The Self-Arbing DEX

Anonymous Author

August 30, 2022

Abstract

We describe how a DEX can include backrunning logic that can be used to keep substantial arbitrage profits for itself.

1 Introduction

With the advent of Layer 2 networks, subnets, supernets etc, we expect to see DEXes that serve local economies without the need for bridging. In these DEXes we expect to see only a small number of different tokens with a handful of pools containing significant liquidity. In such cases a DEX contract can check for arbitrage in a small set of predetermined potential opportunities. This paper describes how this can be done efficiently.

2 Setup and Notation

We assume the DEX consists solely of UniswapV2-style pools. Each pool i is described by the tokens s_i, t_i and the respective reserves respective reserves p_i, q_i. We let R be the mapping from token pairs to reserves so that for every pool i we have $R(s_i, t_i) = p_i, q_i$ and $R(t_i, s_i) = q_i, p_i$.

The DEX administrator has identified a set of opportunities that are simply lists/paths of tokens $[a, b, c, \ldots, a]$ such that the first and last token are the same

and for every adjacent pair in the list $(a, b), (b, c), \ldots$ the exists a pool in the DEX. When a user trades with the DEX the DEX executes the trade the same way that UniswapV2 does so. As part of the same transaction however, the DEX subsequently checks if any arbitrage opportunities are available, and if so executes the arb and gives the profits to stakers.

3 Main Theorem

For the case of all UniswapV2 pools where the trading fee is $(1 - \gamma)$ (with $\gamma = 0.997$ in most deployments) we derive a closed form expression that tells us for any input amount x_0 the amount of tokens x_n we would get if we traded along the path $[a_0, a_1, a_2, \ldots, a_n]$ i.e. trade x_0 tokens of a_0 for x_1 tokens of a_1, then x_1 tokens of a_1 for x_2 tokens of a_2 and so on until we get x_n tokens of a_n. The following theorem shows that x_t has a form that can be updated easily. In the next section we use this observation to compute the optimal amount x_0 to maximize profits on paths of the form $[a_0, a_1, a_2, \ldots a_0]$

Theorem 1. *Given a DEX of UniswapV2 pools with trading fee $1 - \gamma$ and a path of tokens $[a_0, a_1, \ldots, a_n]$, the output amount x_n satisfies*

$$x_n = \frac{K_n x_0}{L_n x_0 + M_n} \tag{1}$$

where K_n, L_n, M_n are defined via the recurrences

$$K_{t+1} = \gamma q_t K_t \tag{2}$$
$$L_{t+1} = p_t L_t + \gamma K_t \tag{3}$$
$$M_{t+1} = p_t M_t \tag{4}$$

where $p_t, q_t = R(a_{t-1}, a_t)$ are the reserves of tokens a_{t-1} and a_t in their pool. and with starting values $K_0, L_0, M_0 = 1, 0, 1$.

Proof. The proof is by induction on the path length. For $n = 1$ we have $p_1, q_1 = R(a_0, a_1)$, $K_1 = \gamma q_1$, $L_1 = \gamma$, $M_1 = p_1$ and $x_1 = \frac{\gamma q_1 x_0}{\gamma x_0 + p_1}$ We know check that the Uniswap invariant is maintained with the new reserves

$$(p_1 + \gamma x_0)(q_1 - x_1) = (p_1 + \gamma x_0)\left(q_1 - \frac{\gamma q_1 x_0}{\gamma x_0 + p_1}\right) = (p_1 + \gamma x_0)q_1 - \gamma q_1 x_0 = p_1 q_1$$

This proves the base case. We now assume that the claim holds for paths of length n and show that it holds for paths of length $n + 1$. By the induction hypothesis when we trade the path $[a_0, a_1, \ldots, a_{n-1}, a_n]$ we know that we have

$$x_n = \frac{K_n x_0}{L_n x_0 + M_n}$$

amount of a_{n-1}. Now let $p_n, q_n = R(a_{n-1}, a_n)$ and consider the Uniswap invariant again

$$p_n q_n = (p_n + \gamma x_n)(q_n - x_{n+1})$$

Plugging in the value of x_n and simplifying we have

$$x_{n+1} = \frac{\frac{\gamma q_n K_n x_0}{L_n x_0 + M_n}}{p_n + \gamma \frac{K_n x_0}{L_n x_0 + M_n}}$$

$$x_{n+1} = \frac{\gamma q_n K_n x_0}{p_n(L_n x_0 + M_n) + \gamma K_n x_0}$$

$$x_{n+1} = \frac{\gamma q_n K_n x_0}{(p_n L_n + \gamma K_n) x_0 + p_n M_n}$$

$$x_{n+1} = \frac{K_{n+1} x_0}{L_{n+1} x_0 + M_{n+1}}$$

where in the last step we used the recurrences for $K_{n+1}, L_{n+1}, M_{n+1}$. □

4 Optimal Arbitrage Amount

Given a cycle $[a_0, a_1, \ldots, a_{n-1}, a_0]$ we can solve for the optimal arbitrage amount by setting to zero the derivative of the profit

$$y(x_0) = x_n - x_0 = \frac{K_n x_0}{L_n x_0 + M_n} - x_0$$

with respect to the input amount x_0. Doing so leads to the solution

$$x_0^* = \frac{\sqrt{K_n M_n} - M_n}{L_n}$$

For an arb to be possible we need $x_0^* > 0$ or, equivalently, $K_n > M_n$.

5 Algorithm

Below is a basic algorithm. Gas optimizations are up to the implementation.

Require: List of opportunity paths a
  for $i \leftarrow 0, \ldots, a.\text{length} - 1$ **do**
    $K, L, M \leftarrow 1, 0, 1$
    for $j \leftarrow 1, \ldots, a[i].\text{length} - 1$ **do**
      $p, q \leftarrow R(a[i][j-1], a[i][j])$
      $M \leftarrow p \cdot M$
      $L \leftarrow p \cdot L + \gamma \cdot K$
      $K \leftarrow \gamma \cdot q \cdot K$
    end for
    if $K > M$ **then**
      $x \leftarrow \frac{\sqrt{KM} - M}{L}$
      Swap x tokens along path $a[i]$
      break
    end if
  end for

6 Additional Notes

6.1 Scaling

The description assumes real numbers. An actual implementation needs proper scaling so that K, L, M do not overflow. One simple way to achieve this is to divide the reserves by a large scaling factor Z, and finally swap $x \cdot Z$ tokens.

6.2 Opportunities

Suppose we have identified the cycle $[a_0, a_1, a_2, a_0]$ as an interesting opportunity. Should we also consider the cycles $[a_1, a_2, a_0, a_1]$ and $[a_2, a_1, a_0, a_2]$ which consist of the same pools? The answer is no: All these cycles allow one to extract the same amount of value. The difference is only on the token being paid out. While cyclical shifts need not be considered, the reverse order cycle $[a_0, a_2, a_1, a_0]$ should always be considered.