CoW Protocol Solver Competition - A 2nd Price Auction Approach

Max Holloway [*]
max@xenophonlabs.com

November 2022

Abstract

The CoW Protocol solver competition today pays a fixed amount of rewards for each batch of matched trades. We analyze the shortcomings of this approach, and we present a new approach to the solver competition that has the potential to simplify solvers' optimization as well as increase the value that the platform is able to deliver to its users.

1 Introduction

CoW protocol provides a trading interface that enables users to maximize the value of their trade's execution. Users submit their orders to a centralized component, called the "driver", which serves as their liason to get optimal order matching and execution. Since optimally matching and executing orders is challenging, the protocol outsources this task to an external group of competing optimizers, called "solvers". The solvers submit their proposed order matching, called a "solution", to the driver, and the driver selects the rule-compliant solution that has the highest value as the winner. The solver who submitted the winning solution then settles the solution on-chain. In exchange for solvers' matching and execution service, the protocol pays them and reimburses their gas expenditure for on-chain execution.

Although solvers are an essential component of the protocol, their incentives do not naturally align with that of the protocol and its users. It is fair to assume that the solvers are profit-maximizing, and currently this profit comes from winning the solver competitions, which can only be achieved by optimizing a function that benefits traders, subject to constraints that make the solution economically viable for the protocol. While this approach has proved to be quite practical, there are a number of pain points for the protocol that could use improvement.

One pain point for the protocol is that the driver is required to estimate the gas cost required to execute the batch solution on-chain. This calculation is difficult to perfect, and it would be better handled by solvers than the protocol. Generally, moving batch execution calculations away from the protocol, and onto the solvers, precludes principal-agent problems and is thus desirable.

Our aim in this paper is to illustrate how the protocol can elicit optimal batches from solvers and compensate them to execute those solutions, under the assumption that the driver knows nothing of the outside world except for what is reported by the solvers. The field of mechanism design provides powerful tools that enable us to realize this goal.

We present desirable properties of the solver competition in section 2, and we provide a practical mechanism that achieves many of these properties in section 3. We then discuss the limitations of our work and avenues for further research in section 4, and we conclude in section 5.

2 Mechanism Desiderata

There are three stakeholders affected by the solver competition: traders, solvers, and the protocol. We provide desiderata that apply generally to all stakeholders, as well as desiderata that apply specifically to

[*]A number of generous individuals contributed to the creation of this document. Thomas Bosman independently discovered a very similar 2nd price mechanism months before I did; Marco Correia presented the idea for a similar capped payment to what it used here; and the entire CoW protocol team has supported this research.

each of these stakeholders.

2.1 General Desiderata

1. No matter what information the protocol asks the solver to report, the solver's dominant strategy is to report that information truthfully. This property is called *Dominant-Strategy Incentive Compatibility* (DSIC). This enables solvers to win by giving the best solutions, rather than reporting an untruthful value to the mechanism in hopes of higher profit.
2. Compatibility with existing solver optimization constraints [Pro]. These constraints make explicit which batches are valid, and these constraints implement important business constraints for the protocol.

2.2 Trader Desiderata

1. Adherence to the existing batch auction rules [Pro].
2. The solution chosen by the protocol should be the one that maximizes trader surplus. This property roughly encapsulates ex-post Pareto Efficiency (PE).

2.3 Solver Desiderata

1. Submitting a batch solution will never yield negative profit. This property is called *Individual Rationality* (IR).

2.4 Protocol Desiderata

1. The protocol should generate a profit, or at the very least, remain profit-neutral. This property is called *Weakly Balanced Budget* (WBB).
2. Solvers handle batch execution and all calculations related to batch execution, including gas consumption and gas price estimates, rather than the driver.
3. The solution chosen should be the solution that maximizes the protocol's objective function with respect to trader surplus, fees, and blockchain costs.

3 A New Solver Competition Mechanism

Here is a mechanism that handles solver solution selection and solver payment.

3.1 Mechanism Overview

Suppose that solver i creates a batch solution s_i that has an associated blockchain execution cost c_i. Assume, for now, that after computing the duple (s_i, c_i), they honestly report this duple back to the driver.

Now the driver computes the *social welfare function* $\mathcal{U} : (S \times \mathbb{R}^+) \to \mathbb{R}$ as

$$\mathcal{U}(s, c) = Q(s) - c,$$

where $Q : S \to \mathbb{R}$ gives the *quality* of a solution s, and c gives the solver's reported cost of the solution. An intuitive definition of solution quality is simply the trader surplus plus the protocol fees, which would have definition

$$Q(s) = \sum_{o \in O} U_o(x(o), y(o)) \cdot z(o) + \sum_{o \in O \cup L} f_o \cdot z(o),$$

where O is the set of user orders, and L is the set of all liquidity orders; all notation is borrowed from the CoW protocol docs [Pro]. Here, Q and c are both represented in units of a numeraire known to solvers, such as `ETH`.

After the solution submission period is completed, the driver computes $\mathcal{U}$ for all solution-cost pairs (s, c). It finds the pair which yields the highest utility, (s_i, c_i), and the pair which yields the second-highest utility, (s_j, c_j); if there is only one valid solution, then the next-best solution is simply the null solution, with no order matches and zero cost. The driver chooses the solver i as the winner, it pays an amount p_i to solver i, and nothing to the losing solvers. We define the amount p_i as

$$p_i = c_i + (\mathcal{U}(s_i, c_i) - \mathcal{U}(s_j, c_j))\,.$$

This payment can be interpreted as the driver paying the best solver the cost of executing their solution, plus the marginal utility that this solver achieved beyond the next best solver, which is a special case of the Clarke Pivot Rule for VCG mechanisms [Wikc].

This payment is dependent on the settlement of the batch into the blockchain. If the solver attempts to settle the batch on-chain, and their settlement transaction reverts, then the payment to them should be equal to the cost that they suffered in the revert.

We have thus described a way for our mechanism to determine a winning solution and to make payments. We omit implementation details, such as when/how to make payments to solvers.

3.2 Mechanism Properties

All of the following good properties apply for a single round of the solver competition.

1. **Dominant Strategy Incentive Compatibility (DSIC) w.r.t. solution utility**. If a solver i finds two solutions, (s_a, s_b), that have blockchain execution costs (c_a, c_b), such that $\mathcal{U}(s_a, c_a) < \mathcal{U}(s_b, c_b)$, then their profit $p_i - c_i$ is at least as high when they report solution $s_i = s_b$ and cost $c_i = c_b$. This follows directly by comparing $p_i - c_i$ for case a and case b. If player i would win the auction with solution b, then they are strongly incentivized to report solution b; if solution b would not win the auction, they are just as well off by reporting solution b as they would be by reporting solution a, since profit is 0 in each case (they receive no payment and pay no blockchain execution cost).

2. **Dominant Strategy Incentive Compatibility (DSIC) w.r.t. cost reporting**. For any given solver i, and a given solution s_i, their profit $p_i - c_i$ is at least as high when they report the honest cost of their solution. That is, they have no incentive to report the cost of executing s_i. This is proved in appendix A.

3. **Individual rationality, modulo non-blockchain costs**. Assuming the solver accurately estimates their own blockchain cost, and that their other costs are negligible, there is no situation whereby a solver would benefit by abstaining from the auction. Notice that this assumption of small non-blockchain costs only affects the solver's willingness to compete; it *does not* affect the solver's willingness to be incentive compatible.

4. **Maximizes social welfare**. By definition, the winning solution and cost, (s_i, c_i), maximize the social welfare function $\mathcal{U}$.

5. **Weak collusion resistance**. This follows from dominant-strategy incentive compatibility: the only Nash Equilibrium is the one in which all solvers give their best solution and its exact cost. Due to this being a unique Nash Equilibrium, there is no room for collusion-equilibria.

6. **Compatibility with positive net-trader-surplus**. Let's define net trader surplus as `trader surplus - blockchain cost`. We can impose a constraint that says the driver will only accept batches with positive net trader surplus. Modifying the auction in this way does not affect any of the aforementioned nice properties: payouts are all zero in the case where no net-positive-trader-surplus solutions are submitted, and payouts are identical to before when such solutions are submitted. In neither of these cases does the solver have an incentive to deviate from their aforementioned truthful dominant strategy.

7. **Compatibility with existing CoW solution constraints**. We can define the valid solution language S to be the set containing any valid solutions, and all of the aforementioned results still hold. In practice, the driver would check that each solver i's reported solution s_i obeys all of the constraints.

While this mechanism has many good properties, it has some issues as well, which we list below.

1. **Imperfect collusion resistance**. Although the mechanism described does not have any Nash Equilibrium that include collusion, solvers could augment the mechanism outside of the protocol, and the augmented game may have collusion equilibria. For instance, if all solvers agreed to report the utility of their best solutions to each other, then only have the highest utility solver participate, then they could artificially boost the component of the payment that comes from utility surplus. This additional payment could be split among the other solvers, and participating in this augmented version of the solver competition would strictly increase each solver's profits. To enforce collusion in the augmented game, one could imagine requiring a slashable stake in order to participate in the augmented game. This becomes less of an issue if we can guarantee that there will be at least one solver that does not participate in the augmented collusion game. Otherwise, we do not know of a solution to this problem.

2. **Higher payment made by the protocol than the current baseline**. When we backtest the amount that the protocol would pay under this mechanism, assuming all solutions are identical to those currently, then the mean payment would be almost two times as large as the current mean protocol payment. This is not an acceptable compromise, and we address potential remedies later in this section.

3. **Solutions may not be economically viable for the protocol**. The net payment made by the protocol, i.e. `sum(fees) - payment`, may be greater than zero, violating the WBB property. This cannot simply be addressed by adding a constraint to the solver optimization problem, since computing the payment requires the top two solutions and costs, and the solver optimizaiton constraints are only allowed to rely on the data present in a batch. This is due to the second-price nature of this mechanism, and a first-price auction would be able to resolve this issue, albeit at the expense of many of the good properties above. We present possible solutions later in this section.

3.3 Mechanism Modifications

3.3.1 Utility Cap on Payments

When we backtest what the payment under our 2nd price mechanism would be for historical batches, we see that its average payment by the protocol is over two times larger than that under the status quo mechanism. When we investigate the distribution of payments made under the 2nd price mechanism, we see that the bulk of the payments are much less than the status quo average payment; however, there are some instances of very large payments by the protocol which lead to astronomically high solution payments, which then dominate the mean payment across all batches.

A natural reaction is to cap the payments. If we simply cap the payment made by the mechanism, this will introduce the risk that a solver's payment will be less than their cost. Instead, we might introduce a cap on the utility surplus component of their payment, for some cap $k > 0$ and top two solvers i, j:

$$p_i = \min\left(k, U(s_i, c_i) - U(s_j, c_j)\right) + c_i.$$

In scenarios where the utility difference between the top two solutions is less than or equl to k, this payment rule is equivalent to that of our original 2nd price mechanism. When the utility difference is greater than k, solvers no longer receive the proceeds of their solution's marginal utility.

Unfortunately, this payment rule *does* preclude incentive compatibility of the mechanism, in utility-difference-greater-than-k scenarios. However, we are able to show in a backtest that if we set the utility cap such that the protocol retains the same status quo mean payment that it does today, then the utility diff payment cap will be approximately `0.312 ETH`, and only approximately `0.692%` of batches would reach this cap. That is, the cap would not make any difference on the strategic behavior of solvers in at least `99%` of the

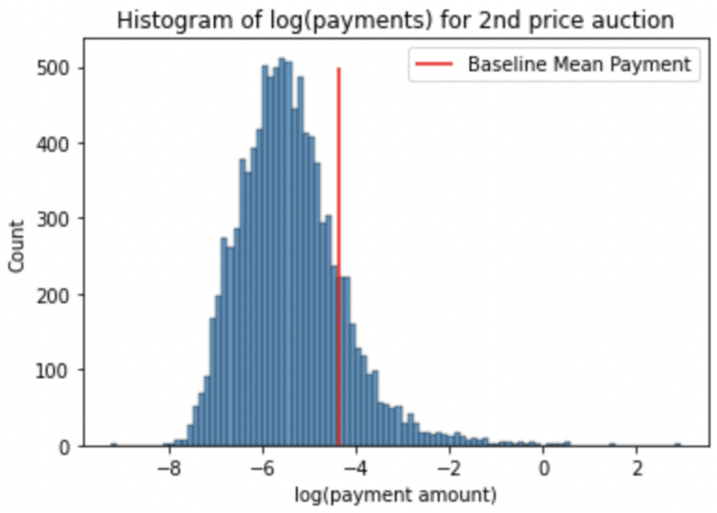

Figure 1: Most payments made under our 2nd price auction mechanism are less than the mean payment of the baseline mechanism. However, there is a fat tail of very high payments that elevates the mean of the 2nd price auction mechanism above the baseline mechanism.

batches. In the batches where gaming could exist, the winning solution will still need to be at least as high utility as the next best solution, and thus the impact on the protocol of gaming would be rather small, even if it occurred. It remains an open research question just how much profit a solver can extract by reporting a solution-cost pair that is suboptimal for the protocol in scenarios where the cap is enforced. We hypothesize that solvers have better optimization opportunities than to optimize for the `0.692%` of batches which hit the utility diff cap.

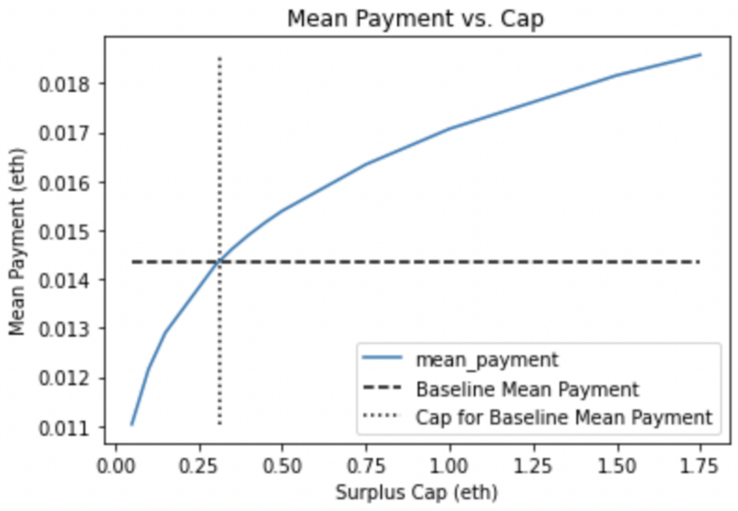

Figure 2: The utility surplus cap of `0.312 ETH` gives us our baseline mean payment of `0.0127 ETH`

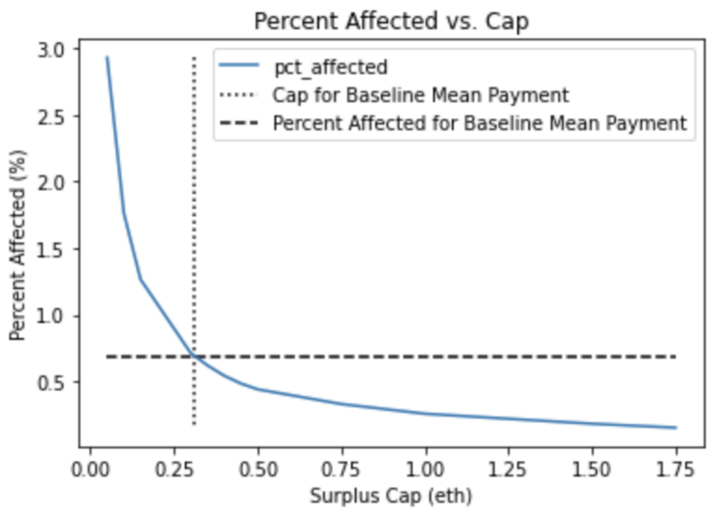

Figure 3: The utility surplus cap of `0.312 ETH` affects only `0.692%` of batches.

3.3.2 Modified Quality Score

Our mechanism is robust enough to handle arbitrary quality functions, so long as they are real-valued. An intuitive decision is to define Q as

$$Q(s) = \sum_{o \in O} U_o(x(o), y(o)) \cdot z(o) + \sum_{o \in O \cup L} f_o \cdot z(o).$$

Thus the utility of a solution, $\mathcal{U}(s, c) = Q(s) - c$, would replicate the objective function in the current solver competition. This utility function implicitly gives equal priority to trader surplus, fees, and cost, and each of these terms is simply additive. However, the design space for utility functions of the form $Q(s) - c$ is vast.

One approach is to determine weights, w_1, w_2, that apply to trader surplus and fees, respectively. Then, if trader surplus is $T(s)$ and fees are $F(s)$ for a solution s, it would follow that $Q(s) = w_1 T(s) + w_2 F(s)$, and $\mathcal{U}(s, c) = w_1 T(s) + w_2 F(s) - c$. This generalization of the baseline objective function gives the protocol a rich space to control its revenues and trader surplus. Some example parameter settings include $w_2 > 1$ if the protocol wants to prioritize its revenue over its costs, and $w_1 < 1$ if the protocol wants to prioritize costs over trader surplus. Perhaps best of all, this objective function is linear, and would likely require very little engineering lift for solvers to upgrade.

This approach can be used as a non-invasive way to build protocol economic viability into the solver competition. By non-invasive, we mean that this modification will not affect any of the positive properties of our 2nd price auction mechanism, and it can be tuned to make the solver competition WBB for the protocol, in expectation. Still, this solution is not as clean as an inviolable protocol WBB constraint placed on a batch, and that remains an open problem.

3.3.3 Slippage Payments

When a solver settles a batch on-chain, they often will get a different price than the one they anticipated when submitting their solution to the driver. When they receive more token than anticipated, this is called *positive slippage*; when they receive less token than anticipated, this is called *negative slippage*. In the current version of CoW protocol, solvers forfeit all positive slippage, and they pay for any negative slippage.

However, this asymmetric slippage payment may potentially lead to gaming by solvers. Since solvers are able to utilize off-chain orderflow, it is possible that they would game the solver competition by settling orders on a private-orderflow smart contract that has no notion of slippage, then separately executing orders that may incur positive or negative slippage, then sending back the required amount to the private-orderflow smart contract, then routing funds back to users. Solvers could make it arbitrarily difficult for the protocol to detect positive slippage, thus pocketing it for themselves. Asymmetric slippage payments should be avoided if the protocol wants to avoid this gaming behavior; a possible replacement is to pay solvers their positive slippage, as well as make them pay their negative slippage.

Unfortunately, slippage changes the solvers' solution-cost reporting strategy in the auction. Since increasing the quoted trader surplus leads to a higher probability of wining the auction, solvers would need to optimize their reported solution s such that it maximizes their expected profit. This profit maximization requires anticipating the next best solver's solution's utility, which undermines the benefit that we get from dominant-strategy incentive compatibility with the second price auction.

This problem is not likely to have a clean solution that remains individually rational for solvers, since the root of this problem is that the quality term in the solver's solution is a random variable, rather than a fixed quantity. A next step on this line of research would be to quantify the historical distribution of slippage, then determine how critical of an issue this is; if slippage is negligible compared to solver profits, then we can make a similar claim to that of capped payments: incentive compatibility breaks, but the economic upside of breaking incentive compatibility is likely too low to make it worth solvers' efforts.

3.4 Recommendations

1. Use a 2nd price auction, as explained in this section.
2. Enforce a utility difference cap at `0.312 ETH`.
3. Adjust the weights of the protocol objective function to reflect the importance of blockchain cost, fees, and trader surplus.
4. Require solvers to pay negative slippage, and allow solvers to keep positive slippage.

4 Discussion

Although we have given some detail regarding a CoW protocol 2nd price solver competition, we admit that our best solution – a 2nd price auction with utility payment caps and a weighted-average quality score – still has open problems that deserve further research. We detail a number of these open problems below.

Collusion prevention. To incentivize solvers to provide the best solution, rather than a solution that is just better than all of their competitors', we need their payment to increase with the utility of their solution. However, our mechanism does not have a notion of what utility is appropriate for a given batch, and thus it uses the relative utility of the top two solutions to determine how much the top solver should receive. This opens the door for collusion behaviors between the top two solvers. Are there other ways that the protocol could increase its payments with respect to utility that do not use the second-best solution's utility?

Repeated games. All of our analysis in this paper is on a single round of solver competition. However, the real solver competition would be repeated indefinitely into the future. This gives rise to repeated-game equilibria that diverge from repeating the stage-game dominant strategy. For instance, a malicious solver Millie can broadcast on Twitter that if any other solvers win a batch, then Millie will under-report her own cost indefinitely into the future, leading her to win all batches into the future; this is called a "grim trigger" [Wika]. Although this strategy would be painfully expensive for Millie in the case where others do not cooperate, it is nevertheless a Nash equilibrium for all to cooperate, leading to worse outcomes for most solvers and for users. Analyzing equilibria in repeated games is notoriously challenging, and it is an open question how we would avert bad repeated game equilibria on the protocol.

Proving revenue equivalence. Although the uncapped version of our mechanism resembles a standard single-unit second-price reverse auction, we have not demonstrated that it satisfies the conditions for the revenue equivalence theorem to hold [Wikb]. We conjecture that revenue-equivalence must be satisfied, but this remains to be proven.

Optimal auctions. Since Myerson's seminal 1981 paper, a rich literature of revenue maximizing auctions has commenced [Mye]. While 2nd price auctions are optimal in prior-free scenarios, we can likely form a good prior on the utility that the top solver will report for a given batch. How could concepts like virtual bids and reserve utilities squeeze down the protocol's costs, while retaining incentive compatibility? Could this approach provide an alternative to the capped utility diff approach?

Assuming 1-2 benevolent solvers. If we assume that a small set of 1-2 solvers will try to maximize utility under all conditions, how much would this improve our worst-case scenarios for things like collusion and untruthfulness due to utility diff cap? Would it be worth spinning up a lean, external entity, whose mandate is to act as a benevolent solver?

Research on the utility diff cap. We chose a utility diff cap that preserved the protocol's cost. If we lowered the utility diff cap further, then this would lead to lower protocol costs, but it would also increase the number of batches that are gameable. If the protocol migrated to this mechanism, it would be worthwhile to research an optimal utility diff cap; our preliminary research showed that we could decrease protocol spending on solutions by up to 10%, while still keeping the proportion of gameable batches below 1.5%.

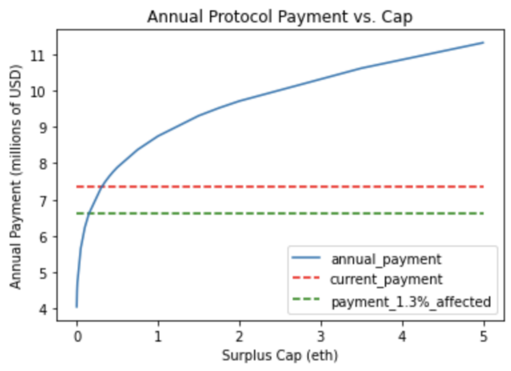

Figure 4: Source: colab notebook [Hol].

Historical slippage. As addressed in section 3.3.3, slippage can impact solvers' solution-cost reporting strategy. We could use historical on-chain data to determine the slippage distribution; if slippage is very small in comparison to solver profits, then it is unlikely that solvers would modify their strategy to account for slippage; if slippage is large, then slippage may have a negative impact on the strategy-proofness of the auction. Which of these is the case?

5 Conclusion

In this paper, we present a new design for a solver competition that draws from mechanism design and auction theory best practices. The key insight is that the protocol pays the winning solver their cost, plus the marginal utility that the solver brings. This auction has a number of nice properties, such as solver truthfulness (over 99% of the time) and maximization of the protocol's utility function. We even provide a number of methods for decreasing the protocol's costs by over $750,000 per year, all while keeping solver truthfulness in over 98.5% of batches. While there are a number of remaining open questions, we believe this auction provides a practical avenue for the protocol to move settlement to its solvers.

A Proof: Dominant Strategy Incentive Compatibility of Reporting a Solution's True Cost

Claim For any player k and fixed solution s_k, is a dominant strategy for player k to report their true cost c_k of settling the solution on-chain.

Proof

Suppose that in the honest case, the winner is i and the second-place solver is j, and (s_i, c_i) and (s_j, c_j) are their true optimal solutions and costs. Our objective is to show that no players have an incentive to deviate from their truthful reporting.

Case A: Winner Misreporting Suppose player i would be first place if they reported honestly. We make no assumption about the honesty of other players' reporting.

Then player i has the following profit from their truthful reporting:

$$\begin{aligned} p_i - c_i &= c_i + (\mathcal{U}(s_i, c_i) - \mathcal{U}(s_j, c_j)) - c_i \\ &= \mathcal{U}(s_i, c_i) - \mathcal{U}(s_j, c_j) \\ &\geq 0. \end{aligned}$$

Suppose that they misreport their cost as $c_{i'}$. If they are no longer the winner, then their new profit is 0, which is never better than their profit as a winner. If they are still the winner, then their new profit is

$$\begin{aligned} p_{i'} - c_i &= c_{i'} + (\mathcal{U}(s_i, c_{i'}) - \mathcal{U}(s_j, c_j)) - c_i \\ &= c_{i'} + ((Q(s_i) - c_{i'}) - \mathcal{U}(s_j, c_j)) - c_i \\ &= (Q(s_i) - c_i) - \mathcal{U}(s_j, c_j) \\ &= \mathcal{U}(s_i, c_i) - \mathcal{U}(s_j, c_j). \end{aligned}$$

If player i is still the winner, then they profit the same amount when mis-reporting and remaining the winner. This demonstrates that when a player wins by reporting honestly, they have no incentive to deviate from reporting honestly. Since we made no assumptions about the reporting of other players, honest cost reporting is a dominant strategy for the winning player.

Case B: Loser Misreporting Suppose player k would lose the competition if they reported cost honestly; that is, $\mathcal{U}(s_k, c_k) < \mathcal{U}(s_i, c_i)$. We make no assumption about the honesty of other players' cost reporting.

Then player k has the following profit from truthful reporting: $p_k - c_k = 0 - 0 = 0$.

Now suppose that player k misreports their cost as $c_{k'}$. If they are still not a winner, their profit remains unchanged. However if they are now a winner, their profit is

$$\begin{aligned} p_{k'} - c_k &= c_{k'} + (\mathcal{U}(s_k, c_{k'}) - \mathcal{U}(s_i, c_i)) - c_k \\ &= c_{k'} + ((Q(s_k) - c_{k'}) - \mathcal{U}(s_i, c_i)) - c_k \\ &= (Q(s_k) - c_k) - \mathcal{U}(s_i, c_i) \\ &= \mathcal{U}(s_k, c_k) - \mathcal{U}(s_i, c_i) \\ &\leq 0. \end{aligned}$$

We have shown that player k only stands to lose money by misreporting their cost. Since we made no assumptions about other players' behavior, honest cost reporting is a dominant strategy for the losing player.

Conclusion

When a player reports their solution and cost, they must be either a winner or a loser. Since honestly reporting the cost is a dominant strategy in both of these cases, it follows that honestly reporting cost is a dominant strategy in all cases.

References

[Hol] Maxwell Holloway. *CoW 2nd Price Auction Data Analysis.* URL: https://colab.research.google.com/drive/1VZP2oqNEeXUgY05TcfGF5TApJzjJoxFZ?usp=sharing.

[Mye] Roger B Myerson. “Optimal auction design”. In: *Mathematics of operations research* 6.1 (), pp. 58–73.

[Pro] CoW Protocol. *The Batch Auction Optimization Problem.* URL: docs.cow.fi/off-chain-services/in-depth-solver-specification/the-batch-auction-optimization-problem.

[Wika] Wikipedia contributors. *Grim trigger — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-October-2022]. URL: https://en.wikipedia.org/w/index.php?title=Grim_trigger&oldid=1055411435.

[Wikb] Wikipedia contributors. *Revenue equivalence — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-October-2022]. URL: https://en.wikipedia.org/w/index.php?title=Revenue_equivalence&oldid=1010949459.

[Wikc] Wikipedia contributors. *Vickrey Clarke Groves mechanism — Wikipedia, The Free Encyclopedia.* [Online; accessed 27-October-2022]. URL: https://en.wikipedia.org/w/index.php?title=Vickrey%E2%80%93Clarke%E2%80%93Groves_mechanism&oldid=1074505319.