Poolshark

alphak3y

Abstract
Poolshark is a noncustodial directional automated market maker implemented for the Ethereum Virtual Machine. In comparison to its predecessors, it allows for liquidity providers to adopt a buy-and-hold strategy using a closed-form solution for liquidity position tracking.

Keywords
Directional Automated Market Maker — Directional Liquidity — Cover Liquidity — Price Liquidity

*__Corresponding author__: alphak3y@protonmail.com

Contents

Introduction

Automated market makers (AMMs) are defined as software that prices liquidity using a pre-defined algorithm. In the context of decentralized finance, $x * y = k$, a formula associated with constant-function market makers (CFMMs), is commonplace due to its well-defined computational and thus blockspace usage. For a pool in which two tokens are paired with one another, the formula maintains that the total value of the first token must always equal the total value of the second token in the pool.

Directional automated market makers (DAMMs) are an extension of automated market makers wherein liquidity in the pool is non–recyclable and there exist discrete and separate liquidity curves for each trading direction.

Motivation

Bidirectional liquidity AMMs, defined as a smart contract containing allowing for liquidity to be traded either from

$$token_0 => token_1 \ \vee \ token_1 => token_0$$

is what is commonly given consideration with respect to the term AMM as of this writing. The way that such a smart contract trades assets using $x * y = k$ math is based both on market demand as well as supply from liquidity providers.

As long as the liquidity remains constant within the pool at a given price, traders execute liquidity swaps to balance the liquidity pool reserves. It is worth noting the value of this smart contract design is to enable liquidity providers to continuously provide liquidity to the market and collect trading fees for doing so.

When this kind of liquidity provision can become disadvantageous to the liquidity provider from a monetary value perspective is when there is large price divergence between the two assets in the liquidity pool. This phenomenon is commonly referred to as *impermanent loss*, however for sake of simplicity, it is easier to define this impermanent loss as

$$ASP_{market} - ASP_{LPposition}$$

where the ASP (i.e. average selling price) difference between the liquid market value and the ASP of the LP (i.e. liquidity provider) position at some constant price.

If we take 1 ETH and spread it across the price range of 3000 DAI per ETH to 5000 DAI per ETH using range-bound liquidity and $x * y = k$, that 1 ETH will be sold at approximately 3868.23 DAI per ETH.

When the market price of ETH becomes 5000 DAI per ETH, a user with range-bound liquidity between 3000 DAI per ETH and 5000 DAI per ETH will experience a value loss of 1131.77 DAI. Here we assume the user to liquidate these holdings to 100% DAI at the price of 5000 DAI per ETH with zero price slippage.

In order to cover these losses, we highlight here a few key options for the liquidity provider:

1) Choose to not participate in bidirectional liquidity pools and favor a buy-and-hold strategy

2) Buy and hold an option, perpetual or other illiquid asset with the potential for a liquidity crunch when covering losses

3) Buy and hold the underlying asset in the bidirectional liquidity pool which the LP expects to appreciate in price over time

1), 2), and 3) in the current context of decentralized exchanges involve diminishing liquidity available for assets such as ETH, DAI, and other decentralized cryptocurrencies in favor of improving individual portfolio outcomes.

Having a way to express 3) by providing range-bound liquidity to a smart contract means A) liquidity can be provided by both traders and fee collectors B) bidirectional liquidity providers require less active management to become delta neutral C) more price diversity in the market with discrete liquidity curves for each trading direction.

This is the clear motivation for the introduction and existence of directional automated market makers (DAMMs): to support the needs of on-chain liquidity seeking a gas-efficient buy-and-hold strategy for across various market conditions.

1. High-level overview

Directional liquidity AMMs, defined as a smart contract allowing for liquidity to be exclusively traded between

$$token_0 => token_1 \oplus token_1 => token_0$$

Directional automated market makers have two key traits:

1) Irreversible liquidity swaps

2) Discrete liquidity curves

The Poolshark Protocol delivers these qualities with two definitive variants, *Cover Pools* and *Price Pools*.

Cover Pools enable the liquidity provider to unlock liquidity as the price moves against the asset which they provided to the liquidity pool.

Price Pools allow for pro-rata price priority, where users executing liquidity swaps receive the lowest price for their chosen trading direction.

Cover Example :

Alice provides DAI in a range of 3000 to 5000 DAI per ETH to offset potential impermanent loss.

Price Example :

Bob provides ETH at a 0.01% discount to the rest of the market in order to exit his ETH position.

1.1 Cover Liquidity Pools

In order to cover this loss of 1131.77 DAI as mentioned in the *Motivation* section, the solution proposed in this whitepaper is to purchase the same amount of ETH back from the market that was sold over this range at the same average price of 3868.23 DAI per ETH.

For each bit of ETH that the liquidity provider receives from the market, they ought to hold this ETH in order to profit from a continual directional increase in price (i.e. impermanent gain). From this point on we shall refer to this description as a *Cover Pool*.

Cover Pools gradually unlock liquidity as the price continues to move against the asset the LP provided. In the case here, the user provides 3868.23 DAI to create a *Cover LP Position* with a lower bound of 3000 DAI per ETH and an upper bound of 5000 DAI per ETH. As the price continues to increase for ETH to DAI and decrease for DAI to ETH, the smart contract will unlock more of the user's LP position to be auctioned off to the market.

The smart contract requires a reference price to determine which liquidity should be unlocked at any given time, and therefore is informed by a time-weighted average price (TWAP) oracle, ideally one whose data originates on-chain. A longer TWAP sample (e.g. last 1 hour) will make the *Cover Pool* less reactive to changes in price, and conversely a shorter TWAP sample (e.g. last 10 minutes) will make the *Cover Pool* more reactive to price changes but better equipped to handle short-term volatility.

In addition to creating a positive spread on the latest TWAP price, the *Cover Pool* will also need to dynamically adjust to the current market price, which is likely plus or minus the current TWAP sample. To account for this adjustment, *Cover Pools* will implement the concept of *Gradual Dutch Auctions* in order to adjust the pool price each consecutive block.

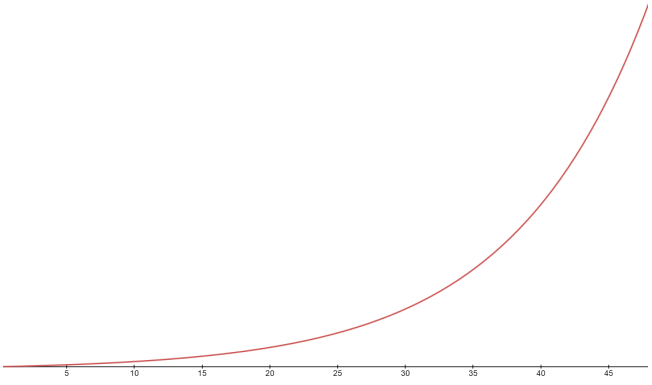

Figure 1. Cover auction price curve

Each time the TWAP moves, a liquidity auction will initiate and create a positive spread for the those swapping against the *Cover Pool* in comparision to the TWAP equal to the *tickSpacing*. For example, if the ETH-DAI *Cover Pool* in

our example has sources a TWAP 0.1% (10 basis points) above 3000 DAI per ETH the ETH to DAI price will start at 3000 + 0.2% and the DAI to ETH price will start at 3000 DAI per ETH. This is done to get slightly ahead of the current market price so the pool can ensure those with a liquidity position in the pool get filled. To be clear, this positive TWAP spread does not alter the price at which the *Cover LP Position* gets filled at but rather slightly speeds up the liquidity unlocking to provide better outcomes over a large succession of ticks and liquidity unlocks.

When liquidity is unlocked by the TWAP, it is unlikely this TWAP will fit the natural market price without any adjustment. To account for this delta, *Cover Pools* will implement the concept of *Gradual Dutch Auctions* in order to increase or decrease the pool price as needed each consecutive block a tick or set of ticks has its liquidity auctioned off.

1.2 Price Liquidity Pools

In current bidirectional range AMMs as of writing, when the LP position has been reached the desired price, it needs to be withdrawn. This is a drastic departure from how traditional limit orders work on centralized exchanges. *Price Pools* provide a closed-form solution for mirroing one-way fills: a claim tick.

The concept of a claim tick is such that there is some price tick T up to which our position has been filled since the time of creation. As the pool crosses each tick and fills the user's Position P, it will mark the global total fee growth at which the tick was crossed.

Since there exist discrete curves for each trading direction, the smart contract is able to retrieve the lowest price for each trading direction. As a result, users executing liquidity swaps will receive the best price available in the smart contract. Liquidity providers receive the benefit of irreversible "take-profit" range orders that can be used for a variety of scenarios.

2. Implementation Details

Directional liquidity relies on a time-keeping mechanism to tell liquidity providers the *Tick* closest to fill completion, which we will refer to as the *Claim Tick*, which will contain any other data needed to fulfill a user receiving the collected assets thus far and/or removing their liquidity from the smart contract. Given the claim tick is known and verified as correct (i.e. it has seen some update since the user created their Position), $x * y = k$ math can be used to determine the amount a Position should receive of both $token0$ and $token1$.

This simple mechanism of liquidity position tracking persists throughout, however there are a variety of special cases to handle depending on where the pool price is at time of mint, collect, or burn. It is important to note that *Cover Pools* and *Price Pools* have completely separate mechanics outside of the similarities outlined in this section.

2.1 Cover Pool Contracts

When first launching a *Cover Pool*, we must first ensure that the *Range Pool* we are referencing has a sufficient TWAP sample length for any user minting a *Cover LP Position*. If our pool uses a TWAP sample length of 1 hour, we must ensure there is 1 hour worth of data currently available.

Once this check passes, we can initialize the pool and allow users to start adding liquidity. By initializing the pool, $Tick_{min}$ and $Tick_{max}$, the minimum and maximum representable price ticks, are created in addition to a $Tick_{latest}$, a *Tick* representing the current TWAP sample.

In order to prevent the state of any liquidity position from being fragmented, we must not allow users to add liquidity on both sides of the current TWAP. For LPs seeking to provide $token1$ in exchange for $token0$, they must provide liquidity at a price greater than the current TWAP, assuming the pool price is represented as

$$price_{pool} = \frac{token_1 \ reserves}{token_0 \ reserves}$$

Likewise, LPs seeking to provide $token1$ in exchange for $token0$ must provide liquidity at a price lesser than the current TWAP. Users seeking to exchange at a more favorable price than the current TWAP should instead use a *Price LP Position*, which is meant to play the role of a take-profit range order.

Having discrete liquidity curves means having separate liquidity data for each swap direction. We will refer to this distinction as $pool0$ and $pool1$, for which $pool0$ will contain $token0$ and $pool1$ will contain $token1$. *Position* and *Tick* data will also have distinct contract storage in order to track the activity of each pool and apply exactly-once liquidity position fills.

2.1.1 Position Updates

Upon minting an LP position in a *Cover Pool*, a $+\Delta_{liquidity}$, or *positive liquidityDelta*, is added to $Tick_{start}$, while conversely a $-\Delta_{liquidity}$ is added to the $Tick_{end}$ *Tick*. If the LP mints a position for $token1$ to $token0$, the $Tick_{start}$ will be at the lower bound and the $Tick_{end}$ will be at the upper bound. Vice versa will be true if the LP is depositing $token0$ for $token1$.

Utilizing standard $x * y = k$ math and virtual reserves, we can calculate the amount of liquidity L that a *Position* holds using the following formula[1]:

$$L = \sqrt{(x + \frac{L}{price_{\text{upper}}})(y + L\sqrt{price_{\text{lower}}})}$$

where $\text{price}_{\text{lower}}$ is strictly less than $\text{price}_{\text{upper}}$ and L represents the square root of k in the equation $x * y = k$.

If there isn't already a *Position* that exists for a user with a chosen $Tick_{\text{lower}}$ and $Tick_{\text{upper}}$, liquidity will be added to the respective ticks and the *Position* will have two values initialized, $accumulateEpochLast$ and $claimPriceLast$. The former tells us at what $accumulateEpoch$ the position was

last updated at, while *claimPriceLast* tells us up to what price the user has claimed their expected *Position* token output.

If a *Position* already exists for a user with a chosen $Tick_{\text{lower}}$ and $Tick_{\text{upper}}$, the current *Position* must be updated first before we can create the new *Position*. This is implemented so the *Position* can be partially filled and then have more liquidity deposited thereafter. Once the *Position* is updated, it will be stored with a $Tick_{\text{lower}}$ representing what remains if $token1$ was initially deposited into the *Positon*, or $Tick_{\text{upper}}$ if $token0$ was initially deposited into the *Position*.

2.1.2 Liquidity Auctions

For a single constant $Tick\ T$, it is unlikely the current TWAP will fully inform the pool of what is a competitive market price. There is a possibility that the market price on anothe liquidity pool, perhaps one that is bidirectional, will be superior for the trader at any given point in time without some adjustment. To remedy this, the concept of a *Gradual Dutch Auction* is implemented in order to naturally fit the *Cover Pool* price to to that of other liquidity pools.

Either a Continuous GDA or a Discrete GDA could be implemented depending on which mechanism is more suitable for a given case. Continuous GDAs would not make all the liquidity available immediately to the market across a single *Tick*. Discrete GDAs, while ostensibly longer to carry out the auction, would split the liquidity across a single *Tick* or set of *Ticks* into multiple auctions. Using either method, there is some decay constant λ which will be used to determine the starting price of the liquidity each consecutive block. If there is any Δ from the expected output amount for the LP, we use the value $amountIn\Delta$ to track this difference on a per liquidity unit basis.

2.1.3 Price Tracking

Price is tracked using a time-weighted average price oracle of some sample length determined by the user launching the pool. Each time the TWAP moves by the amount *tickSpacing*, we must initiate the process of what is referred to as "accumulation". Effectively what this process will do is update the *accumulateEpoch* value on each *Tick* passed between the old TWAP and the new TWAP, informing all *Positions* seeking to claim that each of these ticks has been crossed since the user minted their position. Knowing that a tick was crossed allows the protocol to use simple $x * y = k$ math to determine how much each of $token0$ and $token0$ the *Position* is owed, checking *claimPriceLast* to determine which part of a given *Position* has been filled and/or claimed and which has not.

If for some reason there is a *Tick* which is left either partially or completely unfilled (i.e. the TWAP moved before the market was able to fill the user), we will leverage $amountIn\Delta$ and $amountOut\Delta$ to inform the *Position* on a per liquidity unit basis how much was left unfilled. *amountIn* here represents the desired output token for the *Position* while *amountOut* represents the token the LP deposited. The "accumulation" process will carryover these Δ values across ticks until a Tick is crossed which removes all current liquidity in the pool.

After the "accumulation" process is carried out and all *Tick* contract storage is updated, a new *Tick* will be initialized for the new TWAP price and will be linked to the two nearest ticks. The price for $pool1$ will start at a positive spread of *tickSpacing* in favor of the trader, while the price for $pool0$ will do the same based on the new TWAP. This allows the liquidity unlock for each pool to get slightly ahead of the TWAP and thus increase the chances of successfully filling the entire range of the current liquidity auction.

2.2 Price Pool Contracts

[Coming in v1.0.1]

Acknowledgments

Major thanks to all the wonderful people I have met at each and every Ethereum conference. Without your inspiration and dedication to the space this wouldn't have been possible. Together we can build a brighter future.

Big thanks to 0xnexusflip for being a constant sounding board and equal thanks to everyone on my team who believed in me.

References

[1] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. 2021. Uniswap v3 Core. Retrieved Jan 10, 2023 from https://uniswap.org/whitepaper-v3.pdf

[2] Frankie, Dan Robinson, Dave White, and andy8052. 2022. Gradual Dutch Auctions. Retrieved Jan 10, 2023 from https://www.paradigm.xyz/2022/04/gda