Retrospectively Identifying Profit-Generating Transactions on the Algorand Blockchain

Jonas Gebele, November 06, 2023, Final Presentation Application Project

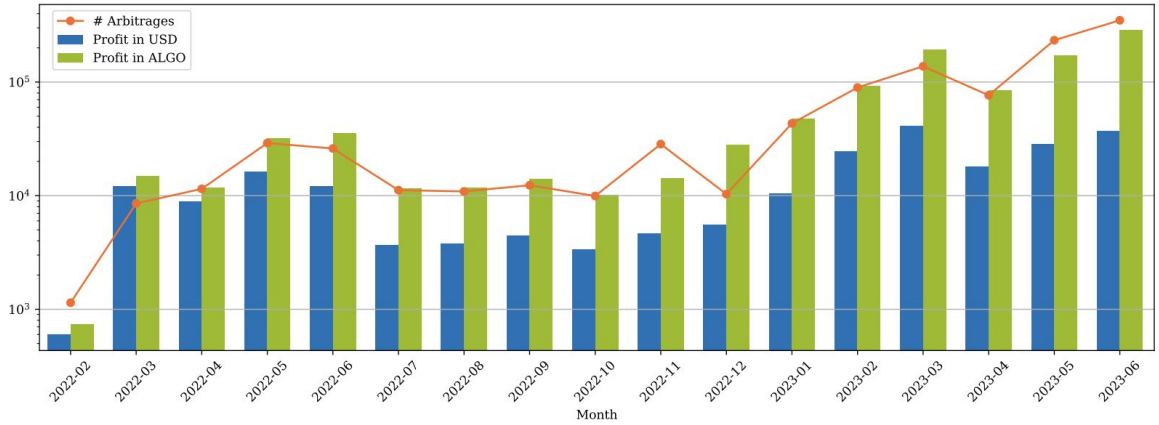
Chair of Software Engineering for Business Information Systems (sebis)
Department of Computer Science
School of Computation, Information and Technology (CIT)
Technical University of Munich (TUM)
wwwmatthes.in.tum.de

Outline

1. Background Information
2. Research Objectives
   2.1. Research Efficient Algorithms for Arbitrage Detection on Decentralized Exchange Graphs
   2.2. Establishing a Methodology for Profitable Execution of Arbitrage Opportunities
   2.3. Recreation of Historical States on the Algorand Blockchain
   2.4. Implementation & Analysis of Developed Algorithms on Historical States
3. Conclusion and Future Work

Background Information

- MEV originally studied within Ethereum, extends to Algorand
- Different blockchains characteristics affect MEV extraction patterns
- Study by Öz et al. (2023) examines Algorand's MEV landscape, highlighting arbitrage patterns and key actors
- Single dominant searcher executed 600,000 across over 1 million arbitrage cases



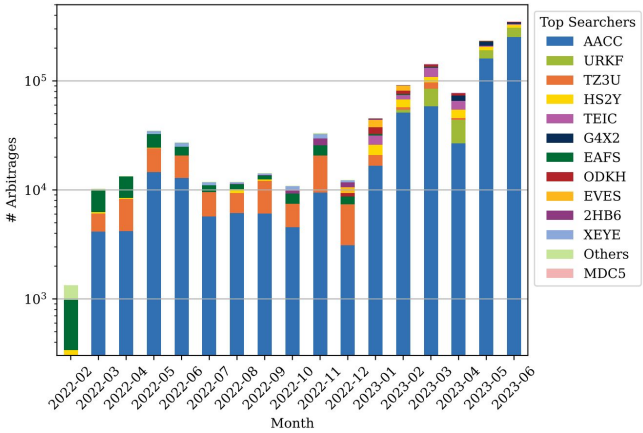
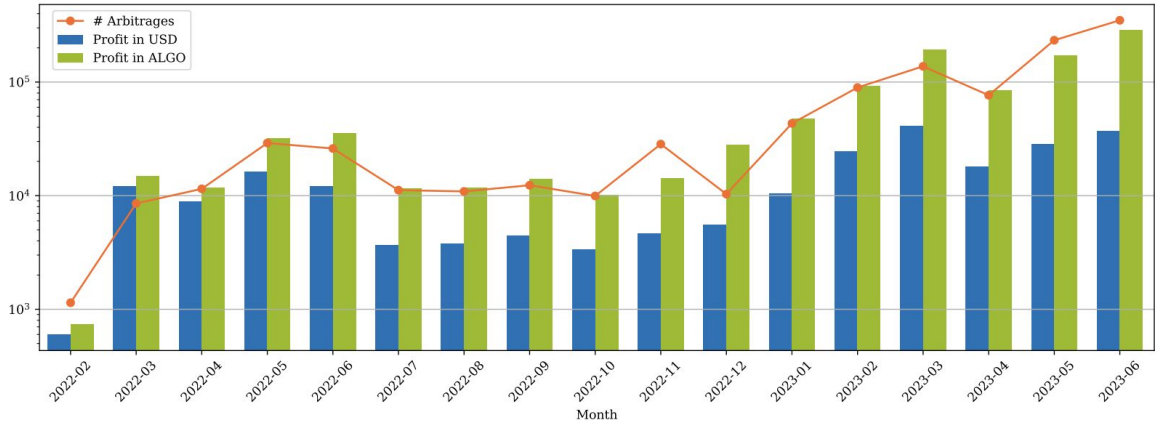
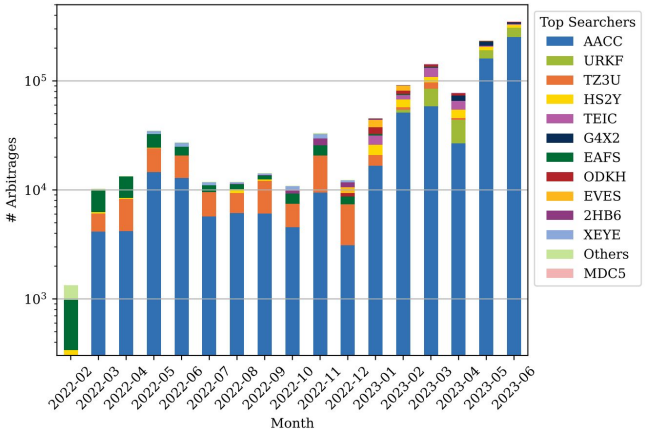


Image Sources: Öz et al. (2023)

Background Information

- MEV originally studied within Ethereum, extends to Algorand
- Different blockchains characteristics affect MEV extraction patterns
- Study by Öz et al. (2023) examines Algorand's MEV landscape, highlighting arbitrage patterns and key actors
- Single dominant searcher executed 600,000 across over 1 million arbitrage cases



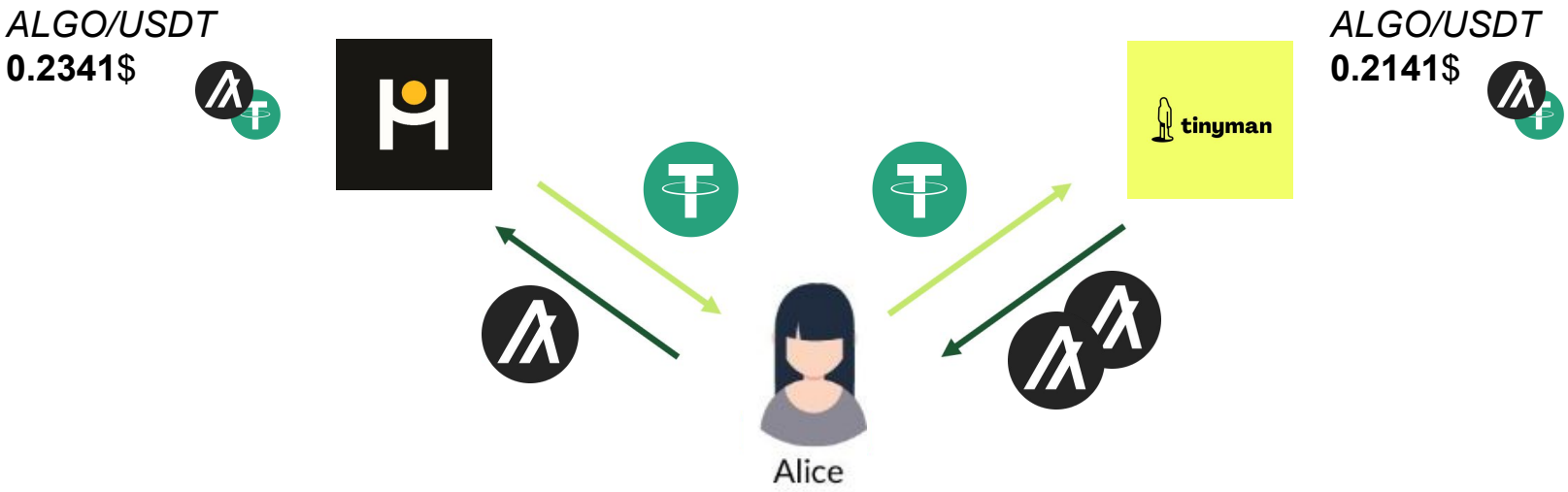


Are the arbitrages taken in an optimal way?
Can we compete with the searchers?

Image Sources: Öz et al. (2023)

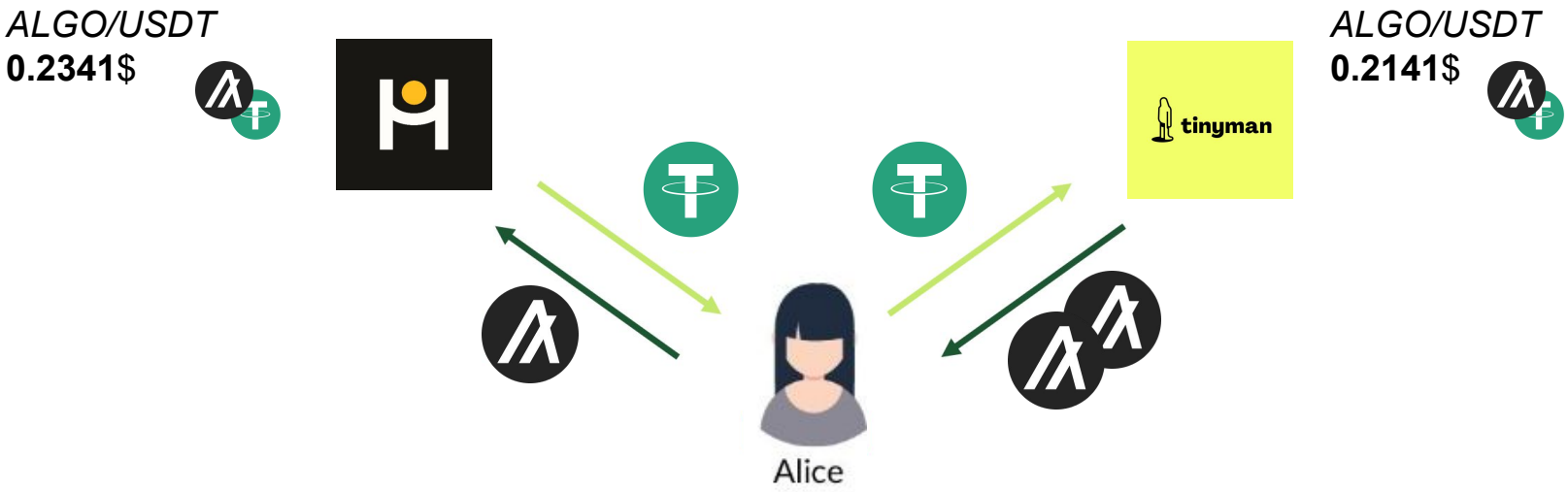
Background Information

Arbitrage



Background Information

Arbitrage



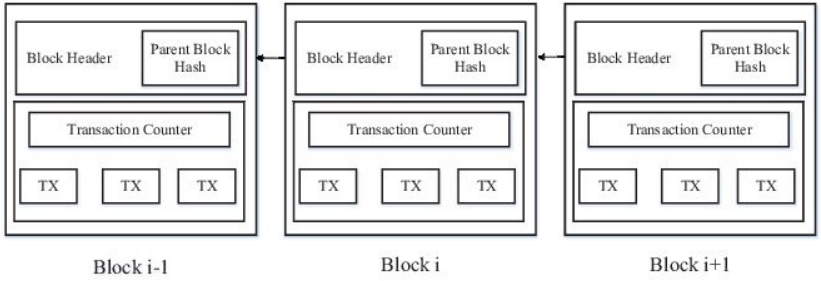
Alternative Types of Arbitrage

- Multi-Hop Arbitrage
- Same Asset-Type Arbitrage

Background Information

Blockchain as State Machine

At any given block, a blockchain has a state, consisting of all balances, smart contracts, etc.



Each block represents a state transition where transactions and smart contracts alter the state.

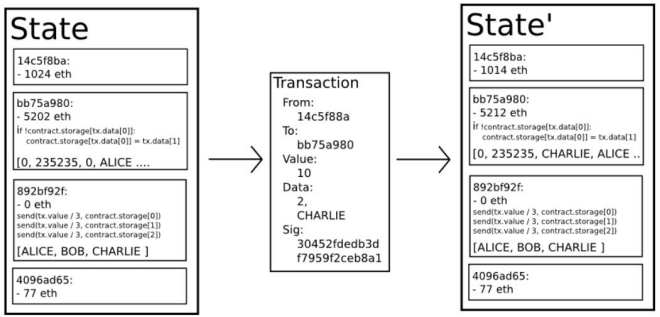


Image Sources: Yiping Liu, China Cyber Security Annual Conference (2021)

Outline

1. Background Information
2. Research Objectives
   2.1. Research Efficient Algorithms for Arbitrage Detection on Decentralized Exchange Graphs
   2.2. Establishing a Methodology for Profitable Execution of Arbitrage Opportunities
   2.3. Recreation of Historical States on the Algorand Blockchain
   2.4. Implementation & Analysis of Developed Algorithms on Historical States
3. Conclusion and Future Work

Research Questions

Research Questions

1. Research Efficient Algorithms for Arbitrage Detection on Decentralized Exchange Graphs
2. Establishing a Methodology for Profitable Execution of Arbitrage Opportunities

Practical Implementation

3. Recreation of Historical States on the Algorand Blockchain

(4. Implementation & Analysis of Developed Algorithms on Historical States)

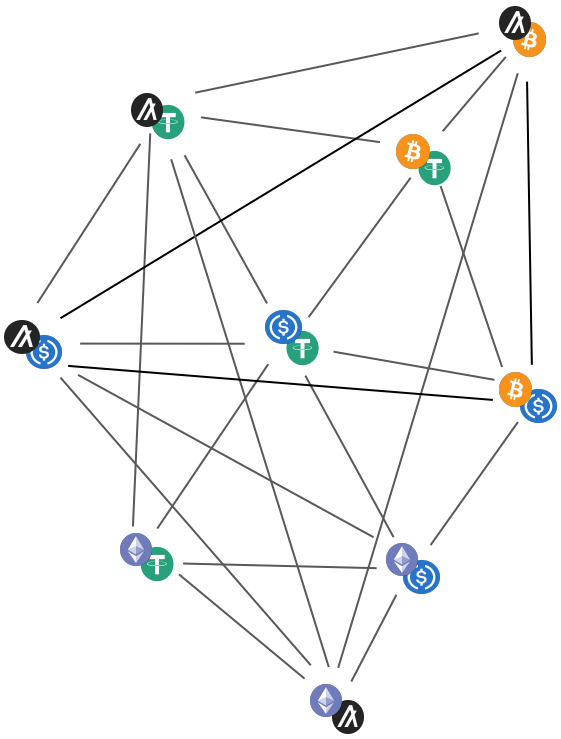
Identification of Profit-Generating Transactions

1. Detecting Profitable Arbitrage Cycles

- Negative Cycle Detection in Directed Graphs
- Trace-Based Discovery of Negative Cycles in Adjacency Sub-Matrices

2. Input Optimization for Profitable Arbitrage Execution

- Binary Search Algorithm
- Convex Optimization



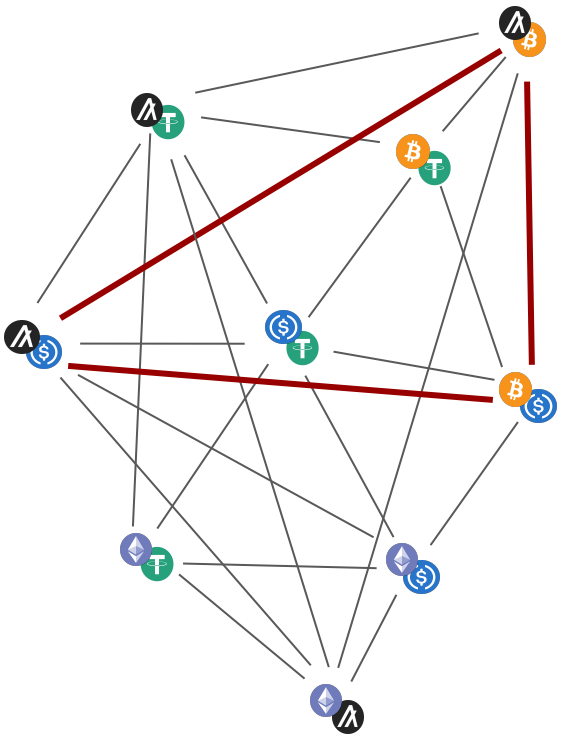
Identification of Profit-Generating Transactions

1. Detecting Profitable Arbitrage Cycles

- **Negative Cycle Detection in Directed Graphs**
- Trace-Based Discovery of Negative Cycles in Adjacency Sub-Matrices

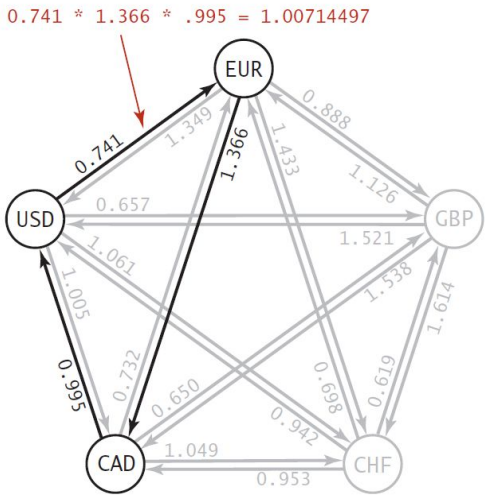
2. Input Optimization for Profitable Arbitrage Execution

- Binary Search Algorithm
- Convex Optimization



Negative Cycle Detection in Directed Graphs

Detection Profitable Arbitrage Cycles



Cast Problem as a **Directed Graph**

- **Assets** (Nodes)
- **Exchanges** given Assets (Vertices)
- **Exchange-Rates** (Weights)

Search for Profitable Arbitrage Cycles

- Exchange-Rate Product > 1

Weight Transformation to Utilize Graph-Search Algorithms

1. Bellman Ford (greedy)
2. Johnson's Algorithm (sparse graphs)
3. Floyd-Warshall Algorithm (dense graphs)

$$x_1 \cdot x_2 \cdot \ldots \cdot x_k > 1$$
$$\Leftrightarrow \frac{1}{x_1} \cdot \frac{1}{x_2} \cdot \ldots \cdot \frac{1}{x_k} < 1$$
$$\Leftrightarrow \ln\left(\frac{1}{x_1} \cdot \frac{1}{x_2} \cdot \ldots \cdot \frac{1}{x_k}\right) < \ln(1)$$
$$\Leftrightarrow \ln\left(\frac{1}{x_1}\right) + \ln\left(\frac{1}{x_2}\right) + \ldots + \ln\left(\frac{1}{x_k}\right) < 0$$

\+ easy to implement & run

\- difficult to scale (scale up)
\- runtime dependencies
\- does not account for liquidity and slippage

Image Source: Robert Sedgewick, Algorithms, 4th Edition

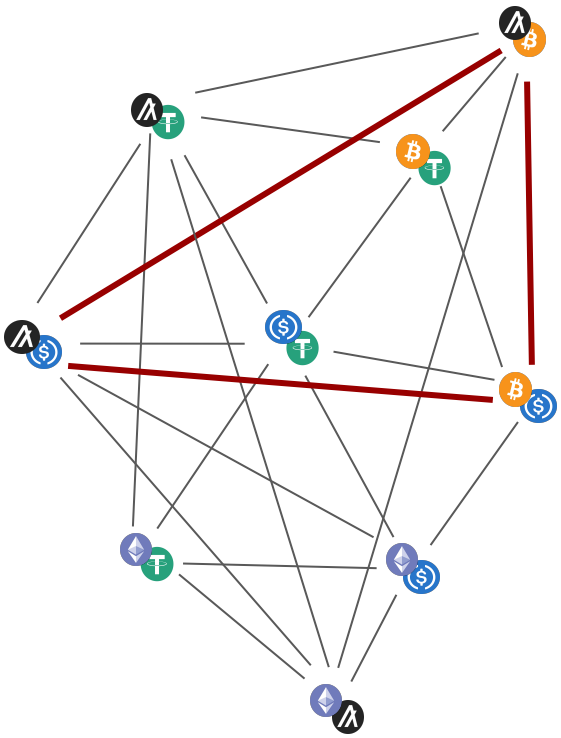
Identification of Profit-Generating Transactions

1. Detecting Profitable Arbitrage Cycles

- Negative Cycle Detection in Directed Graphs
- **Trace-Based Discovery of Negative Cycles in Adjacency Sub-Matrices**

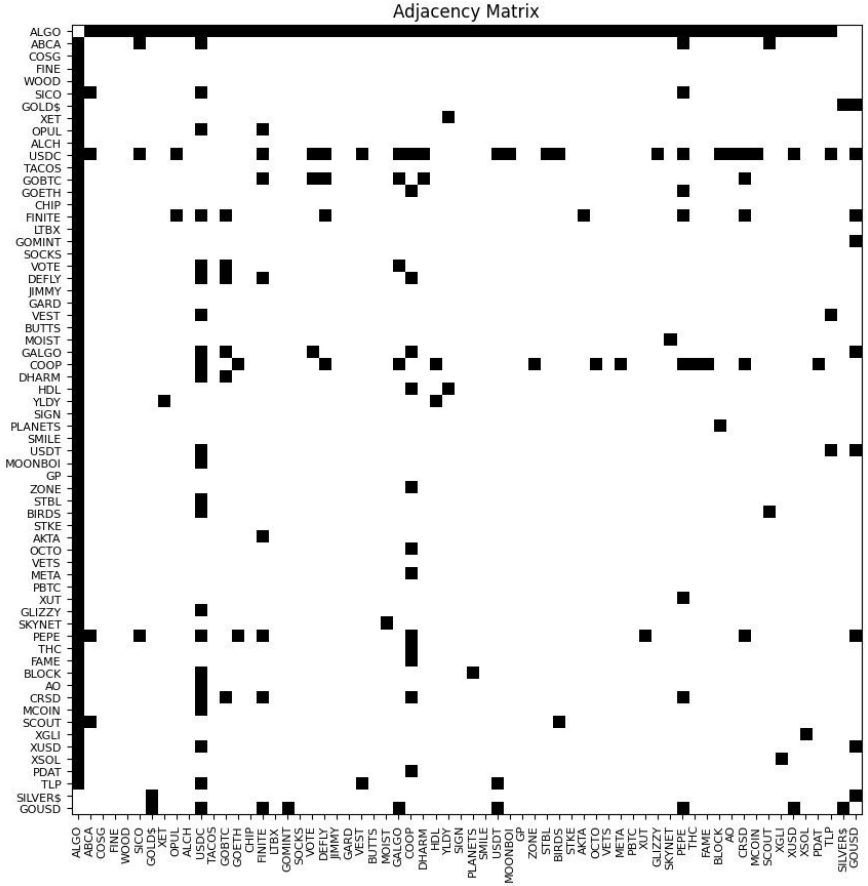
2. Input Optimization for Profitable Arbitrage Execution

- Binary Search Algorithm
- Convex Optimization



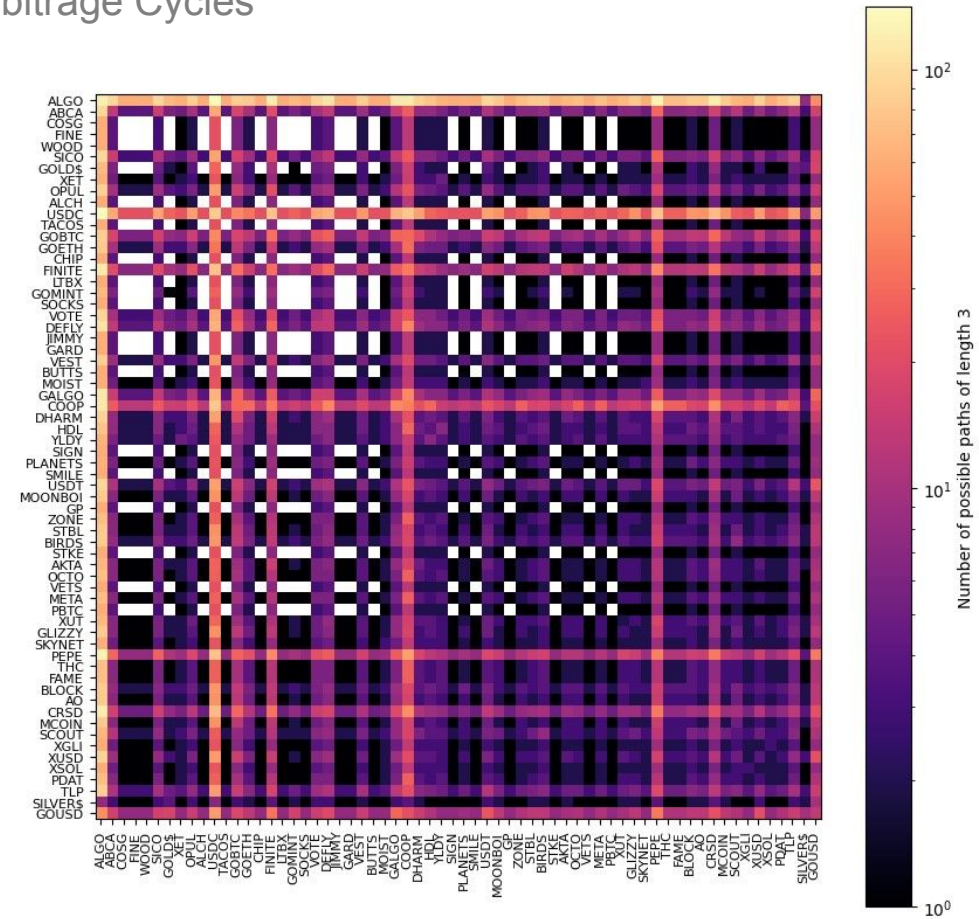
Traced-Based Discovery of Negative Cycles in Adjacency Sub-Matrices

Detection Profitable Arbitrage Cycles



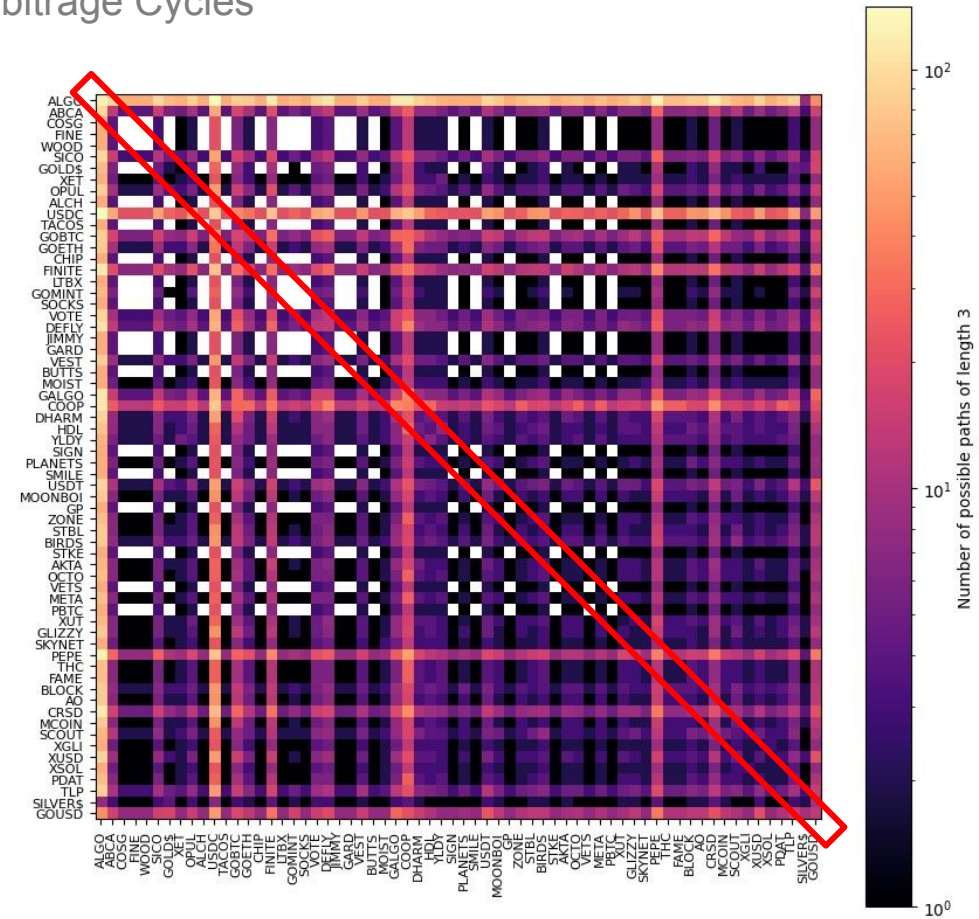
Traced-Based Discovery of Negative Cycles in Adjacency Sub-Matrices

Detection Profitable Arbitrage Cycles



Traced-Based Discovery of Negative Cycles in Adjacency Sub-Matrices

Detection Profitable Arbitrage Cycles



Traced-Based Discovery of Negative Cycles in Adjacency Sub-Matrices

Detection Profitable Arbitrage Cycles

Receiving

Tendering

$$\begin{array}{c} \\ \\ \\ \end{array}\begin{bmatrix} 1 & \boxed{\frac{1}{20000}} & \frac{1}{1000} \\ 20000 & 1 & \boxed{20} \\ \boxed{1000} & \frac{1}{20} & 1 \end{bmatrix}$$

First trade $\xrightarrow{\frac{1}{20000}}$

Second trade $\xrightarrow{20}$

Third trade $\xrightarrow{1000}$

$$\frac{1}{20000} \cdot 20 \cdot 1000 = 1$$

Traced-Based Discovery of Negative Cycles in Adjacency Sub-Matrices

Detection Profitable Arbitrage Cycles

Receiving

Tendering

$$\begin{array}{c} \\ \$ \\ ฿ \\ \Xi \end{array} \begin{array}{c} \begin{array}{ccc} \$ & ฿ & \Xi \end{array} \\ \begin{bmatrix} 1 & \frac{1}{20000} & \frac{1}{1000} \\ 20000 & 1 & 20 \\ 1000 & \frac{1}{20} & 1 \end{bmatrix} \end{array} \begin{array}{c} \\ 1 \\ 20000 \\ 1000 \end{array}$$

$$\begin{array}{c} \\ \$ \\ ฿ \\ \Xi \end{array} \begin{array}{c} \begin{array}{ccc} \$ & ฿ & \Xi \end{array} \\ \begin{bmatrix} 1 & \frac{1}{20000} & \frac{1}{1000} \\ 20000 & 1 & 20 \\ 1000 & \frac{1}{20} & 1 \end{bmatrix} \end{array}$$

Trivial Cycle(s)

$$\begin{array}{c} \\ \$ \\ ฿ \\ \Xi \end{array} \begin{array}{c} \begin{array}{ccc} ฿ & \Xi & \$ \end{array} \\ \begin{bmatrix} \frac{1}{20000} & \frac{1}{1000} & 1 \\ 1 & 20 & 20000 \\ \frac{1}{20} & 1 & 1000 \end{bmatrix} \end{array}$$

Cycle after **one** left-shift

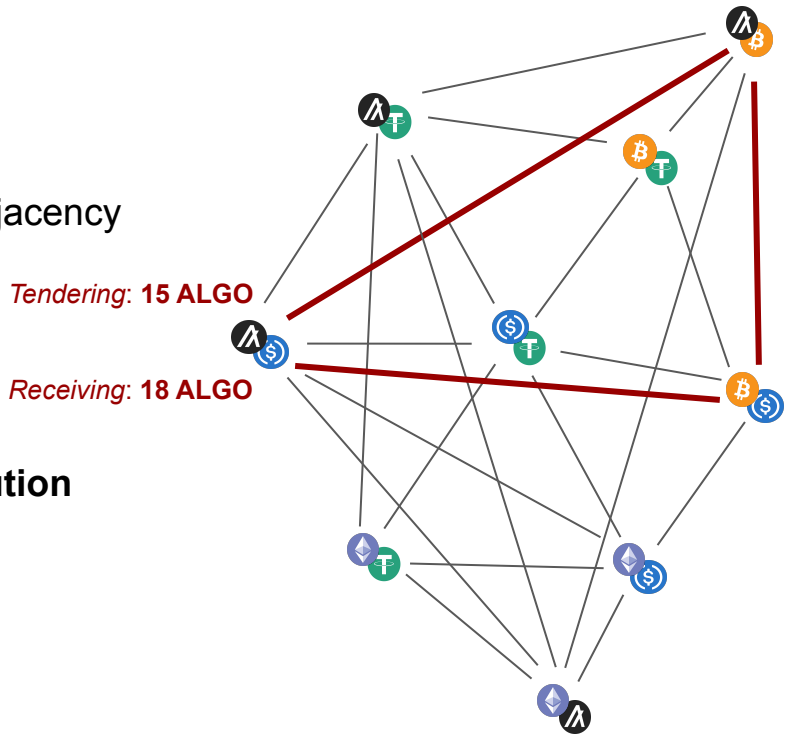
$$\begin{array}{c} \\ \$ \\ ฿ \\ \Xi \end{array} \begin{array}{c} \begin{array}{ccc} \Xi & ฿ & \$ \end{array} \\ \begin{bmatrix} \frac{1}{1000} & 1 & \frac{1}{20000} \\ 20 & 20000 & 1 \\ 1 & 1000 & \frac{1}{20} \end{bmatrix} \end{array}$$

Cycle after **two** left-shift

Identification of Profit-Generating Transactions

1. Detecting Profitable Arbitrage Cycles

- Negative Cycle Detection in Directed Graphs
- Trace-Based Discovery of Negative Cycles in Adjacency Sub-Matrices



2. Input Optimization for Profitable Arbitrage Execution

- Binary Search Algorithm
- **Convex Optimization**

Transaction Input Optimization for Profitable Arbitrage Execution

Convex Optimization

$$\text{maximize} \quad U(\Psi)$$

$$\text{subject to} \quad \Psi = \sum_{i=1}^{m} A_i(\Lambda_i - \Delta_i)$$

$$\phi_i(R_i + \gamma_i\Delta_i - \Lambda_i) \geq \phi_i(R_i), \quad i = 1, \ldots, m$$

$$\Delta_i \geq 0, \Lambda_i \geq 0, \quad i = 1, \ldots, m.$$

Objective Function

Trade Input-Output Constraints

Decentralized Exchange Price Invariants

Positive Input-Output Constraints

```
------------------- ARBITRAGE TRADES + EXECUTION ORDER -------------------

TRADE POOL = HUMBLESWAP_ALGO_USDC
        TENDERING 15347.386040985702 TOKEN 0
        RECEIVEING 2663.3708938162385 TOKEN 1

TRADE POOL = ALGOFI_ALGO_USDC
        TENDERING 2663.3708938162213 TOKEN 1
        RECEIVEING 15898.964361424953 TOKEN 0

TRADE POOL = HUMBLESWAP_ALGO_USDT
        TENDERING 4.738312510320262e-05 TOKEN 0
        RECEIVEING 6.192878195048184e-06 TOKEN 2

TRADE POOL = ALGOFI_ALGO_USDT
        TENDERING 6.192860819319015e-06 TOKEN 2
        RECEIVEING 3.8171442004638095e-05 TOKEN 0

TRADE POOL = HUMBLESWAP_ALGO_goUSD
        TENDERING 6.572196010014498e-07 TOKEN 0
        RECEIVEING 1.7160162158095875e-11 TOKEN 3
```

Research Questions

Research Questions

1. Research Efficient Algorithms for Arbitrage Detection on Decentralized Exchange Graphs
2. Establishing a Methodology for Profitable Execution of Arbitrage Opportunities

Practical Implementation

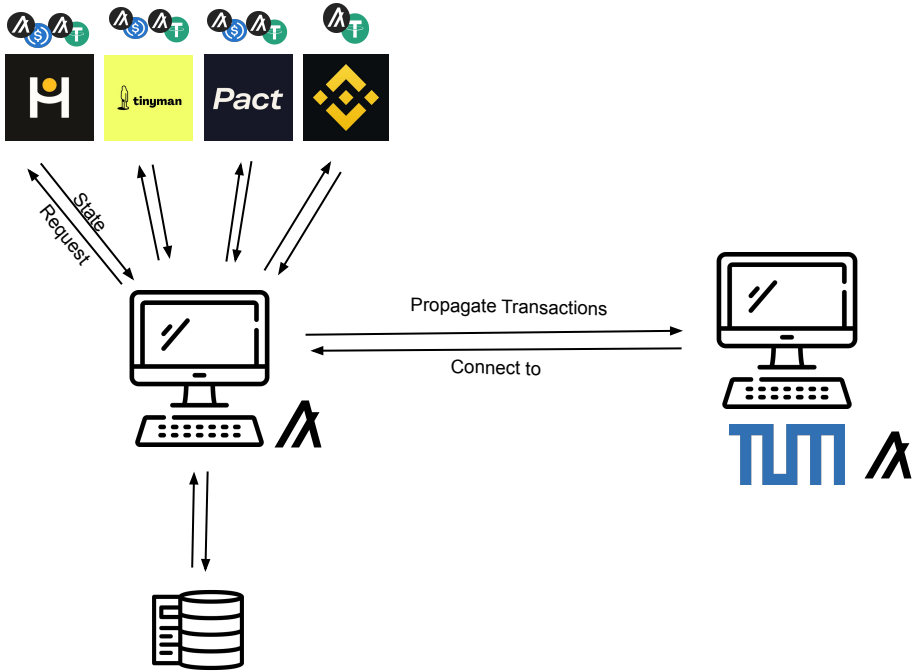
3. Recreation of Historical States on the Algorand Blockchain

(4. Implementation & Analysis of Developed Algorithms on Historical States)

Recreation of Historical States on the Algorand Blockchain

Generation of Past States

- *63 tokens*
- *over 360 markets*




```
0.18465,0.185404,0.194214,0.184586,0.182313,0.184843,0.187997
```

Analysis of Algorithm on Historical States

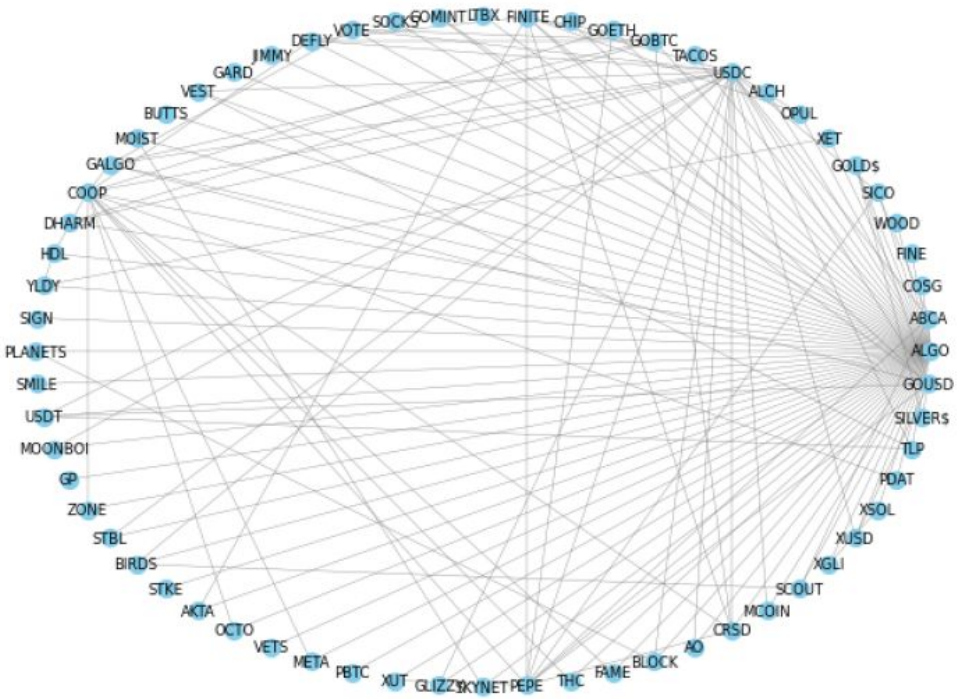
Practical Implementation

Implementation:

```
G = build_graph()
number_hops = [2,3,4]

# Build matrices
for each hop in number_hops:
    get_possible_cycles()
    get_exchange_pairs()
    generate_matrix_combinations()

# Detect arbitrage opportunities
for each block in dataset:
    parse_transactions()
    get_affected_exchanges()
    update_matrix_combinations()
    for each matrix in matrix_combinations:
        shift_matrix()
        calculate_trace()
        if trace > 1:
            calculate_optimal_inputs()
            execute_arbitrage()
```



Analysis of Algorithm on Historical States

Practical Implementation

Implementation:

```python
G = build_graph()
number_hops = [2,3,4]

# Build matrices
for each hop in number_hops:
    get_possible_cycles()
    get_exchange_pairs()
    generate_matrix_combinations()

# Detect arbitrage opportunities
for each block in dataset:
    parse_transactions()
    get_affected_exchanges()
    update_matrix_combinations()
    for each matrix in matrix_combinations:
        shift_matrix()
        calculate_trace()
        if trace > 1:
            calculate_optimal_inputs()
            execute_arbitrage()
```

Output:

```
Found 12 cycles of length 3 or less for USDT (312769)
['312769', '987374809', '312769']
['312769', '987374809', '31566704', '312769']
['312769', '987374809', '0', '312769']
['312769', '672913181', '312769']
['312769', '672913181', '31566704', '312769']
['312769', '31566704', '312769']
['312769', '31566704', '987374809', '312769']
['312769', '31566704', '672913181', '312769']
['312769', '31566704', '0', '312769']
['312769', '0', '312769']
['312769', '0', '987374809', '312769']
['312769', '0', '31566704', '312769']
```

Analysis of Algorithm on Historical States

Practical Implementation

Implementation:

```python
G = build_graph()
number_hops = [2,3,4]

# Build matrices
for each hop in number_hops:
    get_possible_cycles()
    get_exchange_pairs()
    generate_matrix_combinations()

# Detect arbitrage opportunities
for each block in dataset:
    parse_transactions()
    get_affected_exchanges()
    update_matrix_combinations()
    for each matrix in matrix_combinations:
        shift_matrix()
        calculate_trace()
        if trace > 1:
            calculate_optimal_inputs()
            execute_arbitrage()
```

Output:

```
All possible potential pools for the cycles (number = 16):
(31566704, 0)
(987374809, 0)
(0, 987374809)
(672913181, 31566704)
(312769, 987374809)
(0, 31566704)
(312769, 672913181)
(312769, 31566704)
(31566704, 312769)
(987374809, 312769)
(312769, 0)
(31566704, 987374809)
(31566704, 672913181)
(672913181, 312769)
(987374809, 31566704)
(0, 312769)
```

Analysis of Algorithm on Historical States

Practical Implementation

Implementation:

```python
G = build_graph()
number_hops = [2,3,4]

# Build matrices
for each hop in number_hops:
    get_possible_cycles()
    get_exchange_pairs()
    generate_matrix_combinations()

# Detect arbitrage opportunities
for each block in dataset:
    parse_transactions()
    get_affected_exchanges()
    update_matrix_combinations()
    for each matrix in matrix_combinations:
        shift_matrix()
        calculate_trace()
        if trace > 1:
            calculate_optimal_inputs()
            execute_arbitrage()
```

Output:

```
Suitable pools (number = 28):
tinyman2 - 312769 (TLP) <> 987374809 (USDT). Reserves TLP/USDT: 303866.0/531810658.0. Fee: 0.997
pact - 312769 (USDT) <> 672913181 (GOUSD). Reserves USDT/GOUSD: 31436458102.0/36109538099.0. Fee: 0.99003
tinyman2 - 31566704 (TLP) <> 987374809 (USDC). Reserves TLP/USDC: 2315581.0/4052934621.0. Fee: 0.997
tinyman2 - 312769 (USDC) <> 31566704 (USDT). Reserves USDC/USDT: 229838703105.0/230824062879.0. Fee: 0.997
tinyman1 - 312769 (USDC) <> 31566704 (USDT). Reserves USDC/USDT: 8655871851.0/8698188624.0. Fee: 0.997
tinyman1 - 0 (USDC) <> 31566704 (ALGO). Reserves USDC/ALGO: 57788975062.0/581594046755.0. Fee: 0.997
tinyman2 - 31566704 (TLP) <> 987374809 (USDC). Reserves TLP/USDC: 2315581.0/4052934621.0. Fee: 0.997
tinyman2 - 0 (USDT) <> 312769 (ALGO). Reserves USDT/ALGO: 12815919257.0/128950260512.0. Fee: 0.997
tinyman1 - 312769 (TLP) <> 987374809 (USDT). Reserves TLP/USDT: 3818118.0/6674124437.0. Fee: 0.997
tinyman1 - 312769 (TLP) <> 987374809 (USDT). Reserves TLP/USDT: 3818118.0/6674124437.0. Fee: 0.997
tinyman2 - 312769 (USDC) <> 31566704 (USDT). Reserves USDC/USDT: 229838703105.0/230824062879.0. Fee: 0.997
tinyman2 - 0 (TLP) <> 987374809 (ALGO). Reserves TLP/ALGO: 420565.0/7447378198.0. Fee: 0.997
pact - 312769 (USDT) <> 31566704 (USDC). Reserves USDT/USDC: 1287178926.0/1190094506.0. Fee: 0.99003
pact - 0 (ALGO) <> 31566704 (USDC). Reserves ALGO/USDC: 1477236252835.0/146952635698.0. Fee: 0.99003
tinyman2 - 0 (USDT) <> 312769 (ALGO). Reserves USDT/ALGO: 12815919257.0/128950260512.0. Fee: 0.997
pact - 31566704 (USDC) <> 672913181 (GOUSD). Reserves USDC/GOUSD: 184269144680.0/223408396860.0. Fee: 0.99003
tinyman2 - 0 (USDC) <> 31566704 (ALGO). Reserves USDC/ALGO: 1009721762507.0/10151016600260.0. Fee: 0.997
pact - 0 (ALGO) <> 31566704 (USDC). Reserves ALGO/USDC: 1477236252835.0/146952635698.0. Fee: 0.99003
pact - 31566704 (USDC) <> 672913181 (GOUSD). Reserves USDC/GOUSD: 184269144680.0/223408396860.0. Fee: 0.99003
tinyman2 - 0 (USDC) <> 31566704 (ALGO). Reserves USDC/ALGO: 1009721762507.0/10151016600260.0. Fee: 0.997
tinyman1 - 0 (USDT) <> 312769 (ALGO). Reserves USDT/ALGO: 4622916382.0/46663725530.0. Fee: 0.997
pact - 312769 (USDT) <> 672913181 (GOUSD). Reserves USDT/GOUSD: 31436458102.0/36109538099.0. Fee: 0.99003
tinyman1 - 312769 (USDC) <> 31566704 (USDT). Reserves USDC/USDT: 8655871851.0/8698188624.0. Fee: 0.997
pact - 312769 (USDT) <> 31566704 (USDC). Reserves USDT/USDC: 1287178926.0/1190094506.0. Fee: 0.99003
tinyman1 - 0 (USDT) <> 312769 (ALGO). Reserves USDT/ALGO: 4622916382.0/46663725530.0. Fee: 0.997
tinyman2 - 312769 (TLP) <> 987374809 (USDT). Reserves TLP/USDT: 303866.0/531810658.0. Fee: 0.997
tinyman2 - 0 (TLP) <> 987374809 (ALGO). Reserves TLP/ALGO: 420565.0/7447378198.0. Fee: 0.997
tinyman1 - 0 (USDC) <> 31566704 (ALGO). Reserves USDC/ALGO: 57788975062.0/581594046755.0. Fee: 0.997
```

Analysis of Algorithm on Historical States

Practical Implementation

Implementation:

```python
G = build_graph()
number_hops = [2,3,4]

# Build matrices
for each hop in number_hops:
    get_possible_cycles()
    get_exchange_pairs()
    generate_matrix_combinations()

# Detect arbitrage opportunities
for each block in dataset:
    parse_transactions()
    get_affected_exchanges()
    update_matrix_combinations()
    for each matrix in matrix_combinations:
        shift_matrix()
        calculate_trace()
        if trace > 1:
            calculate_optimal_inputs()
            execute_arbitrage()
```

Output:

$$\begin{array}{c} \\ \$ \\ \text{₿} \\ \text{Ξ} \end{array} \begin{array}{c} \begin{array}{ccc} \$ & \text{₿} & \text{Ξ} \end{array} \\ \begin{bmatrix} 1 & \frac{1}{20000} & \frac{1}{1000} \\ 20000 & 1 & 20 \\ 1000 & \frac{1}{20} & 1 \end{bmatrix} \end{array} \quad \begin{array}{c} \\ \$ \\ \text{₿} \\ \text{Ξ} \end{array} \begin{array}{c} \begin{array}{ccc} \text{₿} & \text{Ξ} & \$ \end{array} \\ \begin{bmatrix} \frac{1}{20000} & \frac{1}{1000} & 1 \\ 1 & 20 & 20000 \\ \frac{1}{20} & 1 & 1000 \end{bmatrix} \end{array} \quad \begin{array}{c} \\ \$ \\ \text{₿} \\ \text{Ξ} \end{array} \begin{array}{c} \begin{array}{ccc} \text{Ξ} & \text{₿} & \$ \end{array} \\ \begin{bmatrix} \frac{1}{1000} & 1 & \frac{1}{20000} \\ 20 & 20000 & 1 \\ 1 & 1000 & \frac{1}{20} \end{bmatrix} \end{array}$$

Matrix-Combinations for each

- *hops*
- *feasible path*
- *available pool*
- *shift*

Matrices can be generated and used for look-up during runtime

Parallelization of trace-computation possible

Analysis of Algorithm on Historical States

Practical Implementation

Implementation:

```
G = build_graph()
number_hops = [2,3,4]

# Build matrices
for each hop in number_hops:
    get_possible_cycles()
    get_exchange_pairs()
    generate_matrix_combinations()

# Detect arbitrage opportunities
for each block in dataset:
    parse_transactions()
    get_affected_exchanges()
    update_matrix_combinations()
    for each matrix in matrix_combinations:
        shift_matrix()
        calculate_trace()
        if trace > 1:
            calculate_optimal_inputs()
            execute_arbitrage()
```

Output:

```
TRADE POOL = HUMBLESWAP_ALGO_USDC
        TENDERING 15347.386040985702 TOKEN 0
        RECEIVEING 2663.3708938162385 TOKEN 1

TRADE POOL = ALGOFI_ALGO_USDC
        TENDERING 2663.3708938162213 TOKEN 1
        RECEIVEING 15898.964361424953 TOKEN 0
```

Outline

1. Background Information
2. Research Objectives
   2.1. Research Efficient Algorithms for Arbitrage Detection on Decentralized Exchange Graphs
   2.2. Establishing a Methodology for Profitable Execution of Arbitrage Opportunities
   2.3. Recreation of Historical States on the Algorand Blockchain
   2.4. Implementation & Analysis of Developed Algorithms on Historical States
3. Conclusion and Future Work

Conclusion and Future Work

Conclusion

- Identify profit-generating transactions
  1. Negative Cycle Detection
  2. Input Optimization
- For each step Algorithms with different trade-offs
- Methodology tailored to Algorand's properties and graph-size

Future Work

- Robust & runtime-efficient Implementation
  - Parallelization of matrix calculations
- *(Modification of Algorand node to reduce latency)*
- Run on 4 week dataset
- Quantitative analysis

Jonas Gebele
jonas.gebele@in.tum.de

Technische Universität München
Faculty of Informatics
Chair of Software Engineering for
Business Information Systems

Boltzmannstraße 3
85748 Garching bei München