Uniswap v3 replaces fungible ERC-20 liquidity positions with non-fungible ERC-721 liquidity positions.

Does that mean it no longer supports flexible Uniswap-v2-style liquidity mining? Or that liquidity mining programs will have to be actively managed, selecting specific ranges to incentivize? Or that liquidity mining programs could be gamed by providing huge amounts of inactive liquidity?

No.

Uniswap v3 can support the same kind of liquidity mining as Uniswap v2—incentivizing all active liquidity pro rata, at a constant rate per second—with only relatively modest compromises. It indirectly incentivizes concentration of liquidity, since liquidity providers are rewarded for their share of virtual liquidity (while it is active), rather than the total value of the tokens they supplied. It can even do all this in a single staking contract, allowing people to stake the same liquidity with multiple incentives at the same time and eliminating the need to deploy new contracts for different incentives.

Omar Bohsali (currently an Entrepreneur in Residence at Paradigm) recently received a grant from the Uniswap Grants Program to implement this algorithm. You can see track his progress on GitHub.

This is the first in a series of posts on How I Learned To Stop Worrying and Love Non-Fungible Liquidity.

The next post will talk about how Uniswap v3 positions can be used as collateral.

The billion-dollar algorithm

To understand how the standard liquidity mining algorithm can be adapted for Uniswap v3, we first need to understand how it works for earlier versions of Uniswap.

This algorithm—for efficient incremental calculation of proportional reward distribution—was first described inScalable Reward Distribution on the Ethereum Blockchain, by Bogdan Batog, Lucian Boca, and Nick Johnson. (Hat tip @FrankieIsLost for discovering this paper.)

The first liquidity mining program where rewards were computed on-chain was SNX'sincentive for the sETH Uniswap pool. This was implemented in the Unipool sETH contract, written in 2019 by Anton Bukov. I consider this one of the most influential smart contracts ever written.

Suppose you want to incentivize liquidity in a Uniswap v1 or v2 pool over a particular period. You want to distribute tokens fairly across liquidity providers and linearly over time—say, at a rate of

tokens per second.

To do this, imagine cutting the pool into single-second time slices. For each of those slices, a given liquidity provider should receive

tokens, where

is that individual liquidity provider's balance of liquidity tokens at time

and

is the total quantity of staked liquidity for that pool at time

. Their reward for a period from

to

would be the total sum of their rewards for each of these seconds:

This would be easy enough to compute off-chain. But how can we efficiently compute it on-chain, with only incremental updates every time someone enters or leaves the pool?

If the user's balance

is constant over that period, then the above formula can be simplified to:

We can then decompose that sum into a difference of two sums. (A very similar trick was used for the oracle accumulator introduced in Uniswap v2.)

This means that all we need to track in the staking contract is a single accumulator tracking "seconds per liquidity" since the beginning of the pool:

In the Unipool contract, this accumulator iscalled rewardPerTokenStored

. When anyone stakes or unstakes liquidity—thus changing

—the staking contract updates the

accumulator, by adding

to its previous value. (Technically, in the Unipool contract and most of its forks, the accumulator tracks

, rather than multiplying by the reward rate later.)

When someone stakes liquidity, the contract checkpoints their starting value of the accumulator,

. When they later unstake, the contract looks at the new value of the accumulator

and computes their rewards for that period:

Uniswap v1 and v2 didn't need any built-in support for these incentives, because these numbers (both the accumulator and the checkpointed values) only change when the staking contract is touched.

In addition to laying the groundwork for the yield farming trend—during which it was forked repeatedly—this clever algorithm turned out to be quite versatile. For example, Uniswap v3 uses a similar algorithm to track fees earned by individual positions (with some additional tricks to support concentrated liquidity). While it is most useful on-chain (where efficiency is paramount), it is also useful for simplifying off-chain computations. The UNI retroactive distribution was based on a query that reimplemented this algorithm in SQL.

Time to concentrate

Uniswap v3 complicates the situation, but not unsolveably.

As described in the whitepaper, Uniswap v3 supports concentrated liquidity

—liquidity that is active only while the current price tick (

) is within a particular range (

).

This means that a price movement can change the liquidity balance of staked positions without the staking contract being touched. For a given position with

liquidity, we now need to calculate:

This can be decomposed into:

We can interpret this as secondsPerLiquidityInside = secondsPerLiquidity

- secondsPerLiquidityBelow(lowerTick) - secondsPerLiquidityAbove(upperTick)

.

The first term can be computed using the same global

accumulator discussed above. In the Uniswap v3 contracts, this global accumulator is tracked alongside the price oracle, as secondsPerLiquidityX128

.

To allow us to compute the other two terms, Uniswap v3 checkpoints secondsPerLiquidityOutside

for each tick every time that tick is crossed. "Outside" means on the opposite side of the tick from the current price. This allows us to compute the total secondsPerLiquidity

accrued on either side of the tick at any time. For example, if tickCurrent < i

, then secondsPerLiquidityBelow(i) = secondsPerLiquidity - secondsPerLiquidityOutside(i)

; if tickCurrent >= i

, then secondsPerLiquidityBelow(i) = secondsPerLiquidityOutside(i)

. (This is quite similar to how Uniswap v3 tracks and computes the fees earned above and below ticks.)

For any range, we can therefore use the above formula to compute secondsPerLiquidityInside

for that range. Uniswap v3 does this calculation for you, and exposes the result through the convenient snapshotCumulativesInside

function.

The staking contract can simply snapshot this secondsPerLiquidityInside

when a user stakes. When they later unstake, the staking contract looks at the new secondsPerLiquidityInside

, takes the difference, and multiplies it by

to get the total rewards earned by the position during that period.

Compromises

We do have to make some compromises based on technical limitations of the new system:

- Fuzzy cutoffs

: There is no way to automatically snapshot all of the accumulators at the exact moment that an incentive ends. After that cutoff, the contract cannot always distinguish between liquidity that was staked before the cutoff and liquidity that was staked after. To accomodate this, the contract can apply a decay to the reward rate for anyone who unstakes after the incentive ends. People who want to lock in the exact reward rate would need to unstake before that.

- Unstaked liquidity

: The algorithm uses the total amount of active liquidity in the core contract, which might be higher than the total staked

liquidity. Unstaked liquidity will still be allocated a share of the rewards, as if it was staked but unclaimed. The creator of an incentive could specify a claim deadline after which they would be able to recover all unclaimed rewards.

Conclusion

Far from "breaking" liquidity mining, Uniswap v3 opens up a vast design space for it. While this post describes an algorithm for implementing "classic" liquidity mining on top of Uniswap v3, I think this barely scratches the surface. As described in section 6.3 of the whitepaper, the Uniswap v3 core contracts expose several other indexes (such as secondsOutside

and tickCumulativeOutside

). I'm looking forward to seeing how protocols make use of these new features.

In a future post, I'll address another common concern about building on top of Uniswap v3 positions: how lending protocols can use them as collateral.