A Theory of Robust Hardness for Truthful Mechanism Design

Dave Buchfuhrer

Abstract

Truthful mechanisms are important to mechanism design. Unfortunately, requiring that they also be computationally efficient can result in poor approximation ratios. In some cases, the impossibility of achieving good approximations is a result of ignoring the computational limitations of the players. This has lead to attempts to propose mechanisms which take player computation into account. In order to better understand such mechanisms, we propose a framework for showing hardness of such mechanisms.

1 Introduction

Combinatorial auctions and combinatorial public projects are two basic allocation games. In each one, the mechanism is in control of a set of objects desired by the players, and each player has preferences over subsets of the items. In an auction, the items are allocated to the players, with players bidding to obtain exclusive ownership over a subset of the items. In a public project, a subset of items are provided to all players with each having full access to the items, as with park equipment or museum displays. Players bid and make payments in order to influence which subset is chosen. In either case, we consider the goal of maximizing *social welfare*, the sum of the players' values for the subset they each receive or share.

In both settings, we wish to design *truthful* mechanisms, in which the mechanism asks each player for his valuation and bidding truthfully is a dominant strategy. The only known universal technique for designing such mechanisms is the VCG mechanism. In many cases, the VCG mechanism is the only truthful mechanism [1, 6, 7, 9]. Unfortunately, recent results have shown that efficient VCG mechanisms are sometimes incapable of achieving a good approximation of the social welfare [2, 3, 4, 8, 9].

A striking example for which all efficient truthful mechanisms have a poor approximation ratio is shown in [3]. Consider the somewhat odd example of a single-player public project. In this problem, there is a set S of items, and the mechanism must choose a subset $S' \subseteq S$ of size $|S'| = k$ to allocate to the player. The goal of the mechanism is to maximize the social welfare, which in this case is the player's value. The goal of the player is also to maximize his value. So mechanism design in this case should be very easy, if not trivial.

The single player in this public project has a coverage valuation. A player with a coverage valuation has a special set $\mathcal{U}$ and associates each item $s \in S$ with some set $U_s \subseteq \mathcal{U}$. The value of a set $S' \subseteq S$ to this player is the size of the union of the associated sets, $\left|\bigcup_{s \in S'} U_s\right|$.

So the goal is to find maximum coverage of $\mathcal{U}$ using only k sets from $\{U_s\}$. This is an NP-hard problem, but can be approximated with a factor of $e/(e-1)$ using a greedy algorithm. Unfortunately, if the $e/(e-1)$ approximation algorithm is used as a mechanism, it is not truthful for the following reason.

If the player wants a particular set T other than the one which would have been allocated by truthful reporting, he need only state that his valuation function has $\mathcal{U} = T$ and $U_s = \{\}$ for $s \notin T$ and $U_s = \{s\}$ for $s \in T$. The greedy algorithm would then pick T as the allocation. Unless the greedy algorithm finds the best solution, there exists a T which is of more value to the player than the greedy solution found by truthful reporting. [3] showed that no efficient truthful mechanism can have an approximation ratio better than $\sqrt{|S|}$ unless NP has polynomial circuits.

This hardness result may seem to be just a strange quirk that can easily be ignored, as a single player public project lacks the most important aspect of a game, namely multiple players with competing interests. This strangeness causes an issue for proofs of hardness for the 2-player or n-player versions of public projects with coverage valuations. These cannot be approximated efficiently and truthfully because they contain the single-player version as a special case. Unfortunately, this proof is vulnerable to precisely the same objections as the original single-player proof. A more robust proof must somehow first eliminate the single player case as the source of complexity.

The above counter-intuitive hardness result for single-player public projects relies on an asymmetry in the definition of "efficient truthful mechanism." In order to be efficient, a mechanism cannot perform NP-hard computations. In order to be truthful, there must not exist a lie by which a player could benefit. Finding this lie need not be computationally feasible. For example, in the greedy mechanism example, a beneficial lie requires solving an NP-hard problem by finding an allocation which approximates the maximum coverage by a factor better than $e/(e-1)$.

1.1 The Nisan-Ronen approach

In [8], a "Second Chance" mechanism was suggested to address the above asymmetry. The mechanism can use any allocation algorithm A. The players submit their valuation functions to the mechanism, and A is used to compute an allocation. Next, each player i submits a function f_i which maps the n valuation functions submitted by the players to n new valuation functions for which the player believes A will compute an allocation with better social welfare.

Each f_i is applied to the submitted valuations to arrive at a new collection of valuations. A is then applied to each of these to arrive at n additional candidate allocations. Of the $n+1$ allocations computed by the mechanism, the one with maximum social welfare is chosen. VCG-style payments are used to ensure that each player is incentivized to maximize the social welfare. So any player who wishes to lie should prefer to put that lie into the function f_i.

Unfortunately, the need to compute each player's function f_i requires the mechanism to limit the computation time of f_i to some polynomial n^c. So a player with more computational resources than n^c time would still benefit more by lying than by submitting a good f_i.

We propose a new general class of mechanisms which take advantage of player computation. Instead of trying to get a handle on player computation and limiting it to the resources available to the mechanism, our mechanism makes use of the same resources that players are using to lie. The mechanism supplements its own computation by making queries to the players. For example, in order to lie a player may need to be able to compute which allocation they prefer given a set of per-item prices which they would be charged. We call such a query in a public project a *k-demand query.*

In the public project with one coverage valuation player described above, making a k-demand query with item prices all equal to 0 results in a solution to the problem. By assuming that the player has the computational capabilities necessary to lie, we are able to come up with a mechanism where lying is no longer beneficial. The use of VCG payments incentives the player to maximize social welfare, so as long as the mechanism maximizes social welfare there is no benefit to lying. A further advantage of our approach is that it can be used to model various assumptions of player power by choosing appropriate types of queries.

Proving that a mechanism like the one above finds an exact solution does not require any more new concepts or techniques. But not every pairing of an oracle and a mechanism design problem will result in an exact algorithm. Some classes of queries will not be powerful enough to allow for better mechanisms. The second main contribution of this paper is a framework for analyzing the computational complexity of these allocation problems paired with queries, as well as several substantive results showing hardness within this framework.

1.2 Oracles and reductions

We model the player queries as instance oracles.

Definition 1 (Instance Oracle)**.** *Consider an n-player game A, with instances described by $(a_1, \ldots, a_n)$, where a_i is the private information held by player i. An* instance oracle *O is a black box function such that for some f, on input x it returns the n results $f(x, a_1), \ldots, f(x, a_n)$.*

For example, an oracle for k-demand queries would take prices $p_1, \ldots, p_n$ as the input x and for each i return $f(x, a_i)$, the set of k items which maximizes player i's utility with prices x. Note that this differs from a traditional oracle in that part of the input is fixed to the player valuations. So even if the k-demand query is NP-hard in general, an instance oracle for demand queries would not necessarily allow the mechanism to solve arbitrary NP-hard problems. For example, if in a particular instance, all players happen to have valuation functions $a_i(S) = 0$ for all sets S, the instance oracle could simply return the items with the k

lowest prices, which can easily be computed in polynomial time and therefore adds no power over polynomial computation.

We denote a problem A with associated oracle O by A^O. We can now define complexity classes in terms of problems with oracles. The class IONP is the class of all problems A^O such that $A \in$ NP. In order to show reductions between these problems, we need to modify the idea of a polynomial-time reduction to take the oracles into account. We do so as follows.

We say that A^O reduces to B^Q (or $A^O \leq_{IO} B^Q$) if there is a polynomial time reduction r from A to B such that for any a, $a \in A \Leftrightarrow r(a) \in B$ and O can be used on a to answer queries to Q on $r(a)$. So a reduction pairs a standard reduction (optionally making use of O) from A to B together with a way to simulate Q using O. This means that even showing $A^O \leq_{IO} A^Q$ can require a nontrivial reduction in order to be able to simulate Q on the result of the reduction, as seen in Section 5.1.

A problem $A^O \in$ IONP is IONP-complete if $A^O \in$ IONP and for all $B^Q \in$ IONP, $B^Q \leq_{IO} A^O$. Note that if A in NP-complete, A^O is IONP-complete for any $O \in$ FP. Most of the proofs of IONP-completeness in this paper reduce to NP-complete problems paired with trivial oracles.

These reductions fulfill the standard properties expected of reductions, as can be seen by the following easily proven lemmas.

Lemma 1. *If $A^O \leq_{IO} B^Q$ and B^Q has a polynomial-time solution, then A^O can be solved in polynomial time.*

Lemma 2 (Contrapositive of 1)**.** *If $A^O \leq_{IO} B^Q$ and no algorithm can solve A^O in polynomial time, then no algorithm can solve B^Q in polynomial time.*

Lemma 3. *If $A^O \leq_{IO} B^Q$ and $B^Q \leq_{IO} C^R$, then $A^O \leq_{IO} C^R$.*

1.3 Definitions

In this paper, we study combinatorial auctions and combinatorial public projects. In both of these games, there are n players and m items. Each player i has a valuation function v_i which maps subsets of $[m] = \{1, \ldots, m\}$ to nonnegative values. In an auction, we wish to find an allocation $S_1, \ldots, S_n$ such that $S_i \cap S_j = \emptyset$ for $i \neq j$ and $S_i \subseteq [m]$ for all i which maximizes $\sum_i v_i(S_i)$. In a public project, there is an additional parameter k, and we wish to find a set $S, |S| = k$ which maximizes $\sum_i v_i(S)$.

In both cases, the problems are parametrized by the class of functions from which v_i is chosen. The first class we look at is that of coverage valuations.

Definition 2 (Coverage valuation (PCOV))**.** *A coverage valuation v_i is defined by a universe U_i and m subsets $u_i^{(1)}, \ldots, u_i^{(m)}$ of this universe. The value of a set S is simply the size of the union of the corresponding subsets,*

$$v_i(S) = \left| \bigcup_{j \in S} u_i^{(j)} \right|.$$

We denote the public project problem with coverage valuation players by PCOV.

The other valuation class we study in this paper is capped-additive.

Definition 3 (Capped-additive valuation (AB,PB))**.** *A capped-additive valuation v_i is defined by m item values $v_i^{(1)}, \ldots, v_i^{(m)}$ and a budget cap b_i. The value of a set S is the sum of the item values, unless that sum exceeds the budget cap, in which case the value is the budget cap.*

$$v_i(S) = \min \left(\sum_{j \in S} v_i^{(j)}, b_i \right).$$

The auction problem with capped-additive players is denoted by AB and the public projects problem with capped-additive players by PB.

For these problems, we also define versions with constant numbers of players by adding a subscript with the number of players. So an auction with 2 capped-additive players is denoted by PB_2.

We study two instance oracles. The first is the demand oracle alluded to in Section 1.2.

Definition 4 (Demand oracle (dem)). *A demand oracle takes as input a list of prices $p_1, \ldots, p_m$ and for each player i, returns a set S maximizing $v_i(S) - \sum_{j \in S} p_j$. This definition is the same for both auctions and public projects.*

We also define a similar oracle which is more useful in public projects.

Definition 5 (k-demand oracle (kdem)). *A k-demand oracle takes as input a list of prices $p_1, \ldots, p_m$ and for each player i, returns a set $S, |S| = k$ maximizing $v_i(S) - \sum_{j \in S} p_j$. This oracle is only defined for public projects, as auctions have no parameter k.*

Finally, we define a maximal-in-range algorithm. These are the class of algorithms that can be implemented truthfully using the VCG mechanism. This is an important class, as the VCG mechanism is the only known general way to truthfully maximize social welfare, and remains an active area of study [5, 6, 7, 9].

Definition 6 (Maximal-in-range (MIR)). *An algorithm A is maximal-in-range if there exists a set S of possible allocations, and A always returns the allocation in S that maximizes the social welfare.*

We use techniques from [3] to show that our IONP-hardness results extend to stronger inapproximability results for maximal-in-range algorithms.

1.4 Our Results

Our results are comprised of several reductions, pictured in Figure 1. Reductions between problems shown are indicated by black-headed arrows. All other reductions are to existing NP-hard problems paired with trivial oracles. The IONP-hard public projects also cannot be approximated by a maximal-in-range algorithm to a factor better than $\sqrt{m}$ using the associated oracles unless NP has polynomial circuits.

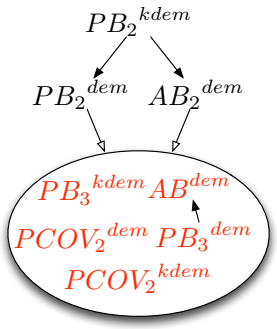

Figure 1: A summary of the results shown in this paper. Arrows indicate a reduction from the problem at the start of the arrow to the one at the end. The circled problems are IONP-hard.

2 Coverage valuations

One of the more surprising results in [3] is that $PCOV_1$ is NP-hard, and that no truthful mechanism can achieve an approximation ratio better than $\sqrt{m}$ unless $NP \subset P/poly$. Clearly, $PCOV_1^{kdem}$ has a polynomial time solution, as a single oracle call can determine the social welfare maximizing set. We show that $PCOV_2^{kdem}$ and $PCOV_2^{dem}$ are IONP-hard.

Theorem 1. *$PCOV_2^{dem}$ and $PCOV_2^{kdem}$ are both IONP-hard.*

Proof. We reduce from vertex cover on a 3-regular graph $G = (V, E)$. By Vizing's theorem, there exists an edge coloring of this graph using 4 colors such that no two adjacent edges share the same color. Furthermore, this proof is constructive and demonstrates how such a coloring can be found in polynomial time. We begin by coloring the edges using colors $1, 2, 3, 4$.

There are $m = |V|$ items. Let $V = \{v_1, \ldots, v_m\}$. Each item j corresponds to the vertex v_j. Player 1 has a valuation function defined by the universe U_1 which is the subset of E which is colored with colors 1 or 2. $u_1^{(j)}$ is the set of edges colored 1 or 2 incident upon v_j. So the value of a set S to player 1 is the number of edges colored 1 or 2 incident upon the vertices corresponding to the values in S. Player 2's value is defined similarly, but for edges colored 3 and 4.

The social welfare for allocating a set S is the number of edges incident upon vertices corresponding to values in S. So there is a social welfare of at least $|E|$ iff there is a vertex cover of size at most k.

Now that we have demonstrated a reduction, we need only show that the *dem* and *kdem* queries can be computed in polynomial time. A query consists of a list of item prices $p_1, \ldots, p_m$. For each player, we must find a set maximizing the difference between the number of appropriately colored edges covered and the sum of the item prices. We achieve this in two steps. We will focus on player 1, as the solution for player 2 can be found in the same manner by symmetry.

First, we examine each connected component in the subgraph consisting of the edges colored 1 or 2. Each of these components must be a path or a cycle. For each component, we compute the $\leq n$ values of what is the maximum utility for that component, choosing at most i items from the component, for i ranging from 0 to the size of the component. These values can be computed by a simple dynamic programming approach as follows.

For a path with ℓ vertices, order the vertices from one end to the other. For each prefix of i of the vertices and number j of items to be chosen, we compute two values. First, we compute the largest cover using j vertices which do not include the ith vertex. Next, we compute the largest cover using j vertices which do include the ith vertex. From these values, it is easy to compute the values for the first $i + 1$ vertices.

The maximum coverage not using vertex $i + 1$ is just the maximum of the two values for using vertices up to i. The maximum coverage using vertex $i + 1$ is either the coverage using $j - 1$ vertices up to i, but not including i, plus the number of edges covered by $i + 1$, or the coverage using $j - 1$ vertices up to and including i, plus the number of edges covered by $i + 1$ but not i (at most one).

For a cycle, we simply choose an arbitrary starting point and order it like a path. Now, we keep track of the maximum values for each prefix, but under the 4 conditions of whether or not we choose the first vertex in the prefix and whether or not we choose the last vertex of the prefix. This allows us to properly keep track of whether the edge joining the first and last vertices in the ordering has already been covered when computing the final step of the dynamic program.

Next, we combine these partial results for each connected component using more dynamic programming. Arbitrarily order the components. We compute the maximum coverage using j items from the first ℓ components by taking the maximum over all i of the maximum coverage using $j - i$ items from the first $\ell - 1$ components, plus the coverage using i items from component ℓ. After computing these values for ℓ from 1 to the number of components, we have the solution to the query.

As this process computes the maximum coverage using at most j items for every j, we can answer a *kdem* query by selecting the result for k items, and we can answer a *dem* query by choosing the maximum result over all j. □

We can now apply techniques from [3] to strengthen the above proof into a proof that no MIR algorithm can approximate ${PCOV_2}^{kdem}$ or ${PCOV_2}^{dem}$ to a factor better than $\sqrt{m}$.

Theorem 2. *No polynomial time maximal-in-range algorithm for ${PCOV_2}^{kdem}$ or ${PCOV_2}^{dem}$ has an approximation ratio of $m^{1/2-\epsilon}$ for any constant ϵ unless $NP \subset P/poly$.*

Proof. It is shown in [9] that any maximal-in-range mechanism for $PCOV_1$ (a special case of $PCOV_2$) must have a subset of the items of size m^α for some constant α that is allocated in every possible way. That is, for every subset of these m^α items, there is an allocation in the range that contains exactly these items, and the other $k - m^\alpha$ items in the allocation are from the other $m - m^\alpha$ items. We use this fact to embed the previous reduction in the m^α items in such a way that the maximal-in-range algorithm solves it exactly.

Assume by way of contradiction that an algorithm A for ${PCOV_2}^{kdem}$ or ${PCOV_2}^{dem}$ achieves an approximation ratio of $m^{1/2-\epsilon}$. Order the items such that the first m^α are the ones that are allocated by A in every possible way. This ordering corresponds to the polynomial advice in $P/poly$. Perform a reduction from a vertex cover instance with m^α vertices to these m^α items as in the proof of Theorem 1, but for each edge, create $k+1$ corresponding items in each set, rather than just 1. This has the effect of multiplying the social welfare by $k+1$. For the other $m-m^\alpha$ items, add $m-m^\alpha$ new items $n_{m^\alpha+1}, \ldots, n_m$ to player 1's valuation's universe U_1 and set $u_1^{(j)} = \{n_j\}$ for $j > m^\alpha$.

Now, A will find a set of k items with social welfare $(k+1)|E|+k-k'$ iff there is a set of k' vertices which covers the original graph. Clearly, if A finds a set with welfare $(k+1)|E|+k-k'$, then this must correspond to a covering of the graph in order to get the $(k+1)|E|$ term. The rest of the welfare must therefore come from having at least $k-k'$ items chosen from the other $m-m^\alpha$. So the covering must have had at most k' items.

Furthermore, if there exists such a covering, it will be found by A, as every possible subset of the m^α items is in As range and A is maximal-in-range. The rest will be filled in by the other $m-m^\alpha$ items, but the way in which these are chosen doesn't matter by construction. So A will solve $PCOV_2$.

Now, we need only see that we can still simulate the oracle queries in order to show that the oracles do not add any extra power beyond that of normal polynomial circuits. We already know how to simulate queries to determine the best coverage choosing j of the m^α items corresponding to the original reduction. We can also choose the best j of the $m-m^\alpha$ items by simply choosing the j with lowest price, as the added value of j of these items is the same to both players regardless of which j are chosen. So we can find the best j items overall by checking all i and finding the best i from the first m^α, and the best $j-i$ from the rest. Thus, we can still compute the *dem* and *kdem* queries in polynomial time. □

3 Three-player public projects

In this section, we examine ${PB_3}^{dem}$ and ${PB_3}^{kdem}$, and show that they are both IONP-hard and cannot be approximated to a factor better than $\sqrt{m}$ by an efficient maximal-in-range mechanism unless NP has polynomial circuits. We begin by proving the following theorem.

Theorem 3. *${PB_3}^{kdem}$ and ${PB_3}^{dem}$ are IONP-hard.*

We reduce from 3-dimensional matching (3DM). A 3DM instance consists of a set of triplets $T \subseteq [k] \times [k] \times [k]$, and the decision problem is does there exist a set $M \subseteq T$, $|M| = k$ such that for each $i \in [3], j \in [k]$, there is an element of M such that its ith coordinate is j?

Starting from such an instance of 3DM, we create the following instance of PB_3. Let $\tau = 2^{\lceil \log_2 |T| \rceil + 1}$ (so $\tau > |T|$). For each item $(\alpha_1, \alpha_2, \alpha_3) \in T$, create an item i such that

- Player 1 values i at $\tau^{3k+1} + \tau^{2k+\alpha_1} - \tau^{k+\alpha_2}$
- Player 2 values i at $\tau^{3k+1} + \tau^{k+\alpha_2} + \tau^{\alpha_3}$
- Player 3 values i at $\tau^{3k+1} - \tau^{2k+\alpha_1} - \tau^{\alpha_3}$

and

- Player 1 has budget $k \cdot \tau^{3k+1} + \sum_{j \in [k]} \tau^{2k+j} - \tau^{k+j}$
- Player 2 has budget $k \cdot \tau^{3k+1} + \sum_{j \in [k]} \tau^{k+j} + \tau^j$
- Player 3 has budget $k \cdot \tau^{3k+1} + \sum_{j \in [k]} -\tau^{2k+j} - \tau^j$

Now, there exists a set $M \subseteq T$ of size k covering each coordinate iff there is a set of k items that gives each player value equal to its budget.

Lemma 4. *The above reduction is a valid reduction from 3-dimensional matching to public projects with 3 capped-additive players.*

Proof. Assume that there exists a set of k items S such that all budgets are met. If player 1's budget is exceeded, then either $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{2k+\alpha_1} > \sum_{j\in[k]} \tau^{2k+j}$ or $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{k+\alpha_2} < \sum_{j\in[k]} \tau^{k+j}$. In the former case, this would cause player 3 to be under budget, a contradiction. In the latter, it would cause player 2 to be under budget, also a contradiction. So we have $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{2k+\alpha_1} = \sum_{j\in[k]} \tau^{2k+j}$ and $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{k+\alpha_2} = \sum_{j\in[k]} \tau^{k+j}$. Now, if $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{\alpha_3} > \sum_{j\in[k]} \tau^{j}$, then player 3 is under budget, and if $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{\alpha_3} < \sum_{j\in[k]} \tau^{j}$ then player 2 is under budget. So $\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{\alpha_3} = \sum_{j\in[k]} \tau^{j}$.

Thus, we have

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{2k+\alpha_1} = \sum_{j\in[k]} \tau^{2k+j} \quad (1)$$

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{k+\alpha_2} = \sum_{j\in[k]} \tau^{k+j} \quad (2)$$

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{\alpha_3} = \sum_{j\in[k]} \tau^{j} \quad (3)$$

Which implies that for every j, there is some $\alpha_1 = j$, some $\alpha_2 = j$ and some $\alpha_3 = j$, implying a 3-dimensional matching. Now, we note that if a 3-dimensional matching exists, then choosing the corresponding items will result in all 3 budgets being met. So a 3-dimensional matching exists iff there is some set of items for which the social welfare is equal to the sum of the players' budgets. □

In order to complete the proof of Theorem 3, we now need show how demand and k-demand queries for instances produced by the reduction can be computed in polynomial time. We show that for any $\ell \in [m]$, the utility maximizing set of size ℓ for any player can be computed in polynomial time. Using this, it is trivial to compute demand and k-demand queries by taking the maximum utility over all ℓ or just for $\ell = k$, respectively.

Lemma 5. *For any set of item prices $p_1, \ldots, p_m$ and any ℓ, it is possible to compute the utility maximizing set of size ℓ for any player.*

Proof. Each player has a value function of the form

$$\sum_{(\alpha_1,\alpha_2,\alpha_3)\in S} \tau^{3k+1} \pm \tau^{(3-i)k+\alpha_i} \pm \tau^{(3-j)k+\alpha_j}$$

for some $1 \le i < j \le 3$. For any $\ell < k$, the player's budget cannot be reached, so the utility maximizing set consists of the ℓ items with maximum individual utility. For $\ell > k$, the budget will be reached by any set of items, so the ℓ items of smallest price are chosen. We now need only to examine the $\ell = k$ case.

In this case, we can ignore the τ^{3k+1} terms, as the will always sum to $k \cdot \tau^{3k+1}$. Now, for each possible value α_i^* for α_i from k to 1, we consider 3 cases. Either zero, one or more than one items with ith coordinate α_i are chosen.

If zero, then we know that if the sign is positive, the budget will be exceeded, and if negative, the budget will not be reached. In either case, it is easy to choose the items with smaller α_i values by either maximizing individual utility or minimizing prices.

If more than one item, we also know that the budget will either be exceeded or not met, so we choose the 2 of lowest price or highest utility, then choose items to fill out the rest of the k with α_i values less than or equal to α_i^*.

Finally, if we choose exactly 1 such item, we move onto the next α_i^* value and choose exactly 1 such item by lowest price or highest utility when we know that the budget will be reached or not.

After we have iterated through all k values of α_i^*, we move on to values α_j^* of α_j. Again we have 3 possibilities, zero, one or more than one items. With one item, we still move on. With zero or more than one items, we again know whether the budget will be exceeded, but it is more difficult to determine how to maximize utility. We use weighted bipartite matching for this purpose.

The graph has one side of k items corresponding to the values of α_i which must each be covered once. On the other side, there are $0, 1$ or k items corresponding to each possible value of α_j. For each $\alpha_j > \alpha_j^*$, there will be exactly one item, as exactly one must be covered. For each $\alpha_j < \alpha_j^*$, there are k items, as we

have not yet determined how many times each of these should be covered. For α_j^*, there are either 0 or k items, depending on whether we want zero or more than one items with this value.

For each pair of nodes corresponding to values α_i and α_j, we add an edge if there is an item with corresponding ith and jth coordinates. We weight the edge by either the utility of that item or the negative of its price, depending on whether we want to maximize utility or minimize price. If there is more than one corresponding item, choose the largest such weight, as preferring this item can only increase the welfare. If necessary to ensure positivity, shift all edge weights to positive values by adding the same value to all of them.

For all items with $\alpha_j > \alpha_j^*$, we know that exactly one of the incident edges must be chosen. To ensure this, add the same large value (say, the sum of all edge weights) to the weight of all incident edges on such nodes. If α_j^* is supposed to be chosen more than once, then do the same for 2 of its nodes.

Finally, compute a maximum weighted k matching. This will result in a set of edges corresponding to an assignment that covers each α_i exactly once, each $\alpha_j > \alpha_j^*$ exactly once, and α_j^* either 0 or more than one time as appropriate while maximizing utility.

The last step is to notice that after all α_j are set to be chosen exactly once, the budget is exactly reached, so the same procedure applies, but with every α_j appearing once in the bipartite graph. Now, we just take the $6k$ assignments computed and choose the one of maximum utility to complete the computation of the oracle result. Since each of the $6k$ computations occurs in polynomial time, the entire procedure terminates in polynomial time. □

As with Theorem 2, $PCOV_2$, the proof of Theorem 3 can be modifed to show that ${PB_3}^{kdem}$ and ${PB_3}^{dem}$ cannot be approximated well by maximal-in-range mechanisms.

Theorem 4. *No polynomial time maximal-in-range algorithm for ${PB_3}^{kdem}$ or ${PB_3}^{dem}$ has an approximation ratio of $m^{1/2-\epsilon}$ for any constant ϵ unless $NP \subset P/poly$.*

Proof. [9] showed that any algorithm for a class of problems including PB_3 which achieves an approximation of at least $m^{1/2-\epsilon}$ must have a set of at least m^α items which are allocated in every possible way for some constant α. As in the proof of Theorem 2, we embed the reduction into these m^α items, and design valuations for the other $m - m^\alpha$ items such that exactly $k - k'$ of them will be chosen in an optimal allocation and the can be considered separately from the m^α items from the original reduction when answering *dem* and *kdem* queries.

The $m - m^\alpha$ items not corresponding to those in the initial proof have a very simple valuation. Player 1 values them at τ^{4k}, and players 2 and 3 value them at 0. Player 1's budget is increased by $(k - k')\tau^{4k}$.

If more than $k - k'$ of these extra items are chosen, then fewer than k' of the original items are chosen, and therefore players 2 and 3 do not reach their budgets. If fewer than $k' - k$ of these extra items are chosen, then player 1's budget cannot be reached. So in the optimal allocation, k' of the original m^α items are chosen, and the other $k - k'$ are chosen arbitrarily from the rest. So this algorithm will find the optimal solution, which will in turn solve the original instance.

Now, we need to see that the ability to compute demand sets of size ℓ for any ℓ is preserved. This is easy, as we need only determine the best way to choose i items from the first m^α and $\ell - i$ from the rest. The contribution from the $m - m^\alpha$ items to any player's value is the same regardless of choice, so the $\ell - i$ of these that are chosen must be those of lowest price. If $\ell - i < k - k'$, then no matter the choice of the rest, the budget is not exceeded, so simply choose the i of greatest utility. If $\ell - i > k - k'$, then the budget is exceeded by these items alone, so the other i chosen must be those of lowest price.

Otherwise, $\ell - i < k - k'$. This leaves the remaining budget and item prices for the m^α items the same as in the original reduction. Therefore, the proof of Lemma 5 demonstrates how to compute the solution to this part of the query. By combining these results, we have computed the query response. □

4 Many player auctions

In this section, we show that $PB^{dem} \leq_{IO} AB^{dem}$, proving that capped-additive auctions with demand queries are IONP-hard.

Theorem 5. $PB^{dem} \leq_{IO} AB^{dem}$

We prove Theorem 5 via the following reduction.

Reduction 1. *We begin with an instance of PB^{dem} in which player i has value $v_j^{(i)}$ for item j and budget cap b_i, and k items are to be chosen. We produce an instance of AB^{dem} with $mn+k$ items and $n+m$ players.*

The first mn items, we label with pairs (i,j) for $1 \leq i \leq n, 1 \leq j \leq m$. The final k items, we label $\alpha_1, \ldots, \alpha_k$. For $i = 1, \ldots, n$,

- *Player i has value $v_j^{(i)}$ for item (i,j) for all j, and value 0 for all other items*
- *Player i has budget b_i*

Let $B = \sum_i b_i$. For $j = 1, \ldots, m$,

- *Player $n+j$ has value B for items (i,j), $i = 1, \ldots, n$, value nB for items α_i, $i = 1, \ldots, k$ and value 0 for all other items*
- *Player $n+j$ has budget nB*

The idea behind reduction 1 is that each item from the public project is turned into n new items and that for each original item, either all of the first n players get a copy, or one of the last m players receives all of the copies. Any of the last m players that does not receive all copies of his item instead receives one of the items $\alpha_1, \ldots, \alpha_k$.

Lemma 6. *Let $S_1, \ldots, S_{m+n}$ be an allocation to an auction produced by Reduction 1. If $S_1, \ldots, S_{m+n}$ has social welfare greater than mnB, then for all but k values j from 1 to m, every item $(i,j), i = 1, \ldots, n$ is allocated to player $n+j$.*

Proof. Suppose by way of contradiction that there exists an allocation with social welfare greater than mnB, yet there are $k+1$ players $n+j$ who don't receive all of the items (i,j). At most k of these players receive some item α_i, and thus have value nB. Thus, one of these players has value at most $(n-1)B$, as it receives at most $n-1$ items with value each B, and has 0 value for all other received items. So the social welfare from players $n+j, j = 1, \ldots, m$ is at most $mnB - B$. The social welfare from players $i = 1, \ldots, n$ is at most B, for a total social welfare of at most mnB, contradicting that the social welfare is greater than mnB. □

Lemma 7. *A public project has social welfare V iff the auction produced by Reduction 1 has social welfare $V + mnB$*

Proof. If the public project has social welfare V, let S be the allocation achieving that welfare. The following allocation has social welfare $V + mnB$ in the auction. For every $i = 1, \ldots, n$, give player i the items $\{(i,j) : j \in S\}$. For every $j \in [m] \backslash S$, give player $n+j$ the items $\{(i,j) : i \in [n]\}$. For the remaining k players $n+j, j \in S$, give them each one of the items $\alpha_i, i = 1, \ldots, k$.

The players $i = 1, \ldots, n$ have values equal to that from the public project, so they contribute V to the social welfare. The players $n+j, j = 1, \ldots, m$ each have value nB, for a total contribution of mnB. Thus, the social welfare of this allocation is $V + mnB$.

Now, suppose that the auction has social welfare $V + mnB$ for some $V > 0$. By Lemma 6, for all but k values j from 1 to m, every item $(i,j), i = 1, \ldots, n$ is allocated to player $n+j$. Let S be the set of k items j from 1 through m such that not every (i,j) is allocated to player $n+j$. As players $n+j, j = 1, \ldots, m$ have total value at most mnB, players 1 through n must have total value at least V. Furthermore, player i has value at most $\sum_{j \in S} v_j^{(i)}$, as the only items that it gets value from that are not given to players $n+j$ are $(i,j), j \in S$. Thus, $\sum_i \sum_{j \in S} v_j^{(i)} \geq V$, so allocation S has social welfare at least V in the original public project. □

Lemma 8. *The demand oracle in a public project can be used to compute answers to demand queries in the auction resulting from Reduction 1.*

Proof. To simulate a query to player i, simply query player i in the public project with price for item j taken from the price (i, j), as these are the only items the player values. Add all other items with negative price to arrive at the query result.

To simulate a query to player $n + j$, there are only a few relevant choices for value. Either the α_i of minimum price is chosen, or the ℓ items (i, j) of lowest price for some ℓ. In addition to these, add all items of negative price to arrive at the proper query result. Clearly, one of these choices maximizes utility. □

As we have shown a valid reduction and the ability to simulate oracle queries, lemmas 6, 7 and 8 together prove Theorem 5.

5 k-demand versus demand

In the previous sections, we demonstrated IONP-hardness. Here, we show how IONPreductions can be used to better understand the relationships between problems of indeterminate hardness. The three problems we examine are ${PB_2}^{kdem}$, ${PB_2}^{dem}$ and ${AB_2}^{dem}$. We demonstrate that ${PB_2}^{kdem}$ can be reduced to the other two, and is therefore a potentially easier problem.

5.1 Public projects self-reduction

In this section we show a self-reduction between PB_2 and itself that allows for *dem* queries to be simulated by *kdem* queries, showing that the *dem* oracle is no more powerful than the *kdem* oracle.

Theorem 6. ${PB_2}^{kdem} \leq_{IO} {PB_2}^{dem}$.

Reduction 2. *We start with an instance of the combinatorial public projects problem with* 2 *capped-additive players. Player i has value $v_j^{(i)}$ for item j and budget b_i. Let $D = 2m \max(b_1, b_2)$. The reduction produces an instance with 2 capped-additive players, where*

- *Player i has value $w_j^{(i)} = v_j^{(i)} + D$ for item j*
- *Player i has budget $c_i = b_i + kD$.*

We assume without loss of generality that for all i, j, $v_j^{(i)} \leq b_i$.

Lemma 9. *A public project has social welfare V iff the public project produced by Reduction 2 has social welfare at least $V + 2kD$.*

Proof. Consider player i's value for a set S of size k. In the original public project, the value is

$$\min\left(\sum_{j\in S} v_j^{(i)}, b_i\right).$$

In the instance produced by the reduction, player i's value is

$$\begin{aligned}\min\left(\sum_{i\in S} w_j^{(i)}, c_i\right) &= \min\left(\sum_{i\in S} v_j^{(i)} + D, b_i + kD\right)\\ &= \min\left(\sum_{i\in S} v_j^{(i)}, b_i\right) + kD\end{aligned}$$

So summing over both players, if there is a social welfare of V for set S in the original auction, there is a social welfare of $V + 2kD$ for S in the auction produced by Reduction 2. □

Lemma 10. *The kdem oracle for a public project can be used to answer dem queries for the public project resulting from Reduction 2.*

Proof. Consider the result of a *dem* query for player i with prices p_j. There are 3 possible cases:

Case 1: It returns a set S, $|S| > k$. In this case, the value of the set to player i is at least $\min((k+1)D, b_i + kD)$. As $D > b_i$, the value of the player is exactly $b_i + kD$, regardless of the items chosen. Thus, the query will simply return a set of size at least $k+1$ of minimum price. This is either the $k+1$ items of lowest price, or all items of negative price if there are more than $k+1$ of these.

Case 2: It returns a set S, $|S| < k$. In this case, the value of the set to player i is at most $\min((k-1)(D+b_i), b_i + kD)$. As $(k-1)b_i < D$, this is simply $(k-1)(D+b_i)$. So the budget does not affect the value in this case. Thus, S consists of the at most $k-1$ items of maximum positive utility.

Case 3: It returns a set S, $|S| = k$. In this case, player i's utility is

$$\min\left(\sum_{j\in S} v_j^{(i)} + D, b_i + kD\right) - \sum_{j\in S} p_j = \min\left(\sum_{j\in S} v_j^{(i)}, b_i\right) + kD - \sum_{j\in S} p_j$$

to find a set of size k maximizing this utility, we simply need a set of size k maximizing $\min\left(\sum_{j\in S} v_j^{(i)}, b_i\right) - \sum_{j\in S} p_j$, which is what the original *kdem* oracle will give us with input p_j and player i.

So by computing the results for each of these three cases, we can choose the one with maximum utility and return it as the result of the *dem* query. □

As we have a valid reduction, and the *kdem* oracle can simulate the *dem* oracle on instances produced by the reduction, we have proven Theorem 6.

5.2 Public projects to auctions

In this section, we reduce from PB_2 to AB_2 in such a way that the *kdem* oracle can again be used to simulate the *dem* oracle.

Theorem 7. ${PB_2}^{kdem} \leq_{IO} {AB_2}^{dem}$.

Reduction 3. *We begin with a public project in which the goal is to choose k items, and player 1 has values $v_1, \ldots, v_m$ and budget b_1, while player 2 has values $w_1, \ldots, w_m$ and budget b_2. Assume without loss of generality that $b_1 \geq v_i$ and $b_2 \geq w_i$.*

Let $W = \sum_i w_i$ and $B = \max(b_1, b_2, W) + 1$.

- *Player 1 has value $v_i + mB$ for item i*
- *Player 1 has budget $b_1 + mkB$*
- *Player 2 has value $mB - w_i$ for item i*
- *Player 2 has budget $b_2 + m(m-k)B - W$*

The original public project has social welfare at least V iff the auction produced has social welfare at least $V + m^2B - W$.

Lemma 11. *In an optimal allocation for the auction produced by Reduction 3, player 1 gets k items and player 2 gets $m-k$ items.*

Proof. Suppose by way of contradiction that player 1 gets fewer than k items. Let S be the set of items player 1 gets. Player 2 has value at most $b_2' = b_2 + m(m-k)B - W$.

Player 1 has value

$$\sum_{i\in S} v_i + mB \leq (k-1)B + (k-1)mB,$$

so the social welfare is at most

$$\begin{aligned} b_2' + (k-1)B + (k-1)mB &= b_2 + m(m-k)B - W + (k-1)B + (k-1)mB \\ &= b_2 + m(m-1)B + (k-1)B - W \\ &\leq b_2 + m(m-1)B + (m-1)B - W \\ &= b_2 + (m+1)(m-1)B - W \\ &= b_2 + (m^2-1)B - W \\ &< m^2B - W. \end{aligned}$$

However, any allocation for which player 1 gets k items and player 2 gets $m-k$ items has social welfare of at least $k \cdot mB + (m-k)mB - \sum_i w_i \geq m^2B - W$.

Now, suppose by way of contradiction that player 1 receives more than k items. Then player 1 has value at most $b_1' = b_1 + mkB$. Player 2 receives a set S of at most $m-(k+1)$ items, so player 2 has value at most

$$\begin{aligned} \sum_{i \in S} w_i' &= \sum_{i \in S} mB - w_i \\ &\leq \sum_{i \in S} mB \\ &\leq (m-k-1)mB \end{aligned}$$

so the social welfare is at most

$$\begin{aligned} b_1 + mkB + (m-k-1)mB &= m^2B + b_1 - mB \\ &< m^2B - (m-1)B \end{aligned}$$

For $m > 1$, $(m-1)B > W$, so

$$m^2B - (m-1)B < m^2B - W.$$

This is again less than the minimum welfare of $m^2B - W$ for giving k items to player 1 and $m-k$ items to player 2. □

Lemma 12. *A public project has social welfare at least V iff the auction produced using Reduction 3 has social welfare at least $V + m^2B - W$.*

Proof. Suppose the original public project has social welfare V. Let S be a set of k items achieving this social welfare. Give S to player 1, and S^C to player 2. Let $V = V_1 + V_2$, where players 1 and 2 have values V_1 and V_2 for S, respectively. In the auction allocation described above, player 1 will have value

$$\begin{aligned} V_1' &= \min\left(\sum_{i \in S} v_i', b_1'\right) \\ &= \min\left(\sum_{i \in S} v_i + mB, b_1 + mkB\right) \\ &= \min\left(\sum_i v_i, b_1\right) + mkB \\ &= V_1 + mkB \end{aligned}$$

for S and player 2 will have value

$$\begin{aligned} V_2' &= \min\left(\sum_{i\in S^C} w_i', b_2'\right) \\ &= \min\left(\sum_{i\in S^C} mB - w_i, \right. \\ &\qquad \left. b_2 + m(m-k)B - W\right) \\ &= \min\left(\sum_{i\in S^C} -w_i, b_2 - W\right) + m(m-k)B \\ &= \min\left(\sum_{i\in S} w_i, b_2\right) - W + m(m-k)B \\ &= V_2 - W + m(m-k)B \end{aligned}$$

So the social welfare of this is

$$\begin{aligned} V_1' + V_2' &= V_1 + mkB + V_2 - W + m(m-k)B \\ &= V_1 + V_2 + m(m-k+k)B - W \\ &= V + m^2B - W \end{aligned}$$

By Lemma 11, we know that player 1 gets exactly k items and player 2 gets $m-k$ items. Let S be the set allocated to player 1 and S^C be the set allocated to player 2. Player 1 has value

$$\begin{aligned} \min\left(\sum_{i\in S} v_i', b_1'\right) &= \min\left(\sum_{i\in S} v_i + mB, b_1 + mkB\right) \\ &= \min\left(\sum_{i\in S} v_i, b_1\right) + mkB \end{aligned}$$

and player 2 has value

$$\begin{aligned} \min\left(\sum_{i\in S^C} w_i', b_2'\right) &= \min\left(\sum_{i\in S^C} mB - w_i, b_2 + m(m-k)B - W\right) \\ &= \min\left(\sum_{i\in S} w_i, b_2\right) + m(m-k)B - W \end{aligned}$$

for a total social welfare of

$$\min\left(\sum_{i\in S} v_i, b_1\right) + \min\left(\sum_{i\in S} w_i, b_2\right) + m^2B - W$$

so $V' = \min\left(\sum_{i\in S} v_i, b_1\right) + \min\left(\sum_{i\in S} w_i, b_2\right)$, and therefore the original public project has social welfare at least V'. □

Now we need only see that the *kdem* oracle on the original public project can be used to answer oracle queries for *dem* in the auction resulting from the reduction in polynomial time.

Lemma 13. *The kdem oracle for a public project can be used to answer dem queries in the auction instance resulting from Reduction 3.*

Proof. Consider the result S of the *dem* query. We will examine 6 cases, 3 for each player.

Case 1a: *dem* returns a set S for player 1, $|S| > k$. In this case, the value to player 1 of any set of size $k+1$ or more is at least $(k+1)mB > b'_1$, so any set of size $k+1$ of more will give the same value. So to maximize the utility, simply choose the $k+1$ items of lowest price. If there are more than $k+1$ items of negative price, choose all items of negative price.

Case 1b: *dem* returns a set S for player 1, $|S| < k$. In this case, the value to player 1 of any set of size $k-1$ or less is at most $(k-1)B + (k-1)mB < kmB < b'_1$. So the budget does not affect the value. Thus, query result is simply the at most $k-1$ items of highest non-negative utility.

Case 1c: *dem* returns a set S for player 1, $|S| = k$. In this case, we have already seen that player 1's utility is $\min\left(\sum_{i\in S} v_i, b_1\right) + mkB$, so the query result is a set of size k maximizing $\min\left(\sum_{i\in S} v_i, b_1\right) - \sum_{i\in S} p_i$, which is exactly what *kdem* gives us as the result for player 1 with prices $p_1, \ldots, p_k$.

Case 2a: *dem* returns a set S for player 2, $|S| > m-k$. In this case, the value to player 2 of any set of size $m-k+1$ or more is at least $(m-k+1)mB - W > b_2 + m(m-k)B - W = b'_2$, so player 2 has value b'_2 regardless of what set is chosen. So as in case 1a, simply choose a set of size at least $m-k+1$ with lowest cost.

Case 2b: *dem* returns a set S for player 2, $|S| < m-k$. In this case, the value to player 2 of any set of size $m-k-1$ or less is at most $(m-k-1)mB \leq m(m-k)B - W < b'_2$. So the budget does not matter. Thus, as in case 1b, we simply choose the at most $m-k-1$ items of highest non-negative utility.

Case 2c: *dem* returns a set S for player 2, $|S| = m-k$. We have already seen that player 2's utility is $\min\left(\sum_{i\in S^C} w_i, b_2\right) + m(m-k)B - W$. So we need only find a set S of size k maximizing $\min\left(\sum_{i\in S^C} w_i, b_2\right) - \sum_{i\in S} p_i$. If we make a query to *kdem* with price $-p_i$ for item i, we get a set T maximizing

$$\min\left(\sum_{i\in T} w_i, b_2\right) - \sum_{i\in T} -p_i = \min\left(\sum_{i\in T} w_i, b_2\right) - \sum_{i\in T^C} p_i + \sum_{i\in[m]} p_i,$$

which in turn maximizes $\min\left(\sum_{i\in T} w_i, b_2\right) - \sum_{i\in T^C} p_i$, as $\sum_{i\in[m]} p_i$ does not depend on the choice of T. Setting $S = T^C$, we have found a set maximizing $\min\left(\sum_{i\in S^C} w_i, b_2\right) - \sum_{i\in S} p_i$.

So in order to solve a *dem* query, we simply need to compute the 3 cases for each player, then choose the best results. □

Lemma 12 shows that we have a valid reduction, and lemma 13 shows that we can simulate the oracle after the reduction, completing the proof of theorem 7.

6 Conclusions

We have developed a model of instance oracles and shown how they can be used to demonstrate that even assuming that each player has the ability to compute worst-case hard functions relevant to their valuations, social welfare maximization can remain hard.

An interesting direction for future research is to show how instance oracle reductions can be used for hardness of approximation for general algorithms or for all truthful mechanisms, rather than just maximal-in-range mechanisms. It would also be interesting to study other games in this model, as well as different instance oracles, including ones which can only provide approximate solutions as query results.

Acknowledgements

Thanks to Shaddin Dughmi, Michael Schapira, John Ledyard and Chris Umans for helpful discussions and feedback in developing the ideas that went into this paper.

References

[1] Sushil Bikhchandani, Shurojit Chatterji, Ron Lavi, Ahuva Mualem, Noam Nisan, and Arunava Sen. Weak monotonicity characterizes deterministic dominant-strategy implementation. *Econometrica*, 74(4):1109–1132, 2006.

[2] Dave Buchfuhrer, Shaddin Dughmi, Hu Fu, Robert Kleinberg, Elchanan Mossel, Christos H. Papadimitriou, Michael Schapira, Yaron Singer, and Christopher Umans. Inapproximability for VCG-based combinatorial auctions. *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '10)*, pages 518–536, 2010.

[3] Dave Buchfuhrer, Michael Schapira, and Yaron Singer. Computation and incentives in combinatorial public projects. *Proceedings of the 11th ACM Conference on Electronic Commerce (EC '10)*, pages 33–42, 2010.

[4] Shahar Dobzinski and Noam Nisan. Limitations of VCG-based mechanisms. *Proceedings of the thirty-ninth annual ACM symposium on Theory of computing (STOC '07)*, pages 338–344, 2007.

[5] Shahar Dobzinski, Noam Nisan, and Michael Schapira. Approximation algorithms for combinatorial auctions with complement-free bidders. *Mathematics of Operations Research*, 35(1):1–13, 2010.

[6] Shahar Dobzinski and Mukund Sundararajan. On characterizations of truthful mechanisms for combinatorial auctions and scheduling. *Proceedings of the 9th ACM conference on Electronic commerce (EC '08)*, pages 38–47, 2008.

[7] Ron Lavi, Ahuva Mu'alem, and Noam Nisan. Towards a characterization of truthful combinatorial auctions. *Proceedings of the 44th Annual IEEE Symposium on Foundations of Computer Science (FOCS '03)*, page 574, 2003.

[8] Noam Nisan and Amir Ronen. Computationally feasible VCG mechanisms. *Journal of Artificial Intelligence Research (JAIR)*, 29:19–47, 2007.

[9] Christos Papadimitriou, Michael Schapira, and Yaron Singer. On the hardness of being truthful. *Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science (FOCS '08)*, 2008.