Graph-Based Arbitrage Detection: Analyzing Negative Cycles in Multi-Asset Crypto Exchange Networks

Bagas Anugrah Putra - 13525075
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: bagasteik2025@gmail.com , 13525075@std.stei.itb.ac.id

***Abstract*—**This study proposes a unified graph-theoretic and optimization-based framework for detecting circular arbitrage opportunities in fragmented cryptocurrency exchange and decentralized finance (DeFi) ecosystems. The core idea is to represent multi-asset trading environments as directed at weighted multigraphs, enabling the transformation of multiplicative return structures into additive edge weights via negative logarithmic mapping. This reformulation allows arbitrage detection to be treated as a shortest-path optimization problem under dynamic market conditions.

To address computational and structural challenges in large-scale trading networks, the framework evaluates exact shortest-path algorithms, including Bellman-Ford and the Shortest Path Faster Algorithm (SPFA), enhanced with efficient predecessor tracking for cycle reconstruction and queue optimization strategies such as Small Label First (SLF) and Large Label Last (LLL). For DeFi-specific architectures involving automated market makers (AMMs), a line graph transformation is introduced to resolve routing ambiguity across liquidity pools and to preserve structural consistency in path evaluation.

The proposed model further incorporates realistic market frictions, including transaction fees, smart contract gas costs, nonlinear price impact derived from constant product market-making functions, and latency effects across heterogeneous blockchain consensus mechanisms. Empirical validation using historical cryptocurrency and foreign exchange datasets demonstrates that while exact graph-based algorithms achieve near real-time detection performance, the feasibility of arbitrage execution is strongly constrained by volatility, slippage, and network latency.

Keywords: Circular arbitrage; cryptocurrency exchange networks; graph theory; shortest path algorithms; decentralized finance (DeFi)

I. INTRODUCTION

The emergence of decentralized ledger technologies has intensified fragmentation across global digital asset markets. Unlike traditional financial systems, where price discovery is consolidated through centralized clearing mechanisms, cryptocurrency trading operates across a distributed network of centralized exchanges (CEXs) and decentralized exchanges (DEXs). This structural decentralization results in persistent and often transient price deviations for identical asset pairs across trading venues, generating conditions conducive to arbitrage-based strategies.

From a macro-structural perspective, these developments can be contextualized through a PESTLE framework, which highlights the multi-dimensional forces shaping market evolution. Political factors such as capital controls and geopolitical uncertainty have accelerated the adoption of decentralized trading infrastructures. Economic pressures, including inefficiencies in traditional financial intermediation and low real yields, have redirected capital flows toward decentralized finance (DeFi) ecosystems. Social dynamics reflect increasing digital financial literacy and broader acceptance of self-custodial asset management. Technological advancements in smart contract execution, scalability solutions, and consensus mechanisms have significantly improved transaction throughput and strategy complexity. Legal ambiguity surrounding digital assets has further enabled regulatory fragmentation, often resulting in jurisdictional arbitrage. Meanwhile, environmental considerations have shifted substantially following the transition from energy-intensive proof-of-work systems to more efficient proof-of-stake architectures, reducing operational costs associated with network participation. Collectively, these factors contributed to the rapid expansion of DeFi liquidity, with total value locked (TVL) increasing from hundreds of millions in 2020 to well over hundreds of billions in subsequent market cycles, thereby amplifying arbitrage intensity within fragmented liquidity landscapes.

Within this environment, arbitrage mechanisms typically emerge as closed-loop trading structures. Triangular arbitrage operates within a single venue by cycling through

a sequence of asset exchanges that ultimately return to the initial asset. Circular arbitrage generalizes this concept by extending the trading sequence across multiple assets and heterogeneous trading venues, thereby increasing both structural complexity and potential return opportunities. However, identifying such cycles in real time is computationally demanding due to the high dimensionality and volatility of order book dynamics across distributed exchanges.

Graph-theoretic modeling provides a rigorous analytical foundation for addressing this problem. By representing assets as vertices and exchange rates as directed weighted edges, the arbitrage detection problem can be reformulated as the identification of negative-weight cycles in a directed graph. This transformation is achieved through logarithmic conversion of multiplicative exchange rates into additive edge weights, enabling the application of classical shortest-path optimization techniques. Within this formulation, the paper further examines algorithmic approaches for cycle detection and evaluates system-level constraints, including latency, variability, liquidity-dependent price formation, and transaction cost structures, all of which critically influence the feasibility of arbitrage execution in real-world decentralized market environments.

II. Theoretical Foundations and Literature Review

To model exchange networks mathematically, let a financial market be represented as a directed weighted multigraph $G = (V, E)$ [1]. The vertex set $V = \{v_1, v_2, \ldots, v_n\}$ represents n unique digital assets, and the edge set E represents the set of directed trading channels between them [1]. An edge $e = (u, v) \in E$ indicates that asset u can be directly exchanged for asset v. If multiple exchanges facilitate trading for the same asset pair, they are modeled as parallel directed edges with distinct weights, making G a multigraph [2]. Standard representations of these structures include adjacency matrices, which are highly efficient for dense networks, and adjacency lists, which optimize space and traversal speed in sparse networks [1].

A path P from a source vertex v_0 to a destination vertex v_k is defined as a sequence of vertices $(v_0, v_1, v_2, \ldots, v_k)$ where each adjacent pair (v_{i-1}, v_i) is connected by a directed edge. A cycle is a path that starts and ends at the same vertex ($v_0 = v_k$). In a weighted graph with additive weights, a negative-weight cycle is defined as a cycle where the sum of the edge weights is strictly less than zero [1]:

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

If a negative-weight cycle is reachable from a source vertex, the shortest path to any vertex on that cycle is undefined. A traveler could traverse the cycle indefinitely to decrease the path cost to $-\infty$. While this property is a pathological failure state for standard single-source shortest path (SSSP) routing, it is the exact mechanism used to detect arbitrage opportunities in financial networks [1].

Historically, the study of shortest paths and cycle detection has produced several foundational algorithms. Alfonso Shimbel (1955) first outlined the core concepts of iterative edge relaxation [3] Lester Ford Jr. (1956) and Richard Bellman (1958) subsequently formalized the classic Bellman-Ford algorithm, which solves the SSSP problem in $O(|V| \cdot |E|)$ time and identifies negative cycles. Edward F. Moore (1959) introduced a queue-based variant of the algorithm, laying the groundwork for the modern Shortest Path Faster Algorithm (SPFA) [3].

To optimize these classical paradigms, several theoretical improvements have been developed. Jin Y. Yen (1970) introduced Yen's improvement, partitioning the edge set to accelerate convergence and detect negative cycles earlier [3]. Bannister and Eppstein (2012) proposed a randomized speedup of the Bellman-Ford algorithm, utilizing random edge permutations to reduce relaxation passes [3]. Jeremy T. Fineman (STOC 2024) achieved a significant theoretical breakthrough, presenting an algorithm that solves the SSSP problem with negative real weights in near-linear time $O(|E| \log^2 |V|)$, representing a major advance over classical $O(|V| \cdot |E|)$ bounds.

Beyond classical shortest path algorithms, several alternative computational models have been applied to financial networks. Maximum-plus product iterations utilize tropical geometry to identify optimal routing paths. Lawler's algorithm solves the minimal cycle ratio problem, finding cycles that optimize specific cost-to-time ratios. Howard's policy iteration algorithm provides a highly efficient, empirically fast approach to solving the minimum cycle ratio problem on doubly weighted graphs.

For highly complex, non-linear trading structures, researchers have formulated the search as a Satisfiability Modulo Theories (SMT) problem, utilizing SMT solvers to prove path feasibility and optimize routing under complex constraints, such as those exploited in the multi-million dollar bZx protocol flash loan attacks. Additionally, quantum computing research has mapped currency arbitrage to Quadratic Unconstrained Binary Optimization (QUBO) problems for execution on D-Wave quantum annealers, and deployed the Quantum Approximate Optimization Algorithm (QAOA) on IBM gate-based quantum simulators using linear cost formulations [6].

III. Mathematical Transformative Proof of Arbitrage Conditions

To systematically identify circular arbitrage, the multiplicative relationship of financial asset exchanges must be converted into an additive relationship that is compatible with shortest-path algorithms [2]. Let each directed edge

$e = (u, v) \in E$ be assigned a nominal exchange rate $R(u, v)$ representing the quantity of asset v obtained per unit of asset u sold [7]. If a trader sequentially executes transactions along a cycle $C = (v_0, v_1, v_2, \ldots, v_k = v_0)$, starting with $Q0$ units of asset $V0$, the final quantity of capital Qk returned is determined multiplicatively [8]:

$$Q_k = Q_0 \cdot R(v_0, v_1) \cdot R(v_1, v_2) \cdots R(v_{k-1}, v_k) = Q_0 \prod_{i=1}^{k} R(v_{i-1}, v_i)$$

An arbitrage opportunity is defined as a closed cycle that yields a positive net return [9]. Normalizing the starting capital to $Q0 = 1$, the mathematical condition for profitable arbitrage is formulated as [10]:

$$\prod_{i=1}^{k} R(v_{i-1}, v_i) > 1$$

Because classical shortest path algorithms rely on the addition of edge weights rather than their product, this multiplicative inequality must be transformed [5]. Applying the natural logarithm function $\ln(x)$ (which is strictly increasing for all $x > 0$ to both sides of the inequality preserves the relationship [6]:

$$\ln\left(\prod_{i=1}^{k} R(v_{i-1}, v_i)\right) > \ln(1)$$

Using the fundamental logarithmic identity $\ln(a \cdot b) = \ln(a) + \ln(b)$, the natural logarithm of a product is expanded into the sum of the individual natural logarithms [7]:

$$\sum_{i=1}^{k} \ln\left(R(v_{i-1}, v_i)\right) > 0$$

This inequality defines a positive-weight cycle (10). To utilize standard SSSP algorithms designed to detect negative-weight cycles, the inequality is multiplied by -1, which reverses its direction [15]:

$$\sum_{i=1}^{k} -\ln(R(v_{i-1}, v_i)) < 0$$

By defining the additive edge weight $w(u, v)$ for each directed edge $(u, v) \in E$ as the negative natural logarithm of the exchange rate [1]:

$$w(u, v) = -\ln(R(u, v))$$

the arbitrage condition is successfully reformulated as finding a cycle C such that [3]:

$$\sum_{(u,v)\in C} w(u, v) < 0$$

This mathematical proof demonstrates that finding a profitable multi-asset arbitrage cycle is equivalent to finding a negative-weight cycle in the transformed directed graph $G = (V, E, w)$.

In real-world order books, trades do not occur at a single mid-price (7). Instead, they execute against specific bid and ask orders [7]. If an exchange quotes the trading pair in units of asset Y per asset X (the X/Y pair), converting $X \rightarrow$ represents selling the base asset, executing at the bid price [7]. Conversely, converting $Y \rightarrow$ represents buying the base asset, executing at the ask price [7]. To reflect this in the graph, directed edge weights are mapped based on the direction of the trade [7]. Table I outlines this mapping.

TABLE I. Order Book Price Quotes and Resulting Graph Edge Weight Mapping

Trading Action/ Conversion Direction	Relevant Order Book Quote	Resulting Graph Edge Weight
Sell base asset X to buy quote asset Y ($X \rightarrow$)	Bid price of X/Y pair	$w(X, Y) = -\ln(\text{Bid}_{X/Y})$
Buy base asset X using quote asset Y ($Y \rightarrow$)	Ask price of X/Y pair	$w(Y, X) = -\ln\left((\text{Ask}_{X/Y})^{-1}\right) = \ln(\text{Ask}_{X/Y})$

IV. Algorithmic Implementations and Cycle Reconstructions

The mathematical transformation defined in Section III allows traders to deploy classical pathfinding and cycle-detection algorithms to locate arbitrage paths in real-time.

A. The Classical Bellman-Ford Algorithm

The classical Bellman-Ford algorithm maintains an array of shortest path estimates d[v] from a single source vertex s to all other vertices $v \in V$, alongside a predecessor array $parent[v]$ to track the path structure. The algorithm runs in three main stages :

```
Algorithm 1: Classic Bellman-Ford
-----------------------------------------
Input : Graph G=(V,E), weights w, src s
Output: True/False negative cycle
-----------------------------------------
1: function BellmanFord(G, w, s):
```

```
2:    for each vertex v in V do:
3:      d[v] ← ∞
4:      parent[v] ← nil
5:    d[s] ← 0
6:
7:    // Iterative Relaxation
8:    for i from 1 to |V| - 1 do:
9:      for each edge (u, v) in E do:
10:       if d[u] + w(u,v) < d[v] then:
11:         d[v] ← d[u] + w(u,v)
12:         parent[v] ← u
13:
14:   // Negative Cycle Detection
15:   for each edge (u, v) in E do:
16:   if d[u] + w(u,v) < d[v] then:
17:       return True
18:   return False
```

B. Walk-to-the-Root Predecessor Backtracking

Detecting the existence of a negative cycle is not sufficient for automated execution; the trading system must reconstruct the exact sequence of vertices that compose the cycle [6]. Simple backward traversal from an arbitrary updated vertex v can fail because v might not be part of the cycle itself, but rather situated on a path downstream from the cycle.

To isolate a node that is guaranteed to reside on the cycle, the algorithm utilizes the "walk-to-the-root" backtracking technique [10]:

```
Algorithm 2: Walk-to-the-Root
-----------------------------------------
Input : Array parent, node v, count |V|
Output: Reconstructed cycle path
-----------------------------------------
1: function WalkToRoot(parent, v, |V|):
2:    curr ← v
3:    for i from 1 to |V| do:
4:      curr ← parent[curr]
5:
6:    cycle ← empty_list()
7:    x ← curr
8:    cycle.push_back(x)
9:    temp ← parent[x]
10:   while temp ≠ x do:
11:     cycle.push_back(temp)
12:     temp ← parent[temp]
13:
14:   cycle.reverse()
15:   return cycle
-----------------------------------------
```

C. Shortest Path Faster Algorithm (SPFA) with Early Termination

The Shortest Path Faster Algorithm (SPFA) optimizes the classical Bellman-Ford paradigm by maintaining a queue Q of vertices whose distance estimates have been successfully updated, avoiding redundant edge relaxations. While standard SPFA only terminates when a node is enqueued V times, an early-termination optimization can be implemented.

By maintaining a predecessor graph D and executing a quick cycle-detection pass (using DFS) every V edge relaxations, the system can identify and execute profitable arbitrage loops before completing the full V relaxation passes.

```
Algorithm 3: Optimized SPFA
-----------------------------------------
Input : Graph G=(V,E), weights w, src s
Output: Path cycle or "No cycle"
-----------------------------------------
1: function SPFA_EarlyTerm(G, w, s):
2:   for each vertex v in V do:
3:     d[v] ← ∞; parent[v] ← nil
4:     inQ[v] ← false; cnt[v] ← 0
5:
6:   d[s] ← 0; Q ← empty_queue()
7:   Q.push(s); inQ[s] ← true
8:   rlx_cnt ← 0
9:
10:  while Q is not empty do:
11:    u ← Q.pop()
12:    inQ[u] ← false
13:
14:    for each neighbor v of u do:
15:      if d[u] + w(u,v) < d[v] then:
16:        d[v] ← d[u] + w(u,v)
17:        parent[v] ← u
18:        rlx_cnt ← rlx_cnt + 1
19:
20:        if not inQ[v] then:
21:          Q.push(v)
22:          inQ[v] ← true
23:          cnt[v] ← cnt[v] + 1
24:          if cnt[v] ≥ |V| then:
25:            return
26:            WalkToRoot(parent,v,|V|)
27:
28:        // Early termination check
29:        if rlx_cnt mod |V| = 0 then:
30:          if DetectCycle(parent) then:
31:            return
32:            WalkToRoot(parent,u,|V|)
-----------------------------------------
```

D. Deque Queue Optimizations: SLF and LLL

- To further optimize SPFA in dense, highly volatile multigraphs, the standard FIFO queue can be replaced with a double-ended queue (deque) Q_{deque} .This structure supports two key heuristics:Small Label First (SLF): When enqueuing a vertex , if $d[v] < d[Q_{\text{deque}}.\text{front}()]$, v is pushed to the front of the deque to prioritize lower-cost paths. Otherwise, it is pushed to the back.
- Large Label Last (LLL): Let $A = \frac{1}{|Q_{\text{deque}}|} \sum_{x \in Q_{\text{deque}}} d[x]$ be the average distance of all vertices currently in the queue. When popping a vertex u from the front of Q_{deque} , if $d[u] > A$, u is moved to the back of the queue. This delays the propagation of high-cost paths and minimizes redundant edge scans.

E. Line Graph Transformations for AMM Pool Disambiguation

In decentralized exchanges (DEXs) like Uniswap, traders execute swaps directly against automated liquidity pools (9). Modeling these platforms using simple token-to-token graphs introduces routing ambiguities. If multiple pools exist for the same asset pair (e.g., a Uniswap V2 pool and a SushiSwap pool for ETH/USDT), representing them as parallel edges in G makes it difficult for SSSP algorithms to preserve pool-specific identity or ensure that sequential transactions execute across valid smart contract routes [9].

To resolve this limitation, the token-level graph $G = (V, E)$ is transformed into a line graph $L(G) = (V_L, E_L)$ [9]. In this formulation, vertices represent individual liquidity pools, and directed edges represent the ability to route trades sequentially between them. Mathematically, let v, $e_1 = (u, v)$ and $e_2 = (v, w)$ be two liquidity pools in the original graph G . In the line graph $L(G)$:

1. A vertex $v_l \in V_l$ is created for each pool (edge) in G .
2. A directed edge $(v_{L,1}, v_{L,2}) \in E_L$ is drawn if and only if the output token of pool e_1 matches the input token of pool e_2 .

Running negative cycle detection algorithms over $L(G)$ ensures that paths represent realistic, pool-specific trading routes, preserving transaction fees and capacities across individual market contracts.

V. Real-World Transactional Friction and Systemic Constraints

While theoretical models assume that detecting a negative cycle guarantees risk-free profit, real-world execution is highly constrained by transaction fees, price slippage, network latency, and liquidity limits.

A. Transaction Fees and Gas Modeling

Every transaction executed on an exchange incurs a fee, which must be incorporated into the edge weights. Proportional fees (e.g., trading commissions in centralized exchanges) act as multiplicative factors that reduce the net exchange rate. Let $f(u, v) \in [0, 1)$ represent the proportional fee rate charged on the trading pair connecting asset u represent the proportional fee rate charged on the trading pair connecting asset u to asset v . The net exchange rate is defined as:

$$R_{\text{net}}(u, v) = R(u, v) \cdot (1 - f(u, v))$$

In decentralized exchanges, smart contract interactions also require gas fees. Let $g_{\text{hop}}(e)$ represent the venue-specific gas cost (e.g., approximately 150,000 gas for Uniswap V2 and 200,000 gas for Uniswap V3). The aggregate gas cost G for a trading cycle C is defined as:

$$G = \sum_{e \in C} g_{\text{hop}}(e)$$

To maintain cycle profitability, the gross arbitrage returns must exceed both the percentage fees and the absolute gas overhead [11].

B. Slippage and AMM Pool Constraints

Price slippage refers to the variance between the expected price of a trade and the actual execution price.[2] This discrepancy is driven by order book depth in CEXs and liquidity pool curves in DEXs [1]. In Automated Market Makers utilizing a Constant Product Market Maker (CPMM) model, the pool maintains token reserves x and such that [1]:

$$x \cdot y = k$$

If a trader executes a swap of Δx units of token X for token Y with a pool fee rate of ϕ, the output quantity Δy is determined by the non-linear relationship:

$$\Delta y = y \cdot \left(1 - \frac{x}{x + (1 - \phi)\Delta x}\right)$$

The effective exchange rate for this transaction is:

$$R_{\text{eff}}(\Delta x) = \frac{\Delta y}{\Delta x} = \frac{y \cdot (1-\phi)}{x + (1-\phi)\Delta x}$$

This rate decays non-linearly as the transaction size Δx increases. This decay illustrates a common "liquidity trap" where an arbitrage path appears highly profitable on paper for tiny transaction sizes, but collapses when larger capital sizes are deployed. The first-order price slippage for small trades satisfies:

$$\text{Slippage} \approx \frac{\Delta x}{x}$$

To protect profit margins, the additive edge weights must be updated dynamically as a function of transaction volume Q : Q:

$$w_{\text{adj}}(u, v, Q) = -\ln\left(R_{\text{eff}}(u, v, Q) \cdot (1 - f(u, v))\right)$$

Applying a first-order Taylor series approximation for small transaction fees f and small slippage rates, the adjusted edge weight becomes:

$$w_{\text{adj}}(u, v, Q) \approx -\ln(R(u, v)) + f(u, v) + \text{Slippage}(u, v, Q)$$

This mathematical representation proves that transaction fees and slippage act as additive penalties to the edge weights, raising the threshold required for a cycle to become profitable.

C. Multi-Objective Routing Optimization

Given these constraints, executing real-world arbitrage becomes a multi-objective optimization problem. To find the optimal route and size, the trading system can optimize a candidate vector F under a Pareto objective framework: F under a Pareto objective framework:

$$F(P, w) = (S, -G, -\Sigma, -R)$$

where S represents user surplus, G is the aggregate gas cost, Σ is the expected slippage, and R represents execution risk. To manage market risk, the system can utilize a Conditional Value at Risk (CVaR) metric with a confidence level of , limiting exposure to price fluctuations and sandwich attacks during execution [4].

D. Latency and Blockchain Consensus Dynamics

Arbitrage opportunities are highly fleeting, meaning execution speed is a critical bottleneck. While optimized implementations of the Bellman-Ford algorithm can detect negative cycles in as little as 0.002 milliseconds, processing speeds are bounded by blockchain network latency [3].

Different consensus protocols introduce unique execution environments. Table II compares the transaction properties of Ethereum and Algorand.

TABLE II. Transaction Properties and Consensus Performance Across Decentralized Networks

Performance Metric / Property	Ethereum Network	Algorand Network
Average Transaction Throughput	~30 transactions per second	Up to 7,500 transactions per second
Block Confirmation Time	~12 seconds	3 - 4 seconds
Transaction Prioritization Mechanism	Fee-based auction (gas priority)	First-Come-First-Serve (FCFS)
Minimum Transaction Cost	Highly variable / volatile	0.001 ALGO

On Ethereum, competing searchers run gas auctions, bidding high priority fees to guarantee their transaction is executed first [8]. Conversely, Algorand operates on a First-Come-First-Serve (FCFS) basis under normal conditions, creating different optimization requirements for trading bots [8]. To minimize latency, institutional traders co-locate their execution servers near the exchanges' matching engines [3].

VI. Quantitative Empirical Experiments and Benchmarks

To validate these graph-based arbitrage systems, performance was evaluated using historical foreign exchange and cryptocurrency data [3].

A. Forex Triangular Arbitrage Case Study

Using the compiled data, the system evaluated two distinct multi-currency trading cycles:

1. Cycle 1 (USD → JPY → GBP → USD): Starting with 1 USD, the capital was sequentially swapped.
   Step 1 (USD → JPY): $1\ \text{USD} \times 113.50 = 113.50\ \text{JPY}.$
   Step 2 (JPY → GBP): $1\ \text{USD} \times 113.50 = 113.50\ \text{JPY}.$

Step 3 (GBP → USD): $0.7718 \text{ GBP} \times R(\text{GBP}, \text{USD}) = 1.0653 \text{ USD}$

This cycle returned 1.0651 USD, yielding a net profit of 5.61% after accounting for decimal approximations.

2. Cycle 2 (EUR → USD → JPY → GBP → EUR): This path evaluated four assets under specific spot rates.
Spot Rates: EUR/USD (1.1860), USD/JPY (113.50), JPY/GBP (0.0068), GBP/EUR (1.1555).[6]
Step 1 (EUR → USD): $1 \text{ EUR} \times 1.1860 = 1.1860 \text{ USD}$.
Step 2 (USD → JPY): $1.1860 \text{ USD} \times 113.50 = 134.61 \text{ JPY}$.
Step 3 (JPY → GBP): $134.61 \text{ JPY} \times 0.0068 = 0.9153 \text{ GBP}$.
Step 4 (GBP → EUR): $0.9153 \text{ GBP} \times 1.1555 = 1.0577 \text{ EUR}$.

This cycle returned 1.0577 EUR, yielding a net profit of 5.77% .

B. Cryptocurrency Market Verification

On January 4, 2025, real-time pricing data was fetched for Bitcoin (BTC), Ethereum (ETH), and Ripple (XRP) to detect triangular arbitrage across three sequences

1. Cycle A: $\text{BTC} \to \text{ETH} \to \text{XRP} \to \text{BTC}$
2. Cycle B: $\text{ETH} \to \text{XRP} \to \text{BTC} \to \text{ETH}$
3. Cycle C: $\text{XRP} \to \text{BTC} \to \text{ETH} \to \text{XRP}$

Because these cycles utilized the same currency pairs in different sequences, they all yielded an identical profit of 0.04%. This small margin highlights the high efficiency of crypto markets, where price discrepancies are corrected rapidly by competing algorithms [2].

C. Graph Neural Network Benchmarks

To evaluate modern heuristic approaches, a GraphSAGE GNN model was trained on 200 five-minute snapshots across KuCoin, Gate.io, Huobi, Bitget, and MEXC, using key cryptocurrencies (BTC, ETH, SOL, XRP, LTC, ADA). Edge feature vectors were constructed by fusing multiple market parameters:

$$\chi_{(u,v)}$$

The model was trained for 50 epochs using binary cross-entropy loss and the Adam optimizer with a learning rate of 0.001. During inference, NetworkX's simple_cycles was used to validate the predicted paths. The model achieved an average F1-score of 0.90, precision of 0.89, recall of 0.92, and AUC of 0.94 [9]. However, the average CPU inference time was 78 milliseconds—significantly slower than exact pathfinding algorithms. Table III compares exact and heuristic detection models.

TABLE III. Evaluation Performance Metrics of Exact and Heuristic Negative Cycle Detection Models

Evaluation Metric/ Parameter	Classic Bellman-Ford	Optimized SPFA (Early Term)	GraphSAGE GNN Model
Mathematical Nature	Deterministic Exact	Deterministic Exact	Heuristic Predictive
Average Detection Latency	~2.40 milliseconds (dense)	0.002 milliseconds	78.000 milliseconds
Worst-Case Complexity	$O(V \cdot E)$	$O(V \cdot E)$	$O(V+E)$ forward pass
Negative Cycle Verification	Guarantees detection	Guarantees detection	Requires post-processing
Hardware Requirements	Standard CPU execution	Standard CPU execution	GPU preferred for latency

VII. Discussion and Future Outlook

The experimental evaluations confirm that while exact algorithms (Bellman-Ford and SPFA) provide deterministic guarantees and minimal latency, their worst-case performance on dense multigraphs remains a bottleneck for high-frequency trading [1]. Shortest path algorithms like Dijkstra's are faster but structurally incapable of handling negative edge weights, which are a characteristic feature of profitable arbitrage cycles [6].

To optimize these systems, developers can deploy hybrid architectures that combine predictive machine learning models with exact pathfinding engines [7]. In this framework, GNNs or light-weight classifiers are used to predict and prune the search space, leaving the final cycle verification to optimized SPFA implementations.

Additionally, the growth of decentralized protocols has led to cross-chain and cross-layer arbitrage opportunities. These systems must route transactions across different blockchain consensus networks (e.g., Ethereum, Solana, Polygon, Algorand), requiring the integration of multi-hop bridges and cross-chain execution risks into the

pathfinding algorithms. Future research will focus on developing distributed, parallelized cycle detection frameworks that can process multi-million transaction graphs across fragmented cross-chain liquidity networks.

VIII. CONCLUSIONS

This paper has presented a mathematically rigorous framework for identifying circular arbitrage opportunities within cryptocurrency markets. By converting multiplicative exchange returns into additive path costs using negative natural logarithms, the challenge of locating profitable circular trades was successfully mapped to finding negative-weight cycles in a directed graph.

While classical algorithms like Bellman-Ford and queue-optimized SPFA can locate these cycles with sub-millisecond latencies, their real-world profitability is heavily constrained by transaction fees, network latency, and non-linear price slippage.

As decentralized finance continues to expand and fragment, combining deterministic pathfinding algorithms with machine learning filters and multi-objective optimization engines presents a promising path forward for high-frequency trading systems. Ultimately, navigating the structural friction of these networks is what separates theoretical paper profits from successful real-world execution.

REFERENCES

[1] N. J. Rusli, "Arbitrage Detection in Financial Markets: A Graph Theory Approach Using the Bellman-Ford Algorithm".

[2] Y. Venkatesh, Dr Anusha T, S. Manoj, and V. P, "Arbitrage Detection in Crypto Markets Using Graph Neural Networks," Dep. Comput. Sci. Eng. SRM Inst. Sci. Technol. Vadapalani Chennai India, [Online]. Available: https://www.atlantis-press.com/article/126016976.pdf

[3] N. S. Johansen, K. O. Nielsen, and R. G. Tollund, "Efficiently Finding Ratio-Optimal Infinite Cycles in Doubly-Priced Timed Automata".

[4] M. Marfinetz, "Hybrid Genetic Algorithm for Optimal User Order Routing: Multi-Objective Solver Optimization in CoW Protocol Batch Auctions," Oct. 24, 2025, arXiv: arXiv:2510.21647. doi: 10.48550/arXiv.2510.21647.

[5] V. Pakholchuk, "Arbitrage Opportunities on Decentralized Exchanges: Application of Graph Neural Networks".

[6] S. Deshpande, E. R. Das, and F. Mueller, "Currency Arbitrage Optimization using Quantum Annealing, QAOA and Constraint Mapping," Feb. 08, 2025, arXiv: arXiv:2502.15742. doi: 10.48550/arXiv.2502.15742.

[7] I. Akouaouch and A. Bouayad, "An innovative approach to identifying triangular arbitrage opportunities in financial markets using the Bellman-Ford algorithm," Bull. Electr. Eng. Inform., vol. 15, no. 3, pp. 2579–2594, Jun. 2026, doi: 10.11591/eei.v15i3.10817.

[8] S. Byrne, "An Exploration of Novel Trading and Arbitrage Methods within Decentralised Finance".

[9] J. A. Berg, R. Fritsch, L. Heimbach, and R. Wattenhofer, "An Empirical Study of Market Inefficiencies in Uniswap and SushiSwap," in Financial Cryptography and Data Security. FC 2022 International Workshops, vol. 13412, S. Matsuo, L. Gudgeon, A. Klages-Mundt, D. Perez Hernandez, S. Werner, T. Haines, A. Essex, A. Bracciali, and M. Sala, Eds., in Lecture Notes in Computer Science, vol. 13412. , Cham: Springer International Publishing, 2023, pp. 238–249. doi: 10.1007/978-3-031-32415-4_16.

[10] Y. Zhang, T. Yan, J. Lin, B. Kraner, and C. Tessone, "An Improved Algorithm to Identify More Arbitrage Opportunities on Decentralized Exchanges," Jun. 24, 2024, arXiv: arXiv:2406.16573. doi: 10.48550/arXiv.2406.16573.

[11] R. Munir, Matematika Diskrit, Revisi ke-7. Bandung: Penerbit Informatika, 2020.

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, 1 Juni 2025

Bagas Anugrah Putra - 13525075