The Value of Uninformed Orderflow to the Uniswap Protocol

Max Holloway *
max@xenophonlabs.com

January 2023

Abstract

We analyze how *uninformed orderflow* – Uniswap orders that are not predictive of future price movements on the Uniswap Protocol – affects the trading fees generated on Uniswap pools. We also present an opinionated framework for how much the protocol should value this uninformed orderflow, as well as recommendations for how the protocol should implement uninformed orderflow incentive programs.

Contents

*Disclosure: The authors do not own `UNI` token, nor are they affiliated with Uniswap Labs or any of its affiliates. This research was funded by a grant from the Uniswap Foundation. Any opinions and results stated here are those of the authors, not of the Uniswap Foundation or its affiliates. Nothing in this paper should be construed as financial advice.

1 Introduction

The Uniswap protocol. The Uniswap protocol is a collection of smart contracts that enable liquidity providers (LPs) to passively make a market on pairs of fungible tokens. LPs deposit tokens into a pool, and traders can place orders against the liquidity in the pool. The price that traders receive is computed in smart contracts based on the state of the pool; importantly, LPs are not required to change their liquidity positions to facilitate trades on the pool. In contrast to limit orderbooks – where market makers generate revenue by quoting higher asking prices and lower selling prices – Uniswap's LPs earn a fee on each order that is proportional to the order's volume.

Since the tokens traded on Uniswap have time-dependent demand, and the price quoted by an Uniswap pool does not utilize a time term, the prices quoted by the pool do not always align with those of external trading venues. In the case of Uniswap V2, the only way of changing the price quoted by a pool is to place an order on the pool [AZR20]. Although this is technically not the case in Uniswap V3, since LPs can set their liquidity positions with different price bounds, the cost of blockspace makes it impractical for LPs to affect price discovery by updating and cancelling liquidity positions [Ada+21]. This marks another difference between Uniswap and the prevailing orderbook-based centralized exchanges, where liquidity providers can remove limit orders to adjust the price without incurring an order cancellation cost. In both Uniswap V2 and V3, price discovery is primarily driven by arbitrageurs.

Under most conditions, taking the other side of an arbitrageur's trade is a very bad deal for LPs. The more general phenomenon of trading between parties with asymmetric information is known in game theory as *adverse selection*. Perhaps obviously, agents with more information about the "true" value of a financial product than their counterparty can use this information to place better trades. And since bilateral exchange of common-value assets is a zero-sum game, it follows that the agent who gets the better deal – typically the agent with better information – profits at the expense of their counterparty. This is relevant in the case of Uniswap, where LPs' trading strategy is fixed by the automated market maker, and arbitrageurs make money at the expense of LPs.

Intuitively, LPs should be more profitable when they face non-arbitrage volume. To study this, we introduce the notion of orderflow information.

Defining Uninformed Orderflow. Before determining the value that uninformed orderflow creates for the protocol, we begin with the simpler problem of determing the value that it creates for Uniswap LPs. We do this by introducing a notion of eventual-correctness, whereby we say an order was uninformed if the LP taking the other side of that order was profitable after some period of time following the order. What follows is the formal definition.

Definition 1 (h-uninformedness)**.** *Let S_t be the true price of the pool's token0 at time t, and let P_t be the pool-given price at time t; these are both random variables. Then for an Uniswap order, we define the net trade vector as the amount of the tokens provided to the pool: $\boldsymbol{a} = [a_0, a_1]^T \in \mathbb{R}^2$, with units of the respective tokens. We can now define what it means for an order to be uninformed.*

For some fixed quantity of time $h \in \mathbb{R}+$, we say that an Uniswap order $\boldsymbol{a}$ placed at time t is h-uninformed if

$$\mathbb{E}[P_{t+h} \mid \boldsymbol{a}] = \mathbb{E}[P_{t+h}].$$

Intuitively, if an order is informed, then our expectation of the future price P_{t+h} will change. We shed more light on this and related definitions in section 2.

Related work. The profitability of Uniswap LPs is not a new topic of research. Angeris et al. provided analytic formulas for the profitability of Uniswap LPs between discrete points in time [Ang+19]. White demonstrated that Uniswap LPs with nearly-zero fees outperform those with higher fees under specific volatility and drift conditions. A number of reports have shed light on the historical profitability of Uniswap V3.

This paper. The aim of this paper is to find the value that uninformed orderflow creates for the protocol, as well as discuss the ways through which the protocol can incentivize this orderflow. We begin in section 2 by finding the marginal revenue that uninformed orderflow creates for LPs, and along with the marginal increase in liquidity that we would expect to come from an increase in revenue. In section 3, we provide an opinionated framework for how much the protocol should value an increase in liquidity. We then give recommendations for how the protocol should incentivize uninformed orderflow – if at all – in section 4. Finally, we provide areas for future work in section **??** and conclude in section 5.

2 Uninformed Orderflow's Value to LPs

Before determining the value that uninformed orderflow creates for the protocol, we begin with the simpler problem of determing the value that it creates for Uniswap LPs. We present two lower bounds on this value. The first lower bound only counts the value created by an uninformed order itself, and the second lower bound counts the value created by sandwich attacks that are caused by uninformed orders.

2.1 Lower Bound I: Uninformed Orderflow's Direct Value to LPs

How much value does an uninformed order create for Uniswap LPs? We define the *token0 volume* of a transaction as $||\mathbf{a}|| = \text{abs}(a_0)$, which has units of token0. Let $\gamma \in (0,1)$ be the fee paid by the trader, $\phi \in \{1/4, 1/5, \ldots, 1/9, 1/10\}$ be the protocol fee. Let the price of the pool at time t be P_t, and let the true price of token0 relative to token1 follow a random continuous-time stochastic process S_t. To find how much LPs should value this orderflow, we utilize the markout metric.

Definition 2 (h-Markout)**.** *For a given time lag h and trade $\boldsymbol{a}$ at time t, we define the markout for the order $\boldsymbol{a}$ as*

$$m_{\boldsymbol{a}} = d \cdot ||\boldsymbol{a}|| \cdot (P_{t+h} - \overline{p}(\boldsymbol{a})),$$

where d is -1 if the LPs are selling token0 and 1 if the LPs are buying token0, and $\overline{p}(\mathbf{a})$ is the execution price of order $\mathbf{a}$, including fees paid to the LP.

Markout embodies the goodness of a trade placed at time t, measured at a time $t+h$ in the future. While a positive value of markout does not imply that an LP's position is profitable long-term, it *does* imply that the LP's trade is profitable at the time h after the trade; a similar statement can be made for negative markout trades. However, this is not a critical flaw to the metric, for the following reason. If we assume that prices follow a zero-drift geometric Brownian motion (GBM), with no mean-reversion terms, then we can also demonstrate that for any lag $h' > h$,

$$\mathbb{E}[\text{markout at time } t + h'] = \text{profit at time } t + h = m_{\mathbf{a}}.$$

This follows from the fact that the markout is simply a scalar addition and multiplication of the underlying price process P_t, and if P_t follows a zero-drift GBM for times greater than $t + h$, our h-markout is the the same in expectation as h' markout. In fact, it is not even necessary that P_t follow a GBM; any continuous-time stochastic process P_t that obeys $\mathbb{E}[P_{t+h'}] = \mathbb{E}[R_{t+h}]$ for any $h' > h$ suffices for markout to be equal in expectation at and after the lag-time h.

Since this relationship often holds empirically, we find it to be an the most practical metric for measuring the value of an order for Uniswap LPs.

Theorem 1 (Expected markout on a single uninformed order)**.** *Given an order* $\boldsymbol{a}$*, its expected markout for LPs,* $\mathbb{E}[m_{\boldsymbol{a}}]$*, is lower bounded by the fees that LPs earn on the order. That is,*

$$\mathbb{E}[m_{\boldsymbol{a}}] > ||\boldsymbol{a}|| \cdot p_{avgPostFee} \cdot (\gamma(1-\phi)).$$

Proof. We begin by obtaining an expression for $\overline{p}(\mathbf{a})$, where x represents an amount of token0 and y represents an amount of token1.

$$\overline{p}(\mathbf{a}) = \frac{y_{intoAmm} \cdot (1-\gamma\phi)}{x_{outOfAmm}} = \frac{(y_{reserveChange}/(1-\gamma)) \cdot (1-\gamma\phi)}{x_{reserveChange}} = \frac{1-\gamma\phi}{1-\gamma} \cdot p_{avgPostFee}.$$

This allows us to represent markout in the following way

$$\begin{aligned} m_{\mathbf{a}} &= (-1) \cdot ||\mathbf{a}|| \cdot (S_{t+h} - \frac{1-\gamma\phi}{1-\gamma} \cdot p_{avgPostFee}) \\ &= ||\mathbf{a}|| \cdot (-S_{t+h} + \frac{1-\gamma\phi}{1-\gamma} \cdot p_{avgPostFee}) \\ &\approx ||\mathbf{a}|| \cdot (-S_{t+h} + (1+(1-\phi)\gamma) \cdot p_{avgPostFee}) \\ &= ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - S_{t+h}}{p_{avgPostFee}}\right), \end{aligned}$$

using the Taylor approximation of $\frac{1-\gamma\phi}{1-\gamma}$ around $\gamma = 0$ for the approximate-equal step.

With this definition for markout, we may now move on to determining the value that an uninformed order $\mathbf{a}$ would create for the LPs of a pool. When $d = -1$, we would have

$$\begin{aligned} \mathbb{E}[m_{\mathbf{a}}] &= \mathbb{E}\left[||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - S_h}{p_{avgPostFee}}\right) \mid \mathbf{a}\right] \\ &= ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - \mathbb{E}\left[S_h \mid \mathbf{a}\right]}{p_{avgPostFee}}\right). \end{aligned}$$

Now we assume that $\mathbb{E}[S_h \mid \mathbf{a}] = \mathbb{E}[P_h \mid \mathbf{a}]$; this assumption should intuitively follow from the existence of arbitrage for sufficiently large h. This gives us the following

$$\begin{aligned} \mathbb{E}[m_{\mathbf{a}}] &= ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - \mathbb{E}\left[S_h \mid \mathbf{a}\right]}{p_{avgPostFee}}\right) \\ &= ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - \mathbb{E}\left[P_h \mid \mathbf{a}\right]}{p_{avgPostFee}}\right). \end{aligned}$$

Now, if we assume that an order is h-uninformed, it follows that $\mathbb{E}[P_t \mid \mathbf{a}] = E[P_{t+h}]$, and we find that

$$\begin{aligned} \mathbb{E}[m_{\mathbf{a}}] &= ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - \mathbb{E}\left[P_h \mid \mathbf{a}\right]}{p_{avgPostFee}}\right) \\ &= ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - \mathbb{E}\left[P_h\right]}{p_{avgPostFee}}\right). \end{aligned}$$

If we assume that $\mathbb{E}[P_{t+h}] = P_t$, then we find

$$\begin{aligned} \mathbb{E}[m_{\mathbf{a}}] &= ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - \mathbb{E}\left[P_h\right]}{p_{avgPostFee}}\right) \\ &= ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot \left(\gamma - \gamma\phi + \frac{p_{avgPostFee} - p_0}{p_{avgPostFee}}\right) \\ &> ||\mathbf{a}|| \cdot p_{avgPostFee} \cdot (\gamma(1-\phi)). \end{aligned}$$

This demonstrates the desired result for $d = -1$, the case where the pool is selling token0. A similar result can be shown for the case of $d = 1$. □

Despite the legwork required for the proof, this result should be rather intuitive: in expectation, Uniswap LPs make at least the LP fee on each unit of uninformed volume in the pool. Technically, LPs make slightly more that this value in expectation, due to the fact that uninformed orders move the pool price. Although this approach has given us a lower bound on the h-markout of an uninformed order, it ignores the h-markout of other orders that are caused by uninformed orders. In particular, the uninformed order h-markout ignores sandwich volume. We now create another lower bound on markout value that takes sandwich attacks into account.

2.2 Lower Bound II: Uninformed Orderflow's Sandwich Value to LPs

We have demonstrated, for a given order, a lower bound on the LPs' expected markout. Interestingly, some uninformed orders can *cause* other orders, and in this case, we should ascribe the value of the *caused* order to the uninformed order that caused it. This is far more than a theoretical distinction, since it occurs in real life in the form of sandwich attacks.

A sandwich attack is a type of economic attack that occur on blockchains that propagate trades via public mempools, such as is the case for orders placed on Ethereum Mainnet through the `app.uniswap.org` trading interface. If a trader *Alicia* specifies a worst-case execution price that is far worse than the price she would get from trading on the last block's state, then it is possible for another trader, Bobby, to place an order before the Alice's order and after Alice's order to "sandwich" the her order. Henceforth, we refer to Bobby's first and second transaction as the front- and back-run transactions, respectively; we will refer to Alice's transaction as the victim transaction. There is extensive literature on sandwich attacks and their mitigations ([KDC22], [Züs21], [Zho+21]).

The reason sandwiches play a relevant role in this analysis is due to the fact that they are typically caused by an uninformed order, typically via unsophisticated users who place orders with sloppy slippage tolerance and broadcast these orders through public channels. Since the front- and back-run orders surrounding the uninformed victim order also must pay fees, it is clear that those orders also lead to similar fees paid to miners. We observe empirically that these front- and back-run orders are themselves balanced, selling the same amount as that which is purchased – and thus do not demonstrate the directionality that would be needed in order for them to be predictive of future price movements. That is to say, front- and back-run orders appear to be uninformed, and thus they yield the same markout for LPs as regular uninformed orderflow.

To find the amount of sandwich value that an uninformed order creates for LPs, we can utilize either of two orthogonal approaches: (a) demonstrate optimal sandwich sizes with respect to uninformed order size and slippage tolerance, and (b) use empirical data to find the amount of sandwich value that historical uninformed orders have created for LPs. We demonstrate both approaches here.

2.2.1 Quantifying Sandwich Value Analytically

For Uniswap V2, [HW22] derived the optimal sandwich size with respect to an uninformed order. The resulting analytical solution provides formulas for computing the optimal sandwich input amounts for the front- and back-running trades.

For orders on Uniswap V2 that have no slippage tolerance specified and which is purchasing token1 (WOLOG), the optimal sandwich order input size is the following:

$$\delta^o_{a_x} = \frac{\delta_{v_x}(1-\gamma)^2 x_0 - (2-\gamma)\gamma x_0^2}{(2-\gamma)\gamma x_0 - \delta_{v_x}(1-\gamma)^2\gamma} + \frac{\sqrt{\delta^2_{v_x}(1-\gamma)^3 x_0(x_0 - (1-\gamma)^2\gamma)}}{(2-\gamma)\gamma x_0 - \delta_{v_x}(1-\gamma)^2\gamma},$$

where $\delta^o_{a_x}$ is the optimal sandwich size, δ_{v_x} is the amount of token0 that the victim is putting into the AMM, and x_0 is the amount of token0 in the pool initially [HW22]. When an order specifies

a slippage tolerance s, then the optimal sandwich amount is the following:

$$\delta^s_{a_x} = \frac{\frac{\sqrt{n(x_0,\gamma,\delta_{v_x},s)}}{1-s} - \delta_{v_x}(1-\gamma)^3 - (2-\gamma)(1-\gamma)x_0}{2(1-\gamma)^2},$$

where

$$\begin{aligned} n\left(x_0,\gamma,\delta_{v_x},s\right) = (1-\gamma)^2(1-s)\left(\delta^2_{v_x}(1-\gamma)^4(1-s) + 2\delta_{v_x}(1-\gamma)^2(2-\gamma(1-s))x_0\right. \\ \left. + (4-\gamma(4-\gamma(1-s)))x_0^2\right). \end{aligned}$$

For an algorithm on how we can utilize this method on Uniswap V3, see appendix A.

We can utilize these optimal sandwich order formulas, along with historical on-chain liquidity data, to find the amount that a sandwiched order **a** would create for LPs, in effect allowing us to give a sandwich correction factor that we could add to our markout formula above.

Although this analytical-like method has an appeal, it still requires an algorithm to compute, it requires us to use historical on-chain liquidity data, and it requires us to make gas price assumptions. This makes the method most apt for *ex ante* estimation of the value of uninformed orderflow. If we can wait to ascribe value to orderflow *ex post*, then we can utilize the empirical approach below in section 2.2.2.

2.2.2 Quantifying Sandwich Value Empirically

As an alternative to determining optimal sandwiching analytically, we can utilize historical sandwich data to measure the amount of value created for LPs as a result of an uninformed order getting sandwiched.

Data collection methodology. To collect this data, we use the Uniswap V3 subgraph, and the analysis presented here is performed specifically on the ETH-USDC 0.05% fee Uniswap V3 pool. We check that a transaction is sandwiched by utilizing the same methodology as Johnny Chuang and Anderson Chen in their Uniswap bounty [CC22]. We deem that an order o has been sandwiched if the following conditions hold: (0) there is an address *Addy* that placed two orders in the same block as o, (1) the orders on either side of o were placed by *Addy*, (2) *Addy*'s order before o was in the same direction as o, (3) *Addy*'s order after o was in the opposite direction as o, (4) the orders on either side of *Addy*'s order are do not utilize an aggregator or router contract, and (5) the orders on either side of *Addy*'s have a token0 amount within 50% of each other. We apply this filter to all of the swaps in all of the Ethereum blocks from July 31, 2022 - Dec 31, 2022 to create a dataset of sandwiches.

Results. We find that sandwiched orders create an immense amount of value for the LPs of the protocol. The important figures can be found in appendix B.1, and we present some of the most important figures here.

We also present plot 2.2.2, showing the volume of the sandwich victim and front-/back-run orders. We see that there are two regions of sandwiches, with a cutoff at approximately 100,000 USDC of meat volume. Roughly speaking, when the meat volume is less than 100,000 USDC, the bun volume is correlated to the meat volume (left plot); when the meat volume is greater than 100,000 USDC, the bun volume is not correlated to the meat volume. This arises due to there being two regions of sandwich optimization. When the meat volume is less than small (e.g. less than 100,000 USDC) the sandwich bot must balance its LP fees paid vs its revenue from the sandwich attack. When the meat volume is large (e.g. greater than 100,000 USDC) the sandwich bot will place orders large enough to hit the meat order's maximum slippage.

By inspection, we can tell from these graphs that a sandwiched order of volume $10^{5.8}$ can realistically cause anywhere from 100,000 USDC to 10,000,000 USDC. Although we can approximate the amount of sandwich volume that a sandwiched order causes, the value varies widely, and we

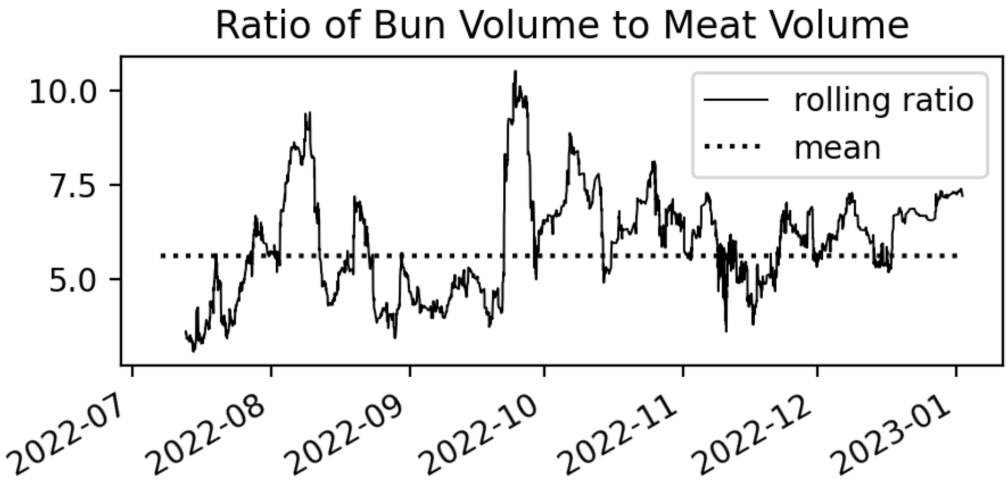

Figure 1: The 100-sandwich rolling ratio of bun volume to meat volume for the ETH-USDC 0.05% pool.

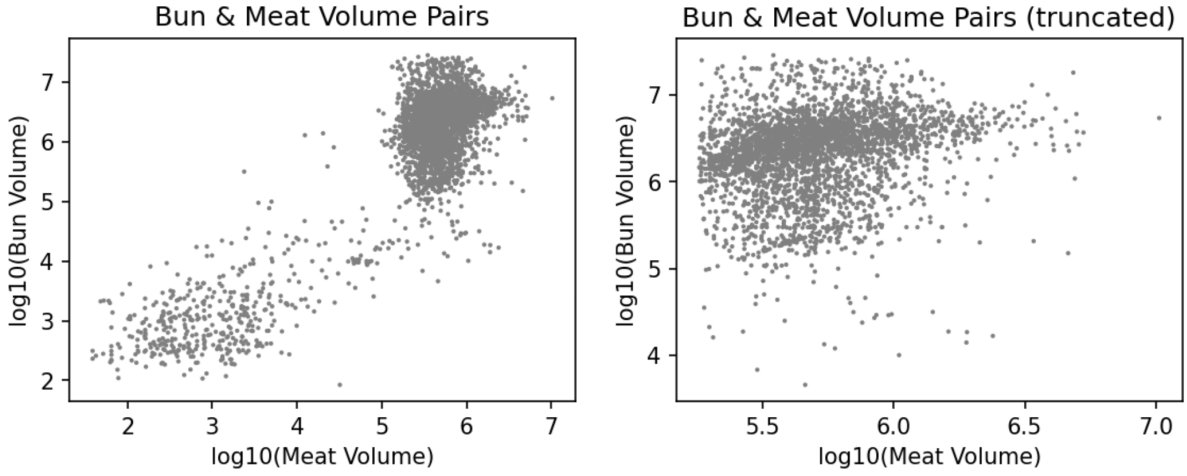

Figure 2: Scatter plots of (meat, bun) volume pairs. The left plot shows all of the sandwich data, and the right plot only considers sandwiches where the meat volume is greater than 100,000 USDC.

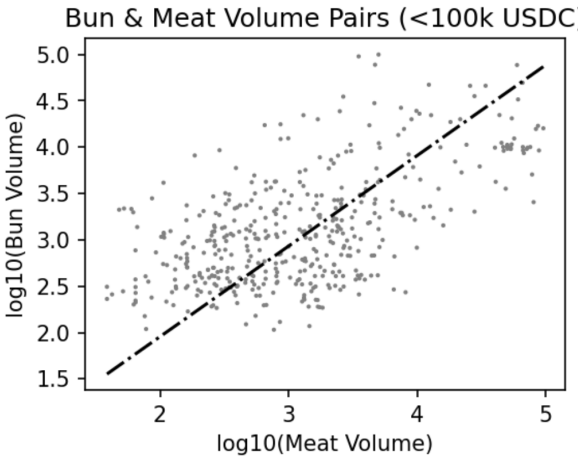

Figure 3: Scatter plots and line of best fit for (meat, bun) volume pairs for meat volume less than 100,000 USDC.

should not give a closed-form approximation of the value of sandwich fees generated from a single sandwiched order. Instead, it is more fair to say that, by inspection of the ratio plot 2.2.2, on-average victim volume causes between 4 and 7 times as much front- and back-run volume.

3 Uninformed Orderflow's Value to the Protocol

How much value does uninformed orderflow create for the protocol? Here we put forward an opinionated approach based on the expected protocol revenues that would materialize in a world with a nonzero protocol fee.

3.1 A Protocol Fee Based Approach

The simplest approach that we have for determining the value that uninformed orderflow creates for the protocol is to analyze, under various fee switch scenarios, how much revenue the protocol earns as a result of uninformed orderflow.

Utilizing the sandwich-adjusted value of uninformed orderflow, we can calculate a lower bound for the value that would go to the protocol as $v_{proto}(x) = (1 + m_{sandwich}) \cdot x \cdot \gamma\phi$, for an order of volume x and a sandwich multiple $m_{sandwich}$. The sandwich multiple can be computed by analyzing historical sandwich attacks and setting it as

$$m_{sandwich} = \frac{\text{dollars of front-run and back-run volume}}{\text{(dollars of uninformed volume) - (dollars of front-run and back-run volume)}}. \quad (1)$$

Computing the numerator and the left-hand side of the denominator of equation 1 is straightforward, but computing the right-hand side of the denominator is less obvious. Even if we know a order's observed markout, it is impossible to determine if that order was informed. With that said, we can create a lower bound on $m_{sandwich}$ by creating an upper bound on the dollars of uninformed value, which itself is tractable. See appendix C for further details on orderflow information bounds.

For the USDC-ETH-0.05% Uni V3 pool, we estimate the following value of $m_{sandwich}$:

$$m_{sandwich} = \frac{\text{17\% of all pool volume}}{\text{(37\% of all pool volume)} - \text{(17\% of all pool volume)}} = 0.85. \quad (2)$$

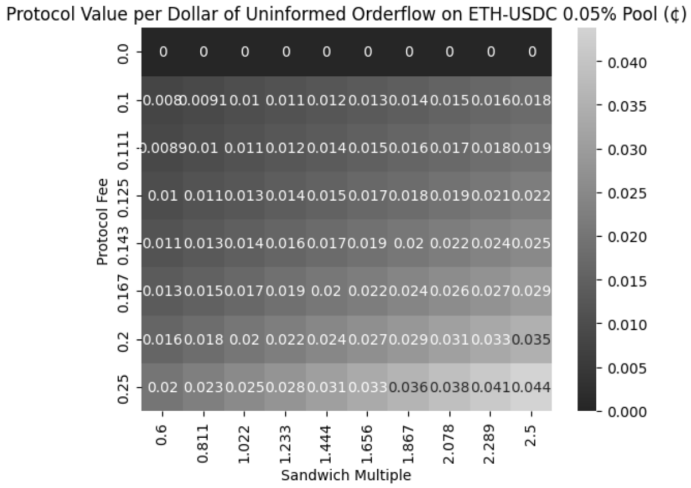

Figure 4: The value per dollar of uninformed orderflow for various protocol fees and sandwich multiples.

That is, each unit of volume will beget approximately 0.85 more units of uninformed volume due to sandwiching. Recall from section 2.2, this method of attributing front- and back-running volume to an order given only the order's size is extremely inaccurate for single order, but can be used across many sandwich orders.

Next, we can utilize many potential protocol fees alongside the sandwich multiple to find the value that the protocol would make in the case where there was a protocol fee. See the figure 3.1 for a visual depiction.

With current estimates of the sandwich multiple at 0.85 on the USDC-ETH-0.05% pool, the current value to the protocol is shown for each fee tier in figure 3.1.

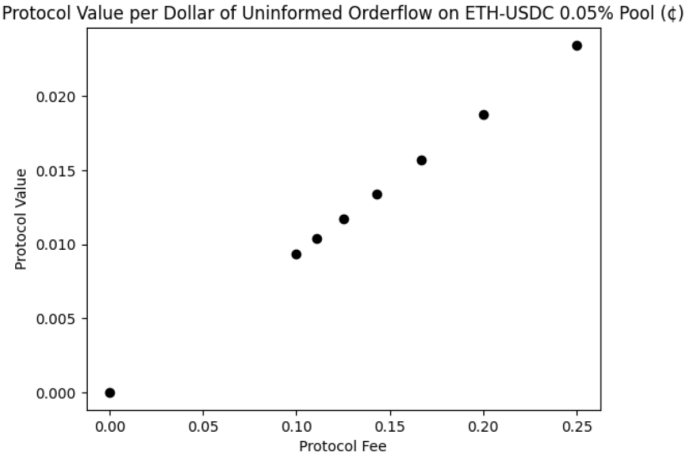

Figure 5: The value per dollar of uninformed orderflow at the current empirically-observed sandwich multiple of 0.85.

It is important to note that these estimates are contingent on there being roughly the same amount of liquidity in a pool before and after a protocol fee is turned on. Due to the technical and

political complexity of the protocol fee, we refrain from analysis of the impact that the protocol fee would have on liquidity. Nevertheless, it is intuitive that the amount of liquidity would decrease in pools that have higher protocol fees, and this would lead to smaller sandwich volumes. Our chart assumes that there is no relationship between the sandwich multiple and the protocol fee, when this is not true.

Thus, our dear reader now approaches a fork in the road. If one believes that small values of the protocol fee would not lead to a meaningful decrease in liquidity, then figure 3.1 demonstrates a lower bound on the average amount of protocol revenue generated from each dollar of uninformed orderflow. This would allow us to conclude our analysis with a simple, yet very protocol-fee-dependent value describing the value of uninformed orderflow.

On the other hand, if one believes that an implementation of the protocol fee would lead to a meaningful decrease in liquidity, then the orderflow values here would be over-approximations of the value that uninformed orderflow creates for the protocol. The next step in this line of research would be to model the relationship between the protocol fee and liquidity, then to model the relationship between the sandwich multiple in liquidity. While this is an excellent opportunity for future research, one will find no answers on that topic in this paper.

Aside from this approach ignoring the effect of the protocol fee on liquidity, it also possesses the obvious drawback that, at the time of writing, there is no protocol fee. This approach would thus lead us to the unfortunate result that uninformed orderflow currently has zero value for the protocol. Readers can decide, according to their own protocol fee political leanings, if this is too harsh of an underestimate of uninformed orderflow value to the protocol.

3.2 Non-approaches

We now list a number of intuitive, yet flawed ways of valuing orderflow for the protocol. We specifically advise that the protocol not utilize the following methodologies for evaluating the value of uninformed orderflow.

Non-approach 1: Liquidity is inherently valuable.

Another approach to valuing orderflow is to see how much it would affect liquidity, then ascribe a value on liquidity itself. Accumulating more liquidity would put Uniswap in a position of strength relative to its competitors. For instance, increased Uniswap liquidity leads to an increased share of dex aggregator volumes, which leads to decreased dex aggregator volumes for Uniswap's dex competitors. However, ascribing a number to the value this creates for the Uniswap protocol would be a perilous task, since even in a world where Uniswap defeats all competitors, the protocol must generate revenue in order for that market dominance value to accrete. Thus we would need to fall back to our previous protocol fee based approach.

Non-approach 2: Uniswap could generate revenue from other sources.

While it is also possible that the Uniswap protocol could generate revenue from other than the protocol fee, it would be inappropriate for us to speculate on the existence, let alone the size, of those revenue opportunities. The protocol may revisit this approach if there is a tangible proposal for generating protocol revenues in a way other than the protocol fee.

Non-approach 3: Uninformed orderflow increases the value of the token.

Yet another approach for quantifying the value that uninformed orderflow creates for the protocol would be to determine how much value uninformed orderflow would accrete to the token. While on its surface this approach resembles a value-based management approach to valuing uninformed orderflow, it is clear that uninformed orderflow itself will not create value for tokenholders unless there is a revenue source for the token. In order to utilize this method of valuing orderflow, we would either need to default back to our initial approach of assuming there is a protocol fee, or we would need to assume there is another source of revenue, which we refute in non-approach 2.

A similar, yet just as inapplicable, train of thought is to determine the relationship between uninformed orderflow and the price of the UNI token. An increase in UNI token value would allow

us to grow the protocol's liquidity and volume. Yet, this begs the question of why UNI token holders would want to increase the usership of the protocol, considering the fact that usership itself does not accrete in revenue.

If one is instead optimizing not for the UNI token value, but instead for the UNI token value *for current holders*, then this approach is more coherent, since increasing the UNI token value would allow current holders to sell the token and realize a gain. Unfortunately for these aspiring tokenholders, we will not entertain this use case as an approach that the protocol should use to value uninformed orderflow.

3.3 Takeaways

We provide an approach to measuring the value that the protocol would receive from uninformed orderflow, assuming a future state where protocol fees are collected. We provide a demonstration of the amount of value that would be created for LPs if a protocol fee were put in place. The main drawback to this approach is that we do not model the effect that an increase in protocol fee would have on pool liquidity; we anticipate that some readers to find this objectionable.

Although introducing the protocol fee requires us to make assumptions about future governance actions, we argue that this is the only coherent way of placing value on uninformed ordeflow. All other intuitive methods – the inherent value of uninformed orderflow, Uniswap generating non-protocol fee revenue, and token price – suffer from other issues that make them unsuitable as a method for valuing uninformed orderflow.

4 How to Incentivize Uninformed Orderflow

Assuming we know how valuable a dollar of uninformed orderflow is for the protocol, we can now discuss ways that the protocol might incentivize uninformed orderflow. The incentive design space is large, and we focus only on two approaches: direct user incentives and user interface incentives.

4.1 Goals

4.2 Approaches

5 Conclusion

A Computing an Optimal Sandwich

Here we give a simple algorithm for computing optimal sandwich attacks on Uniswap V3. In words, we do roughly the following.

We check if the optimal sandwich order size surpasses the current tick; if it does, then the order size is the amount needed to get to the next tick, plus the result of the optimal arbitrage formula recursed on the next optimal sandwich solution; if it does not, then use the optimal sandwich solution. If in the recursive step or the base case the trade is unprofitable, then set the optimal amount to 0. This algorithm's correctness comes from the fact that sandwich profits w.r.t. trade size are single-peaked, and thus we can optimistically cross as many ticks as necessary to get to the optimal sandwich that ignores gas costs, and if we are ever unprofitable, we can simply lower our sandwich input amount.

See the following figure for python-like pseudocode implementation.

```
def optimal_sandwich_input(victim_order, curtick, pool_data, gas_fee_per_tick,
                                          gas_fees_incurred=0):
    """
    Assuming there's a victim order that's buying token0, here's
    an algorithm for computing the optimal sandwich input size of
    token1. A similar algorithm can be used to compute the optimal
    sandwich input when the victim is selling token0.
    """
    # optimal sandwich size if trading on Uniswap V2
    v2_size = v2_optimal_sandwich_size(victim_order, pool_data)
    curtick_max_size = get_size_until_tick_crossing()

    if v2_size > curtick_max_size:

        # find the optimal rest of the trade to perform, under the
        # optimistic assumption that it's worth it to trade on this
        # tick
        gas_fees_incurred += gas_fee_per_tick
        rest_of_order_size = optimal_rest_of_sandwich(victim_order, curtick+1,
                                                      pool_data, gas_fee_per_tick,
                                                      gas_fees_incurred)

        if trading_this_tick_itself_is_profitable(gas_fee_per_tick,
                                                  curtick_max_size):
            return curtick_max_size + rest_of_order_size
        else:
        return 0

    else:
        if trading_this_tick_itself_is_profitable(gas_fee_per_tick, v2_size):
        return v2_size
        else:
            return 0
```

B Data

B.1 Sandwich Data

C Identifying Uninformed Orders

References

[Ada+21] Hayden Adams et al. "Uniswap v3 Core". In: *Tech. rep., Uniswap, Tech. Rep.* (2021).

[Ang+19] Guillermo Angeris et al. "An analysis of Uniswap markets". In: *arXiv preprint arXiv:1911.03380* (2019).

[AZR20] Hayden Adams, Noah Zinsmeister, and Dan Robinson. "Uniswap v2 Core". In: (2020).

[CC22] Johnny Chuang and Anderson Chen. *Uniswap Bounty #19 - MEV, Sandwich Attacks and JIT*. Accessed 4 Jan 2022. 2022. URL: https://coinomo.notion.site/Uniswap-Bounty-19-34df5d8d69e54b2ba10f5b5799758d26.

[HW22] Lioba Heimbach and Roger Wattenhofer. "Eliminating Sandwich Attacks with the Help of Game Theory". In: *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 2022, pp. 153–167.

[KDC22] Kshitij Kulkarni, Theo Diamandis, and Tarun Chitra. "Towards a theory of maximal extractable value i: Constant function market makers". In: *arXiv preprint arXiv:2207.11835* (2022).

[Zho+21] Liyi Zhou et al. "High-frequency trading on decentralized on-chain exchanges". In: *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2021, pp. 428–445.

[Züs21] Patrick Züst. *Analyzing and Preventing Sandwich Attacks in Ethereum*. 2021.