Implementing an Algorithm to Determine Feasibility of and Implement Reduced Forms of Independent Bidder, Single Item Auctions

Yuxuan Ke
Department of Computer Science
Yale College
New Haven, CT 06520, U.S.A.
yuxuan.ke@yale.edu

May 1, 2019

Abstract

We implement an algorithm by [1] that determines the feasibility of a reduced form mechanism for a given single item, bidder independent auction. In doing so we work out some necessary details for the implementation of the algorithm not covered in the cited work. Based on the algorithm’s output, we then formulate some conjectures for special cases of the reduced form.

1 Introduction and Background

Consider a situation whereby we have a seller seeking to auction a single item, and multiple bidders, each with a finite number of possible bids. Each bidder is assumed to sample from their set of possible bids using a known probability distribution, and the bidders bid independently. The seller’s objective would then be to design an auction mechanism that would maximize the expected value of some objective function over the vector of submitted bids.

The details of how the mechanism would be defined are beyond the scope (and not the concern) of this work, but we note that in order to describe the resulting auction mechanism, the seller has to provide an allocation rule that, given a vector of bids by the bidders, specifies how to allocate the item to the bidders. This can be thought of as a probability distribution over the bidders. However, the number of possible bid vectors is
exponential in the number of bidders, which makes it unwieldy and impractical to specify a probability distribution over the bidders for each possible bid vector.

For reasons beyond the scope of this work, it suffices to provide a reduced form of the mechanism, which states for each bid by each bidder, the conditional probability that the bidder wins the item given that they submit that bid, over all other possible bid vectors submitted by the remaining bidders. However, not every reduced form can be feasibly implemented, and the problem arises of determining whether a given reduced form can be feasibly implemented, and if so, how. An algorithm to decide this and find an implementation is given by [1], and implementing this algorithm is the main focus of this work.

The primary and sole reference of this work is [1], which may be referred to for further clarification regarding any of the concepts discussed.

2 Preliminaries and Notation

For our auction, there will be a single item and a single seller who wishes to auction an item to one of \( m \) bidders. Each bidder \( i \) will be assumed to have \( k_i \) distinct possible bids, which will be denoted by \( A_{i,j} \) for \( j \in [k_i] \). In addition, we denote by \( p_{i,j} \) the probability that bidder \( i \) submits the bid \( A_{i,j} \). Denote by \( \mathcal{A} = \{ A_{i,j} \mid i \in [m], j \in [k_i] \} \) the set of all types, and let \( \pi : \mathcal{A} \rightarrow [0,1] \) denote the virtual form of interest, and without loss of generality assume that the \( A_{i,j} \) are ordered within each \( i \) according to \( \pi \), so for all \( i \),

\[
\pi(A_{i,1}) \geq \pi(A_{i,2}) \geq \ldots \geq \pi(A_{i,k_i-1}) \geq \pi(A_{i,k_i})
\]

For each \( \pi \), the corresponding virtual mechanism \( \hat{\pi} : \mathcal{A} \rightarrow [0,1] \) is defined as follows:

\[
\hat{\pi}(A_{i,j}) = \pi(A_{i,j}) \sum_{l=j}^{k_i} p_{i,l}
\]

We say a reduced form \( \pi \) is feasible if there exists an allocation mechanism that on every possible bid vector outputs a probability distribution on the bidders (in order to allocate the item) such that the marginal probabilities of each bidder obtaining the item with a given bid are equal to the reduced form probabilities. Theorem 4 of the reference work is can then be re-stated as follows:

**Theorem 2.1.** Let \( \pi \) be a reduced form and \( \hat{\pi} \) be defined as above. Let \( S_x = \{ A_{i,j} \in \mathcal{A} : \hat{\pi}(A_{i,j}) \geq x \} \). Then \( \pi \) is feasible if and only if

\[
\sum_{A_{i,j} \in S_x} p_{i,j} \cdot \pi(A_{i,j}) \leq 1 - \prod_{i} \left( 1 - \sum_{j:A_{i,j} \in S_x} p_{i,j} \right)
\]
We also require the definition of a hierarchical mechanism as described in the reference work:

**Definition 2.2.** A hierarchical mechanism is a mechanism induced by the following process: fix some function $H : A \rightarrow [|A| + 1]$. On a bid vector $(A_{1,j_1}, A_{2,j_2}, ..., A_{m,j_m})$, if $H(A_{i,j_i}) = |A| + 1$ for all $i \in [m]$, the seller does not give the item to any bidder. Otherwise the seller allocates the item to the bidder $i$ whose bid had the lowest value of $H(A_{i,j_i})$, picking randomly in the event of a tie.

We note the special value $|A| + 1$, this corresponds to LOSE in the reference work.

In addition we say that a hierarchical mechanism is *strict* if its restriction to the preimage of $[|A|]$ is injective; in other words the preimage of each element of $[|A|]$ is a set with at most one element. We also say that a hierarchical mechanism is *partially-ordered with respect to π* if for all $i, j_1, j_2$ such that $1 \leq j_1 \leq j_2 \leq k_i$, we have $H(A_{i,j_1}) \leq H(A_{i,j_2})$.

Theorem 5 from [1] then states that every feasible reduced form $\pi$ can be expressed as a probability distribution over at most $|A| + 1$ strict, partially ordered w.r.t $\pi$ hierarchical mechanisms, and an algorithm is provided to determine the exact distribution, as well as the choice of hierarchical mechanisms. It is this algorithm that we have implemented, which we will discuss further in the subsequent section.

### 3 Implementation

This section will be divided into two parts. First we will review the algorithm as it is described in [1], and then we will examine some of the implementation details and changes made to the originally specified algorithm.

#### 3.1 Review of Algorithm

The algorithm described by the reference work is based on a geometric view of the reduced form as a vector in $\mathbb{R}^{|A|}$. It is noted that the strict, partially ordered w.r.t $\pi$ hierarchical mechanisms form the corners of a polytope, and that $\pi$ lies on the interior of this polytope. Then, by Carathéodory’s theorem, $\pi$ can be expressed as a linear combination of at most $|A| + 1$ corners of the polytope.

The polytope has exponentially many corners and boundary hyperplanes, so the algorithm provided requires only a *separation oracle* (SO) which, given a point, returns a boundary hyperplane separating the point and the interior of the polytope (if no such hyperplane exists, the point must be in the interior, and SO returns *None*), and a *corner oracle* (CO) which, when given a set of boundary hyperplanes of the polytope, returns
a corner of the polytope that is in the intersection of these boundary hyperplanes. The algorithm proceeds as follows, where variables in bold represent vectors:

**Algorithm 1** Algorithm for writing \( \mathbf{x} \) as a combination of at most \( n + 1 \) corners.

Initialize: \( i \leftarrow 1, y \leftarrow 0, z \leftarrow x, c_i \leftarrow 0, a_i \leftarrow 0, E = \emptyset \)

Invariants: \( c = \sum_{i} c_i, y = \frac{1}{c} \sum_{i} c_i a_i \) (or 0 if \( c = 0 \)) , \( c y + (1 - c) z = x \)

if \( SO(x) \neq \text{None} \) then
  Output False
end if

while \( c < 1 \) do
  \( a_i \leftarrow \text{CO}(E) \)
  if \( a_i = z \) then
    \( c_i \leftarrow 1 - c \)
    Output \( c_1, \ldots, c_{n+1}, a_1, \ldots a_{n+1} \)
  else
    \( d \leftarrow \max\{d \mid \text{SO}((1 + d)z - da_i) = \text{None}\} \)
    \( E \leftarrow E \cup \text{SO}((1 + d + \epsilon)z - (d + \epsilon)a_i) \)
    \( c_i \leftarrow (1 - \frac{1}{1+c}) (1 - c) \)
    \( z \leftarrow (1 + d)z - da_i \)
    \( y \leftarrow \frac{c}{c+c_i} y + \frac{c_i}{c+c_i} a_i \)
    \( c \leftarrow c + c_i \)
    \( i \leftarrow i + 1 \)
  end if
end while

Informally, this algorithm works by maintaining the “running total” \( y = \frac{1}{c} \sum_{i} c_i a_i \) and a “residual” \( z \), subject to the invariant \( c y + (1 - c) z = x \). Then \( z \) is pushed onto the intersection of a growing set of hyperplanes \( E \), until finally, when enough hyperplanes have been added to \( E \), \( z \) must be a corner.

The original description of the algorithm also discusses a boundary oracle which, when given a hyperplane, is able to determine if the hyperplane is a boundary hyperplane of the polytope or not. However, this boundary oracle is only ever used as a subroutine of the corner oracle in order to determine if the hyperplanes in the set \( E \) are legitimate. This boundary oracle was implemented (and used as described), but we note that its function is primarily to serve as a sanity check for the corner oracle and that if the corner and separation oracles are used as in the algorithm above, and work as described, the boundary oracle’s functionality is not important for the algorithm.

It remains to discuss how the separation and corner oracles work. For the corner oracle, Proposition 3 of the reference work notes that the boundary hyperplanes are the
equality cases of the following sets of inequalities:

\[ \pi(A_{i,j}) \geq \pi(A_{i,j+1}) \quad i \in [m], j \in [k_i - 1] \quad (1) \]

\[ \pi(A_{i,k_i}) \geq 0 \quad i \in [m] \quad (2) \]

\[ \sum_{i=1}^{m} \sum_{j \leq x_i} p_{i,j} \cdot \pi(A_{i,j}) \leq 1 - \prod_{i=1}^{m} \left( 1 - \sum_{j \leq x_i} p_{i,j} \right) \quad x_i \in [k_i - 1] \quad (3) \]

\[ \sum_{A_{i,j} \in A} p_{i,j} \cdot \pi(A_{i,j}) \leq 1 \quad (4) \]

Now for each hyperplane of the form (3) we let \( S_l = \{ A_{i,j} \in A : j \leq x_i \} \). It was shown that in order for the hyperplanes to intersect on the boundary of the polytope, the sets \( S_l \) must be nested, and so we must have that the sets form a chain of inclusion \( S_1 \subseteq S_2 \subseteq \ldots \subseteq S_t \) for some \( t \). We can define the “layers” as the successive differences:

\[ L_1 = S_1, L_l = S_l \setminus S_{l-1} \quad \text{for each} \ l = 2, \ldots, t \]

Then, according to [1], the corner oracle can pick any strict, partially ordered w.r.t. \( \pi \) hierarchical mechanism satisfying the following conditions:

1. \( A_{i,j} \in L_l, A_{i',j'} \in L_{l'}, l < l' \Rightarrow H(A_{i,j}) < H(A_{i',j'}) \).
2. \( A_{i,j} \in S_l, A_{i',j'} \notin S_l \Rightarrow H(A_{i,j}) < H(A_{i',j'}) \).
3. If the hyperplane \( \pi(A_{i,j}) = \pi(A_{i,j+1}) \) is in \( E \), then either \( H(A_{i,j}) = H(A_{i,j+1}) = |A| + 1 \) or \( H(A_{i,j}) \geq H(A_{i,j+1}) - 1 \).
4. If the hyperplanes \( \pi(A_{i,j}) = \pi(A_{i,j+1}), \pi(A_{i,j+1}) = \pi(A_{i,j+2}), \pi(A_{i,j+2}) = \pi(A_{i,j+3}), \ldots, \pi(A_{i,k_i-1}) = \pi(A_{i,k_i}), \pi(A_{i,k_i}) = 0 \) are all in \( E \), then \( H(A_{i,j}) = |A| + 1 \).

We note that in the third and fourth cases, the partial ordering of the hierarchical mechanism would imply, in the fourth case, that \( H(A_{i,j'}) = |A| + 1 \) for any \( j < j' \leq k_i \). In the third case, if \( H(A_{i,j}) \neq |A| + 1 \), it would imply \( H(A_{i,j}) = H(A_{i,j+1}) - 1 \).

As for the separation oracle, the reference work states that it can be implemented using the inequalities of the form (1) and (2) above, as well as those from 2.1 in the preceding section. The separation oracle would check if a given point satisfies each of the given inequalities, returning None if every inequality is satisfied, otherwise it would return the inequality that is violated.

Implementing the corner and separation oracles as described above would then, in theory, allow us to compute the distribution of hierarchical mechanisms needed to implement the given reduced form using Algorithm 1.
3.2 Implementation Details

In this section we address some inaccuracies and technical issues with the above description of the algorithm, as well as design choices in implementing the algorithm. In particular we will examine the four inequalities that form the boundary of the polytope, which we will call the boundary inequalities or boundary planes, as well as the four conditions for the hierarchical mechanism which we call the hierarchical mechanism conditions.

3.2.1 Separation Oracle

In implementing the separation oracle, we initially used the boundary inequalities (1) and (2) as well as the inequalities from 2.1 as described by [1]. However, it was found that using this separation oracle, during the running of Algorithm 1, \( z \) would sometimes have values exceeding 1, which ended up giving erroneous results. Hence our implementation of the separation oracle required the additional inequalities \( \pi(A_i,1) \leq 1 \) for each \( i \in [m] \).

In addition, we modify the separation oracle to take a list of hyperplanes \( E \) as an argument, and the separation oracle will not return any hyperplane found in the given list \( E \). This was necessary due to imprecision cause by floating-point calculation, which caused the separation oracle to return again the equation of a hyperplane that had been previously returned.

3.2.2 Corner Oracle

The main difficulty in implementing the corner oracle is finding the strict, partially ordered hierarchical mechanism that satisfies all the four conditions, a task that the reference work refers to as ‘easy to do efficiently’. In order to explain our implementation, we first consider equivalence classes on the set of types \( A \) induced by a set of hyperplanes \( E \):

**Definition 3.1.** Given some set of hyperplanes \( E \), \( i \leq m \) and types \( A_{i,j_0}, A_{i,j_s} \), we say \( A_{i,j} \sim_E A_{i,j'} \) if there exist some hyperplanes of the form (1) such that we can find indices \( j_1, j_2, \ldots, j_{s-1} \) such that the planes \( \pi(A_{i,j_q}) = \pi(A_{i,j_{q+1}}) \) are in \( E \) for all \( q = 0, \ldots, s - 1 \).

In addition, we allow \( A_{i,j_s} \) to be 0 in the definition above and the hyperplanes to be of type (2). We denote an equivalence class with a bar over the type: \( \bar{A}_{i,j} \) denotes the equivalence class containing \( A_{i,j} \).

Now we find the hierarchical mechanism as follows:

1. We fill the function values of \( H \) in ascending order, from 1 to \( |A| + 1 \), inserting the value \( |A| + 1 \) when appropriate.
2. Starting from the lowest layer \( L_0 \), fill values of \( H \) in increasing order.
3. If we are currently on the layer $L_l$ and encounter a type $A_{i,j}$ such that $\bar{A}_{i,j} \not\subseteq L_l$, then we defer assignment of $A_{i,j}$ until all types belonging to other bidders in $L_l$ have been filled, and then assign values to the equivalence class $A_{i,j}$. Remove any members of the equivalence class from the subsequent layer (if any).

4. If we are currently on the layer $L_l$ and encounter a type $A_{i,j}$ such that $A_{i,j} \sim E 0$, then defer assignment of $A_{i,j}$ until all types belonging to other bidders in $L_l$ have been assigned, then assign $|A| + 1$ to all unassigned types and return.

5. If we are currently on layer $L_l$ and encounter $i_1 \neq i_2, j_1, j_2$ such that $\bar{A}_{i_1,j_1}, \bar{A}_{i_2,j_2} \not\subseteq L_l$, then defer assignment of types belonging to $i_1$ and $i_2$ until all types belonging to other bidders in $L_l$ have been filled, and then assign the value of $|A| + 1$ to all unassigned types and return.

Note that in this implementation, we consider the additional inequalities of the form that we added to the separation oracle $\pi(A_{i,1}) \leq 1$ to be planes of type (3), only that we allow $x_i$ to be zero, and hence may sum over an empty set for some $i$. If exactly one $x_i$ is equal to one, this reduces to the additional inequality, and by the nesting of the layers, must be the first layer - hence the hierarchical mechanism would satisfy $\pi(A_{i,1}) = 1$ if any such hyperplane were in $E$.

We also note that the four conditions provided by [1] are too strict and that the strictness of inequalities for conditions (1) and (2) must be relaxed in order to accommodate the possibility that both types are assigned the value $|A| + 1$.

3.2.3 Limitations

There are some limitations with the code which we will discuss in this subsection. Firstly there was a need to interconvert between 2-dimensional arrays and 1-dimensional arrays, and due to the fact that `np.flatten()` does not work when the lists in the 2-dimensional arrays are of different length, the code only functions in cases where each bidder has the same number of types.

Also, about 25% of the time, the code does not function as intended, and the mechanism implemented based on the distribution of the hierarchical mechanisms does not have identical reduced form probabilities to the original reduced form. While some effort was put into locating the source of the error, this did not prove fruitful and it is believed that the error is due to rounding issues caused by imprecision in floating-point calculations.

However, despite the implementation not functioning all the time, it is still instructive to examine the times when the algorithm does work, and what sort of hierarchical mechanisms it generates in order to decompose the original reduced form.
4 Results and Discussion

The table below illustrates an example run of the algorithm, along with the hierarchical mechanisms found. The virtual mechanism used was generated randomly by assigning a random allocation rule for each possible bid vector.

<table>
<thead>
<tr>
<th>Bidder</th>
<th>Type</th>
<th>( p_{i,j} )</th>
<th>( \pi(A_{i,j}) )</th>
<th>( H_1 )</th>
<th>( H_2 )</th>
<th>( H_3 )</th>
<th>( H_4 )</th>
<th>( H_5 )</th>
<th>( H_6 )</th>
<th>( H_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>( A_{1,1} )</td>
<td>0.02</td>
<td>0.52</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>( A_{1,3} )</td>
<td>0.3</td>
<td>0.348</td>
<td>3</td>
<td>2</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>( A_{1,2} )</td>
<td>0.68</td>
<td>0.512</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>( A_{2,3} )</td>
<td>0.74</td>
<td>0.46</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>5</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>( A_{2,2} )</td>
<td>0.04</td>
<td>0.516</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>( A_{2,1} )</td>
<td>0.22</td>
<td>0.8</td>
<td>1</td>
<td>6</td>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The reduced forms corresponding to each of the hierarchical functions above are as follows:

<table>
<thead>
<tr>
<th>Bidder</th>
<th>Type</th>
<th>( H_1 )</th>
<th>( H_2 )</th>
<th>( H_3 )</th>
<th>( H_4 )</th>
<th>( H_5 )</th>
<th>( H_6 )</th>
<th>( H_7 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( A_{1,1} )</td>
<td>0.04</td>
<td>1</td>
<td>1</td>
<td>0.74</td>
<td>0.74</td>
<td>0.74</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( A_{1,3} )</td>
<td>0.78</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>( A_{1,2} )</td>
<td>0.78</td>
<td>1</td>
<td>1</td>
<td>0.74</td>
<td>0.74</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>( A_{2,3} )</td>
<td>0.02</td>
<td>0</td>
<td>0.3</td>
<td>0.3</td>
<td>0.3</td>
<td>0.98</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( A_{2,2} )</td>
<td>0</td>
<td>0</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( A_{2,1} )</td>
<td>1</td>
<td>0</td>
<td>0.3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

We note that \( H_4 \) and \( H_5 \) are identical, and \( H_4 \) has a coefficient of zero. The reason for this is that after the third iteration of the algorithm (after \( H_3 \) is processed), the vector \( z \) lies on a boundary hyperplane of the polytope not contained in \( E \), as does the corner \( a_i \) that was returned. Hence \( (1 + d)z - da_i \) lay on a boundary hyperplane not contained in \( E \). Due to rounding errors caused by floating point precision, the separation oracle returned this boundary hyperplane, which resulted in a coefficient of zero. This boundary hyperplane was then added to \( E \). Due to our modification to the separation oracle ensuring that it did not return any boundary hyperplane in \( E \), subsequent calls proceeded as intended and the rest of the algorithm continued smoothly.

In order to determine the accuracy of the algorithm, we compute the 2-norm of the differences of the weighted sum of the reduced forms in the second table and the original reduced form \( \pi \) that was randomly generated. The 2-norm was \( 7.06 \times 10^{-10} \), which is close enough to zero to suggest that it was due to rounding error.

The presence of the zero coefficient is not a coincidence. We note that this occurs when the randomly generated feasible reduced form always awards the item to some bid-
We define a fully allocative reduced form accordingly: one where any corresponding mechanism must always allocate the item. In other words, inequality (4) must be tight:

\[ \sum_{i,j} p_{i,j} \cdot \pi(A_{i,j}) = 1 \]  

(5)

Then the following proposition holds true:

**Proposition 4.1.** Any fully allocative, feasible reduced form can be implemented as a distribution over \(|A|\) strict, partially ordered w.r.t. \(\pi\) hierarchical mechanisms.

We provide a sketch of a proof: note that the fully allocative reduced form obeys equation (5), which is a boundary hyperplane of the polytope. Hence we can start Algorithm 1 with the hyperplane corresponding to equation (5) in \(E\), and it should then “corner” \(z\) in one fewer iteration, which would result in a distribution over \(n\) strict, partially ordered w.r.t. \(\pi\) hierarchical mechanisms.

We form a similar conjecture for reduced forms that are not fully allocative. Define the **trivial mechanism** as the mechanism that always does not allocate the item (it simply throws the item away every time). Then our conjecture is as follows:

**Conjecture 4.2.** Any feasible reduced form can be implemented as a distribution over \(|A|\) strict, partially ordered w.r.t \(\pi\), hierarchical mechanisms and the trivial mechanism.

This conjecture can be strengthened by requiring that the hierarchical mechanisms be fully allocative. If this conjecture were indeed true, then implementing a reduced form can perhaps be done more simply - first by finding a basis of fully allocative hierarchical mechanisms, and then by expressing the desired reduced form as a linear combination of these basis vectors. The fact that the hierarchical mechanisms are fully allocative ensures that the coefficients do not sum to more than 1. However, ensuring that the coefficients are all positive is a nontrivial matter, and if the selection of the basis can be done to ensure this is the case, then the mechanism can be implemented more easily using Gauss-Jordan Elimination.

**References**