Stablecoin Arbitrage: Two-Level Search and Cross-Exchange Algorithm Analysis

Kevin Litvin, Ario Katchooi, Karan Singh
Simon Fraser University

November 2025

1 Introduction

1.1 Problem Statement

Stablecoin arbitrage refers to the exploitation of price discrepancies for stablecoins across multiple cryptocurrency exchanges. Despite stablecoins being pegged to fiat currencies (like USD or CAD), fragmentation of liquidity and latency in updating prices cause their exchange rates to differ from one platform to another [1, 2]. These differences create opportunities for profit, but exploiting them is non-trivial due to transfer fees, exchange latency, and coin volatility.

1.2 Mathematical Formulation

The arbitrage environment is modeled as a directed graph $G = (V, E)$:

- Each node $v \in V$ represents a unique (exchange, stablecoin) pair. For example, (Kraken, USDT), (Coinbase, USDC), and (Coinbase, CAD stablecoin).
- Each directed edge $e = (v_i, v_j) \in E$ represents the possibility of transferring value from node v_i to node v_j. Every edge has an associated transfer cost W_{ij}, defined as

  $$W_{ij} = f_{\text{fee}} + f_{\text{vol}}(t)$$

  where f_{fee} covers all transactional and withdrawal/deposit fees, while $f_{\text{vol}}(t)$ captures slippage and volatility risk during the execution window t [2, 1].

For each node v, let P_v denote the quoted price in the reference fiat currency. The arbitrage condition between nodes is:

$$\Delta_{ij} = P_{v_i} - P_{v_j} > W_{ij}$$

or, for ratio-based arbitrage,

$$\frac{P_{v_i}}{P_{v_j}} > 1 + W_{ij}$$

Whenever the net price difference exceeds the total cost of transfer, a viable arbitrage opportunity is detected.

1.3 Objective

The goal is to develop an automated agent that can:

- Continuously monitor prices and costs across multiple exchanges and stablecoins,
- Execute a two-level search (comparing all pairs of stablecoins between every pair of exchanges),
- Traverse optimal arbitrage cycles (using Dijkstra or A* search with a suitable heuristic for volatility/time),
- Adaptively update to maximize realized profit while minimizing exposure to adverse market movements.

The work explores not only the identification of profitable paths at a single time-point but also assesses algorithmic robustness under live price feeds, incorporating transaction fees, latencies, and potential cross-currency conversions. Challenging market scenarios—including rapidly shifting prices and illiquid coins—are considered to benchmark and compare solution performance [2, 1, 3].

2 Implementation

2.1 Overview of Algorithms

Our system models the multi-exchange stablecoin arbitrage environment as a directed weighted graph. The two-level search algorithm enables agents to systematically compare all possible pairs of stablecoins across all pairs of exchanges, seeking sequences where the price differential exceeds the cumulative costs (fees, slippage, volatility) [2, 1]. We employ two algorithms:

- **Dijkstra's Algorithm:** For finding the least-cost path.
- **A* Search Algorithm:** Adds a volatility/time cost heuristic to prioritize safer cycles [3].

2.2 Pseudocode

Listing 1: Stablecoin Arbitrage: Two-Level Comparison (A* variant)

```
class ExchangeNode:
    def __init__(self, exchange, stablecoin, price):
        self.exchange = exchange
        self.stablecoin = stablecoin
        self.price = price
        self.edges = []

class ArbitrageAgent:
    def __init__(self, graph):
        self.graph = graph
```

```
def find_arbitrage_paths(self, start_node):
    frontier = PriorityQueue()
    frontier.put((0, start_node))
    visited = set()
    while not frontier.empty():
        cost, current = frontier.get()
        if current in visited:
            continue
        visited.add(current)
        for neighbor in current.edges:
            transfer_cost = neighbor.weight # cost + volatility
            price_diff = current.price - neighbor.price
            if price_diff > transfer_cost:
                record_opportunity(current, neighbor, price_diff, transfer_cost)
            future_risk = heuristic(current, neighbor)
            total_cost = transfer_cost + future_risk
            frontier.put((cost + total_cost, neighbor))
```

2.3 Implementation Decisions

- **API Integration:** Modular Python classes fetch live price feeds from exchange APIs, normalizing data [1].
- **Graph Construction:** Nodes for each (exchange, stablecoin) combination; edge weights model cost/volatility [**?**].
- **Performance Optimization:** Batched/cached API calls, heuristics for risk, and dynamic portfolio adjustments.
- **Two-Level Search:** All stablecoin pairs are considered, not just identical coins.

3 Methodology

3.1 Research Questions

- Does two-level search provide consistently higher profit and better robustness than traditional cross-exchange single-coin search [1, 2]?
- In what scenarios does A* with volatility/time heuristic outperform Dijkstra's?
- How does algorithmic performance scale with instance size?
- Are some instance classes particularly challenging (e.g., high correlation, liquidity drops, extreme fees) [1]?
- If suboptimal, how close are solutions to the optimum?

3.2 Experimental Design

Classes of test instances:

- **Synthetic instances:** Handcrafted small networks with controlled parameters.
- **Live API data:** Real-time Kraken/Coinbase feeds.
- **Adversarial benchmarks:** Highly volatile, illiquid, or fee-asymmetric scenarios based on literature.

Parameters varied: number of nodes, coins, transfer cost/volatility, agent portfolio size, API limits.

3.3 Metrics

- Number of arbitrage opportunities found
- Cumulative simulated profit
- Optimality gap from theoretical maximum
- Computational resources/cost
- Solution stability with respect to parameter changes

3.4 Sources and Benchmarks

Synthetic instances are documented in the appendix. Live data from exchanges; challenging scenarios are drawn from published research [2, 3]. All code/instance generators are open in the project repository.

4 Experimental Setup

4.1 System Environment

- **Programming Language:** Python 3.11
- **Libraries:** pandas, numpy, requests, SQLAlchemy, matplotlib
- **OS:** Ubuntu 22.04 LTS
- **Processor:** Intel Core i7-11800H (8c/16t)
- **Memory:** 32GB RAM
- **Internet:** Required for live API data (Kraken, Coinbase)

4.2 Codebase Organization

- `/src/`: Algorithms, connectors, data models
- `/experiments/`: Scripts and analysis notebooks
- `/data/`: Synthetic instances, logs, sample API data
- `requirements.txt`: Locked dependencies

4.3 API Integration and Reproducibility

Kraken and Coinbase APIs are queried via public endpoints. Data normalization and error checks are applied. Experiments are version-controlled (Git), with logs and seeds for reproducibility.

4.4 Limitations

Network and rate limits affect some experiments. Synthetic and adversarial runs are local for stability.

5 Experimental Results

5.1 Performance Visualization

Experiments were run using both synthetic and real-time data for multiple stablecoins and 2–10 exchanges.

5.2 Algorithm Comparisons

- Two-level search detected more cycles/profit than traditional methods [1, 3].
- A* (heuristic) yielded the best results in volatile/large scenarios [1, 2].
- Single-level approaches missed viable cycles, especially under volatility or API lag.

5.3 Table: Representative Results

Instance Size	Algorithm	Cycles	Avg. Net Profit	CPU Time (s)
4 exchanges	Dijkstra	8	\$21.7	0.36
4 exchanges	A*	9	\$24.3	0.44
8 exchanges	Dijkstra	22	\$48.2	1.11
8 exchanges	A*	25	\$53.7	1.27

5.4 Other Results

A* solutions were typically 95–99% optimal unless faced with extreme volatility/illiquidity. Performance scaled near-linearly until 100 nodes, after which rate/APIs bottlenecked.

6 Robustness Enhancements and System Extensions

6.1 Addressing Exchange Flaws

To reduce systemic risks and better reflect real-world trading conditions [**?**], our design roadmap includes:

- **Exchange Diversification:** Integrating both reputable and less-reliable exchanges, using advanced risk filters to exploit opportunity while detecting/avoiding hazardous platforms (e.g., withdrawal freezes, price manipulation).
- **Fiat/Stablecoin Universe Expansion:** Adding support for stablecoins pegged to emerging market currencies. For these, additional cost, volatility, and failure risk components are integrated into profit estimates.

6.2 Resilient API and Data Quality Control

We plan to:

- Build adaptive rate-limiting, retry, and normalization middleware to absorb API failures and maintain uninterrupted detection.
- Incorporate live anomaly detection that flags and deprioritizes exchanges showing frequent errors, price glitches, or systemic operation problems.

6.3 Dynamic Risk Modeling

Future versions of the volatility-time heuristic and edge weighting will simulate not only historical slippage and transfer costs, but also failed transfer events and payout uncertainty, using these to guide path selection in adversarial or high-risk conditions.

6.4 Performance and Scalability

- Move core routines to parallelized or asynchronous execution to process larger networks and reduce latency, especially when handling slow APIs.
- Use batch data collection and proactive cache invalidation to minimize redundant calls, boosting both efficiency and detection rate.

6.5 Continual Learning and Adaptive Management

- Add continual learning agents to update fee, volatility, and risk models using real (not just simulated) transaction outcomes.
- Apply reinforcement learning or metaheuristics for cycle selection and path planning in non-stationary/adversarial markets.

7 Research Design Enhancements and Benchmarks

- Stress-test the system with volatile, low-liquidity coins and high-adversity datasets, not just top stablecoins or "safe" exchanges.
- Benchmark robustness by directly measuring realized loss, frequency of missed cycles, and recovery from API or exchange failures before and after applying new risk and error management strategies.

By targeting both technical and systemic flaws, and enhancing real-time risk management and computational performance, our research aims to develop a robust and flexible arbitrage agent suited for both mainstream and adversarial environments [**?**].

8 Conclusions

Two-level search, especially with A* heuristic, provides the best profit and robustness for stablecoin arbitrage. Poor performance followed from highly correlated or illiquid markets and unpredictable APIs. Excellent performance stemmed from exchanges with uncorrelated price feeds and asymmetric fee structures. Future work should add DEXs, multi-agent strategies, and continual learning for risk mitigation [2, 1, 3].

9 Bibliography

References

[1] BlockApps, "Mastering Cross-Exchange Arbitrage with Stablecoins," 2024.

[2] N. Author, "Arbitrage among Stablecoins," Clemson University Thesis, 2022.

[3] Group 39, "Recursive MA-CBS," CMPT 417 Undergraduate Sample Report (2021).

[4] R. Oant and A. Coroiu, "Crypto Advisor: A Web Application for Spotting Cross-Exchange Cryptocurrency Arbitrage Opportunities," CSEDU 2023, pp. 238–246.

[5] S. Nakamoto, "Bitcoin A peer-to-peer electronic cash system", 2008.

[6] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*, 1968.

[8] CCXT Library, `https://github.com/ccxt/ccxt`

[9] Kraken API Docs, `https://docs.kraken.com`

[10] Coinbase API Docs, `https://docs.cdp.coinbase.com`