Note: this post is the result of a collaboration between team members of CoW Protocol @marco , @tom , Alex and myself) and @voyta.eth , a very active community member.

Introduction

At CoW Protocol, multiple orders can be matched against each other, or jointly against external liquidity - a major advantage over DEX aggregators where orders are treated in isolation. However, this flexibility comes with a potential for unfair solutions in which surplus is shifted illegitimately from one order to another, i.e., an order that was necessary for generating a certain surplus might end up not “receiving” it.

In this post, we aim to raise awareness on this phenomenon by first presenting a simple example of an instance where surplus can be unfairly shifted between different user orders. In order to address this, we then describe a test that detects this particular issue; if added to the rules of the solver competition, such a test would help provide stronger fairness guarantees to the users, and would be one more step towards our broader goal of providing users with the best and “most fair” trades. Finally, towards the end of this post, we also address limitations of the proposed test and point towards further steps that need to be taken in order for the solver competition to be in close alignment with the protocol’s goals.

A simple example of unfair surplus shifting

Consider the following two user orders and two liquidity orders:

- User order o1 is selling 3000 USDC for at least 1 ETH
- User order o2 is selling 3000 DAI for at least 1 ETH
- Liquidity order l1 is buying 3000 USDC for 1.01 ETH
- Liquidity order l2 is buying 3000 DAI for 1 ETH

For simplicity, we assume that fees match exactly the execution costs, and thus a solution that decides to match user order o1 with liquidity order l1, and user order o2 with liquidity order l2, has an objective function value that is equal to the total surplus generated. It is easy to see that in case we choose to execute the two CoWs (Coincidence of Wants), we get a total surplus of 0.01 ETH.

An unfair solution

We now present a solution that we believe is very clearly “unfair”. The solution is presented in the picture below, where we use the convention that an order is depicted with a directed arrow starting from the buy_token and pointing to the sell_token of the order.

[

1042×294 22.6 KB

](https://europe1.discourse-cdn.com/business20/uploads/cow/original/1X/95819718f5a1844f49443fff86f915913f5de167.png)

The clearing exchange rates are given as ETH/USDC = 3000 and DAI/USDC = 1.01.

This solution can be objectively considered unfair as it shifts surplus between orders. User order o1 should have gotten a surplus of 0.01 ETH given there is a liquidity order (l1) trading directly with it. Instead, the solution proposes that user order o1 gets no surplus whereas user order o2 gets a surplus of 0.01 ETH.

Note that the objective function is simply equal to the total surplus generated, which is 0.01 ETH.

A key property that is violated here is what we call “token conservation per user order”. More precisely, if we were to “split” the graph into two graphs, each containing the trading cycle of each user order, then it would be easy to see that the trading cycle of o1 is “losing” 0.01 ETH. In other words, some tokens are extracted out of that trading cycle.

One way to demonstrate this is by using the exchange rates as follows (where, as a reminder, the exchange rate of an order is defined as the ratio of the sell amount over the buy amount):

- Trading cycle of o1:

exchange_rate(o1) × exchange_rate(l1) = (3000 / 1) * (1.01 / 3000) = 1.01.

- Trading cycle of o2:

exchange_rate(o2) × exchange_rate(l2) = (3000 / 1.01) * (1 / 3000) ≅ 0.99.

A fair solution

The desirable solution to the above batch would be to give this 0.01 ETH to user order o1, as already mentioned, and as shown in the picture below.

[

1042×294 22.7 KB

](https://europe1.discourse-cdn.com/business20/uploads/cow/original/1X/391a7468c18a3ac79cf2927b2904a53941cd7348.png)

In this case, the clearing exchange rates would be ETH/USDC ≅ 2970 and DAI/USDC ≅ 0.99. It is also easy to see that the product of exchange rates in each trading cycle is now equal to 1:

- Trading cycle 1:

exchange_rate(o1) × exchange_rate(l1) = (3000 / 1.01) * (1.01 / 3000) = 1.

- Trading cycle 2:

exchange_rate(o2) × exchange_rate(l2) = (3000 / 1) * (1 / 3000) = 1.

Note that again, as before, the objective function is simply equal to the total surplus generated, which is 0.01 ETH; this indeed reveals that the objective function itself cannot distinguish between a fair and an unfair solution, both having the same surplus.

Local token conservation test

To mitigate solutions such as the unfair one presented above, we need to ensure that for every user order α, no external tokens “enter” or “exit” the trading cycles that contain α (and consequently, no surplus can be shifted between orders on independent trading cycles). We refer to this condition as “token conservation per order" or “local token conservation”.

More formally, for each user order α, we require that a certain convex combination of the products of exchange rates over all trading cycles that contain user order α is equal to 1:

where λ(C) and r(C) are well-defined non-negative weights and exchange rates, respectively, of a trading cycle C in the trading subgraph C(α) of user order α (see here for a proper definition of these terms; andhere a more in-depth write-up).

Roughly speaking, this weighted sum (which, more precisely, is a convex combination, i.e., the sum of λ’s adds up to 1) ensures that the resources (i.e., AMMs/liquidity orders) used to execute a given batch are “accessed” by all user orders in the “same rate”, and thus result in a fair execution.

Benefits

This test would filter out a certain kind of unfair solutions that may otherwise be considered valid, and also prevent some internal buffer attacks. There are no known drawbacks compared to the status quo.

Implementation

The test could simply be implemented in the backend code. For solvers that use private liquidity, it requires trusting that the advertised traded amounts match the settled transactions, which needs to be verified retrospectively.

Conclusion & Outlook

As of now, the above test, which is already described in thedocumentation, is not explicitly done when validating solutions, and is only enforced on a solver-by-solver basis, since all solver teams have been asked to ensure that the solutions they currently submit do pass the test. Nevertheless, we believe that it could be a formal test that every solution is required to pass before considering it valid.

Of course, this test does not solve all known issues with unfair surplus shifting. In particular, it does not address the fundamental bargaining problem that arises when there is a perfect CoW between two user orders (or in general, with more user orders creating a single trading cycle). In such cases, there is usually an interval of candidate prices that all generate the same total surplus, and thus are considered equivalent by the current objective function. Again, currently all solver teams participating in the competition have been asked to ensure that the prices in their computed solutions follow what we call “market prices”, and the CoW Protocol team has been closely monitoring executed batches to ensure that this is indeed the case.

A fundamentally stronger and arguably more complex solution to all unfair surplus shifts can be given by enforcing some version of envy-freeness. The solver team of CoW Protocol has been working on this topic, which will be addressed in another post in the near future.

Until then, we believe that, on top of the proposed test presented above, “social consensus” rules should be in place so as to ensure that a solver executing a settlement that unfairly shifts surplus could potentially be penalized.