The Potential of Self-Regulation for Front-Running Prevention on DEXes

LIOBA HEIMBACH, ETH Zurich, Switzerland
ERIC SCHERTENLEIB, ETH Zurich, Switzerland
ROGER WATTENHOFER, ETH Zurich, Switzerland

The transaction ordering dependency of the smart contracts building *decentralized exchanges (DEXes)* allow for predatory trading strategies. In particular, front-running attacks present a constant risk for traders on DEXes. Whereas legal regulation outlaws most front-running practices in traditional finance, such measures are ineffective in preventing front-running on DEXes due to the absence of a central authority. While novel market designs hindering front-running may emerge, it remains unclear whether the market's participants, in particular liquidity providers, would be willing to adopt these new designs. A misalignment of the participant's private incentives and the market's social incentives can hinder the market from adopting an effective prevention mechanism.

We present a game-theoretic model to study the behavior of traders and liquidity providers in DEXes. Our work finds that in most market configurations, the private interests of traders and liquidity providers align with the market's social incentives — eliminating front-running attacks. However, even though liquidity providers generally benefit from embracing the market that prevents front-running, the benefit is often small and may not suffice to entice them to change strategy in reality. Thus, we find that inert liquidity providers might require additional incentives to adopt innovative market designs and permit the market's successful self-regulation.

1 Introduction

The emergence of *decentralized finance (DeFi)* on Ethereum [46] greatly enhanced the interest in cryptocurrency applications. DeFi is a blockchain-based form of finance that utilizes *smart contracts* to offer many traditional financial instruments, but without relying on financial intermediaries. A prime example thereof are *decentralized exchanges (DEXes)*. While traditional exchanges match individual buyers and sellers with the limit order book mechanism, a DEX algorithmically sets the exchange rate for a trade. To this end, DEXes store liquidity for exchanges between individual cryptocurrency pairs in smart contracts, referred to as *liquidity pools*. The trade size and the respective cryptocurrency pair's amount and ratio of reserves control the price. The pool charges a fee for every trade which is proportional to the trade's input amount and distributed pro-rata amongst the pool's *liquidity providers*.

DEXes are becoming increasingly popular. Yet, the rise of DEXes does not come without caveats, leading to the characterization of the Ethereum peer-to-peer network as a dark forest. Predatory trading bots prey on user transactions in Ethereum's *mempool*, the public waiting area for transactions. Predatory trading schemes exploit the lack of privacy given to transactions prior to their execution. Moreover, the smart contracts that build DEXes are dependent on the transaction order. Generally, these attacks involve the attacker *front-running* a victim's transaction. One of the most frequently observed strategies exploiting this dependency on transaction ordering is the *sandwich attack* [41] which we describe in section 3. We focus on sandwich attacks in this work, as this is the only front-running attack directly impacting the welfare of liquidity providers.[1] Such an attack occurs when a trading bot front- and back-runs a victim's transaction, forcing the trade to execute at an unfavorable price. Between 13 April and 12 May 2023, 239,446 transactions were sandwiched on Ethereum blockchain's DEXes and these sandwich attacks generated a profit in excess of US$55M [9].

Given the severity of front-running attacks on DEXes, the market is seeking mechanisms that can prevent such attacks. While front-running attacks are outlawed in traditional finance, the anonymity of market participants

[1] Apart from sandwich attacks, there are destructive front-running attacks [27]. Thereby, the attacker searches for trades that exploit arbitrage opportunities and front-runs these and is indifferent to whether the victim's transaction executes. The DEXes volume remains unchanged.

and the absence of a central authority do not allow for an effective regulatory approach for DEXes. Therefore, they require new innovative solutions to prevent front-running attacks. In response, multiple approaches to prevent front-running and, more generally, transaction reordering manipulation are currently under development [29]. These approaches generally ensure that transaction contents are private from the public until an order is agreed upon. Further, some designs are already being adopted [5, 6].

Even though successful market designs preventing front-running attacks increase market efficiency [34], there may still be insufficient incentives to adopt these prevention schemes. Liquidity providers, who potentially benefit from the additional trading volume from sandwich attacks, might be reluctant to embrace such a market. Examples, where the divergence of private and social incentives has led to adoption failure exist in traditional limit order book exchanges [20].

In this work, we develop a game-theoretical model to study whether traders and liquidity providers would embrace DEXes that incorporate a front-running prevention mechanism. Our repeated game consists of two hypothetical liquidity pools, one allowing sandwich attacks and one implementing a sandwich attack prevention scheme. Traders and liquidity providers distribute across the two pools, thereby maximizing their private incentives. Liquidity providers try to maximize their fees earned and the traders seek to execute their order at the best possible price. Thus, while the presence of sandwich attacks leads to additional trades from the attack, it reduces the volume of ordinary traders as they receive a poor price.

We find that in equilibrium, the players exclusively utilize the liquidity pool with front-running prevention for most market configurations — indicating that the market can fix itself. Yet, market conditions exist where the private incentives of liquidity providers and the system's social incentives are misaligned. Finally, we show that even when the private incentives of liquidity providers align with the market's social incentives, the benefit of embracing the new market is often negligible. Therefore, despite the alignment, the market may have to provide additional incentives to entice liquidity providers to adjust their strategy.

2 Related Work

Front-running has long been prevalent in traditional finance [18, 22, 25] where the regulator is tasked with banning such practices [35, 38]. The lack of a central authority in DeFi, however, means that the market must regulate itself. Thus, we study whether the private incentives of market participants obstruct the adoption of innovative market designs preventing front-running.

Eskandir et al. [27] are the first to systematize work surrounding front-running on DeFi. In a similar line of research, Daiain et al. [24] study the risks of front-running on DEXes. They observe traditional forms of predatory trading behaviors adapting to the blockchain ecosystem. Park [40] further shows that the pricing rule of most DEXes gives rise to intrinsically profitable front-running opportunities. By analyzing the market participants' private incentives to prevent front-running, we build on these earlier works and, in particular, study sandwich attacks, as they influence the welfare of traders and liquidity providers.

The prevalence of front-running attacks on DEXes is first quantified by Qin et al. [41]. Zhou et al. [50] study sandwich attacks both analytically and empirically. Both demonstrate the risk stemming from front-running attacks on DEXes. Our work, on the other hand, focuses on whether the market participant's private incentives are disruptive to the adoption of market designs preventing front-running attacks and the associated risks.

Recently, many suggestions for DEX front-running prevention schemes have emerged. For a comprehensive overview, we refer the reader to Heimbach and Wattenhofer [29]. Their works provide an overview of state-of-the-art prevention mechanisms and find that current schemes do not meet the blockchain ecosystem's requirements.

We summarise the core ideas behind the most relevant suggestions in the following. The simplest schemes, tune the transaction parameters to prevent specific attacks on specific protocols [28, 49]. Further, several suggestions propose that transactions are sent to a trusted third party that is put in charge of ordering the transactions fairly [1, 2, 5–8, 10, 17, 43]. A parallel line of work, instead of relying on a single entity, trust a generally

permissoned committee to order the transactions in a fair manner [15, 16, 21, 23, 30–33, 36, 37, 39, 42, 47, 48] — preventing front-running. Finally, several schemes set the order of transaction by first having users commit to their transactions on-chain and then only revealing the transaction contents later in a second phase [19, 26, 44]. All of these schemes, thus, aim to preserve the privacy of the transaction contents until an execution order is agreed upon. In this work, instead of assessing or designing prevention mechanisms, we study a market with an ideal prevention mechanism to analyze whether the private incentives would steer market participants to accept such a market design.

Budisch et al. [20] examine the incentives of exchanges' to embrace market design innovations that eliminate latency arbitrage and HFT trading. Their work finds that adoption failures arise in traditional limit order book exchanges. In particular, the divergence of private and social incentives hinders the market from accepting new market designs. We study the incentives of liquidity providers in DEXes and show that their private incentives generally align with the system's social interests: demonstrating that liquidity providers can be incentivized to adopt new market designs preventing front-running attacks.

3 Preliminaries

In the following, we detail the trading mechanism of the biggest DEXes and introduce the sandwich attack specifics.

3.1 Automated Market Maker

As its name suggests, trade execution on *automated market makers (AMMs)* is automatic, and the price is controlled by an algorithm with liquidity being supplied by individual liquidity providers rather than brokers or market makers. A host of AMM variants exist, each with its specific pricing mechanism. We focus on the most widely adopted subclass of AMMs: *constant product market makers (CPMMs)* [4]. For each tradeable cryptocurrency pair, the CPMM stores assets of both cryptocurrencies in a liquidity pool. The CPMM then ensures that the product between the amounts of the two reserved pool currencies stays constant. This property ensures that the price for swapping between these pairs mimics the behavior of a demand curve of a normal good. Both Uniswap and Sushiswap, two of the biggest DEXes, employ the CPMM for pricing. The original CPMM design, as deployed by Uniswap V2 [13] and Sushiswap [11], utilizes the same liquidity for the pool's entire price range. Consider a pool $X \rightleftharpoons Y$ between X-tokens and Y-tokens with respective reserves x and y. Then the pool's marginal price indicating the pool's current price for X-token in terms of Y-token is $P = y/x$ [13]. Further, the pool's liquidity is defined as $L = \sqrt{x \cdot y}$. This liquidity needs to support trading along the entire price range $(0, \infty)$ in the original CPMM design.

In the newest Uniswap design (V3) [14], liquidity providers choose the price range $[P_a, P_b]$ for which they provide liquidity. This concentration of liquidity is intended to increase the capital efficiency, as the liquidity only needs to support trade execution in the corresponding price range. Liquidity providers can only choose from a discrete set of price range boundaries that are defined by the pool's initialized *ticks*. Between each pair of initialized ticks, the CPMM only needs to maintain enough reserves to support trading between the price boundaries. One can simulate a constant product pool with adjusted larger reserves, referred to as *virtual reserves*, between any pair of neighboring initialized ticks.

For a price range $[P_a, P_b]$ between two neighbouring initialized ticks, the *liquidity* inside the tick is given by L and the *marginal price* is P. The CPMM ensures that the constant product of the virtual reserves x and y stays constant, i.e., $x \cdot y = k = L^2$, where k is the constant product of the reserves in the considered price interval. As on Uniswap V2, the marginal price is $P = y/x$ and the liquidity is $L = \sqrt{x \cdot y}$ [14]. Thus, the virtual reserves are

then given by $x = L/\sqrt{P}$ and $y = L \cdot \sqrt{P}$. For the sake of simplicity, we focus on trading between two neighboring initialized ticks and refer to virtual reserves simply as reserves in this work.[2]

The exchange rate received by traders is dependent on their trade size and the number of tokens reserved in the liquidity pool. Consider, again, a liquidity pool between X-token and Y-token, $X \rightleftharpoons Y$. We denote the respective initial (virtual) reserves prior to any trading as x and y, respectively, and use ξ and v for the reserves of X and Y at any time during the trading process. Thus, if a trader adds an infinitesimal amount $d\xi$ to the pool, the following amount of Y is extracted

$$dv = -\frac{xy}{\xi^2} d\xi.$$

This expression follows directly from the constant product property. Note that the sign convention we choose is relative to the pool, i.e., $\Delta v < 0$ for a trader buying Y-tokens. Further, observe that the price per Y-token increases with the input amount. Thus, traders have to pay more per desired token the larger their trade is, resulting in *expected slippage* which is the difference between the pool's marginal price and the actual price received by the trader. Note that the expected slippage is lower in more liquid pools, i.e., those with larger stored reserves.

From the infinitesimal price it follows that a trader wishing to sell δ_x X-tokens will receive δ_y Y-tokens, where

$$\delta_y = -\int_x^{x+(1-f)\delta_x} \frac{-x \cdot y}{\xi^2} d\xi = y - \frac{x \cdot y}{x + (1-f)\delta_x} = \frac{y(1-f)\delta_x}{x + (1-f)\delta_x},$$

and f is the transaction fee which is charged relative to the input amount δ_x and is distributed pro-rata to the tick's liquidity providers. The sign in front of the integral is negative as the trader receives the Y-tokens extracted from the pool. Note that the $(1-f)\delta_x$ in the upper integral bound corresponds to the amount of X added to the pool after deduction of the transaction fee. Thus, post-execution the reserves of X and Y will be $x + (1-f)\delta_x$ X-token and $y - \delta_y$ Y-tokens.

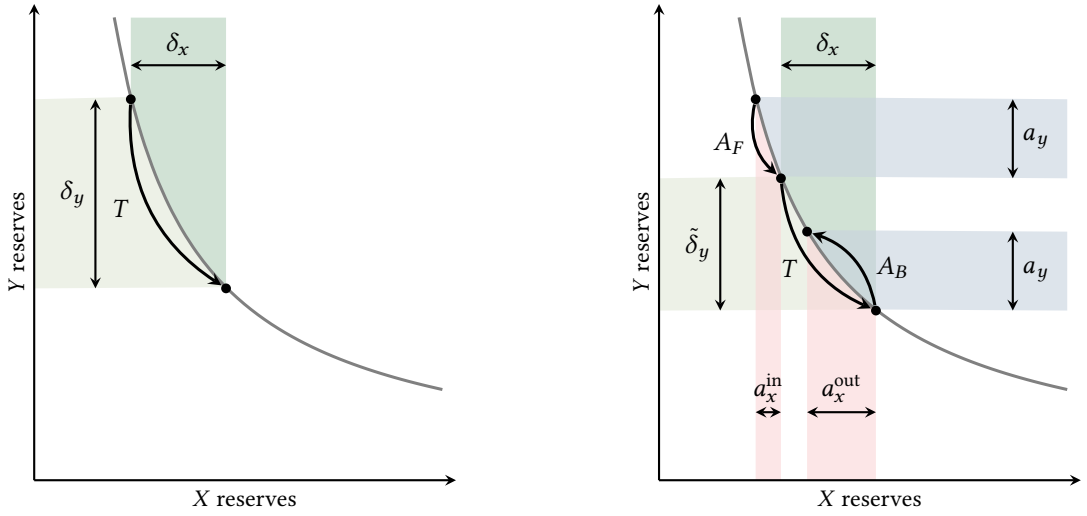
The time at which a trade executes is unclear to the trader, as their transactions will only be confirmed upon block inclusion. In the meantime, other transactions changing the pool's state might occur. The change in the pool's state introduces a difference between the trader's expected price at the time of submission and the actual price at the time of execution. This price change is known as *unexpected slippage*. To ensure the price of the transaction does not deviate significantly from the expectation, traders specify a *slippage tolerance*, indicating the maximum unexpected price movement they are willing to accept. A trade expecting δ_y Y-tokens at the time of transaction submission will only execute if it receives no less than $(1-s)\delta_y$ Y-tokens, where s is the slippage tolerance. Typical slippage tolerances are $s < 0.03$ [45].

3.2 Sandwich Attacks

A too small slippage tolerance results in frequent transaction failure. However, the slippage tolerance also gives an opening for sandwich attacks. On Ethereum, users broadcast their transactions to the network. The transaction waits in the mempool until it is included in a block by a validator. During this time, the transaction is visible to predatory trading bots and runs the risk of being front- and back-run as part of a sandwich attack. Predatory trading bots scan the mempool's inflowing transaction stream searching for profitable sandwich attack opportunities.

As validators control the ordering in a block, sandwich attackers can provide validators with the necessary (financial) incentives to achieve their desired transaction ordering. In fact, *front-running-as-a-service* schemes, such as Flashbots [6] and Eden network [5], facilitate this interaction between sandwich attackers and validators. On the other hand, these services can also be used for front-running prevention, but users must deliberately seek them out rather than being truly incorporated in the market design.

[2]As our analysis focuses on a time-frame where the fair market price between X- and Y-tokens remains constant (cf. Section 4), trading is also likely to occur within a small price range and thus will likely remain within one tick. To cover trading across ticks, one can easily reapply our analysis. Note that trading within a tick on Uniswap V3 is mostly the same as on Uniswap V2.



(a) The execution of transaction T without a sandwich attack. The transaction T receives the Y-assets at the expected price.

(b) The execution of transaction T with a sandwich attack. Transaction T is first front-run by transaction A_F and then back-run by transaction A_B.

Fig. 1. Execution of victim transaction T in pool $X \rightleftharpoons Y$. without (cf. Figure 1a) and with (cf. Figure 1b) sandwich attack. In the presence of an attack the trader receives fewer Y-tokens $\tilde{\delta}_y < \delta_y$ while the attacker makes a profit, i.e., $a_x^{\text{in}} < a_x^{\text{out}}$.

A sandwich attack involves the attacker front-running the victim's transaction, exchanging X-token for Y-token in transaction A_F. The attacker's front-running transaction purchases the same asset as the victim: Y-token. Thereby, the attacker drives up the asset Y's price. The following victim transaction T then buys Y-token at a higher price and further inflates Y's price. To conclude the attack, the attacker back-runs the victim's transaction, selling Y-assets at the inflated price with transaction A_B.

To provide a conceptual understanding of sandwich attacks, we visualize a victim's trade T without a sandwich attack in Figure 1a. Figure 1b then shows how a sandwich attack alters the transaction T. We observe that without the sandwich attack, the victim expects a greater output δ_y (cf. Figure 1a) than the output $\tilde{\delta}_y$ it receives in the presence of a sandwich attack (cf. Figure 1b). The attacker's front-running inflates Y-asset's price. Further, we observe that the attacker's output a_x^{out} of the back-running transaction A_B exceeds the attacker's input a_x^{in} (cf. Figure 1b). The difference $a_x^{\text{out}} - a_x^{\text{in}}$ presents the attackers profit, as the attacker's input a_y to transaction A_B is the output of transaction A_F.

Lastly, we note that at first glance liquidity providers appear to benefit from sandwich attacks as they lead to increased trading volume, and therefore, collected fees. However, ordinary traders could reduce their trading activity in the presence of sandwich attacks, as they receive a worse price than the market price. We will study this interplay by analyzing the effects of sandwich attacks on the utility of both traders and liquidity providers.

4 Model

We model a system with two liquidity pools Pool_N and Pool_W. Both pools facilitate exchanges for the same cryptocurrency pair: $X \rightleftharpoons Y$. While a scheme to prevent sandwich attacks is implemented in Pool_N, sandwich attacks are common practice in Pool_W. With our model, we will study whether DEX participants are able to self-regulate and adopt a DEX with front-running prevention in place.

There are four types of players in our model: traders, liquidity providers, sandwich attackers, and price arbitrageurs. Traders and liquidity providers strive to maximize their personal utility (cf. Section 4.3 and Section 4.4)

across two liquidity pools Pool_N and Pool_W. In maximizing their utilities, traders and liquidity providers account for the effects of sandwich attacks and price arbitrageurs.

Without sandwich attacks, trades in Pool_N execute at the expected price.[3] In Pool_W, on the other hand, sandwich attack bots make an attack whenever it is profitable (cf. Section 4.1). We denote the fraction of the total liquidity placed in Pool_N by p. Thus, the reserves in Pool_N are $x_N = p \cdot x$ X-tokens and $y_N = p \cdot y$ Y-tokens, and $x_W = (1-p)x$ X-tokens and $y_W = (1-p)y$ Y-tokens in Pool_W. Given the price $P_{X\to Y}$ of X-token, we have $P_{X\to Y} = \frac{y}{x} = \frac{y_N}{x_N} = \frac{p\cdot y}{p\cdot x} = \frac{y_W}{x_W} = \frac{(1-p)y}{(1-p)x}$.

We emphasize that the social incentives of our system are to completely adapt Pool_N without front-running. In the presence of sandwich attacks in Pool_W, the trades of ordinary traders do not execute at the effective market price but rather at an unfavorable rate. Further, we purposefully exclude incentives of sandwich attackers and price arbitrageurs when discussing the system's incentives. Including their incentives would turn the game into a zero-sum game. Thus, in the presence of profitable sandwich attacks and price arbitrages, the remaining market participants (traders and liquidity providers) collectively lose money.

Further note that throughout, we assume the slippage tolerance s ($0 < s < 1$) to be fixed, i.e., all victim transactions have the same slippage tolerance. This assumption is reasonable as most DEXes on Ethereum, as, while the slippage tolerance can be modified, most apply a default setting [3, 11, 12]. Similarly, the transaction fee f ($0 < f < 1$) is identical in both pools. Finally, we disregard the *gas fee*, the fee paid to validators for block inclusion on the Ethereum blockchain, for all players in our analysis. The gas fee would add a fixed cost to every trade and liquidity movement. However, for the sake of simplicity and as the gas fee is not part of the CPMM market mechanism itself, we assume the gas fee to be zero.

4.1 Sandwich Attackers

Sandwich attackers observe the inflowing transactions in Pool_W. Upon noticing a trade entering the mempool of Pool_W, they compute the maximal input for the sandwich attack and assess the attack's profitability. The maximal input infers the maximal acceptable price movement on the trader, such that the trade still executes. Attackers conduct any such profitable attack. We find the maximal input of a sandwich attack and study their profitability in Section 5.1.

The victim submits an order T_W wishing to exchange $\delta_{x,W} > 0$ X-tokens in Pool_W for Y-tokens and sets a slippage tolerance s. When submitting the trade T_W, the victim is estimated to receive $\delta_{y,W}$ Y-tokens, i.e., the number of tokens the victim would receive if no other trade is executed beforehand. On the other hand, when a sandwich attack occurs, the attacker front-runs the victim with transaction A_F exchanging $a_x^{\text{in}} > 0$ X-tokens for a_y Y-tokens. Now the victim's transaction executes at a worse price. To finish the attack, the attacker exchanges a_y Y-tokens for a_x^{out} X-tokens in the back-running attack transaction A_B.

We define the attacker's utility as their profit:

Definition 4.1. The attacker's utility $U^A(\delta_{x,W}, f, s, p, x, y)$ is given by

$$a_x^{\text{out}}(\delta_{x,W}, f, s, p, x, y) - a_x^{\text{in}}(\delta_{x,W}, f, s, p, x, y).$$

Here, $a_x^{\text{in}}(\delta_{x,W}, f, s, p, x, y)$ is the input of the front-running transaction and $a_x^{\text{out}}(\delta_{x,W}, f, s, p, x, y)$ is output of the back-running transaction.

We will assume that if a profitable sandwich attack exists, it executes successfully. A bot must have access to the necessary funds and achieve its desired transaction ordering which can be accomplished through *front-running-as-a-service* platforms such as Flashbots [6]. These services further guarantee their users that a transaction

[3]Note while it is possible for there to be several trades in a single block, we can assume them to only amount to natural price fluctuations. In the time frame of a block, they can be assumed to be negligible [28].

will only be included in a block if it executed successfully. Therefore, it is reasonable to assume that profitable sandwich attacks execute successfully.

4.2 Price Arbitrageurs

We consider a time window, during which the external market price between the pools' two cryptocurrencies is constant. Then, price arbitrageurs ensure that the pool's price returns to $P_{X\to Y}$ after every trade sequence (either an individual victim transaction in Pool_N or a victim transaction together with a sandwich attack in Pool_W). Thus, price arbitrageurs balance the market after any set of trades such that it reflects the fair market price. Letting the pool return to its initial state allows us to study the system analytically in the presence of an infinitely long trade flow as opposed to a fixed set of trades.

4.3 Traders

Our game captures a continuous stream of trade orders. Traders wish to sell X-tokens for Y-tokens, as they have a personal use for Y-tokens, and thereby, associate a relative benefit $\alpha > 0$ with Y-tokens. The private benefit associated with the number of Y-tokens a trader buys, $\tilde{\delta}_{y,\bullet}$, is thus given by $(1+\alpha)\tilde{\delta}_{y,\bullet}$ in $\text{Pool}_\bullet$. The trader *strategy space* is $S^T = \{(\delta_{x,N}, \delta_{x,W}) | \delta_{x,N}, \delta_{x,W} \in \mathbb{R}^{\geq 0}\}$, where $\delta_{x,N}$ is the trade input in Pool_N and $\delta_{x,W}$ is the trade input in Pool_W. Thus, traders pick a non-negative trade size in both pools. In our game, traders set their trade sizes across both pools to maximizes their personal benefit.

We model many individual traders as a continuous stream of trade orders. Here, we assume that all traders have the same relative benefit α, and also consider distribution on α to capture non-uniformity among traders.

In Pool_N, where there are no sandwich attacks, the tokens received by traders $\tilde{\delta}_{y,N}$ equals the expected trade output $\delta_{y,N}$, i.e., $\tilde{\delta}_{y,N} = \delta_{y,N}$. On the other hand, in Pool_W traders experience sandwich attacks which reduce the expected output.[4]

In addition to the benefits traders obtain from the received Y-tokens, they also associate a cost with the trade's input, which is given by $P_{X\to Y}\delta_{x,\bullet}$. Here, $\delta_{x,\bullet}$ is the trade input in X-tokens, and $P_{X\to Y}$ is the fair exchange rate from X-tokens to Y-tokens. By combining the trader benefit and cost in both pools, we obtain their utility in Definition 4.2. Note that while we focus on trades from X to Y, by symmetry, the analysis applies directly in the opposite direction.

Definition 4.2. The trader's utility $U^T(\delta_{x,N}, \delta_{x,W}, \alpha, f, s, p, x, y)$ for a trade with input $\delta_{x,N} \geq 0$ in Pool_N and input $\delta_{x,W} \geq 0$ in Pool_W is given by

$$(1+\alpha)\delta_{y,N}(f,p,x,y) - \frac{y}{x}\delta_{x,N} + (1+\alpha)\delta_{y,W}(f,p,x,y,s) - \frac{y}{x}\delta_{x,W},$$

Here, $\delta_{y,N}(f,p,x,y)$ and $\delta_{y,W}(f,p,x,y,s)$ are the outputs of the trade in the each pool.

In our model, trades execute across both pools to maximize the trader utility U^T. Given a distribution on the relative benefit α, the trading volume in either pool depends on the pool's reserve, transaction fee, and slippage tolerance.

4.4 Liquidity Providers

Liquidity providers supply reserves to the two pools. Knowledge of the trader's utility is assumed for liquidity providers. Further, liquidity providers are aware of the behavior of sandwich attackers and price arbitrageurs. We consider the liquidity providers to be rational, i.e., they optimally place their liquidity across the pools such

[4]Traders assume for there to be a sandwich attack for every transaction in Pool_W. As sandwich attacks only execute when they are profitable, there is not always a sandwich attack. However, this is only the case for small transactions (cf. Section 5.1) and unrealistic parameter configurations (cf. Section 6), and it is, therefore, negligible.

that they maximize their received fees. The system has $n \in \mathbb{N}$ liquidity providers. Both the number of liquidity providers and the system's total reserves are fixed during the time of this analysis. A liquidity provider LP_i for $i \in [0, \dots, n-1]$ holds a proportion l_i ($0 < l_i \leq 1$) of the total liquidity $L = \sqrt{x \cdot y}$, where x and y are the system's total reserves. We note that $\sum_{i=0}^{n-1} l_i = 1$.

Liquidity provider LP_i's *strategy space* is given by all possible distributions of their liquidity across both pools: $S_i^{LP} = \{(q_i l_i L, (1-q_i) l_i L) | 0 \leq q_i \leq 1\}$. More precisely, a liquidity provider LP_i can choose the proportion q_i of their liquidity in Pool_N. They automatically place the remaining proportion $1 - q_i$ of their liquidity in Pool_W. Knowing the distribution of the remaining liquidity $(1 - l_i)L$ across the pools and behavior of the other market participants, the liquidity provider chooses the strategy that maximizes the received fees. We define the liquidity provider's utility as the earned fees:

Definition 4.3. The utility $U^{\text{LP}}(f, \alpha, s, x, y, q_i, l_i)$ of liquidity provider LP_i that places $q_i l_i L$ liquidity in Pool_N and $(1 - q_i) l_i L$ in Pool_W represents the fees collected in both pools.

Our game starts with an arbitrary initial liquidity distribution. One after the other, liquidity providers can change their personal liquidity distribution. The system is in a *Nash equilibrium* if no liquidity provider can improve their utility by unilaterally changing their liquidity distribution (strategy). We will loosen the restriction on equilibria and also consider *ε-equilibria*, where a liquidity provider only changes strategy if it increases their utility by a factor greater than $1 + \varepsilon$ ($\varepsilon \geq 0$). We analyze the system with this relaxation on equilibria, as inert liquidity providers are unlikely to change strategies for infinitesimal utility increases due to the effort involved. This adjustment allows us to analyze whether the potential private benefits of liquidity providers suffice.

5 Strategies

The optimal strategies of sandwich attackers and price arbitrageurs are straightforward. Sandwich attackers always execute the largest possible profitable attack, i.e., the attack inferring the maximal acceptable price movement on the trader (cf. Section 5.1) and price arbitrageurs re-balance the market after every trade sequence.

Traders set their trade sizes across both pools optimally to maximize their utility, knowing the pools' liquidity, transaction fee, and the potential presence of sandwich attacks (cf. Section 5.2). Finally, liquidity providers distribute their liquidity to maximize the received fees. Liquidity providers account for the effects altering their liquidity distribution has on the trading volume of the three other market participants (cf. Section 5.3).

5.1 Sandwich Attack Profitability

A sandwich attacker only executes an attack whenever it is profitable, i.e., when U^A is positive (cf. Definition 4.1). We find that the sandwich attackers profit for a front-running transaction of size a_x^{in} can be calculated analytically in Lemma 5.1.

Lemma 5.1. *The sandwich attacker's profit from an attack of size a_x^{in} to the front-running transaction on a victim's transaction $\delta_{x,W}$ can be given analytically.*

Proof. First, the sandwich attacker swaps a_x^{in} and receives

$$a_y = -\int_{(1-p)x}^{(1-p)x+(1-f)a_x^{\text{in}}} \frac{-x \cdot y}{\xi^2} d\xi,$$

in the front-running transaction A_F. Then the trader sells $\delta_{x,W}$ and in return receives

$$\tilde{\delta}_{y,W} = -\int_{(1-p)x+(1-f)a_x^{\text{in}}}^{(1-p)x+(1-f)(a_x^{\text{in}}+\delta_{x,W})} \frac{-x \cdot y}{\xi^2} d\xi.$$

Finally, the sandwich attacker uses a_y Y-tokens to buy a_x^{out} X-tokens in its back-running transaction A^B. Due to the transaction fee f being applied to the input, only $\tilde{a}_y = (1-f)a_y$ of the initially swapped a_y re-enters the

pool. Therefore, we write

$$\tilde{a}_y = \int_{(1-p)x+(1-f)(a_x^{\text{in}}+\delta_{x,W})}^{(1-p)x+(1-f)(a_x^{\text{in}}+\delta_{x,W})-a_x^{\text{out}}} \frac{-x \cdot y}{\xi^2} d\xi,$$

where the sign change in front of the integral is the result of Y-assets being returned to the pool.

The amount of X the attacker holds after the transaction a_x^{out} can be found by equating the two integrals for a_y and $\tilde{a}_y$, using $\tilde{a}_y = (1-f)a_y$, and solving for a_x^{out}. This yields the profit of the sandwich attacker

$$U^A = a_x^{\text{out}} - a_x^{\text{in}} = \frac{(1-f)^2 a_x^{\text{in}}((1-p)x + (1-f)(a_x^{\text{in}} + \delta_{x,W}))^2}{((1-p)x)^2 + (2-f)(1-f)(1-p)x \cdot a_x^{\text{in}} + (1-f)^3 a_x^{\text{in}}(a_x^{\text{in}} + \delta_{x,W})} - a_x^{\text{in}}.$$

□

We will analyze the conditions under which profitable sandwich attacks exist. First, we determine a bound for victim's trade size $\delta_{x,W}$ such that a profitable sandwich attack exists (cf. Lemma 5.2). From Lemma 5.2 we can follow that a profitable attack only exists, if the victim's trade size $\delta_{x,W}$ exceeds a fee dependent threshold $\delta_x^{\min} = f(1-p)x/(1-f)^2$. Hence, only relatively large trades are prone to sandwich attacks.

Lemma 5.2. *A sandwich attack of size a_x^{in} is only profitable if the trader's transaction size exceeds*

$$f((1-p)x + a_x^{in}(1-f))/(1-f)^2.$$

Proof. The expression for $\delta_x^{\min}$ follows from Lemma 5.1 by solving $U^A = 0$. The minimum transaction size for which a profitable sandwich attack exists is obtained by setting $a_x^{\text{in}} = 0$. □

Next, we explore what limits the attacker's maximum profit to show that it is optimal for sandwich attackers to execute the attack with maximal input size, i.e., the attack that infers the maximal acceptable price movement, as dictated by the slippage tolerance of the trader. In Lemma 5.3 we show that the attacker's maximal input a_x^s for which a victim's transaction still executes can be calculated analytically.

Lemma 5.3. *The sandwich attacker's maximal input, a_x^s, for a transaction exchanging $\delta_{x,W}$ X-tokens with slippage tolerance s such that the victim's trade still executes can be given analytically.*

Proof. We consider a sandwich attack with initial input a_x^{in} to the front-running transaction. The output of the victim transaction selling $\delta_{x,W}$ becomes

$$\tilde{\delta}_{y,W} = -\int_{(1-p)x+(1-f)a_x^{\text{in}}}^{(1-p)x+(1-f)(a_x^{\text{in}}+\delta_{x,W})} \frac{-x \cdot y}{\xi^2} d\xi.$$

The victims transaction will, however, only go through, if

$$\tilde{\delta}_{y,W} \geq (1-s)\delta_{y,W}$$

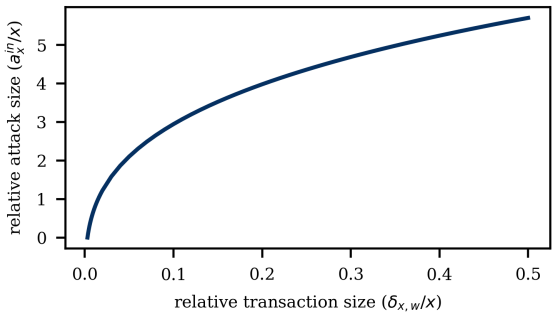
$$-\int_{(1-p)x+(1-f)a_x^{\text{in}}}^{(1-p)x+(1-f)(a_x^{\text{in}}+\delta_{x,W})} \frac{-x \cdot y}{\xi^2} d\xi \geq (1-s)\left(-\int_{(1-p)x}^{x+(1-f)\delta_{x,W}} \frac{-x \cdot y}{\xi^2} d\xi\right).$$

Thus, the attacker's maximal input a_x^s increases the slippage incurred by the victim to their tolerance, i.e., $\tilde{\delta}_{y,W} = (1-s)\delta_{y,W}$. Solving for a_x^s, we find that the maximal input is

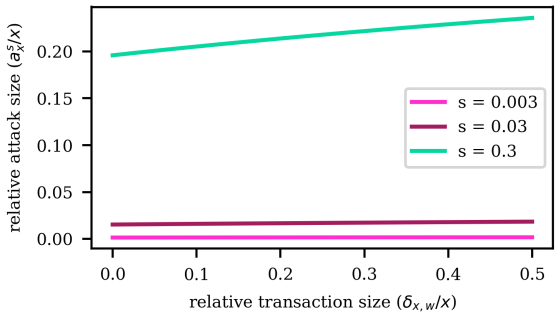
$$a_x^s = \frac{1}{2}\left(\frac{\sqrt{\delta_{x,W}^2(1-f)^2 + \frac{4(1-p)x((1-p)x+\delta_{x,W}(1-f))}{1-s}} - 2(1-p)x}{1-f} - \delta_{x,W}\right).$$

□

However, for very large slippage tolerances the size of the sandwich attack is limited. To see this we can consider the asymptotic behaviour of Lemma 5.1 in the limit of very large attack sizes a_x^{in}: $\lim_{a_x^{\text{in}} \to \infty} U^A =$



(a) Attack size a_x^{in} achieving max profit U^A vs. the victim's trade size. a_x^{in} is found by maximizing the attacker's profit w.r.t. a_x^{in}.



(b) Maximum sandwich attack size dependent on the victim's transaction size and slippage tolerance.

Fig. 2. Limits on the sandwich attack size in terms of profitability (left) and slippage tolerance (right) for $f = 0.3\%$. Note the vast difference in scale of the vertical axis, demonstrating that the attack is limited by the slippage tolerance.

$\lim_{a_x^{\text{in}} \to \infty} -f a_x^{\text{in}} \to -\infty$. We, thus, analyze whether the slippage tolerance or profitability limits the sandwich attack size and plot the sandwich attack size that achieves the maximum profit U^A as a function of the victim's transaction size $\delta_{x,W}$ in Figure 2a. Figure 2b shows the sandwich attacker's maximal input a_x^s as a function of the the victim's transaction size $\delta_{x,W}$ for different slippage tolerances. Note the vast difference in scale, demonstrating that the sandwich attack size is clearly limited by the slippage tolerance. Thus, in realistic market configurations, the sandwich attackers always execute the attack with maximal possible input size a_x^s.

5.2 Trade Sizes

Traders wish to maximize their utility $U^T(\delta_{x,N}, \delta_{x,W}, \alpha, f, s, p, x, y)$ (cf. Definition 4.2), i.e., the difference between the benefit from receiving Y-tokens and the trade's costs. The utility function accounts for sandwich attacks in Pool_W and assumes that the transaction output is reduced by the slippage tolerance. We show in Lemma 5.4 that the optimal transaction size, maximizing utility U^T, in Pool_N ($\delta_{x,N}^{\text{opt}}$) and Pool_W ($\delta_{x,W}^{\text{opt}}$) can be expressed analytically. Observe that the transaction size is proportional to the pool's reserves of X-token. Further, we can see that the effects of the slippage tolerance on the trade size are identical to those of the transaction fee. Thus, the combination of transaction fee f and slippage tolerance s in Pool_W is from the trader's perspective equivalent to a larger transaction fee equaling $f + s - f \cdot s$ in Pool_N. Therefore, the transaction size in Pool_N is always larger than in Pool_W, and we follow that the trader's utility is maximized for $p = 1$. We also note that the optimal transaction size increases with α and decreases with the transaction fee f, as well as, where applicable, the slippage tolerance s.

Lemma 5.4. *A trade of size* $\delta_{x,N}^{opt} = \max(0, p \cdot x(\sqrt{(1+\alpha)(1-f)} - 1))$ *maximizes a trader's utility* U^T *in* Pool_N *and in* Pool_W *the optimum is at* $\delta_{x,W}^{opt} = \max(0, (1-p)x(\sqrt{(1+\alpha)(1-s)(1-f)} - 1))$.

Proof. Without loss of generality, we maximize the trader's utility in each pool independently and start with Pool_N. The trader's utility in Pool_N is given by

$$U_N^T = (1+\alpha)\frac{(1-f)\delta_{x,N} p \cdot y}{(1-f)\delta_{x,N} + p \cdot x} - \frac{y}{x}\delta_{x,N}.$$

We differentiate the trader's utility in Pool_N, U_N^T, with respect to the transaction size $\delta_{x,N}$ to find the transaction size $\delta_{x,N}^{\text{opt}}$ maximizing the trader's utility. We obtain

$$\partial_{\delta_{x,N}} U_N^T = \frac{(1+\alpha)(1-f)p^2 \cdot x \cdot y}{(\delta_{x,N}(1-f) + p \cdot x)^2} - \frac{y}{x},$$

and the two zero crossing of $\partial_{\delta_{x,N}} U_N^T$ are:

$$p \cdot x(\pm\sqrt{(1+\alpha)(1-f)} - 1).$$

For our parameters, $x, y, \alpha > 0$, $0 < f < 1$, $0 \leq p \leq 1$, the second derivative, $\partial^2_{\delta_{x,N}} U_N^T$, is only negative for the following zero crossing

$$\delta_{x,N}^{\max} = p \cdot x(\sqrt{(1+\alpha)(1-f)} - 1).$$

Thereby, $\delta_{x,N}^{\max}$ maximizes the traders utility. The trader optimally sells $\delta_{x,N}^{\text{opt}} = \max(0, \delta_{x,N}^{\max})$ in Pool_N.

We proceed analogously as above for Pool_W and find the trader the optimally places $\delta_{x,W}^{\text{opt}} = \max(0, \delta_{x,W}^{\max})$ in Pool_W. In the previous,

$$\delta_{x,W}^{\max} = (1-p)x(\sqrt{(1+\alpha)(1-s)(1-f)} - 1).$$

The trade inputs to maximize the trader's utility can, thus, be determined analytically and are given by $\delta_{x,N}^{\text{opt}} = \max(0, \delta_{x,N}^{\max})$ and $\delta_{x,W}^{\text{opt}} = \max(0, \delta_{x,W}^{\max})$. □

With the help of Lemma 5.4, we can obtain bounds for relative benefit α, such that traders benefit from trading in Pool_N and Pool_W. A trader executes a trade in Pool_N, as long as their α exceeds $\alpha > \alpha_N^{\min} = f/(1-f)$. Notice that this bound only depends on the transaction fee f. In Pool_W, a trader will only execute a trade if $\alpha > \alpha_W^{\min} 0 = f + s - s \cdot f/((1-f)(1-s))$.

To summarise, traders can analytically determine the transaction size that optimizes their utility given the pools' parameters. As expected, the traders' utility is maximized when all liquidity is in Pool_N. In the following section, we will investigate how liquidity providers distribute their liquidity across the two pools', knowing that traders execute optimal transactions.

5.3 Liquidity Distribution

A liquidity provider's utility directly corresponds to the received fees (cf. Definition 4.3). We, therefore, first quantify the system's total fees given traders with relative benefit α. Note that the total fees include the fees received from traders' transactions, sandwich attacks (whenever applicable), and price arbitrage.

In Lemma 5.5 we find that the total fees are proportional to p. If the fee gradient with respect to p is zero, all liquidity distributions maximize the game's fees. Otherwise, the game's fees are maximized, either when all liquidity is in Pool_N ($p = 1$) or when all liquidity is in Pool_W ($p = 0$).

Lemma 5.5. *The total transaction fees $F(f, \alpha, s, y, p)$ collected across both pools for a traders with relative benefit α are proportional to p.*

Proof. We consider four intervals in the following:

$0 < \alpha \leq \frac{f}{(1-f)}$, $\frac{f}{(1-f)} < \alpha < \frac{f+s-s\cdot f}{(1-f)(1-s)}$, $\alpha > \frac{f+s-s\cdot f}{(1-f)(1-s)}$, and $U^A \leq 0$, $\alpha > \frac{f+s-s\cdot f}{(1-f)(1-s)}$ and $U^A \geq 0$,

where $U^A(\delta_{x,W}, f, s, p, x, y)$ is the sandwich attacks' profitability (cf. Definition 4.1).

Following from Lemma 5.4, we conclude that there are no trades executed in either pool and, thereby, no fees collected in either pool for $\alpha \leq \frac{f}{(1-f)}$.

We continue with the second interval, e.g., $\frac{f}{(1-f)} < \alpha \leq \frac{f+s-s\cdot f}{(1-f)(1-s)}$. Following from Lemma 5.4 traders exclusively execute trades in Pool_N on this interval. The fees collected in Pool_N for a transaction by a trader with relative benefit α are

$$\begin{aligned} F_N(f,\alpha,s,y,p) =& f\left(\delta_{x,N}^{\text{opt}}\cdot\frac{y}{x}+\frac{p\cdot y(1-f)\delta_{x,N}^{\text{opt}}}{p\cdot x+(1-f)\delta_{x,N}^{\text{opt}}}\right)\\ =& p\cdot y\cdot f\left(1+\frac{1}{1+(\sqrt{(1+\alpha)(1-f)}-1)(1-f)}\right)(\sqrt{(1+\alpha)(1-f)}-1)\end{aligned}$$

Y-tokens. In the previous, $\delta_{x,N}^{\text{opt}}\cdot\frac{y}{x}$ is the trader's transaction size in Y-tokens in Pool_N and $\frac{p\cdot y\delta_{x,N}^{\text{opt}}}{p\cdot x+(1-f)\delta_{x,N}^{\text{opt}}}$ is the size of the price arbitrageur's transaction.

In the third interval, e.g., $\alpha > \frac{f+s-s\cdot f}{(1-f)(1-s)}$ and $U^A \leq 0$, traders execute trades in both pools (cf. Lemma 5.4). However, there is no profitable sandwich attack, due to the small trade size in Pool_W (cf. Lemma 5.2). The fees collected in Pool_N are again given by $F_N(f,\alpha,s,y,p)$, but liquidity providers collect additional fees in Pool_W. The fees collected in Pool_W for a transaction by a trader with relative benefit α are given by

$$\begin{aligned} F_W^{U^A\leq 0}(f,\alpha,s,y,p) &= f\left(\delta_{x,W}^{\text{opt}}\cdot\frac{y}{x}+\frac{(1-p)y(1-f)\delta_{x,W}^{\text{opt}}}{(1-p)x+(1-f)\delta_{x,W}^{\text{opt}}}\right)\\ &= (1-p)y\cdot f\left(1+\frac{1}{1+(\sqrt{(1+\alpha)(1-f)(1-s)}-1)(1-f)}\right)(\sqrt{(1+\alpha)(1-f)(1-s)}-1)\end{aligned}$$

where

$$\delta_{x,W}^{\text{opt}}\cdot\frac{y}{x}$$

is the trader's transaction size in Y-tokens in Pool_W and

$$\frac{p\cdot y\cdot\delta_{x,W}^{\text{opt}}}{p\cdot x+(1-f)\delta_{x,W}^{\text{opt}}}$$

is the size of the price arbitrageur's transaction that returns the pools to its initial state.

Finally, we analyze the fourth interval, e.g., $\alpha > \frac{f+s-s\cdot f}{(1-f)(1-s)}$ and $U^A > 0$. In this interval trades execute in both pools and sandwich attacks execute in Pool_W. Thus, in addition to the fees $F_N(f,\alpha,s,y,p)$ collected in Pool_N, we also consider the fees collected in Pool_W from traders, price arbitrageurs and sandwich attackers for the liquidity provider utility. In the presence of sandwich attacks, the fees in Pool_W are given by

$$\begin{aligned} F_W^{U^A>0}(f,\alpha,s,y,p) =& \left(\left(\delta_{x,W}^{\text{opt}}+a_x^s\right)\frac{y}{x}+\frac{(1-p)y(1-f)\left(\delta_{x,W}^{\text{opt}}+a_x^s\right)}{(1-p)x+(1-f)\left(\delta_{x,W}^{\text{opt}}+a_x^s\right)}\right)\\ =& \frac{(1-p)y\cdot f}{2(1-f)\sqrt{1-s}}\left(\frac{4(1-s)}{n_2(f,\alpha,s)+(1-f)n_1(f,\alpha,s)\sqrt{1-s}}-n_2(f,\alpha,s)+(1-f)n_1(f,\alpha,s)\sqrt{1-s}\right)\end{aligned}$$

where $\left(\delta_{x,W}^{\text{opt}}+a_x^s\right)\frac{y}{x}$ combines the trader's transaction size in Pool_W and the bot's front-running transaction size in Y-tokens (cf. Lemma 5.3). In the previous,

$$n_1(f,\alpha,s)=\sqrt{(1+\alpha)(1-s)(1-f)}-1$$

and

$$n_2(f,\alpha,s) = \sqrt{4 + 4(1-f)n_1(f,\alpha,s) + (1-f)^2 n_1^2(f,\alpha,s)(1-s)}.$$

Further,

$$\frac{(1-p)y(1-f)\left(\delta_{x,W}^{\text{opt}} + a_x^s\right)}{\left((1-p)x + (1-f)\left(\delta_{x,W}^{\text{opt}} + a_x^s\right)\right)}$$

is the combined size of the attacker's back-running transaction and the transaction that returns the pool to its initial state. Through a combination, we obtain that the fees collected across both pools are given by

$$F(f,\alpha,s,y,p) = \begin{cases} 0 & 0 < \alpha \leq \frac{f}{(1-f)} \\ F_N(f,\alpha,s,y,p) & \frac{f}{(1-f)} < \alpha \leq \frac{f+s-s\cdot f}{(1-f)(1-s)} \\ F_N(f,\alpha,s,y,p) + F_W^{U^A \leq 0}(f,\alpha,s,y,p) & \alpha \geq \frac{f+s-s\cdot f}{(1-f)(1-s)} \text{ and } U^A \leq 0 \\ F_N(f,\alpha,s,y,p) + F_W^{U^A > 0}(f,\alpha,s,y,p) & \alpha \geq \frac{f+s-s\cdot f}{(1-f)(1-s)} \text{ and } U^A > 0 \end{cases}$$

Thus, we can conclude that that F is proportional to p for every $\alpha > 0$. □

Note that Lemma 5.5 not only holds for a trade order from a single trader with relative benefit α and the associated trade orders from sandwich attackers and arbitrageurs. It holds for an infinite sequence of trade orders with the same α along with the associated orders from sandwich attackers and arbitrageurs. The previous follows from price arbitrageurs returning the pool to its initial price $P_{X\to Y}$ after every trade sequence. We conclude that the total fees collected for a continuous stream of trade orders originating from a homogeneous set of traders with the same relative benefit α is proportional to p.

Lemma 5.5 gives the system's total fees $F(f,\alpha,s,y,p)$. By virtue of the proportionality of the total fees F in both p and y, the fees received by an individual liquidity provider LP_i with liquidity $(q_i l_i L, (1-q_i)l_i L)$ are given by $F_i(f,\alpha,s,y,q_i,l_i) = l_i \cdot F(f,\alpha,s,y,q_i)$. Therefore, the optimal liquidity distribution for an individual liquidity provider also has all liquidity in Pool_N ($q_i = 1$) or all liquidity in Pool_W ($q_i = 0$), whenever the gradient of the fee with respect to q_i is non-zero. A liquidity provider will redistribute their liquidity optimally, i.e., such that their utility is maximized, whenever they can increase their received fees by more than a factor of $1+\varepsilon$.

6 Game Equilibria

Before discussing the quantitative model, we will give an intuitive explanation. First, we note that liquidity providers profit from large trading volumes, irrespective of their origin. As the sandwich attackers extract their profits from the traders, ordinary traders will reduce their trading volume if the attacks become too lucrative. Therefore, whether a pool with sandwich attacks is the equilibrium boils down to whether the increased trading volume the attackers generate can offset the diminished trading activity of regular traders. The regime where the equilibrium lies in the pool with attackers is, thus, characterized by large sandwich attacks but with a very small resulting profit for the attackers.

We will now substantiate this qualitative picture by locating the game's ε-equilibria to identify which pool is favored by the market actors. A liquidity distribution is an ε-equilibria if no liquidity provider can increase their utility by more than a factor of $1+\varepsilon$ by adjusting the liquidity distribution. For $\varepsilon = 0$, any ε-equilibrium is considered a Nash equilibrium. We analyze the game's equilibria assuming a fixed (mean) benefit for the traders, α, in Section 6.1 and discuss the heterogeneous case in Section 6.2.

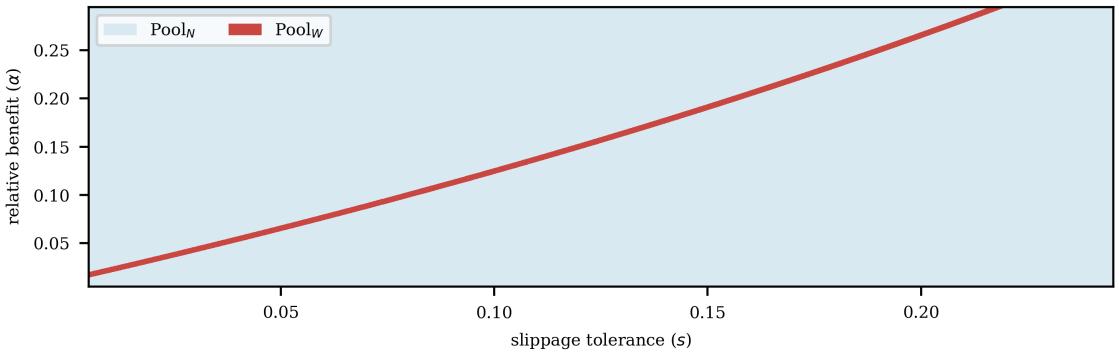


Fig. 3. The Nash equilibrium dependent on the slippage tolerance and the relative benefit. Almost everywhere, i.e., in blue areas, the Nash equilibrium is Pool$_N$, and the game's Nash equilibrium is the social optimum. Only on the red area, the Nash equilibrium is Pool$_W$. We note that the vertical cross-section of the red area is less than $0.01\Delta\alpha$. We set $x = 5'000'000$ X, $y = 5'000'000$ Y and $f = 0.003$.

6.1 Homogeneous Traders

We start by analyzing the game's equilibria given a homogeneous trader set, i.e., all traders have the same relative benefit α. In the simplest case, $\partial_p F = 0$, all liquidity distributions are both ε-equilibria and Nash equilibria (cf. Lemma 6.1).

Lemma 6.1. *The only Nash equilibria if $\partial_p F \neq 0$ are $p \in \{0, 1\}$. If $\partial_p F = 0$, all liquidity distributions are ε-equilibria in a homogeneous traders game.*

Proof. If the fees gradient is non-zero ($\partial_p F \neq 0$), the maxima is located at either corner point of the interval, as fees are proportional to p (cf. Lemma 5.5), and the fees gradient is non-zero, the maxima is located at either corner point of the interval. Else, if the fees gradient $\partial_p F$ is zero, the fees across the entire interval are constant. Therefore, all liquidity distributions are ε-equilibria. □

In Lemma 6.1 we further show that for $\partial_p F \neq 0$ the only possible Nash equilibria are the two corner cases: all liquidity in Pool$_N$ ($p = 1$) or in Pool$_W$ ($p = 0$). This follows from the proportionality of the fees to p which means that the sign of $\partial_p F$ dictates the location of the Nash equilibrium. In Figure 3, we visualize the dependence of this equilibrium on the slippage tolerance and relative benefit.

We notice that in areas where either the trader's relative benefit α or the slippage tolerance is high, Pool$_N$ is the Nash equilibrium. When α is comparatively large, so is the traders' transaction size. Liquidity providers, therefore, receive a substantial amount of fees from ordinary traders and sandwich attacks would decrease the pool's trading volume more than the volume created by the attacker. Thus, all liquidity is in Pool$_N$. The same holds when the slippage tolerance is high compared to the trader's benefit. Trades no longer execute in Pool$_W$ (cf. Lemma 5.4) or their size in Pool$_W$ is too small for there to be a profitable sandwich attack (cf. Lemma 5.2). Thus, there is basically no volume in Pool$_W$.

There is a small area in between where Pool$_W$ is the equilibrium. Here, the slippage tolerance is just small enough not to exceed the bound given in Lemma 5.4 and the trader's transaction size in Pool$_W$ is just large enough to allow for a profitable attack (cf. Lemma 5.2). Here, liquidity providers' private incentives are maximized in the presence of sandwich attackers.

While the sign of the gradient $\partial_p F$ dictates the location of the Nash equilibrium, it is not sufficient to determine if it is an ε-equilibrium. As we show in Theorem 6.2, it is the relative difference between the fees the liquidity

provider earns with their current distribution and the maximum fees they can collect that dictate whether the liquidity provider will change strategy.

Theorem 6.2. *A liquidity distribution is an ε-equilibrium if there is no liquidity provider with initial liquidity distribution $(q_i l_i L, (1-q_i) l_i L)$, such that*

$$\frac{\max\{F(f,\alpha,s,y,0), F(f,\alpha,s,y,1)\} - F(f,\alpha,s,y,q_i)}{F(f,\alpha,s,y,q_i)} < \varepsilon.$$

Proof. For a liquidity distribution to qualify as an ε-equilibrium, no liquidity provider must see a possibility to increase their expected fees by more than than a factor $1+\varepsilon$ through adjusting their liquidity distribution. Further, we know from Lemma 6.1 that a liquidity provider receives the most fees either when all their liquidity is in Pool_N or all their liquidity is in Pool_W. Thus, the maximum relative increase to an LP's fees, with current liquidity distribution $(q_i l_i L, (1-q_i) l_i L)$, is given as

$$\frac{\max\{F(f,\alpha,s,y,0), F(f,\alpha,s,y,1)\} - F(f,\alpha,s,y,q_i)}{F(f,\alpha,s,y,q_i)}.$$

In case the previous fraction does not exceed ε for any liquidity provider, the current distribution is a Nash equilibrium. □

Currently, all liquidity is in markets that allow for sandwich attacks. Therefore, we are especially interested in identifying market configurations that are ε-equilibria for all initial liquidity distributions, as in this situation, a new market with a front-running protection mechanism would not attract any liquidity even if it were to maximize the liquidity provider's private incentives. The maximum relative change in fees for a given market configuration is given as $|\Delta_F|$, where $\Delta_F = \partial_p F / F_{\min}$ and $F_{\min} = \min(F(p=0), F(p=1))$. Independent of a liquidity provider's initial distribution, the relative benefit of switching strategy cannot exceed ε in case $|\Delta_F| < \varepsilon$. The sign of Δ_F corresponds to the sign of the fee's gradient $\partial_p F$ and therefore indicates the position of the Nash equilibrium.

We simulate the dependence of Δ_F on the slippage tolerance and relative benefit in Figure 4. For comparatively large slippage tolerances, the magnitude of Δ_F is large. Independent of the liquidity in Pool_W, the trading volume is either zero when $\alpha < \alpha_W^{\min}$ or relatively small when sandwich attacks are not profitable, i.e., $U^A < 0$. Therefore, switching strategies by moving liquidity from Pool_W to Pool_N leads to a sizable increase in fees in this part of the parameter space, and we do not expect any ε-equilibria in Pool_W for this parameter range.

Turning to more realistic areas of the parameter space ($U^A > 0$), we notice that Δ_F's magnitude is small. The dark blue line in Figure 4 shows where $\Delta_F = 0.01$, indicating that all initial liquidity distributions are ε-equilibria for $\varepsilon = 0.01$ below this line for $U^A > 0$. Thus, all liquidity providers who only change strategies for a relative benefit larger than 1% would not be inclined to move their liquidity. We follow that any liquidity distribution is an ε-equilibrium for a significant proportion of the parameter space even for small ε. Therefore, liquidity providers are largely indifferent to whether the market utilizes a front-running protection mechanism and might require additional financial incentives to migrate their liquidity to pools with front-running protection mechanisms.

In Appendix A, we further simulate and discuss a liquidity provider's maximal utility increase obtained by shifting from their initial liquidity distribution to the optimum and quantify the minimum ε such that they move their reserves.

6.2 Heterogeneous Traders

In this section, we show where the ε-equilibria fall in a game with heterogeneous traders. We model a trader's relative benefit α as a random variable A with probability mass function $\psi_A(\alpha)$. The game is in an ε-equilibrium for any probability mass function $\psi_A(\alpha)$ that fulfills the condition provided in Theorem 6.3. Theorem 6.3 assumes that the random variable A is discrete. Note, however, that it could be adapted to the continuous case.

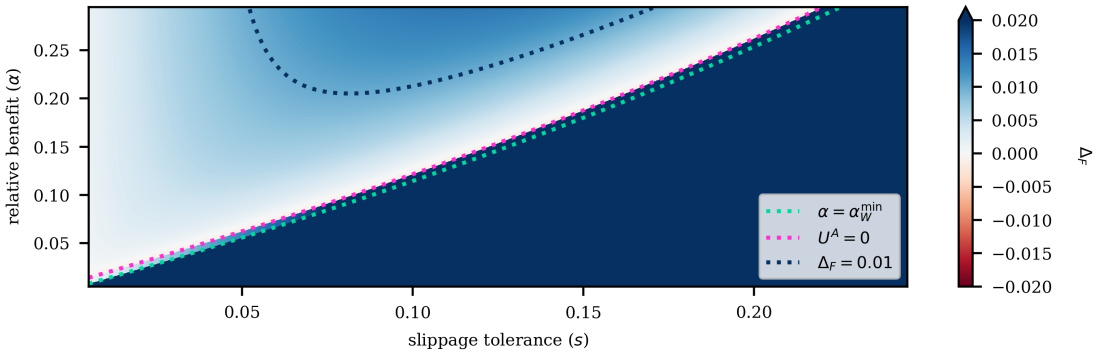


Fig. 4. Simulation of Δ_F across both pools depending on the trader's relative benefit and the slippage tolerance. In blue areas the Nash equilibrium is Pool_N, in red areas it is Pool_W. Δ_F is cut off at 0.02 for better visibility. Below the dotted cyan line no trading takes place in Pool_W, and below the dotted magenta line sandwich attacks are not profitable. We set $x = 5'000'000$ X, $y = 5'000'000$ Y and $f = 0.003$.

Theorem 6.3. *A liquidity distribution in a system with heterogeneous traders with distribution $\psi_A(\alpha)$ is an ε-equilibrium if there is no liquidity provider with initial liquidity distribution $(q_i l_i L, (1-q_i) l_i L)$, such that*

$$\frac{\max\left\{\sum_\alpha \psi_A(\alpha) F(f,\alpha,s,y,0), \sum_\alpha \psi_A(\alpha) F(f,\alpha,s,y,1)\right\} - \sum_\alpha \psi_A(\alpha) F\left(f,\alpha,s,y,q_i\right)}{\sum_\alpha \psi_A(\alpha) F\left(f,\alpha,s,y,q_i\right)} < \varepsilon.$$

Proof. We proceed similarly to Theorem 6.2 and note that as long as no liquidity provider must see a possibility to increase their expected fees by more than than a factor $1+\varepsilon$ through adjusting their liquidity distribution, the configuration is a Nash equilibrium. The fees received by a liquidity provider LP_i are given by

$$\sum_\alpha \psi_A(\alpha) F\left(f,\alpha,s,y,q_i\right),$$

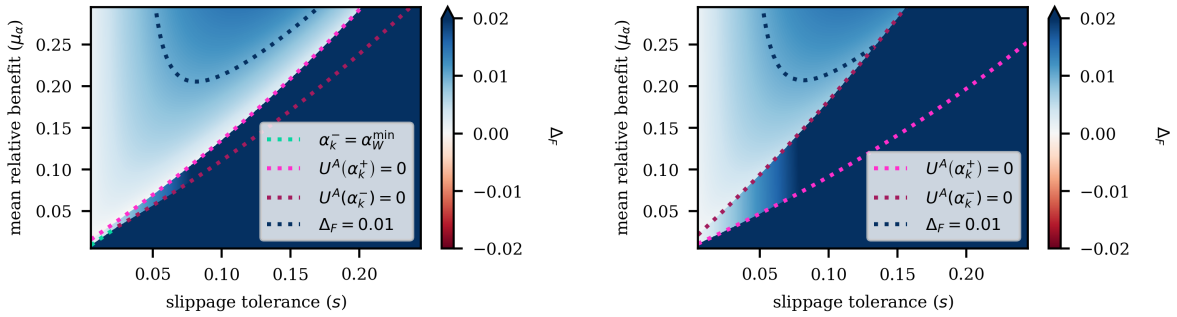
where $F\left(f,\alpha,s,y,q_i\right)$ is given by Lemma 5.5. We, thus, follow that the fees received by liquidity provider LP_i are also proportional to q_i in the heterogeneous case. Further, it holds that liquidity provider receives the most fees either when all their liquidity is in Pool_N, all their liquidity is in Pool_W or the fees are constant for all liquidity distribution. Thus, the maximum relative increase to an LP's fees, with current liquidity distribution $(q_i l_i L, (1-q_i) l_i L)$, is given as

$$\frac{\max\left\{\sum_\alpha \psi_A(\alpha) F(f,\alpha,s,y,0), \sum_\alpha \psi_A(\alpha) F(f,\alpha,s,y,1)\right\} - \sum_\alpha \psi_A(\alpha) F\left(f,\alpha,s,y,q_i\right)}{\sum_\alpha \psi_A(\alpha) F\left(f,\alpha,s,y,q_i\right)} < \varepsilon.$$

In case the previous fraction does not exceed ε for any liquidity provider, the current distribution is a Nash equilibrium. □

Further, we predict that for most probability distributions, extreme values of the system's fees gradient $\partial_p F$ will be averaged out. We, therefore, presume that Pool_W will present a Nash equilibrium for even fewer market configurations and that a large proportion of the game's parameter space is an ε-equilibrium for small ε's. To back up this assumption, we simulate the Δ_F for a two-point distribution with the following probability mass function:

$$\psi_A(\alpha) = \begin{cases} \frac{1}{2} & \text{if } \alpha = \alpha_k^- = \left(1 - \frac{1}{k}\right)\mu_\alpha, \\ \frac{1}{2} & \text{if } \alpha = \alpha_k^+ = \left(1 + \frac{1}{k}\right)\mu_\alpha, \end{cases}$$



(a) We set $k = 10$.

(b) We set $k = 3$.

Fig. 5. Visualization of Δ_F across both pools for a heterogeneous trader distribution $\psi_A(\alpha)$ depending on mean relative benefit and the slippage tolerance. In blue areas the Nash equilibrium is Pool_N, in red areas it is Pool_W, and in the white area in-between all liquidity distributions are Nash equilibria. Δ_F is cut off at 0.02 for better visibility and the dotted dark blue line visualizes where $\Delta_F = 0.01$. Where relevant we show the boundaries profitability boundaries for traders and sandwich attackers. The dotted cyan line indicates where $\alpha_k^- = \alpha_W^{\min}$, i.e, no trading of traders with the respective relative benefit takes place in Pool_W below the line. The dotted magenta line shows where $U^A(\alpha_k^+) = 0$ and the dotted purple line shows where $U^A(\alpha_k^-) = 0$, i.e, sandwich attacks are not profitable on trades with the respective α below these lines. We set $x = 5'000'000$ X, $y = 5'000'000$ Y and $f = 0.003$.

in Figure 5 for $k = 10$ (cf. Figure 5a) and $k = 3$ (cf. Figure 5b). As expected Δ_F resembles the homogeneous case more closely, when the two points of the distribution are close to each other, i.e., for the higher values of k. Note that for $k = \infty$ the two-point distribution becomes a one-point distribution, i.e., the homogeneous case.

We further notice that a more significant area of the parameter space has Pool_N as the Nash equilibrium when the slippage tolerance is large, i.e., for $U^A(\alpha_k^-) < 0$. Half the traders have a lower relative benefit than μ_α and, thereby, these traders will only execute transactions in Pool_W for smaller slippage tolerances. We further note that the area of the parameter space, where Pool_W is the Nash equilibrium grows smaller as k decreases. While there remains a small area of the parameter space that has Pool_W as the Nash equilibrium for for $k = 10$ (cf. Figure 5a), this completely disappears for $k = 3$ (cf. Figure 5b). The particular combination of requirements that must be met for Pool_W to be the Nash equilibrium is never achieved as the distance between the two points of the distribution grows. However, while we expect that for most heterogeneous games Pool_W will never be a Nash equilibrium, it remains an ε-equilibria even for small ε whenever the mean relative benefit μ_α is comparatively large, i.e., for $U^A(\alpha_k^-) > 0$.

7 Social Incentives and Self-Regulation

In our model, whenever the derivative of the liquidity provider's utility is positive, i.e., $\partial_p U^{LP} = \partial_p F > 0$, the system's Nash equilibrium maximizes social welfare. Our analysis demonstrates that Pool_N is the Nash equilibrium for the vast majority of the parameter space (cf. Figure 3). Only in a small area, Pool_W is the Nash equilibrium. We, thus, conclude that for $\varepsilon = 0$, social welfare is maximized in most market configurations. Therefore, markets preventing front-running generally align the private incentives of liquidity providers with the system's incentives.

However, liquidity providers are currently in markets without front-running protections. Thus, an innovative DEX preventing front-running attacks must attract liquidity from other markets. Our analysis highlights that even though placing all liquidity in Pool_N maximizes the private incentives of liquidity providers for the majority of market configurations, the benefit from adjusting a liquidity distribution is often only small. The market, therefore,

cannot rely on the inert liquidity providers to revise their liquidity distribution. Therefore, added incentives may be required for the successful adoption of a such market. A new DEX, implementing a front-running prevention scheme, could, for example, distribute its native token to liquidity providers as an added incentive. Similar benefits have been distributed at the launch of new DeFi platforms [11].

8 Conclusion

Our game-theoretical study of the incentives of traders and liquidity providers to adopt a DEX with a new market design preventing front-running attacks shows that generally, the private incentives of both traders and liquidity providers align the market's social incentives — eliminating front-running attacks. In the absence of a central authority, market participants must experience a personal benefit for the successful adoption of such a design. Therefore, this alignment of the liquidity provider's private incentives and the market's social incentives is promising.

However, our analysis also finds that generally, the increase in the liquidity provider's utility from moving to a market preventing front-running is small. Liquidity providers are usually not nimble market participants, and, therefore, the prospect of a small utility increase might not suffice. Successful self-regulation of the market to prevent front-running attacks is likely to require additional initial financial incentives to gain the attention of liquidity providers.

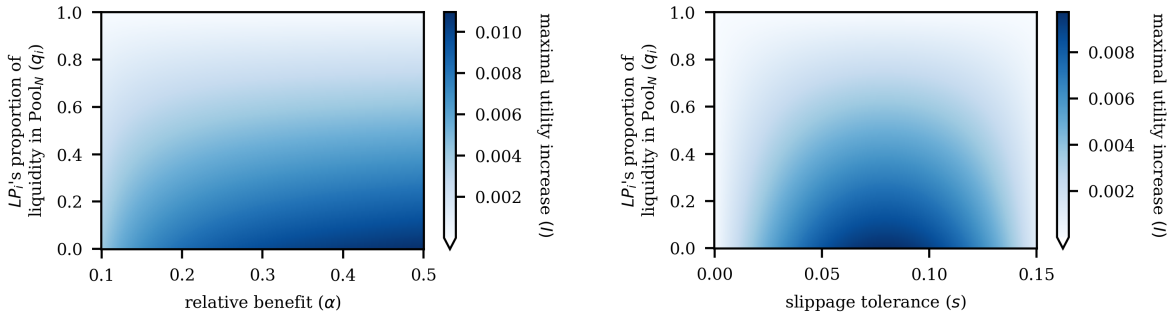
References

[1] Automata network. https://www.ata.network/ (2023)
[2] Cowswap. https://cowswap.exchange/ (2023)
[3] Curve. https://curve.fi/ (2023)
[4] Dexs volume. https://defillama.com/dexs (2023)
[5] Eden. https://www.edennetwork.io/ (2023)
[6] flashbots. https://docs.flashbots.net/ (2023)
[7] Gnosis protocol. https://gnosis.io/ (2023)
[8] Openmev. https://openmev.xyz/ (2023)
[9] Sandwich overview. https://eigenphi.io/mev/ethereum/sandwich (2023)
[10] Secretswap. https://secretswap.net/ (2023)
[11] Sushiswap. https://sushi.com/ (2023)
[12] Uniswap. https://uniswap.org/ (2023)
[13] Adams, H., Zinsmeister, N., Robinson, D.: Uniswap v2 core (2020)
[14] Adams, H., Zinsmeister, N., Salem, M., Keefer, R., Robinson, D.: Uniswap v3 core. Tech. rep., Uniswap (2021)
[15] Asayag, A., Cohen, G., Grayevsky, I., Leshkowitz, M., Rottenstreich, O., Tamari, R., Yakira, D.: A fair consensus protocol for transaction ordering. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP). pp. 55–65 (2018). https://doi.org/10.1109/ICNP.2018.00016
[16] Baird, L.: The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance. Swirlds Tech Reports SWIRLDS-TR-2016-01, Tech. Rep (2016)
[17] Bentov, I., Ji, Y., Zhang, F., Breidenbach, L., Daian, P., Juels, A.: Tesseract: Real-time cryptocurrency exchange using trusted hardware. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. p. 1521–1538. CCS '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3319535.3363221, https://doi.org/10.1145/3319535.3363221
[18] Bernhardt, D., Taub, B.: Front-running dynamics. Journal of Economic Theory **138**(1), 288–296 (2008)
[19] Breidenbach, L., Daian, P., Tramèr, F., Juels, A.: Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In: 27th {USENIX} Security Symposium ({USENIX} Security 18). pp. 1335–1352 (2018)
[20] Budish, E., Lee, R.S., Shim, J.J.: A Theory of Stock Exchange Competition and Innovation: Will the Market Fix the Market? NBER Working Papers 25855, National Bureau of Economic Research, Inc (2019)
[21] Cachin, C., Mićić, J., Steinhauer, N.: Quick order fairness. In: Financial Cryptography and Data Security (FC), Grenada (2022)
[22] Comerton-Forde, C., Tang, K.M.: Anonymity, frontrunning and market integrity. The Journal of Trading **2**(4), 101–118 (2007)
[23] Constantinescu, A., Ghinea, D., Heimbach, L., Wang, Z., Wattenhofer, R.: A fair and resilient decentralized clock network for transaction ordering. arXiv preprint arXiv:2305.05206 (2023)

[24] Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 910–927. IEEE (2020)
[25] Danthine, J.P., Moresi, S.: Front-Running by Mutual Fund Managers: A Mixed Bag. Review of Finance **2**(1), 29–56 (1998)
[26] Doweck, Y., Eyal, I.: Multi-party timed commitments. arXiv preprint arXiv:2005.04883 (2020)
[27] Eskandari, S., Moosavi, M., Clark, J.: Sok: Transparent dishonesty: front-running attacks on blockchain. In: Financial Cryptography and Data Security (FC), St. Kitts, Saint Kitts and Nevis (February 2019)
[28] Heimbach, L., Wattenhofer, R.: Eliminating sandwich attacks with the help of game theory (2022)
[29] Heimbach, L., Wattenhofer, R.: SoK: Preventing Transaction Reordering Manipulations in Decentralized Finance. In: 4th ACM Conference on Advances in Financial Technologies (AFT), Cambridge, Massachusetts, USA (September 2022)
[30] Kelkar, M., Deb, S., Kannan, S.: Order-fair consensus in the permissionless setting. IACR Cryptol. ePrint Arch. **2021**, 139 (2021)
[31] Kelkar, M., Deb, S., Long, S., Juels, A., Kannan, S.: Themis: Fast, strong order-fairness in byzantine consensus. Cryptology ePrint Archive, Report 2021/1465 (2021), https://ia.cr/2021/1465
[32] Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for byzantine consensus. In: Annual International Cryptology Conference. pp. 451–480. Springer (2020)
[33] Kursawe, K.: Wendy, the good little fairness widget: Achieving order fairness for blockchains. In: Proceedings of the 2nd ACM Conference on Advances in Financial Technologies. pp. 25–36 (2020)
[34] Manahov, V.: Front-running scalping strategies and market manipulation: Why does high-frequency trading need stricter regulation? Financial Review **51**(3), 363–402 (2016)
[35] Markham, J.W.: Front-running - insider trading under the commodity exchange act. Catholic University Law Review **38**, 69 (1988-1989)
[36] Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. p. 31–42. CCS '16, Association for Computing Machinery, New York, NY, USA (2016)
[37] Momeni, P., Gorbunov, S., Zhang, B.: Fairblock: Preventing blockchain front-running with minimal overheads. In: Security and Privacy in Communication Networks: 18th EAI International Conference, SecureComm 2022, Virtual Event, October 2022, Proceedings. pp. 250–271. Springer (2023)
[38] Moosa, I.: The regulation of high-frequency trading: A pragmatic view. Journal of Banking Regulation **16**(1), 72–88 (2015)
[39] Orda, A., Rottenstreich, O.: Enforcing fairness in blockchain transaction ordering. Peer-to-peer Networking and Applications **14**(6), 3660–3673 (2021)
[40] Park, A.: The conceptual flaws of constant product automated market making (2021), available at SSRN: 3805750
[41] Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 198–214. IEEE (2022)
[42] Reiter, M.K., Birman, K.P.: How to securely replicate services. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(3), 986–1009 (1994)
[43] Stathakopoulou, C., Rüsch, S., Brandenburger, M., Vukolić, M.: Adding fairness to order: Preventing front-running attacks in bft protocols using tees. In: 2021 40th International Symposium on Reliable Distributed Systems (SRDS). pp. 34–45. IEEE (2021)
[44] Tatabitovska, A., Ersoy, O., Erkin, Z.: Mitigation of transaction manipulation attacks in uniswap (2021)
[45] Wang, Y., Züst, P., Yao, Y., Lu, Z., Wattenhofer, R.: Impact and User Perception of Sandwich Attacks in the DeFi Ecosystem. In: ACM CHI Conference on Human Factors in Computing Systems (CHI), New Orleans, LA, USA (May 2022)
[46] Wood, G., et al.: Ethereum: A secure decentralised generalised transaction ledger (2014)
[47] Zhang, H., Merino, L.H., Estrada-Galinanes, V., Ford, B.: Flash freezing flash boys: Countering blockchain front-running. In: 2022 IEEE 42nd International Conference on Distributed Computing Systems Workshops (ICDCSW). pp. 90–95. IEEE (2022)
[48] Zhang, Y., Setty, S., Chen, Q., Zhou, L., Alvisi, L.: Byzantine ordered consensus without byzantine oligarchy. In: 14th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 20). pp. 633–649 (2020)
[49] Zhou, L., Qin, K., Gervais, A.: A2mm: Mitigating frontrunning, transaction reordering and consensus instability in decentralized exchanges (2021)
[50] Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-frequency trading on decentralized on-chain exchanges (2020)

A Maximal Utility Increase

We quantify a liquidity provider's minimum ε such that they move their reserves given an initial distribution of liquidity in this section. In Figure 6 we visualize for which ε a liquidity provider LP_i with a proportion q_i of their liquidity in Pool_N does not change strategy depending on the traders' relative benefit (cf. Figure 6a) and the slippage tolerance (cf. Figure 6b). In case $\varepsilon > I$, where I denotes the liquidity provider's maximal relative utility increase for a strategy change, a liquidity provider does not adjust their distribution. Thus, the smaller I, the higher the likelihood for the current configuration to be an ε-equilibrium.



(a) Liquidity provider's maximal utility increase dependent on their current liquidity distribution and the traders' relative benefit. We set $s = 0.05$.

(b) Liquidity provider's maximal utility increase dependent on their current liquidity distribution and the slippage tolerance. We set $\alpha = 0.2$.

Fig. 6. A liquidity provider's maximal utility increase I. Configurations where $I < \varepsilon$ are ε-equilibria resulting in the liquidity provider staying put. We set $x = 5000000\ X$, $y = 5000000\ Y$ and $f = 0.003$.

We first notice that the further a distribution is from a Nash equilibrium, the higher the maximal utility increase. For example, we see that for a high relative benefit, Figure 6a, the maximal utility increase I is highest when q_i is small, and thereby far from the Nash equilibrium. We observe the same pattern in Figure 6b. A liquidity provider with a near-optimal liquidity distribution, thus, has little to gain from changing strategy. While this is intuitive, it is also promising as liquidity providers have higher incentives to migrate to the Nash equilibrium whenever their liquidity distribution varies largely from the equilibrium liquidity distribution. b Additionally, we observe that in areas where the slippage tolerance is low or trading is less profitable for traders due to high slippage tolerances (cf. Lemma 5.4), a liquidity provider's maximal utility gain is small (cf. Figure 6b). The possibility for only a small increase in a liquidity provider's utility from adjusting their liquidity position when the slippage is low stems from both pools being similar for very low slippages. Thus, the received fees are similar in both pools. When the slippage is close to the bound provided by Lemma 5.4, it is also close to the intermediary Nash equilibrium, i.e., $\partial_p F = 0$ (cf. Figure 4). Therefore, the difference in the fees received in either pool is small.