Below, I try to outline what I think is an underexplored MEV research direction.

Often when blocks are built, the ordering doesn't take retail users' preferences over ordering into account. For example, in an ETH/USDC sandwich we know that what a sandwicher pays to the builder must be less than what the user lost in slippage. Simple heuristic changes don't necessarily solve this (as I will show below), can we allow users to express preferences such that the final block is representative of the utility of all users involved?

Paying for better execution

In a permissionless system, we assume that actors are rational so we assume block builders order transactions maximise payment to coinbase (let's leave aside the repeated game for now). So builders including sandwiches makes complete sense - users pay the same amount as long as their transaction is included, but sandwichers only pay if they are able to sandwich a user meaning that the builder is incentivised to think only about including the transaction and allowing for the sandwich.

This isn't an consequence of the bundling abstraction either. Let's imagine there are three transactions submitted to the builder:

- A and B: two swaps buying ETH on Uniswap and only paying for inclusion
- C: a swap selling ETH paying the coinbase address as an increasing function of the price at which the swap is executed

A rational builder orders the transactions (A,B,C) or (B,A,C) as this gives C the best possible price and consequently the builder the highest payment. Clearly, this gives B worse execution than orderings which place C before B.

Now let's imagine B also expresses preference over execution, but pays a higher rate for price improvement than C (this time for lower prices of course).

The ordering would then end up being (C,B,A). This ordering is clearly worse for C than before. What one would expect is that B and C - aware that other swappers are also paying for execution - raise their rate of payment to the builder until almost all the surplus from price improvement goes to the validator.

The plus side is that this would mean that users don't get sandwiched much (assuming everyone values the asset being traded into at roughly the same price). The down side is that users now end up with a net utility which is very close to their utility if they ended up being executed at their slippage limit so it's like they're being sandwiched anyway. This is particularly bad when you consider situations in which random ordering would have given a better price than the slippage limit even without payment for execution. (e.g. the swap is the only ETH/USD trade in the block and the trade is too small to warrant a sandwich because of gas & LP fees).

This is an intuitive argument. Did I miss something which means that users paying for execution is actually a feasible direction of research? What can be changes can be made to transactions and payments to make this feasible?

Sequencing rules

An interesting approach is something like what this paper proposes. That is, constraining builders to a sequencing rule in which swaps in the same direction can only follow each other if they are executing at a better price than what was available at the top of the block OR there are no swaps of the other direction following them (i.e. they are not being used to move the price for someone else).

This gives users a guarantee that they aren't being given a worse price for someone else's benefit.

Of course there are still are more less favourable orderings which users can pay for within this kind of rule and being in the long tail of same-direction swaps at the end isn't desirable. There is definitely also room for multi-block MEV strats here. There is also the question of what happens when users submit more complicated orders like arbitrages touching multiple pools.

With ordering rules and payment for execution quality as a tool, what's the best outcome we can achieve? ("Best outcome" for swappers that is

). What's the best sequencing rule? (Edit: there is wider exploration of the space of ordering rules inthis thread)