directed acyclic graph obtained after deleting all cycles from the graph given in Figure 5 using the algorithms described above. Note that the weight on each edge in the graph given in Figure 6 shows the total tax paid by a particular buyer to a particular seller (total tax is the sum of all the tax values involved in multiple transactions between them). For example, as one can observe in the edge from vertex A to vertex C in Figure 6, the total tax involved between them after

Fig. 5: A known case of *circular trading*



deleting many transactions to remove the cycle given in Figure 5 using the proposed algorithm is ₹ 25 lakhs. It is important to note that, here the set of transactions that makes up the sum of ₹ 25 lakhs is the point of interest to tax authorities.

Fig. 6: The output DAG



V. Conclusion

In this paper, we formalized the infamous tax evasion technique called *circular trading*. In *circular trading*, a group of traders do heavy fictitious sales and(or) purchase transactions in a circular manner among themselves, without any value addition, $ie.$, the input tax and the output tax due to the fictitious transactions remains the same. The problem of removing the hence formed cycles is important as the tax authorities can easily detect malicious transactions once the

cycles are removed. Here, we proposed an algorithm to remove such cycles by making use of an important observation that the amount of tax payable by a dealer due to fictitious sales and purchases transactions is almost zero. In future, we try to define centrality measures for detecting the key players in $circular\ trading$. In addition, we plan to investigate whether there are more effective ways for removing cycles.

VI. Acknowledgment

We are very grateful to the Telangana state government, India, for sharing the commercial tax dataset which is used in the proposed work.

References

[1] V. Van Vlasselaer, T. Eliassi-Rad, L. Akoglu, M. Snoeck and B. Baesens. *"Gotcha! Network-based fraud detection for social security fraud,"* Journal of Management Science, Jul. 2016.

[2] E. Dillon, *The Fraudsters – How Con Artists Steal Your Money,* Merlin Publishing, Chapter 7, Sep. 2008.

[3] Godbole Committee, *"Report on Economic Reforms of Jammu and Kashmir"*, Ministry of Finance, Government of Jammu and Kashmir, 1998.

[4] A. Schenk and O. Oldman, *Value Added Tax: A Comparative Approach*, 1st ed., Cambridge Tax Law Series, Jan. 2007.

[5] G. K. Palshikar and M. M. Apte, *"Collusion set detection using graph clustering,"* Data Mining and Knowledge Discovery, vol. 16, Issue 2, Apr. 2008, pp. 135–164.

[6] M. Franke, B. Hoser and J. Schröder, *"On the analysis of irregular stock market trading behavior,"* Data Analysis, Machine Learning and Applications, Jan. 2007, pp. 355–362.

[7] K. Golmohammadi, O. R. Zaiane, and D. Daz, *"Detecting stock market manipulation using supervised learning algorithms,"* Data Science and Advanced Analytics, IEEE, Nov. 2014, pp. 435–441.