Retrospectively Identifying Profit-Generating Transactions on the Algorand Blockchain

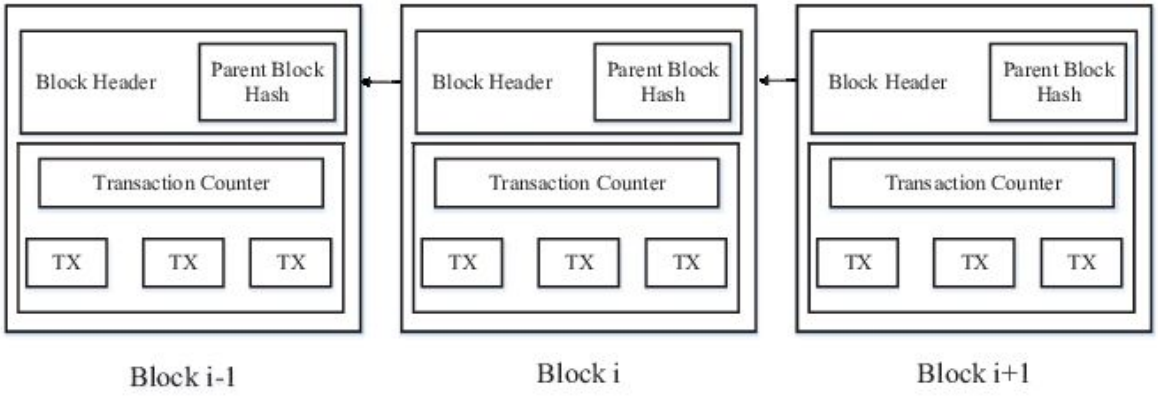
Jonas Gebele, June 05, 2023, Kick-off Presentation Application Project

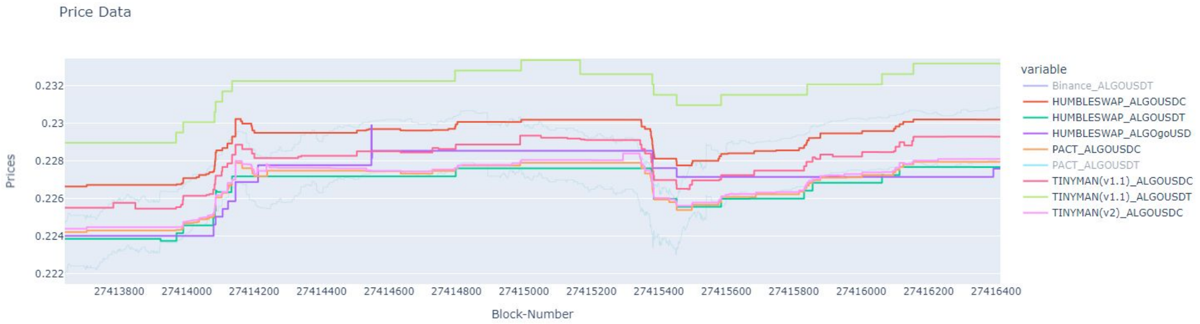
Chair of Software Engineering for Business Information Systems (sebis)
Department of Computer Science
School of Computation, Information and Technology (CIT)
Technical University of Munich (TUM)
wwwmatthes.in.tum.de

Outline

Motivation and Background Information

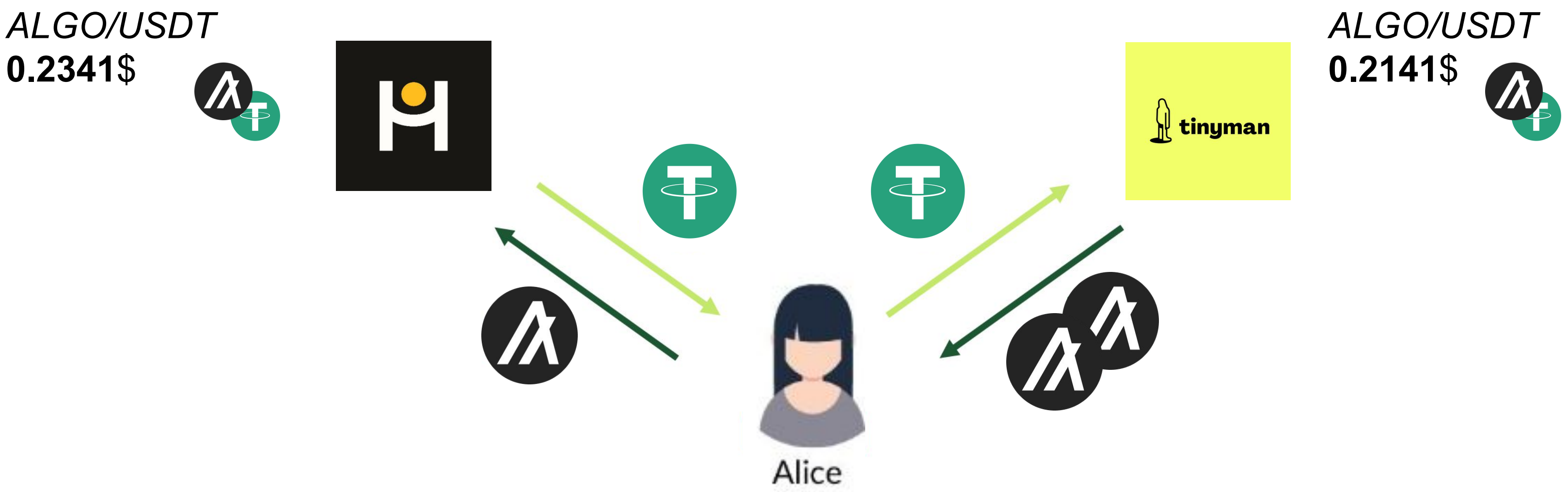
Algorand as a State Machine





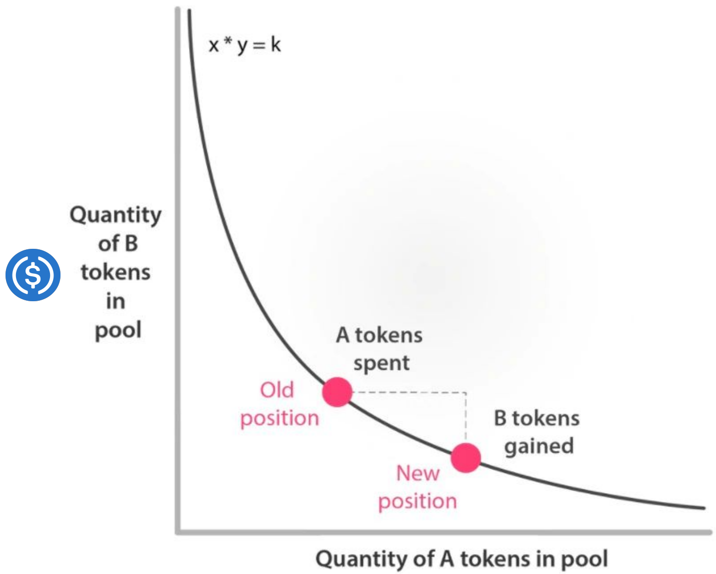
Motivation and Background Information

Decentralized Exchange Arbitrage



Motivation and Background Information

Constant Product Market Maker (CPMM) DEX's



CPMMs (Constant-Product Market Maker)

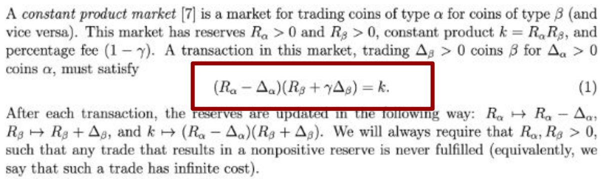
Example

ALGO ~ X = 1000

USDC ~ Y = 200

k = X * Y = 1000 * 200 = **200 000**
implied price: Y/X = 0.2 USDC per ALGO

Buy with 5 USDC ALGO (Swap USDC for ALGO)



A *constant product market* [7] is a market for trading coins of type α for coins of type β (and vice versa). This market has reserves $R_\alpha > 0$ and $R_\beta > 0$, constant product $k = R_\alpha R_\beta$, and percentage fee $(1-\gamma)$. A transaction in this market, trading $\Delta_\beta > 0$ coins β for $\Delta_\alpha > 0$ coins α, must satisfy

$$\boxed{(R_\alpha - \Delta_\alpha)(R_\beta + \gamma\Delta_\beta) = k.} \quad (1)$$

After each transaction, the reserves are updated in the following way: $R_\alpha \mapsto R_\alpha - \Delta_\alpha$, $R_\beta \mapsto R_\beta + \Delta_\beta$, and $k \mapsto (R_\alpha - \Delta_\alpha)(R_\beta + \Delta_\beta)$. We will always require that $R_\alpha, R_\beta > 0$, such that any trade that results in a nonpositive reserve is never fulfilled (equivalently, we say that such a trade has infinite cost).

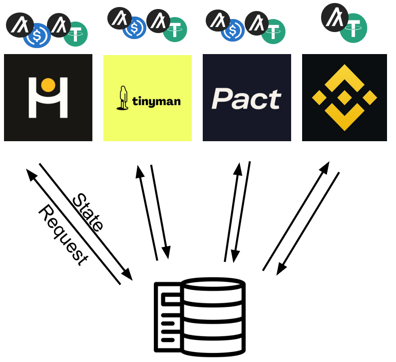
200'000 = (1000-**output**) * (200+5) = 975.61 * 205 = 200'000
output = 24.39 *(ALGO tokens)*

new implied price: Y/X = 205/975.61 = 0.21012 USDC per ALGO

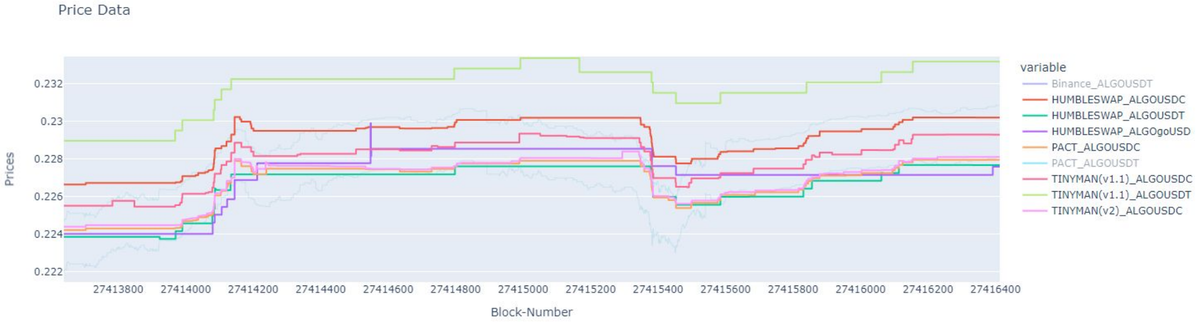
Outline

Problem Statement

Recap Guided Research: Price-Deviations between DEXs on Algorand



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



Problem Statement

Recap Guided Research: Upper- and Lower-Bound Estimation of Arbitrage-Related MEV

Upper Bound - Maximal Theoretical Extractable Value through DEX-CEX Arbitrage

Lower Bound - Easily Extractable Value through cross-DEX Arbitrage

Algorithm for Lower-Bound cross-DEX Arbitrage Estimate

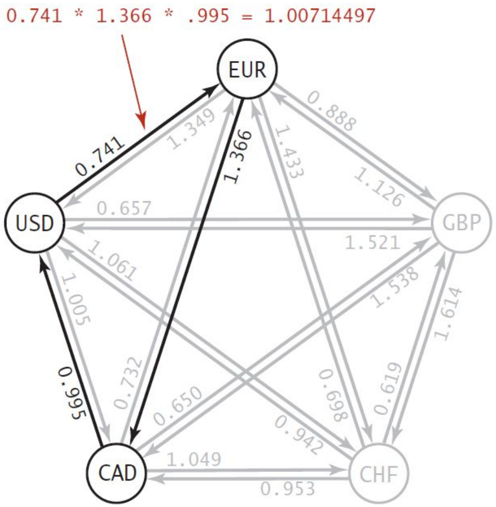
1. Search for DEX's with theoretical MEV with prices being below deviation boundary
2. Search for DEX's with theoretical MEV with prices being above deviation boundary
3. For each set of DEX's of 1. and 2. select the maxima
4. Perform transactions on both markets using the input of the smaller MEV

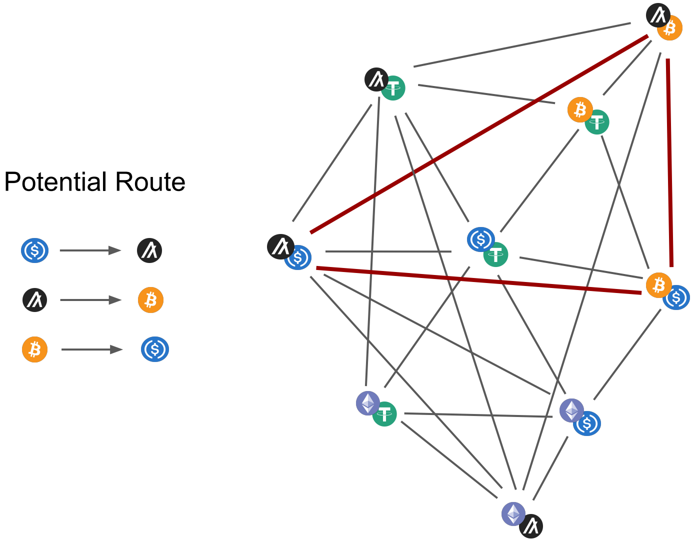
Outline

Find Profit Generating Transactions for a Historical State

Arbitrage Algorithms in Traditional Markets

Multigraph (Tokens as Nodes, Markets as Edges)





Bellman-Ford Algorithm for existence of negative-cycles (= profitable arbitrage routes)
Modified Bellman-Ford for actual paths

Find Profit Generating Transactions for a Historical State

<u>Open Research Question on efficient Arbitrage on CPMMs</u>

How to adapt Bellman-Ford efficiently to Arbitrage on CPMMs?

- (Modified) Bellman-Ford is deterministic, terminates after finding the first negative-cycle
- No obvious indication on profitability of a cycle

How to find the optimal input for the arbitrage for different pool-sizes?

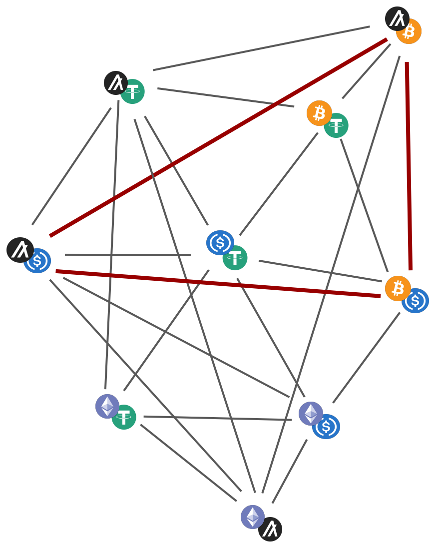
- Constrained optimization problem
- Solvers vs approximations vs binary search

Find Heuristics for Graph Pruning.

Enable branching in arbitrage routes.

Integrate transactional state-dependency.

- Multiple arbitrage opportunities within one state



Generate Past States of DEX Pools

Problem:

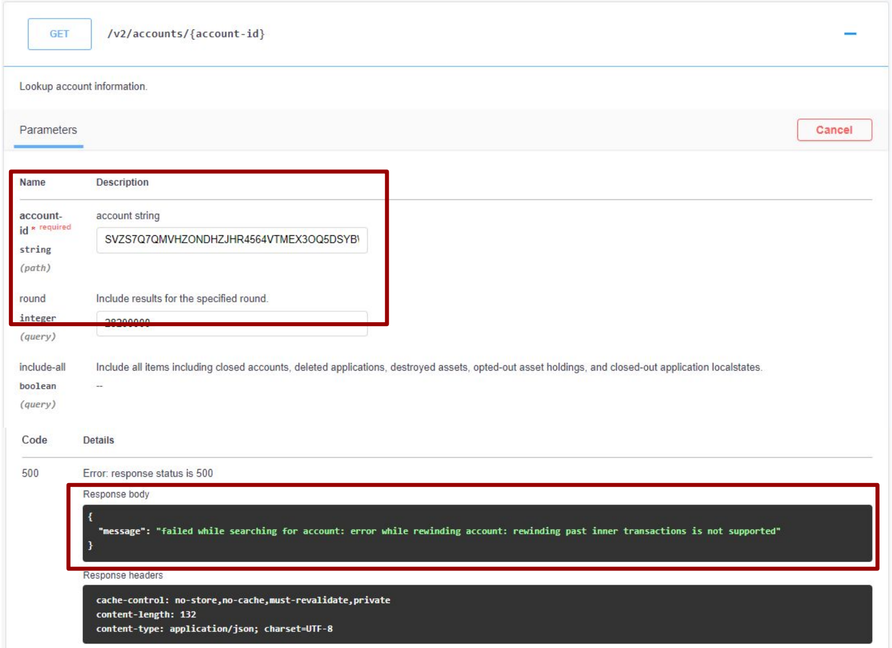
Algorand Indexer fails to query past states of smart-contracts

Goals:

Recreate DEX states starting from contract creation up to current round

Parse Transactions for DEX interactions and simulate pool-changes

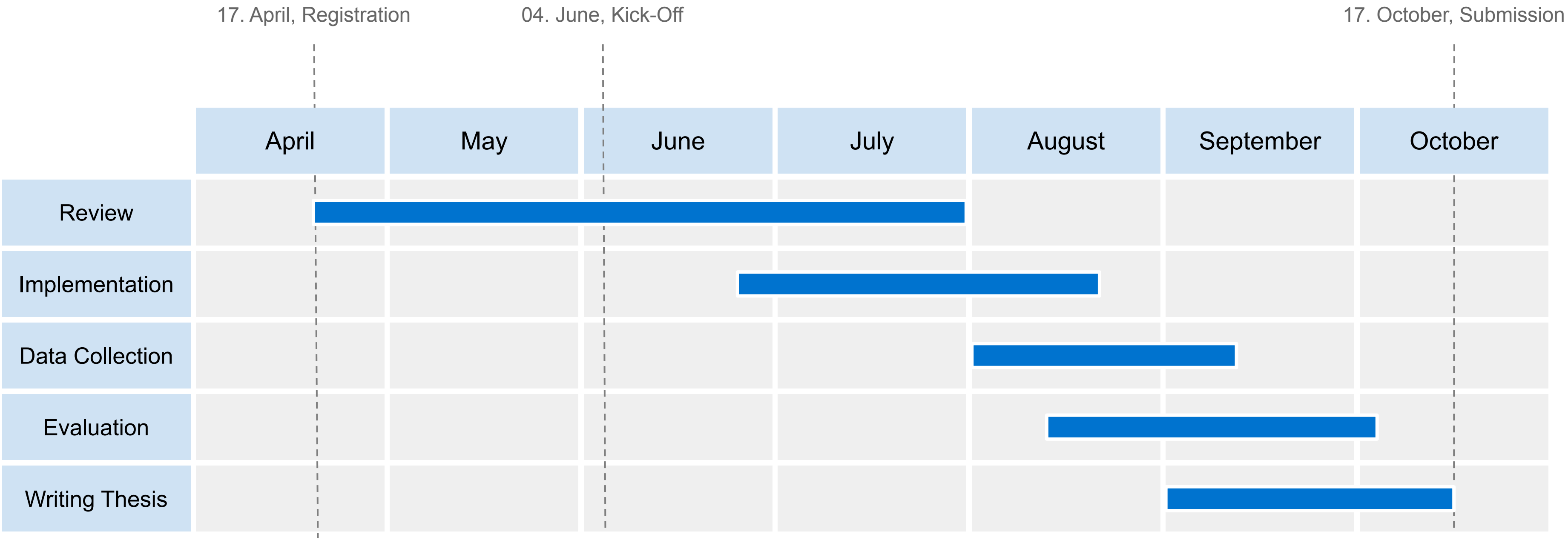
Store Network of DEX’s and interactions in Graph-Database to make efficiently queryable



Algorand Indexer fails to recreate past states

Timeline and Future Work

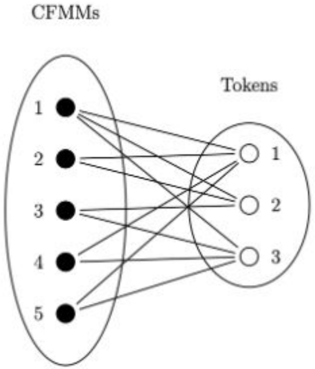
Timeline

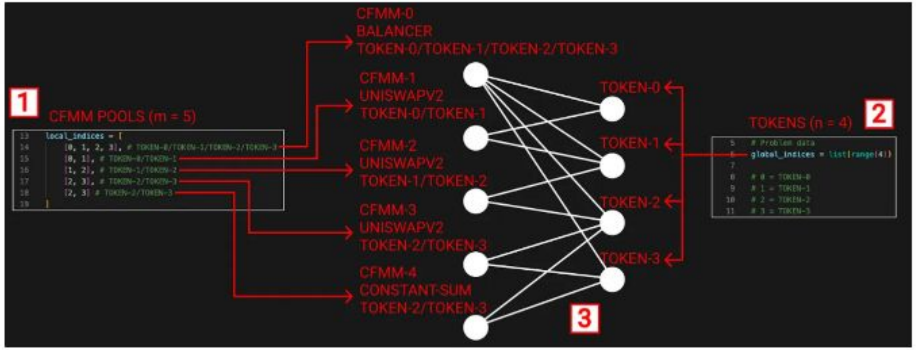


Jonas Gebele
jonas.gebele@in.tum.de

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

Boltzmannstraße 3
85748 Garching bei München





$$
\begin{aligned}
x_1 \cdot x_2 \cdots x_k &> 1 && (1)\\
\Leftrightarrow \frac{1}{x_1} \cdot \frac{1}{x_2} \cdots \frac{1}{x_k} &< 1 && (2)\\
\Leftrightarrow \ln\left(\frac{1}{x_1} \cdot \frac{1}{x_2} \cdots \frac{1}{x_k}\right) &< \ln(1) && (3)\\
\Leftrightarrow \ln\left(\frac{1}{x_1}\right) + \ln\left(\frac{1}{x_2}\right) + \cdots + \ln\left(\frac{1}{x_n}\right) &< 0 && (4)
\end{aligned}
$$

A *constant product market* [7] is a market for trading coins of type α for coins of type β (and vice versa). This market has reserves $R_\alpha > 0$ and $R_\beta > 0$, constant product $k = R_\alpha R_\beta$, and percentage fee $(1-\gamma)$. A transaction in this market, trading $\Delta_\beta > 0$ coins β for $\Delta_\alpha > 0$ coins α, must satisfy

$$(R_\alpha - \Delta_\alpha)(R_\beta + \gamma\Delta_\beta) = k. \quad (1)$$

After each transaction, the reserves are updated in the following way: $R_\alpha \mapsto R_\alpha - \Delta_\alpha$, $R_\beta \mapsto R_\beta + \Delta_\beta$, and $k \mapsto (R_\alpha - \Delta_\alpha)(R_\beta + \Delta_\beta)$. We will always require that $R_\alpha, R_\beta > 0$, such that any trade that results in a nonpositive reserve is never fulfilled (equivalently, we say that such a trade has infinite cost).

```
// given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint amountOut) {
    require(amountIn > 0, 'UniswapV2Library: INSUFFICIENT_INPUT_AMOUNT');
    require(reserveIn > 0 && reserveOut > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');
    uint amountInWithFee = amountIn.mul(997);
    uint numerator = amountInWithFee.mul(reserveOut);
    uint denominator = reserveIn.mul(1000).add(amountInWithFee);
    amountOut = numerator / denominator;
}
```

$$
x\left(\frac{pool_{balancesin1}\, pool_{balancesin2}\, pool_{balancesin3}\, pool_{balancesout1}\, pool_{balancesout2}\, pool_{balancesout3}}{x\left(pool_{balancesin1}+x\right)^{2}\left(-\frac{pool_{balancesin1}\, pool_{balancesout1}}{pool_{balancesin1}+x}+pool_{balancesin2}+pool_{balancesout1}\right)^{2}\left(-\frac{pool_{balancesin2}\, pool_{balancesout2}}{-\frac{pool_{balancesin1}\, pool_{balancesout1}}{pool_{balancesin1}+x}+pool_{balancesin2}+pool_{balancesout1}}+pool_{balancesin}+pool_{balancesout2}\right)^{2}} - \frac{-\frac{pool_{balancesin3}\, pool_{balancesout3}}{-\frac{pool_{balancesin2}\, pool_{balancesout2}}{-\frac{pool_{balancesin1}\, pool_{balancesout1}}{pool_{balancesin1}+x}+pool_{balancesin2}+pool_{balancesout1}}+pool_{balancesin3}+pool_{balancesout2}}+pool_{balancesout3}}{x^{2}}\right) - 1 + \frac{-\frac{pool_{balancesin3}\, pool_{balancesout3}}{-\frac{pool_{balancesin2}\, pool_{balancesout2}}{-\frac{pool_{balancesin1}\, pool_{balancesout1}}{pool_{balancesin1}+x}+pool_{balancesin2}+pool_{balancesout1}}+pool_{balancesin3}+pool_{balancesout2}}+pool_{balancesout3}}{x}
$$

$$
\Delta_1^* = \frac{\sqrt{\gamma^n \prod_{i=1}^{n} R_{i,i+1} * R_{i+1,i}} - \prod_{i=1}^{n} R_{i,i+1}}{\sum_{i=1}^{n} \gamma^i \left(\prod_{j=1}^{i-1} R_{j+1,j}\right)\left(\prod_{j=i+1}^{n} R_{j,j+1}\right)}
$$