Consolidated vs Fragmented

How Alpaca Beats the Optimal Liquidity Aggregator

El Ranchero

November 17, 2020

1 Introduction

AlpacaSwap is an evolution of automated market maker protocols that provides optimal liquidity by consolidating all assets into a single pool. This ensures that liquidity of popular assets is not fragmented across a number of pairs, creating the first viable, plug-and-play liquidity pool for DApps.

Aggregating all liquidity in a single pool confers significant benefits to traders compared to fragmented liquidity pools in the form of reduced slippage, reduced gas, and reduced fees. In this paper, we illustrate how Alpaca reduces slippage, even in comparison to the *optimal* liquidity "aggregator" in a world of fragmented liquidity.

2 Problem Set Up

The math gets *extremely* cumbersome very quickly in any non-trivial scenario. Therefore, we will restrict ourselves to a relatively simple example, although even in this case the math rapidly balloons. Nevertheless, the optimal liquidity property of Alpaca holds in the general case.

Let us compare two scenarios. For simplicity, our universe will have three assets, A, B, and C. [1] Without loss of generality, when considering a single pair trade, Alpaca reduces to the Uniswap invariant formula, so we will stick to comparing swapping an amount DA of asset A for an amount DB for

[1] Incidentally, if we restrict ourselves to simply two assets, Alpaca and Uniswap are isomorphic, even if the latter has multiple pairs of the same assets. But that obviously is not anywhere near reality.

asset B. As this paper will demonstrate, dB differs between these scenarios, to the detriment of the Uniswap trader.

Scenario 1: Fragmented Uniswap Liquidity. In this scenario, there are three pools, corresponding to each of the three possible pairs. We denote X_i to be pool i's reserves of asset X and dX_i to be the amount of asset X traded in or out of pool i. Without loss of generality, let us assume pool 1 is comprised of A and B, pool 2 of A and C, and pool 3 of B and C.

Scenario 2: Consolidated Alpaca Liquidity. In this scenario, all liquidity is consolidated into a single pool (the Mother Of All Pools). To make the comparison valid, the reserves of asset X is $\sum_i X_i$, where X_i is as in the previous scenario.

3 Fragmented Uniswap Trade

The hero of our tale wishes to trade dA and maximize the amount he receives, dB. It is hopefully obvious to the reader that trading entirely in pool 1 (comprised of A and B) would be highly suboptimal compared to the second scenario where all of A and B are in a single pool. Instead, let us assume our precocious protagonist observes that he can execute part of his trade by trading A for C in pool 2 and C for B in pool 3 with the aim of minimizing slippage and overall trading costs.

Therefore, there are two main legs of the trade. The first leg is executed directly in pool 1. He trades dA_1 in and gets dB_1 out. Using the standard Uniswap constant product invariant notation, we can easily see that

$$\begin{aligned} k_1 &= A_1 \cdot B_1 \\ k_1 &= (A_1 + dA_1)(B_1 - dB_1) \\ dB_1 &= B_1 \left(1 - \frac{A_1}{A_1 + dA_1}\right) \end{aligned}$$

Leg two trades dA_2 in to pool 2 and gets dC_2 out, then trades in dC_3 in to pool 3 and gets dB_3 out. Noting that $dC_2 = dC_3$, we iteratively apply the

above formula to find

$$dB_3 = B_3 \left(1 - \frac{C_3}{C_3 + C_2 \left(1 - \frac{A_2}{A_2 + dA_2} \right)} \right)$$

The total amount of asset A in, then, is

$$DA = dA_1 + dA_2$$

and the total amount of asset B out is

$$DB = dB_1 + dB_3$$

Finding the optimal combination of leg 1 and leg 2 is an optimization problem. We restate it as follows:

$$\begin{aligned} \underset{x}{\text{maximize}} \quad & f(dA_1, dA_2) \\ \text{subject to} \quad & g(dA_1, dA_2) = 0 \\ & dA_1, dA_2 \geq 0 \end{aligned}$$

where

$$\begin{aligned} f(dA_1, dA_2) &= DB = dB_1 + dB_3 \\ g(dA_1, dA_2) &= dA_1 + dA_2 - DA \end{aligned}$$

Luckily, this follows the standard form of Lagrangian multipliers [2]. Unfortunately for our intrepid trader — and any "liquidity aggregator" application — the solution is quite complex, and effectively intractable for any on-chain contract.

Our Lagrangian function is then

$$\mathcal{L}(dA_1, dA_2, \lambda) = f(dA_1, dA_2) + \lambda \cdot g(dA_1, dA_2)$$

[2] `https://en.wikipedia.org/wiki/Lagrange_multiplier`

which we solve by setting the Lagrangian gradient to 0:

$$\begin{aligned}\nabla_{dA_1,dA_2,\lambda}\mathcal{L}(dA_1, dA_2, \lambda) &= \left(\frac{\partial\mathcal{L}}{\partial dA_1}, \frac{\partial\mathcal{L}}{\partial dA_2}, \frac{\partial\mathcal{L}}{\partial\lambda}\right) \\ &= \begin{pmatrix} \frac{k_1}{(A_1+dA_1)^2} + \lambda \\ \frac{k_2 \cdot k_3}{(A_2 \cdot C_3 + (C_2+C_3)dA_2)^2} + \lambda \\ dA_1 + dA_2 - DA \end{pmatrix} \\ &= 0,\ \lambda \neq 0\end{aligned}$$

Starting the first two equations in this system, we solve for dA_1:

$$dA_1 = \sqrt{\frac{k_1}{k_2 k_3}}\,(A_2 \cdot C_3 + (C_2 + C_3)dA_2) - A_1$$

Combining this result with equation 3 of the system, we solve for dA_2:

$$dA_2 = \frac{A_1 + DA - \sqrt{\frac{k_1}{k_2 k_3}} A_2 C_3}{1 + \sqrt{\frac{k_1}{k_2 k_3}}\,(C_2 + C_3)}$$

As the reader can easily see, the algebra even for this simple scenario is getting monstrous. Instead of losing ourselves in tedious algebra, for this paper let us simplify things somewhat further by assuming that $X_i = n, \forall i, X \in \{A, B, C\}$. I suggest the brave use Mathematica or another symbolic algebra system if they wish to verify the more general case.

Then,

$$\begin{aligned}
dA_2 &= \frac{n + DA - \frac{1}{n} \cdot n \cdot n}{1 + \frac{1}{n}(n+n)} \\
&= \frac{1}{3}DA \\
dA_1 &= DA - dA_2 \\
&= \frac{2}{3}DA \\
dB_1 &= B_1 \left(1 - \frac{A_1}{A_1 + dA_1}\right) \\
&= n \left(1 - \frac{1}{1 + \frac{2}{3}\frac{DA}{n}}\right) \\
dB_3 &= B_3 \left(1 - \frac{C_3}{C_3 + C_2\left(1 - \frac{A_2}{A_2 + dA_2}\right)}\right) \\
&= n \left(1 - \frac{1}{2 - \left(\frac{1}{1 + \frac{1}{3}\frac{DA}{n}}\right)}\right)
\end{aligned}$$

The values for dA_1 and dA_2 make intuitive sense — dA_2 suffers from "twice" the slippage, and thus is half the size of dA_1 (note: this is an over-simplification).

Even after many simplifications, the algebra is offensive to the eyes. But alas, that is the result of our hero's foolhardy quest to beat Alpaca in a world of fragmented liquidity. Still, the hero marches on.

Let's see how he fares when $DA = \frac{3}{100}n$, a reasonably large trade, but certainly not one out of the question in the fast-moving world of DeFi. Then, the total amount he recieves, DB, is

$$DB = dB_1 + dB_3 = \boxed{\frac{1}{34}n}$$

Now all that is left is to see how that compares to the results he would see using Alpaca instead.

4 Consolidated Alpaca Trade

In the world of Alpaca, our hero doesn't need to worry about optimization across multiple pools. Instead, trading any pair is as simple as trading against the Alpaca pool. To make the comparison apples-to-apples, instead of having pairs of n for each possible pair, as in our simplified scenario 1, we instead have $2n$ of each asset in the Alpaca pool for the same total asset base.

As mentioned previously, Alpaca's constant geometric mean curve can (with some small algebraic substitutions) be reduced to the Uniswap constant product curve when looking at any given pair. $DA = \frac{3}{100}n$ is swapped in and DB (different in value from DB in the previous scenario) is received out according to our familiar formula:

$$\begin{aligned} DB &= B\left(1 - \frac{A}{A + DA}\right) \\ &= 2n\left(1 - \frac{2n}{2n + \frac{3}{100}n}\right) \\ &= 2n\left(1 - \frac{2}{2 + \frac{3}{100}}\right) \\ &= \boxed{\frac{6}{203}n} \end{aligned}$$

Much simpler! So, how did our hero do?

5 Comparing the results

Let us compare the amount our fearless trader received in the two scenarios:

$$\frac{\text{scenario 1}}{\text{scenario 2}} = \frac{\frac{1}{34}n}{\frac{6}{203}n} = \frac{203}{204} \approx 99.5\%$$

So in scenario 1, he receives **0.5% less** of asset B for his efforts — an increase in slippage of 50 basis points! That is an extraordinarily high amount for a trade.

But that's not all! I've been quite generous to scenario 1 thus far and excluded things like gas and fees. Let's take a realistic tally of the additional costs that fragmented liquidity impose on our poor trader:

1. *Higher slippage*
   $\sim$50 bps for the optimal trade. But let's be honest — the most likely scenario is a suboptimal routing of orders between different venues according to an imprecise heuristic algorithm. Remember, 50 bps in this example is the *global maximum*, so it only can get worse.

2. *Additional trading fees*
   The second leg of the Uniswap scenario requires two trades. That imposes an additional cost on the total trade, equal to the fraction of the trade done in the two hops times the fee. For a fee of $\phi = 30$ basis points, that is a 10 basis point incremental fee.

3. *Increased gas*
   The Uniswap scenario required 3 trades rather than one. Needless to say, this can get quite expensive, and can only be optimized somewhat. The fractional cost varies by trade size, but for a typical trade this will be at least 20 basis points additional cost.

4. *Aggregator fees*
   If our trader is a contract, or just doesn't want to learn a not insignificant amount of optimization theory, he will be reduced to using a liquidity aggregator app, which imposes additional fees. Let's call it 5 basis points.

6 Putting it all together

The bottom line: Our trader is easily paying more than 1% in additional fees solely because of how inefficiently capital is deployed when liquidity is fragmented across pairs. That's a disaster.

The general result: For any given asset base, Alpaca *strictly dominates* the fragmented alternative. You can take that to the bank.