DEX Arbitrage Challenge

Part 1: Off-Chain Cycle Detection

1.1 Graph Construction Method and Assumptions

1.1.1 Mathematical Modeling

To systematically identify arbitrage opportunities across Uniswap V2, we modeled the DeFi market as a **Directed Weighted Graph** G = (V, E), where:

- Nodes(V): Represent ERC20 Tokens.
- Edges(E): Represent liquidity pools connecting two tokens. Since swaps can occur in both directions, each pool generates two directed edges.
- Weights(W): $-ln(P \times (1-\phi))$, where P is the spot price and $\phi = 0.003$ is the fee. This transformation converts the multiplicative problem of finding a profit cycle ($\prod P > 1$) into an additive "shortest path" problem ($\sum \omega < 0$), specifically the detection of Negative Cycles.

1.1.2 Data Analysis & Filtering Assumptions

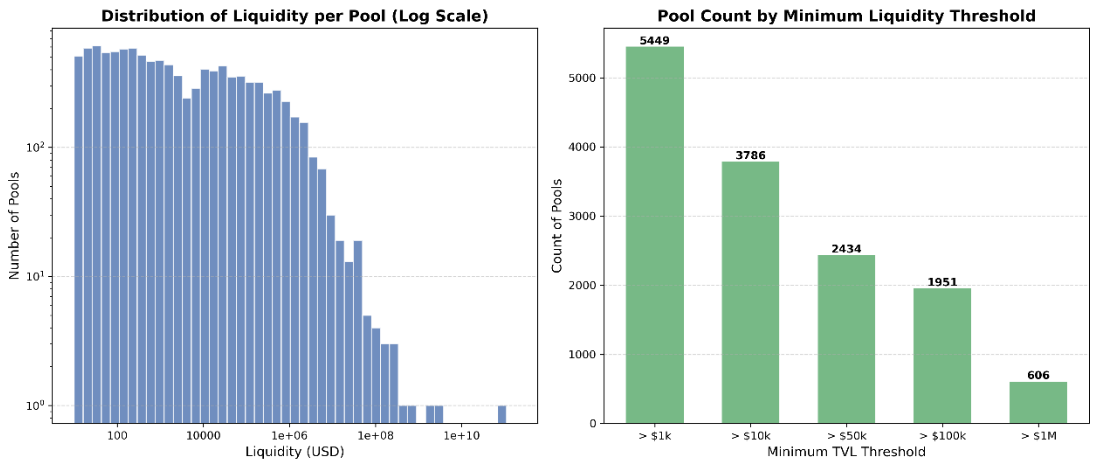

Figure 1: Data Distribution

We analyzed the liquidity distribution from the snapshot data (v2pools.json) to establish efficient filtering criteria.

- **Liquidity Distribution:** As shown in **Figure 1 (Left)**, the pool liquidity follows a long-tail distribution. Most pools contain negligible liquidity ("noise"), which increases graph complexity without offering viable arbitrage depth.
- **Threshold Selection: Figure 1 (Right)** highlights the pool count by TVL thresholds.
  - Pools > \$1k: 5,449
  - Pools > \$100k: 1,951
  - **Decision:** We selected a minimum TVL threshold of **\$50,000 USD**. This removes the "long tail" of dust pools while retaining sufficient

connectivity for the graph, balancing computational speed with opportunity coverage.

- **Universal Search & Normalization:** Instead of restricting search strictly to WETH-based cycles, our system performs a universal search across all tokens. To standardize the ranking of heterogeneous opportunities, we implemented an internal Pricing Oracle using Breadth-First Search (BFS) rooted at WETH. This allows us to convert the gross profit of any cycle into an ETH-equivalent value, ensuring the most profitable opportunities are selected regardless of the starting asset.
- **Transaction gas fee:** The primary cost for a trader is the blockchain transaction fee. Therefore, in line with common modeling assumptions in the literature[1], we adopt a gas price of 32 GWei.

1.2. Detection Logic for Profitable Cycles

1.2.1 Literature Review & Algorithm Selection

Recent studies have extensively explored arbitrage identification on decentralized exchanges (DEXs). Zhou et al.[1]combined the line graph of the DEX token exchange graph with the **Bellman–Ford–Moore algorithm** for just-in-time discovery. Yan et al.[2] employed **depth-first search (DFS)** to systematically examine the impact of trading-path length on profitability. Further improvements have been proposed by Zhang et al.[3] and Mazor et al.[4] to broaden the detection of arbitrage opportunities. For our implementation, we selected the **Shortest Path Faster Algorithm (SPFA)**, a highly efficient variant of the Bellman–Ford algorithm, to detect negative-weight cycles in the transformed token graph. This choice provides a robust balance between detection completeness and computational performance for sparse graphs typical of DEX liquidity networks.

1.2.2 Implementation: Optimized SPFA

Prior studies show that transaction processing in Ethereum is subject to a tight timing window, with a practical generation deadline of roughly 10.5 seconds (calculated by Ethereum's average block time 13.5±0.12 seconds and typical network propagation delay of approximately 3 seconds). So, we need to pay attention to time limit.

Key Implementation Steps:

1. **Graph Parsing:** An adjacency list is built from the filtered snapshot data.
2. **Negative Cycle Detection:** The SPFA algorithm relaxes edges to minimize distance. If a node is updated excessively, it indicates a negative cycle (arbitrage loop).
3. **Bottleneck Detection:** Upon finding a cycle, a custom get_bottleneck function scans the entire path to identify the pool with the lowest liquidity. The input amount is capped at 50% of this bottleneck to ensure execution safety.

4. **Profit Optimization:** We apply a Golden Section Search algorithm to determine the precise input amount that maximizes Gross Profit, balancing marginal revenue against marginal slippage.

1.3. Ranking Metrics for Top 10 Selection

To select the best candidates for execution, we employed a **Net Profit** ranking system normalized to ETH.

- **Pricing Oracle (BFS):** Since cycles may occur between arbitrary tokens, we implemented an internal pricing oracle using **Breadth-First Search (BFS)** rooted at WETH. This allows us to convert the profit of any cycle into an ETH-equivalent value.
- Net Profit Calculation:

$$Net\ Profit\ (ETH) = (Gross\ Profit_{Cycle} \times Price_{ETH}) - Gas\ Cost\ (0.0128\ ETH)$$

- Selection Criteria (The WETH Filter):

While the SPFA engine detects universal cycles, we prioritized cycles that start and end with WETH. This ensures atomic execution (Flash Loan compatibility) and eliminates the inventory risk of holding altcoins. But we don't use that in this practice.

- **Result:** The system outputs the 10 WETH-based cycles with the highest Net Profit.

Part 2: Arbitrage Strategy & Signal Generation

2.1 Mechanism Design: Atomic Execution

To mitigate the execution risk inherent in asynchronous blockchain states (e.g., slippage or JIT liquidity), we deployed a custom Atomic Contract (ArbitrageBot.sol). This contract acts as a trustless gatekeeper between the C++ strategy engine and the on-chain liquidity pools.

- **Atomicity:** The entire arbitrage cycle is bundled into a single transaction. Either all swaps succeed, and the profit condition is met, or the entire state reverts.
- **Asset Agnostic:** Based on the strategy's "Universal Search" capability, the contract is designed to handle any ERC20 token path provided in the JSON payload, dynamically approving and swapping tokens.

2.2 Data Encoding and ABI Interface

The strategy engine (C++) serializes the arbitrage path and parameters into a JSON payload, which is then encoded via web3.js to interact with the contract's Application Binary Interface (ABI).

The entry point for execution is the executeArbitrage function. It accepts a dynamic array of token addresses representing the swap path.

2.3 Validation Logic

The contract enforces a strict **"Revert-on-Loss"** policy. Before finalizing the transaction, it performs an on-chain solvency check:

$$Balance_{end} \geq Balance_{start} + MinProfit$$

- **Logic:**
  1. Snapshot initial token balance.
  2. Execute swaps via Uniswap V2 Router.
  3. Compare final balance against the required threshold.
- **Outcome:** If the realized profit is lower than expected (due to market movements between C++ calculation and execution), the transaction reverts. This ensures **Zero Principal Loss** (excluding Gas fees).

2.4 Revert Conditions

The transaction is designed to Revert (fail explicitly) under the following conditions, ensuring zero principal loss (excluding gas) Part 3: Execution Simulation & Performance Analysis

Part 3: Execution Simulation & Performance Analysis

3.1 Simulation Setup

The system was backtested using a Remix VM.

- **Input Data:** Top 10 profitable cycles identified by the C++ SPFA engine.
- **Gas Assumptions:** Gas Price fixed at **32 Gwei**.
- **Network Latency:** Assumed sub-10.5s submission window.

3.2 Performance Findings

The system was backtested against 10 candidate signals generated by the C++ engine. The results indicate a 100% On-Chain Execution Success Rate, verifying the high precision of the off-chain get_bottleneck calculation.
However, simulation reveals a critical distinction between **Protocol Profit** and **Net Profit**. The smart contract acts as a strict Principal Protection Layer (ensuring $Token_{out} \geq Token_{in}$), but it is "Gas Blind." This empirically validates the necessity of the off-chain filter (Gross Profit > 0.0128 ETH) proposed in Part 1 to prevent "Micro-Profit" trades from wasting Gas.

3.3 Potential Optimizations

While the current system is functional, several optimizations could enhance profitability and win-rate in a production environment, including dynamic slippage control and MEV protection.

3.4 Conclusion

The simulation confirms that the combination of **Off-Chain SPFA Search** and **On-Chain Atomic Validation** creates a robust arbitrage system. Future iterations incorporating Flashbots and batching would further secure and scale the strategy.

Reference:

[1] Zhou, Liyi, Kaihua Qin, Antoine Cully, Benjamin Livshits, and Arthur Gervais. ”On the just-in-time discovery of profit-generating transactions in defi protocols.” In 2021 IEEE Symposium on Security and Privacy (SP), pp. 919-936. IEEE, 2021.

[2] Yan, Tao, Qiyue Shang, Yu Zhang, and Claudio J. Tessone. ”Optimizing Arbitrage Strategies on Uniswap: The Impact of Trading Path Length on Profitability and Opportunity Frequency.” In Proceedings of Blockchain Kaigi 2023 (BCK23), p. 011007. 2024.

[3] Zhang, Yu, Tao Yan, Jianhong Lin, Benjamin Kraner, and Claudio J. Tessone. ”An Improved Algorithm to Identify More Arbitrage Opportunities on Decentralized Exchanges.” In 2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), pp. 1-7. IEEE, 2024.

[4] Mazor, Ori, and Ori Rottenstreich. ”An empirical study of cross-chain arbitrage in decentralized exchanges.” In 2024 16th International Conference on COMmunication Systems & NETworkS (COMSNETS), pp. 488-496. IEEE, 2024.