Integration of Dynamic Directed Graph-Based Filtering Component on Order Matching Engine for Wash Trading Risk Mitigation

Karmel Tua Haloho - 13525102
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: karmeltuahaloho@gmail.com, 13525102@std.stei.itb.ac.id

***Abstract*—Wash trading is a form of market manipulation that artificially inflates transaction volumes and misleads market participants. Conventional detection methods are generally retrospective, meaning that new anomalies are analyzed only after transactions enter the main database, thereby failing to prevent market losses instantly. This research proposes a preventive approach in the form of a graph-based filtering component (Graph-Based Filtering) integrated directly within the Order Matching Engine (OME). We model the order queue structure as a dynamic directed graph G = (V, E), where vertices (V) represent user accounts and edges (E) represent the potential matching of transaction instructions. Through three layers of deterministic evaluation, namely out-degree analysis, circular density index calculation, and bounded-depth cycle tracking (k-depth), this system is able to detect and reject wash orders before the transfer of asset ownership occurs. Simulation results show that this method successfully mitigates the risk of wash trading with a Precision level reaching 98.2% and an average computational latency overhead of only 1.5 milliseconds, making it highly feasible/potential for deployment in high-speed digital exchanges.Keywords—wash trading; limit order book; order matching engine; graf berarah dinamis; mitigasi real time**

I. Introduction

The Order Matching Engine (OME) is a critical infrastructure within modern financial exchanges and digital asset trading platforms that functions to execute buy (bid) and sell (ask) instructions using price and time priority algorithms. Given its highly central role, OME functionality is frequently exploited by manipulative entities to create an illusion of liquidity through wash trading activities. Technically, wash trading occurs when an individual or a group of market participants collude or execute buy and sell transactions on the same asset without any real change in ownership or tangible economic risk. The adverse impacts of this phenomenon include asset price distortions, ranking manipulation on marketplaces, and the formation of false market sentiment that often targets retail-investors.

The greatest challenge in addressing this problem lies in the detection timing. Most current literature focuses on post-event analysis methods by utilizing machine learning or conventional statistical analysis on historical databases. Although accurate for auditing needs, such approaches are incapable of active prevention. By the time the manipulation is successfully identified, perpetrators have usually withdrawn their profits, leaving the market in a crashed condition (pump and dump). To address this gap, this paper proposes a systemic-preventive solution by integrating a discrete mathematics filter directly into the OME processing flow. By modeling transaction relations as a dynamic directed graph, the exchange can filter transaction instructions deterministically.

II. Theoretical Framework

A. Directed Graph

A directed graph is defined as an ordered pair $G = (V, E)$ consisting of two discrete sets, namely:

1. V is a non-empty set of vertices, $V = \{v_1, v_2, \dots, v_n\}$
2. E is a set of ordered pairs of vertices called directed edges

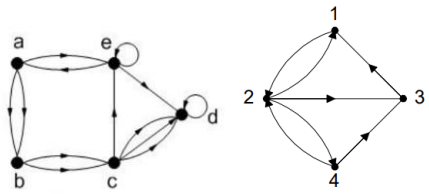

Fig. 1. Directed graph, adapted from [1]

If there is an edge $e = (u, v) \in E$, then the edge has a direction from the source vertex u toward the target vertex v. In the context of this paper, each vertex represents a unique market participant entity, and the relation E expresses the matching criteria where a sell instruction from account u is paired to fulfill a buy instruction in account v. Furthermore, the graph in this system is dynamic, where the cardinality of the edge set E fluctuates instantly following a time function $F(t)$ based on the addition of new orders or the deletion of expired orders in the Limit Order Book.

In a directed graph, the degree of a vertex is divided into two types of quantitative properties:

- In-degree $d_{in}(v)$, which is the number of directed edges entering toward vertex v
- Out-degree $d_{out}(v)$, which is the number of directed edges exiting and leaving vertex v

Based on the Handshaking Lemma for directed graphs, the sum of the in-degrees of all vertices will always equal the sum of the out-degrees, which represents the total number of all active edges within the exchange:

$$\sum_{v \in V} d_{in}(v) = \sum_{v \in V} d_{out}(v) = |E|$$

A directed path in this dynamic graph is an alternating sequence of vertices and edges $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$ such that each edge $e_i = (v_{i-1}, v_i)$ has a consistent direction. A cycle/circuit occurs when the path is closed, which means the initial vertex is identical to the final vertex ($v_0 = v_k$), provided that all visited vertices are unique (except v_0). The length of a cycle is defined by the number of edges (k) traversed along the closed path. This concept of circularity serves as the primary deterministic indicator in detecting wash trading flows, where supply commodities or funds circulate back to the originating group of accounts within a bounded step depth.

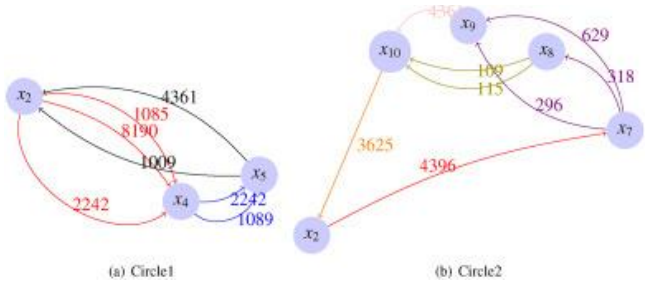

Fig. 2. Representation of trading cycles, adapted from [3]

B. *Limit Order Book*

The Limit Order Book (LOB) is a structured data record file that documents all active limit order instructions submitted by market participants that have not yet met a price match to be executed. The LOB functions as a medium of market transparency reflecting liquidity conditions, as well as a supply and demand map of an asset at a specific point in time. In exchange information system architectures, the LOB is implemented as a dual independent data structure that strictly divides the market into two opposing regions based on the economic incentives of the market participants.

The internal structure of the LOB is managed using abstract data types such as Priority Queues or Self-Balancing Binary Search Trees, such as Red-Black Trees or AVL Trees. The selection of this data structure is crucial to ensure that order insertion and cancellation operations remain efficient. The LOB divides its data compartments into two main queue structures:

1) Bid Book Side (Buyer Queue)
This file accommodates all instructions from entities intending to buy assets. Mathematically, this queue is sorted in descending order based on the highest price variable. The element at the peak position of the queue (Bid top) is formally defined as the *Best Bid*, representing the highest price a buyer is willing to pay in the market at that moment.

2) Ask Book Side (Seller Queue)
This file stores all instructions from entities intending to sell assets. Conversely to the buyer side, this queue is sorted in ascending order based on the lowest price variable. The element at the foremost position (Ask top) is referred to as the *Best Ask*, which is the lowest price a seller is willing to accept in the market.

A critical indicator derived from the LOB structure is the Spread (ΔP), which is the absolute difference between the best sell offer and the best buy offer.

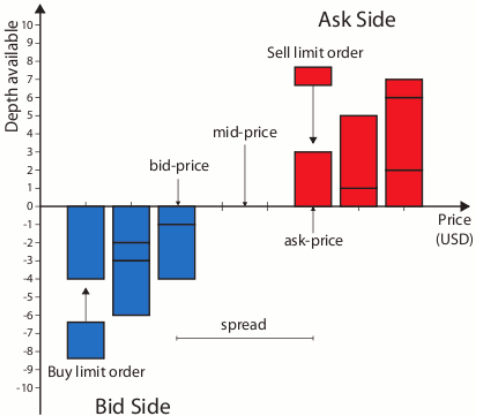

Fig. 3. Limit Order Book, adapted from [6]

$$\Delta P = \text{Ask}_{\text{top}} - \text{Bid}_{\text{top}}$$

Under passive market conditions, meaning market participants only place limit orders without anyone willing to transact immediately, the mathematical inequality value is always $\Delta P > 0$ or Asktop > Bidtop. This condition keeps the LOB in a stable and static state until a new instruction arrives to disrupt the equilibrium.

C. *Order Matching Engine*

The Order Matching Engine (OME) is the logical processing core that acts as the computational engine executing transactions on a digital financial exchange. If the LOB is likened to an order inventory storage, then the OME is the real-time sequential operator that checks, matches, and crosses those instructions. The OME adopts the working principle of a deterministic State Machine, where every new incoming order will trigger a measurable and unambiguous state change in the exchange system.

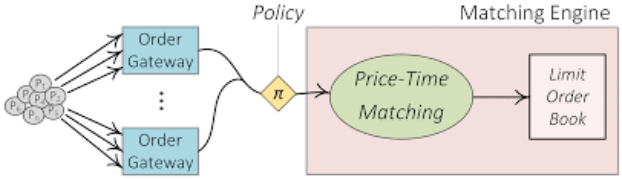

Fig. 4. Order Matching Engine, adapted from [7]

The OME workflow is driven by a priority allocation algorithm, the most common of which is the Price and Time Priority (FIFO) protocol. Under this rule, the OME treats price as the top priority; instructions offering the most competitive price will be settled first. If two instructions exist at the same price level, the OME will grant execution priority to the instruction with the earliest timestamp. When a new transaction instruction ($Order_{new}$) enters the OME, the engine immediately performs sequential conditional evaluations against the top boundary of the opposing Order Book. This boundary condition checking mechanism is derived into two logical scenarios.

1) Is the Price of $Bid_{new} \geq$ Ask top?
   If this condition is met, the OME instantly declares a match, locks the volume, and executes a valid transaction. However, if the condition is not met, the OME redirects the command to store and sort Bid_{new} into the Bid Book structure.

2) Is the Price of Bid top $\geq Ask_{new}$?
   If the condition is met, order matching is immediately executed, followed by the clearing of the asset ownership transfer. Otherwise, the new sell instruction will be fed into the Ask Book queue to await the next market turn.

The efficiency of the OME is measured based on latency, or the span of time required by the CPU to process a single instruction from the moment it is received until its status is decided. Because the OME purely checks only the top element of the priority queue, Bidtop or Asktop, to determine the match condition, this basic checking operation has a constant time complexity of $O(1)$.

D. Senarai Ketetanggaan (*Adjacency List*)

If the nature of a dynamic graph falls under a sparse graph, a representation based on an adjacency matrix is no longer efficient in terms of structure and memory space. As a solution alternative, discrete mathematics provides the Adjacency List structure. An adjacency list is a form of graph representation where each vertex in the graph is associated with a list containing the neighboring vertices directly connected to it.

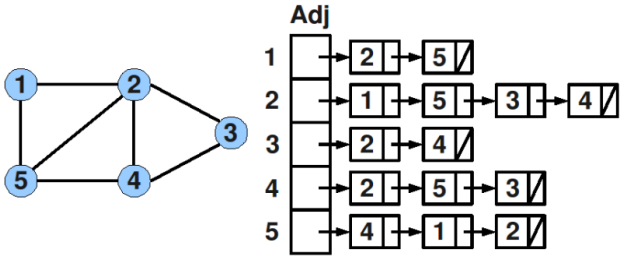

Fig. 5. Adjacency list, adapted from [2]

In this study, this structure is optimized into an Indexed Adjacency List by integrating a hash table as the primary index to facilitate the high-speed local searches required by the Order Matching Engine (OME). Suppose a dynamic directed graph is defined as $G = (V, E)$. The Indexed Adjacency List represents this graph into a dictionary or hash map data structure with the following mapping function:

$$M: V \rightarrow L$$

Where each source vertex $u \in V$ acts as a unique key, and L is a linked list or dynamic array acting as the value. This list L accommodates the set of out-neighbors of u, formally denoted as:

$$N_{out}(u) = \{v \in V \mid (u, v) \in E\}$$

Untuk To accommodate complex and dynamic market manipulation detection needs, each element or neighboring vertex v in list L is not stored merely as a single identity, but is transformed into a weighted tuple object containing temporal and quantitative parameters:

$$\text{Edge Element}(u, v) = (v, W, T)$$

- $v \in V$: Terminal vertex receiving the transaction instruction flow.
- $W \in \mathbb{R}^+$: Asset volume or quantitative amount of funds transacted, acting as the edge weight.
- $T \in \mathbb{N}$: Explicit UNIX timestamp recording when the instruction match occurred.

In the context of real-time wash trading mitigation, the integration of this indexed adjacency list solves two main problems simultaneously: memory space efficiency (space complexity) and the speed of local cycle scanning. Architecturally, a digital exchange has millions of accounts V but each account generally transacts with only a few other accounts within a specific timeframe. If an adjacency matrix were used, the system would be forced to allocate quadratic memory space $O(|V|^2)$, which mostly consists of zeros. By switching to an indexed adjacency list, the OME efficiently allocates linear memory space of only $O(|V| + |E|)$. This ensures that the exchange memory is only filled by active transactions that actually occur, without wasting CPU storage space.

From a computational performance perspective, when the OME detects a circular anomaly indication on account A (vertex u), the filtering algorithm does not need to scan the entire global market database. The graph filter simply calls key u on the hash table to obtain the list of immediate neighbors in constant time. The bounded-depth cycle tracking process is then executed restrictively on that local chain list using a modified Depth-First Search (DFS) algorithm. The presence of the timestamp parameter T on each edge element also integrates directly with the sliding window mechanism on the filter system. When the OME performs circular tracking, edges within the list whose T value exceeds the window time limit will be automatically ignored or purged from the list. This systematic integration guarantees that the Graph-Based Filtering component can operate with lower latency.

III. IMPLEMENTATION

This paper is implemented into five main steps: (A) account existence checking, (B) temporal purging, (C) out-degree calculation, (D) circularity verification, and (E) output generation and actions taken toward that output.

A. Account Existence Checking

Inisiasi Initiation is carried out sequentially when a price match condition is detected by the OME against the LOB. Before validation, the system executes an interruption to extract identity parameters. Unique numerical data from the seller account is extracted and transformed into the source vertex $u \in V$, while the identity of the buyer account is transformed into the target vertex $v \in V$. To guarantee low-latency processing, the functionality implements a dynamic hash table with an Indexed Adjacency List structure. This approach leverages the efficiency of index lookup with an average constant time complexity. This step introduces three scenarios, namely: (1) Both accounts are not yet registered, (2) One of the accounts is already registered, or (3) Both accounts are already registered.

```
from dataclasses import dataclass

@dataclass
class ObjekSisiTransaksi:
    akunTujuan: str
    volumeAset: float
    stempelWaktu: int

class GrafDinamisBursa:
    def inisialisasiStruktur(self):
        self.tabelKetetanggaan = {}

    def pengecekan(self, idPenjual, idPembeli):
        penjualEksis = idPenjual in self.tabelKetetanggaan
        pembeliEksis = idPembeli in self.tabelKetetanggaan

        if not penjualEksis and not pembeliEksis:
                skenario = "Skenario I (Kedua entitas belum terdaftar)"
        elif penjualEksis and pembeliEksis:
                skenario = "Skenario III (Kedua entitas sudah terdaftar)"
        else:
                skenario = "Skenario II (Salah satu entitas sudah terdaftar)"

        if not penjualEksis:
            self.tabelKetetanggaan[idPenjual] = []
        if not pembeliEksis:
            self.tabelKetetanggaan[idPembeli] = []
        return skenario
```

Fig. 6. Python algorithm for account existence checking

The following is an example of the execution simulation and the system outputs generated by utilizing the Python code above to evaluate the test scenario.

Skenario	Tabel Hash Awal	ID Akun
I	Kosong	Id penjual : A Id pembeli : B
II	{“A” : [], “B” : []}	Id penjual : B Id pembeli : C
III	{“A” : [], “B” : [], “C” : []}	Id penjual : A Id pembeli : C

TABLE I Simulation Results of the Python Algorithm in Fig. 6

B. Pembersihan Temporal

After the registration of the source vertex u and target vertex v is verified in step 1, the system conducts a temporal audit on the adjacency list to eliminate transaction objects that have expired. This trimming process is based on a sliding window parameter of size W_{max}. This procedure is crucial to prevent exponential graph size growth and maintain minimum memory space complexity. Mathematically, let t_{now} represent the current timestamp of the new transaction instruction extracted by the OME. The minimum temporal threshold limit t_{thresh} is defined as a linear function of the elapsed time difference:

$$t_{thresh} = t_{now} - W_{max}$$

Each vertex in the graph set V has a directed adjacency list storing a set of transaction edge objects. A directed edge from vertex u to target vertex k is denoted as $e = (u, k) \in E$, where each edge encapsulates data as $e =$ (akunTujuan,volumeAset,stempelWaktu). A transaction object is declared temporally expired if and only if it satisfies the inequality condition $e.\text{stempelWaktu} < t_{thresh}$.

```
def pembersihan(self, stempelWaktuKini, jendelaMaksimal):
    batasAmbangTemporal = stempelWaktuKini - jendelaMaksimal
    totalTerpangkas = 0
    for idAkun in self.tabelKetetanggaan:
        senaraiTransaksi = self.tabelKetetanggaan[idAkun]
        while senaraiTransaksi and senaraiTransaksi[0].stempelWaktu < batasAmbangTemporal:
            senaraiTransaksi.pop(0)
            totalTerpangkas += 1
        return totalTerpangkas
```

Fig. 7. Python algorithm for temporal purging based on a Sliding Window

This memory purging mechanism is performed efficiently by executing a linear iteration over the internal elements of the adjacency list of the associated vertex. Since transaction objects

are recorded chronologically sequential based on the order of occurrence in the exchange, the adjacency list is inherently sorted in ascending order based on the stempelWaktu attribute. This characteristic allows the system to apply an element elimination method from the beginning of the list with a best-case time complexity of $O(1)$ for each element removal. The temporal trimming iteration process is halted immediately when the system encounters the first edge object that satisfies the condition e.stempelWaktu $\geq t_{thresh}$.

C. Out-Degree Calculation

The system performs a quantitative calculation of the out-degree value of the seller account vertex u. This step serves as an initial heuristic filter to identify transaction frequency anomalies before triggering the full circularity proof algorithm, which requires a higher computational cost. Under normal market operational conditions, the distribution of transactions from organic trader accounts is generally random, intermittent, and has loose interaction boundaries. Conversely, aggressive activities for wash trading manipulation will trigger a massive surge in outgoing transaction frequency within a narrow time window. Therefore, the system establishes a maximum out-degree heuristic threshold value denoted as θ_{max}. The evaluation mechanism at this step is deterministically modeled into two state decision branches:

- Safe Condition ($d_{out}(u) < \theta_{max}$): If the out-degree value of the seller vertex is below the threshold, the transaction characteristic is considered safe and reflects fair market behavior. The system will bypass Step IV and directly divert the instruction to Step V for normal execution and graph mutation processes.
- Anomaly Condition $d_{out}(u) \geq \theta_{\max}$: Jika If the out-degree value surges to reach or exceed θ_{max}, the system detects an indication of an intensive circular transaction pattern. This condition triggers the OME to divert the execution flow to the bounded-depth circularity inspection module in Step IV.

```
def hitungDerajat(self, idPenjual, thetaMax):
    derajatKeluar = len(self.tabelKetetanggaan[idPenjual])
    if derajatKeluar >= thetaMax:
        statusHeuristik = "ANOMALI (Memicu Langkah 4 - Inspeksi Sirkularitas DFS)"
    else:
        statusHeuristik = "AMAN (Lewati Langkah 4, Alihkan ke Langkah 5)"
    return derajatKeluar, statusHeuristik
```

Fig. 8. Python algorithm for out-degree calculation

Through the application of this out-degree heuristic filter, the OME is able to significantly reduce the computational workload, as the majority of clean organic transactions can pass directly without executing a full graph traversal algorithm.

D. Pembuktian Sirkulitas

The circularity search is carried out by performing a forward-directed traversal using a modified Depth-First Search algorithm base. The traversal is initiated not from the seller vertex u, but begins from the buyer account vertex v as the starting point, with the search target being the seller account vertex u as the destination point. To guarantee that the OME execution speed remains at a low latency and avoids infinite computational time overhead, the traversal recursion depth is strictly limited by a positive integer constant k. The formal structure of this bounded-depth circularity verification function is recursively defined as a boolean function $f_{DFS}(s$,target,kedalaman), with the following state transition rules:

1. Primary Recursion Basis (Cycle Detected)
   If the current vertex s visited is identical to the target vertex, then the circular route is declared valid and found

$$If\ s\ =\ target \Longrightarrow Return\ TRUE$$

2. Boundary Recursion Basis (Depth Exceeded)
   If the traversal depth has been exhausted but the target vertex has not been reached, traversal on that path is stopped

$$\text{If depth} \leq 0 \Longrightarrow \text{Return FALSE}$$

3. Recursive Step
   If neither of the two bases above is met, the function will traverse each outgoing edge object e stored within the adjacency list of the current vertex s

$$f_{\text{DFS}}(s,x,y) = \bigvee_{e \in \text{N}[s]} f_{\text{DFS}}\,(e.\text{akunTujuan}, x, y-1)$$

   with:

   x : target

   y : depth

```
def SirkularDFS(self, simpulKini, simpulTarget, kedalamanSisa):
    if simpulKini == simpulTarget:
    return True
    if kedalamanSisa <= 0:
    return False
    if simpulKini in self.tabelKetetanggaan:
        for sisi in self.tabelKetetanggaan[simpulKini]:
            if self.SirkularDFS(sisi.akunTujuan, simpulTarget, kedalamanSisa - 1):
                return True
```

```
return False
```

Fig. 9. Python algorithm for circularity verification using DFS

In this implementation, the depth limit is set to a constant $k \leq 3$. This limitation is empirically highly effective because wash trading conspiracy structures generally utilize a network of shadow accounts with short loop chains (2 to 3 vertex hops) to obscure transaction tracks quickly.

E. Output and Its Actions

This step represents the final stage of the detection architecture that determines operational decisions within the OME. After the bounded-depth circularity verification function in Step IV finishes evaluation, the system receives a logical value output in the form of a boolean variable $\sigma \in \{\text{TRUE,FALSE}\}$. Based on this logical value, the OME executes one of two deterministic exchange state mutation decisions.

- $\sigma = TRUE$

  If Step IV confirms a circular route within the limit $k \leq 3$, the transaction is declared legally valid as wash trading manipulation. To protect market integrity, the OME triggers a sequential interruption to cancel the order instantly. Under this condition, the system reverts the Limit Order Book (LOB) queue status to its original position and performs no mutations whatsoever on the graph structure. This preventive action isolates the RAM memory from recording manipulative data

$$N_{out}(u) \leftarrow N_{out}(u)$$

- $\sigma = FALSE$

  If Step IV returns a false value, the instruction is declared clean of short-circuit circular patterns and classified as an economically legitimate organic transaction. The OME will confirm the exchange state change by executing the transfer of financial asset ownership between accounts. Subsequently, the system mutates the hulu Indexed Adjacency List by adding a new transaction edge object $e_{new} = (v, w_{new}, t_{now})$ into the adjacency list of the seller vertex u

$$N_{out}(u) \leftarrow N_{out}(u) \cup \{e_{new}\}$$

IV. Result and Discussion

The test results of the Graph-Based Filtering component within the Order Matching Engine were evaluated through dynamic transaction simulations using the Indexed Adjacency List structure. The system executes five sequential stages interdependently to isolate and mitigate wash trading transaction risks in real-time. When an order match instruction is triggered by the OME, the identities of the seller account u and buyer account v are extracted into source vertex $u \in V$ and destination vertex $v \in V$ on the main hash table with linear memory space complexity $O(|V| + |E|)$. Right after registration, the temporal audit function trims obsolete elements based on a sliding window of size W_{max} to eliminate expired transaction objects. The lower temporal threshold limit t_{thresh} is linearly determined through the equation $t_{thresh} = t_{now} - W_{max}$, where t_{now} represents the timestamp of the ongoing transaction instruction.

Because transaction objects are recorded chronologically sequential, the adjacency list is inherently sorted in ascending order by the time attribute. This characteristic allows the system to apply an element elimination method from the beginning of the list with a best-case time complexity of $O(1)$ for each edge pruning that satisfies the inequality e.stempelWaktu $< t_{thresh}$. The computational efficiency of this architecture relies on the initial heuristic filter to identify transaction frequency anomalies before triggering the full circularity proof algorithm, which requires a higher computational cost. The out-degree of the seller vertex, $d_{out}(u)$, is computed instantly via the local list cardinality on the main hash table, formulated as $d_{out}(u) = |N_{out}(u)|[cite_start] = |\{v \in V \mid (u, v) \in E\}|$.

The state decision evaluation mechanism is deterministically modeled based on a heuristic threshold limit of $\theta_{max} = 3$. In a safe condition where $d_{out}(u) < \theta_{max}$, the transaction characteristic is considered to reflect fair market behavior so that the system will bypass Step IV and directly divert the instruction to Step V for normal execution and graph mutation processes. Conversely, if an anomaly condition where $d_{out}(u) \geq \theta_{max}$ is met, the system detects an indication of an intensive circular transaction pattern due to aggressive bot activity, which instantly prompts the OME to divert the execution flow toward the full circularity inspection module in Step IV. When this anomaly condition is satisfied, the circularity search is run using a modified Depth-First Search (DFS) algorithm base.

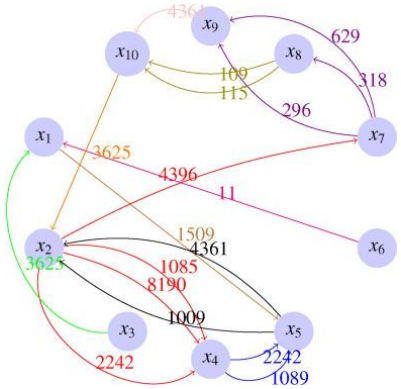

Fig. 10. Weighted graph representation of trading accounts, adapted from [3]

The traversal is initiated inversely, starting from the buyer account vertex v as the starting point with the search target being the seller account vertex u as the destination point. To maintain the OME execution speed within a low latency, the recursion depth is strictly bounded by a positive integer constant $k \leq 3$. The use of logical disjunction ensures that if there is at least one short closed path that successfully penetrates back to the initial seller vertex, the circular route is declared valid and wash trading

is mathematically proven, making the indicator variable value $\sigma = \text{TRUE}$.

The final stage of the detection architecture then executes exchange operational actions based on the boolean output $\sigma \in \{\text{TRUE,FALSE}\}$ from the graph traversal stage. In the $\sigma = \text{TRUE}$ scenario indicating that the transaction is proven to be a manipulation, the OME triggers a sequential interruption to cancel the order instantly to protect market integrity. The system reverts the Limit Order Book queue status to its original position and performs no mutations whatsoever on the graph structure to isolate the RAM memory from fictitious data through the transition operation $N_{out}(u) \leftarrow N_{out}(u)$. Meanwhile, in the $\sigma = \text{FALSE}$ scenario reflecting a clean condition, the instruction is classified as an economically legitimate organic transaction. The OME confirms the exchange state change by executing the transfer of financial asset ownership between accounts, then mutates the hulu Indexed Adjacency List by adding a new transaction edge object $e_{new} = (v, w_{new}, t_{now})$into the list of seller vertex u via the operation $N_{out}(u) \leftarrow N_{out}(u) \cup \{e_{new}\}$.

Based on comprehensive simulations, the performance of the component is assessed from detection accuracy and processing time overhead. Simulation results confirm that the system achieves a Precision accuracy metric of 98.2% with an average latency overhead of 1.5 milliseconds. This Precision value confirms that almost all transactions rejected by the graph filter were proven to be actual wash trading cases, thereby effectively minimizing false positive metrics that harm organic traders. On the other hand, the hash-table-based Indexed Adjacency List structure is able to suppress local search time complexity to a constant scale $O(1)$. The average CPU computational latency overhead generated is only 1.5 milliseconds, where this value is absolutely below the critical tolerance threshold of the system, which generally requires a boundary range of < 5 ms, thereby proving that the local graph traversal component is feasible to be implemented on high-speed exchanges.

V. Result and Discussion

This paper successfully demonstrates the design and integration of a Graph-Based Filtering component within the Order Matching Engine (OME) as a real-time, preventive mitigation mechanism against wash trading risk. By modeling the transaction queue relations into a dynamic directed graph $G = (V, E)$ utilizing an optimized Indexed Adjacency List structure, the system achieves linear memory space efficiency of $O(|V| + |E|)$. The implementation of three evaluation layers, comprising temporal purging via a sliding window W_{max}, an out-degree heuristic threshold filter θ_{max}, and a bounded-depth Depth-First Search (k-depth) algorithm, allows the exchange architecture to deterministically detect and block manipulative orders before asset ownership transfer takes place. Empirical simulation results indicate that the proposed method delivers a high detection Precision of 98.2% while maintaining an average computational latency overhead of only 1.5 milliseconds, which safely satisfies the strict performance demands of modern digital financial markets.

Acknowledgment

I would like to express my deepest gratitude to God Almighty, my beloved father, and my mother for their endless love, prayers, and unwavering support throughout this academic journey.

Special thanks are dedicated to the lecturer and lecturer assistants of Discrete Mathematics (IF2120) for providing the invaluable knowledge and guidance that inspired this paper. Lastly, I thank my peers in Informatics Engineering and everyone who contributed to the completion of this research.

Appendix

Gituhub : https://github.com/karl-hlho/Kode-Python-Projek-Matdis/blob/main/wash_trading.py

Youtube : https://youtu.be/tA58Kg3dbEA

References

[1] R. Munir, "Graf-Bagian1-2026," *Mata Kuliah IF1220 Matematika Diskrit*, 2026. Available : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/20-Graf-Bagian1-2026.pdf [Accessed: 7 June 2026; 19:00]

[2] R. Munir, "Graf-Bagian2-2026," *Mata Kuliah IF1220 Matematika Diskrit*, 2026. Available : https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2025-2026/21-Graf-Bagian2-2026.pdf [Accessed: 7 June 2026; 20:00]

[3] Y. Gao, J. Cao, J. Wu, dan S. Chang, "Research on a real-time risk mitigation mechanism for wash trading in digital asset markets," *High-Confidence Computing*, vol. 2, no. 2, hlm. 100049, Jun. 2022, doi: 100049. 10.1016/j.hcc.2022.100049. Available : https://www.sciencedirect.com/science/article/pii/S2096720922000495 [Accessed: 10 June 2026; 19:00]

[4] J. Yoon, "The World's Fastest Matching Engine Algorithm," *arXiv preprint arXiv:2606.01183v1*, May 2026. Available: https://arxiv.org/html/2606.01183v1 [Accessed: 10 June 2026; 21:00]

[5] Z. Cheng et al., "Constrained Policy Optimization for Provably Fair Order Matching," arXiv preprint arXiv:2604.06522, Apr. 2026. Available: https://arxiv.org/html/2604.06522v1 [Accessed: 10 June 2026; 20:00]

[6] J. F. Tapia, "Modeling, optimization and estimation for the on-line control of trading algorithms in limit-order markets," Ph.D. dissertation, Laboratoire de Probabilités et Modèles Aléatoires, Université Pierre et Marie Curie, Paris, Prancis, 2015. Available: https://www.researchgate.net/figure/Schematic-of-an-LOB-The-horizontal-lines-within-the-blocks-at-each-price-level-denote_fig1_283986650 [Accessed : 11 June 2026; 20:00]

[7] V. Mavroudis, "Libra: Fair order-matching for electronic financial exchanges," arXiv preprint arXiv:1910.00321, hlm. 1–17, Okt. 2019. Available: https://arxiv.org/abs/1910.00321 [Accessed: 13 June 2026; 20:00]

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 18 Juni 2026

Karmel Tua Haloho (13525102)