Specification for Implementing the Budget-Constrained Product-Mix Auction Solver

September 30, 2019

1 Definitions

1.1 Quantities and prices

We may interchangeably use *quantity* or *units* to refer to some amount of a good. Quantities of goods for budget-constrained auctions are nonnegative real numbers $x \in \mathbb{R}^+$. A *quantity vector* $\mathbf{x} = (x_1, \ldots, x_N)$ is an N-dimensional vector of quantities, where $x_j \in \mathbb{R}^+$ represents the quantity of good j. We will typically write $\mathbf{x}$ for a quantity vector allocated to an individual bid and $\mathbf{q}$ for a quantity vector sold by the auction as a whole.

Prices are nonnegative real numbers $z \in \mathbb{R}^+$. A *price vector* is an N-dimensional vector $\mathbf{z} = (z_1, \ldots, z_N)$. We will typically write $\mathbf{v}$ for a price vector offered by a bid and $\mathbf{z}$ for a price vector determined by an auction.

1.2 Supply

A *good* is represented by a natural number $j \in \{1, \ldots, N\}$.

A *supply curve* for good j is a step function σ_j, represented by a list of step widths $s_j^q \in \mathbb{R}^+$ and heights $\mu_j^q \in \mathbb{R}^+$, for some step indices q. That is,

$$\sigma_j(x) = \sum_q \mu_j^q \chi_j^q(x)$$

where χ_j^q is the indicator function for the q^{th} step and e_j^q is the left endpoint of the step, given by

$$\chi_j^q(x) = \begin{cases} 1, \text{ if } e_j^q < x \leq e_j^q + s_j^q \\ 0, \text{ otherwise} \end{cases} \qquad e_j^q = \sum_{1 \leq k \leq q-1} s_j^k.$$

The supply curve should be nondecreasing, i.e. $\mu_j^{q'} \geq \mu_j^q$ for all $q' > q$.

A *supply* σ consists of a supply curve σ_j for each good j.

The *size* of supply curve σ_j is the sum of its step lengths:

$$R_j = \text{size}(\sigma_j) = \sum_q s_j^q$$

This represents the maximum quantity of good j available to be sold.

1.3 Bids

A *bid* consists of:

- a *bid label* $i \in \{1, \ldots, M\}$;
- a total *budget* $b^i \in \mathbb{R}^+$ that can be spent on the goods;
- a price vector $\mathbf{v^i} = (v_1^i, \ldots, v_N^i)$ of unit prices for each good.

1.4 Auction

A *budget-constrained auction* consists of:

- a supply σ;
- a set of bids.

2 Auction rules

Before we can describe the process used to solve an auction, we must specify the rules that govern such auctions. In particular, we will show how one determines the winning bids and what quantities of goods they get, for fixed auction prices. This culminates in a definition of what it is to be a solution to an auction.

2.1 Classifying bids

Suppose bid i has maximum budget b^i and offers unit bid prices $\mathbf{v^i} = (v_1^i, \ldots, v_N^i)$, and that $\mathbf{z} = (z_1, \ldots, z_n)$ is a vector of positive auction prices. We assume that there is at least one $z_i > 0$, and will in fact consider only those goods i such that $z_i > 0$. The goods i such that $z_i = 0$ will simply not be allocated. [1] Let

$$r_{max} = \max_{\substack{j \in \{1,\ldots,N\} \\ z_j \neq 0}} \frac{v_j^i}{z_j} \qquad\qquad G = \operatorname*{argmax}_{\substack{j \in \{1,\ldots,N\} \\ z_j \neq 0}} \frac{v_j^i}{z_j}$$

so r_{max} is the highest bid-price to auction-price ratio, and G is the set of all goods that achieve this ratio (of which there can be one or several).

Then:

- Bid i is *winning* if $r_{max} \geq 1$ (i.e. if there is some j such that $v_j^i \geq z_j$).
- Bid i doesn't win any quantity of any good if $r_{max} < 1$ (i.e. if for all j we have $v_j^i < z_j$).

[1] Individual bids may specify prices of 0, of course. However, an auction price of 0 for some good can happen only when all bids offer 0 for that good, which results in "ignoring" the good and simply allocating no quantity of it.

We can now further classify a winning bid into three different categories:

- If $r_{max} > 1$ and G consists of a single good j_0, then the bid is said to be *non-marginal*. Such a bid must receive all of its budget worth of good j_0.
- If $r_{max} > 1$ and G contains two or more goods, the bid is said to be *marginal in the goods* G. Such a bid may receive any linear combination of the goods in G that is worth exactly its budget.
- If $r_{max} = 1$, the bid is said to be *marginal in its budget and in the goods* G, regardless of the size of G. Such a bid may receive any linear combination of the goods in G that is worth anywhere between 0 and all of its budget.

2.2 Valid quantity assignments

We now specify formally what a bid should receive in each of the cases defined above. We say that $\mathbf{x^i} = (x^i_1, \ldots, x^i_N)$ *is a valid quantity vector for bid* i provided that:

- If i is a non-winning bid, then $x^i_j = 0$ for all j.
- If i is a winning non-marginal bid for good j, then $x^i_j = \frac{b^i}{z_j}$, and $x^i_{j'} = 0$ for all $j' \neq j$.
- If i is a winning bid marginal in goods G, then:
  1. $\sum_j x^i_j z_j = b^i$ (the bid spends its entire budget);
  2. $x^i_j = 0$ for $j \notin G$ (the bid does not receive any non-preferred goods);
  3. $x^i_j \geq 0$ for $j \in G$ (the bid may receive any preferred goods).
- If i is a winning bid marginal in the budget and in goods G, then:
  1. $0 \leq \sum_j x^i_j z_j \leq b^i$ (the bid spends up to its budget);
  2. $x^i_j = 0$ for $j \notin G$ (the bid does not receive any non-preferred goods);
  3. $x^i_j \geq 0$ for $j \in G$ (the bid may receive any preferred goods).

An *assignment of goods to bids* is a set of quantity vectors $\{\mathbf{x^i}\}$, one for each bid i, so that x^i_j gives the quantity of good j assigned to bid i. Let $q_j = \sum_i x^i_j$ be the total quantity of good j assigned across all bids.

We say that $\{\mathbf{x^i}\}$ *is a valid assignment (at prices* $\mathbf{z}$*)* provided that:

- for every good j, $q_j \leq R_j$, i.e. the total quantity of good j assigned does not exceed the supply of good j;
- for every bid i, $\mathbf{x^i}$ is a valid quantity vector for bid i.

A *solution* to an auction is a pair of a price vector $\mathbf{z}$ (the auction prices at which goods are sold) and an assignment of goods to bids $\{\mathbf{x^i}\}$ (the quantities received by each bid) which is valid at prices $\mathbf{z}$.

3 Solving an auction

The above section defined what it means to be a valid solution to an auction. In the following, we give the objective function to be maximised in order to determine the optimal solution, and explain the search algorithm for finding such an optimal solution.

3.1 Auctioneer's objective function

We seek to find a solution consisting of an auction price vector $\mathbf{z}$ and an assignment of goods to bids $\{\mathbf{x^i}\}$ that maximises the auctioneer's profit, provided the demands of the bids are satisfied. More precisely, let

$$\psi_{\mathbf{z}}(\{\mathbf{x^i}\}) = \sum_j (z_j q_j - \hat{\sigma}_j(q_j)) \text{ where } q_j = \sum_i x_j^i$$

be the profit made by the auctioneer using the auction price vector $\mathbf{z} = (z_1, \ldots, z_N)$ and selling the total quantity $\mathbf{q} = (q_1, \ldots, q_N)$. Here $\hat{\sigma}_j(q)$ is the cost to the auctioneer of selling q_j units of the good j, given by

$$\hat{\sigma}_j(q) = \int_0^q \sigma_j(x)\, \mathrm{d}x$$

where σ_j is the supply curve step function.

We seek to determine

$$\operatorname*{argmax}_{\mathbf{z},\{\mathbf{x^i}\}} \left(\psi_{\mathbf{z}}(\{\mathbf{x^i}\}) \mid \{\mathbf{x^i}\} \text{ valid assignment at price vector } \mathbf{z}\right).$$

We proceed by first constructing a set of price vectors S amongst which the optimal auction prices must be found, then for each $\mathbf{z} \in S$ we search for a valid assignment that maximises $\psi_{\mathbf{z}}$.

In general the optimal price vector is not guaranteed to be unique. Similarly, for a given price vector there may be multiple valid assignments of goods to bids that yield the maximum profit at those prices. The program will choose arbitrarily to resolve such ambiguity.[2]

3.2 Determining candidate auction prices

In order to obtain a set S of candidate auction prices amongst which the optimal price vector is guaranteed to be, it is enough to consider points in price space that are obtained by intersecting subsets of the indifference hyperplanes generated by the bids. This will be described in more detail in the following subsection. The algorithm will determine a set of candidate auction price vectors using only the auction's bids; supply curve information is not relevant.

[2] In principle, one could fairly easily define a tie-breaking rule to select a unique price vector (such as choosing the lowest prices based on a lexicographic ordering of goods). Similarly, one could choose amongst the valid assignments based on preferences of which goods to allocate, and specify how to ration marginal bids to produce fairer results.

Two different strategies for finding candidate auction prices have been implemented: one that exhaustively finds all indifference hyperplane intersections, while the other adds a heuristic that reduces the number of points to try. The next subsection explains the general idea in greater detail and specifies an exhaustive search over all the hyperplane intersections. The following subsection covers the heuristic method.

3.2.1 Exhaustive method

A bid i with price vector $\mathbf{v^i}$ generates two types of indifference hyperplanes:

- The *hod for good j* is the hyperplane with equation $x_j = v_j^i$. This splits $\mathbb{R}^N$ between bids that offer more than what the bid does for good j (the region in which $x_j > v_j^i$) and on the other side bids that offer less.
- The *flange for goods j and j'* (where $j \neq j'$) is the hyperplane with equation $\frac{x_j}{v_j^i} - \frac{x_{j'}}{v_{j'}^i} = 0$. This splits $\mathbb{R}^N$ between the bids that would prefer buying good j on one side and good j' on the other.

These hyperplanes divide up price space into Unique Demand Regions (UDRs). Within a UDR, every bid's demand is always for a unique good. On the boundaries, bids may be indifferent between receiving multiple goods. The optimal value of the objective function must lie at a vertex of a UDR, since otherwise the auctioneer can increase the prices (and hence their profit) without changing the allocation of goods to bids.

Let H be the set of all such hyperplanes for all the bids, i.e. all the hods (for all bids and all goods) and flanges (for all bids and all pairs of distinct goods). The set of candidate price vectors S will be the set of unique points that can be constructed as intersections of N hyperplanes from H.

Consider all the linear systems expressing the intersection of N distinct hyperplanes from H (where N is the number of goods sold at the auction). Each such system gives a set of N equations that a point $(x_1, \ldots, x_N)$ has to satisfy to lie at the intersection of the N hyperplanes. Some hyperplanes will be hods, others flanges. Such a system with $N = 3$ and a single bid with prices $(2, 4, 5)$ might be:

$$\begin{aligned} x_1 &= 2 \\ \frac{x_1}{2} - \frac{x_3}{5} &= 0 \\ x_2 &= 4 \end{aligned} \tag{1}$$

Linear systems can be described in matrix form, for example, the above system can be formulated as trying to find $(x_1, x_2, x_3) \in \mathbb{R}^3$ such that:

$$\begin{pmatrix} 1 & 0 & 0 \\ \frac{1}{2} & 0 & -\frac{1}{5} \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \\ 4 \end{pmatrix}$$

Such a system has a unique, non-zero solution if and only if the matrix is invertible, or equivalently, has rank N. The intersection is then non empty but reduced to a single point, which will be a candidate auction price vector for the rest of the algorithm.

The algorithm however does not consider systems that do not have at least the equation for one hod in them, as for such a system, the right hand side of the matrix equation above would be

$$\begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

which means that if there is a solution to the system that we call $\mathbf{x} \in \mathbb{R}^n$, then $\lambda\mathbf{x}$ is a solution too for any $\lambda \in \mathbb{R}$, since $A(\lambda\mathbf{x}) = \lambda(A\mathbf{x}) = \lambda \cdot 0 = 0$ (if A is the $N \times N$ matrix representing the system). In particular, the intersection would not be reduced to a single point.

All valid linear systems are examined. Those systems whose matrix has rank N are solved to find an intersection. The other systems are simply dropped.

3.2.2 Heuristic for searching fewer candidate prices

This heuristic reduces the number of candidate price vectors to check relative to the exhaustive method of finding all indifference hyperplane intersections described in the previous subsection. However, its correctness is not obvious, and is outside the scope of this document.

An *interaction* is an ordered N-tuple of bids $(i_1, \ldots, i_N)$, where N is the dimension (number of goods). Bids may appear more than once in a single interaction. For example, $(1, 1, 2)$ is the interaction that uses bid 1 to determine the prices of two goods, and uses bid 2 to determine the other price.

A *sequence* $s_1 \ldots s_N$ is a permutation of the set of goods $\{1, \ldots, N\}$ representing the order in which to consider the goods. For example, 3 1 2 is a sequence for 3 goods.

Intuitively, the heuristic works by specifying how to determine a candidate price vector from an interaction and a sequence, where the sequence specifies the order in which to consider the goods, and the bids in the interaction determine the prices for each good in turn. We generate the set of all candidate price vectors from all interactions and all sequences.

Given a bid i with prices $\mathbf{v}^i$, the *rebased bid prices* $\mathbf{w^i}$ fixing good j to have price p are given by:

$$w^i_{j'} = \begin{cases} v^i_{j'} \cdot p / v^i_j, \text{ if } v^i_j > p, j \neq j' \\ v^i_j, \text{ if } v^i_j > p, j = j' \\ v^i_{j'}, \text{ otherwise} \end{cases}$$

That is, if the bid's price for j is greater than p, we multiply the bid's prices for other goods by the ratio of p to the bid's price for j; otherwise we leave the bid unchanged.

An interaction and sequence together may determine a single candidate price vector $\mathbf{p}$, or they may fail to determine a candidate. The candidate price vector, if any, is determined by iterating through the bids in the interaction and the goods in the sequence, according to the following algorithm:

At step 1, the candidate price p_{s_1} for the first good in the sequence, s_1, is fixed to be the price given by the first bid in the interaction, i.e. $p_{s_1} = v^{i_1}_{s_1}$. For each other bid in the interaction we then replace its prices with the rebased bid prices fixing good s_1 to have price p_1. (The rebased prices are used by future steps when computing the candidate price vector for the interaction/sequence pair, but computation of candidate price vectors for other pairs uses the original prices.)

At each subsequent step l, the candidate price p_{s_l} for good s_l is fixed to be the price given by the l^{th} bid in the interaction (with its prices having been rebased once for each previous step), i.e. $p_{s_l} = v^{i_l}_{s_l}$. However, we must check that earlier bids in the interaction maintain their preferred good. If $v^{i_l}_{s_l} < v^{i}_{s_l}$ for some $i \in \{i_1, \ldots, i_{l-1}\}$, we reject the interaction/sequence pair and do not yield a candidate price vector. Otherwise, we again rebase the other bids in the interaction, fixing good s_l to have price p_l, and continue with the next step.

When all N steps are complete, assuming the interaction/sequence pair was not rejected, we have a candidate price vector $\mathbf{p}$. By performing these steps for every possible interaction and every possible sequence, recording the distinct candidates where the algorithm was successful, we obtain a set S of candidate price vectors.

For example, suppose we have three bids with prices of $\mathbf{v}^1 = (10, 0, 10)$, $\mathbf{v}^2 = (20, 6, 0)$ and $\mathbf{v}^3 = (0, 15, 15)$. For the interaction $(1, 2, 3)$ and sequence 3 1 2 we proceed as follows:

- Start by fixing the price of good 3 to be $p_3 = v^1_3 = 10$. Rebase the other bids: $\mathbf{v}^2$ is unchanged because $0 = v^2_3 < 10$, but now $\mathbf{v}^3 := (0, 10, 15)$.
- Next fix the price of good 1 to be $p_1 = v^2_1 = 20$. We may continue, and rebasing does not change the other bids, because this is the highest bid price on good 1.
- Finally, fix the price of good 2 to be $p_2 = v^3_2 = 10$. We may continue, because this is the highest bid price on good 2. (If instead we started with $v^2_2 > 10$, we would have rejected the interaction/sequence.)
- Thus we find the candidate price vector $(20, 10, 10)$.

Note that where the interaction consists of the same bid repeated N times, the bid price vector will always be a candidate price vector. Thus the set of candidate price vectors always includes all the price vectors of all the bids in the auction. It may include other price vectors due to rebasing.

3.3 Exploring quantity space

At this point of the algorithm, we have a set S of candidate auction price vectors to try, so we can take each one in turn and evaluate the options for selling goods

to satisfy the bids. In contrast to the previous phase, the solver will at this point make use of the supply curves in addition to the bids. We suppose we have a fixed auction price vector $\mathbf{z}$, and seek to construct a valid assignment of goods to bids $\{\mathbf{x^i}\}$ that maximises the objective function ψ_z.

For any winning non-marginal bid for good j, there are no options to explore. Such a bid receives $\frac{b^i}{z_j}$ units of good j, where b^i is the bid's budget and z_j is the auction price for good j. Similarly, a non-winning bid simply receives no units.

For winning marginal bids, we have to explore multiple possible choices for quantity assignments. We search a tree of possible assignments, where each branch corresponds to a particular choice of $\mathbf{x^i}$ for some bid i. Each of these choices leads to a different branch in the search tree; going one level deeper in the tree means exploring choices for one bid. Of course, we cannot consider all possible assignments (as there are infinitely many); instead we consider a finite set that is guaranteed to contain a profit-maximising assignment.

The search tries *all* possible choices for the order in which to consider marginal bids, as assignment of quantities to bids changes the state of the algorithm, and the respective updates for different bids do not necessarily commute (processing bid i_1 then i_2 does not necessarily lead to the same quantity assignments as processing i_2 then i_1).[3]

If at any point q_j (the total quantity of good j sold so far) exceeds R_j (the available supply of good j) we abandon the corresponding branch of the search tree. It may be that there are no possible assignments that satisfy the winning bids given the available supply, in which case we will reject the candidate price vector entirely. There will always be some price vector for which this is not the case (since increasing prices reduces demand).

At the end of the search, we have a tree where each node corresponds to a partial definition of an assignment $\{\mathbf{x^i}\}$. The leaves of the tree represent sequences of choices that led to a state where all bid demands are satisfied, so they represent complete assignments $\{\mathbf{x^i}\}$ that are potential solutions to the auction. We can then evaluate the objective function at each of these assignments to find the one that maximises profit for the current prices.

The following subsections explain the search process in more detail.

3.3.1 Non-marginal bids

For each non-marginal winning bid i that prefers good j, set $x^i_j = \frac{b^i}{z_j}$ and $x^i_{j'} = 0$ for $j' \neq j$. For each non-winning bid i, set $x^i_j = 0$ for all j.

This gives a partially defined assignment $\{\mathbf{x^i}\}$, which is used as the root of the search tree.

It is possible that the non-marginal bids alone demand more than the available supply, in which case we reject the candidate price vector immediately.

[3]Since the number of permutations of marginal bids is large, we optimize this slightly by identifying sets of marginal bids that are marginal in the same way (i.e. all marginal in a common set of goods, or all marginal in the budget and a common set of goods). We then find an assignment of quantities for the set of bids, and subsequently distribute the goods between the bids.

3.3.2 Marginal bids

During the search, we have a partial assignment of goods to bids represented by the current path in the search tree. In particular, we keep track of the total quantity q_j of each good that has been assigned so far.

For any marginal bid i that might receive some quantity of good j, we consider all of the following choices for x_j^i (in different branches of the search tree):

- no units of good j: $x_j^i = 0$;
- the rest of the bid's budget's worth of good j: $x_j^i = \frac{b^i}{z_j}$ (if the supplies allow it, i.e. provided $q_j + \frac{b^i}{z_j} \leq R_j$, where q_j is the amount of good j sold so far);.
- any quantity less than the rest of the bid's budget's worth of good j, which, when added to the total quantity of that good sold so far, reaches the end of a supply curve step (i.e. any $x_j^i < \frac{b^i}{z_j}$ such that $q_j + x_j^i = \sum_{q \in \{1,\ldots,k\}} s_j^q$ for some k).

The third possibility includes the case where the bid receives all the remaining supply of the good (i.e. $q_j + x_j^i = R_j$). In general, it may also be useful to exhaust a supply step by selling to the current bid, but not sell more units of that good (e.g. if the next supply step is significantly more expensive).

If the bid is marginal in several goods, we handle the goods in all possible orders, exploring every combination of the above possibilities.[4]

Once the bid has been assigned a complete quantity vector $\mathbf{x^i}$, we check that $\mathbf{x^i}$ is a valid quantity vector (e.g. if the bid must spend its entire budget, we check that it has done so). If so, we add the quantity vector to the partial assignment $\{\mathbf{x^i}\}$ and proceed with another bid. If not, we abandon the branch of the search.

3.3.3 Example of marginal bid choices

For example, suppose we have auction prices of $\mathbf{z} = (2, 3)$, and a bid i with a budget of 6 and prices of exactly $\mathbf{v^i} = (2, 3)$. If the supply of both goods is large and we are not near the end of any steps, we will try (all of) the following possibilities for $\mathbf{x^i}$:

- $(0, 0)$ (no units of either good)
- $(0, 2)$ (no units on good 1, exhaust budget on good 2)
- $(3, 0)$ (exhaust budget on good 1)

[4] We could investigate some optimizations to the search, as some branches are strictly better than others. For example, if the height of the current supply curve step has exceeded the auction price for a good, it is better to sell no units of that good to that bid where possible.

Now suppose that we have 2 more units of good 1 left on its current supply curve step, and $\frac{1}{2}$ a unit of good 2 on its current supply curve step (and in both cases we have more units available at higher prices). We will additionally try:

- $(2, 0)$ (exhaust supply step on good 1, no units of good 2)
- $(2, \frac{1}{2})$ (exhaust supply steps on goods 1 and 2)
- $(2, \frac{2}{3})$ (exhaust supply step on good 1, exhaust budget on good 2)
- $(0, \frac{1}{2})$ (exhaust supply step on good 2, no units of good 1)
- $(\frac{9}{4}, \frac{1}{2})$ (exhaust supply step on good 2, exhaust budget on good 1)

Each of these possibilities will lead to a separate branch of the search tree, examining possible assignments to the other marginal bids.

If a supply step is not filled entirely and the budget of a bid receiving some quantity of the corresponding good is not exhausted, profits can be increased. So the optimal point has to be among all those permutations of quantity points that either exhaust budgets or are located at the end of supply steps.

3.3.4 Example of search tree

Suppose we have two bids with a budget of $b^1 = b^2 = 6$, bid prices of $\mathbf{v^1} = (2, 3)$, $\mathbf{v^2} = (4, 6)$ and we are evaluating the candidate auction prices $\mathbf{z} = (2, 3)$. Suppose the total available supply is $R_1 = 4, R_2 = 2$ with a single step on each supply curve. Observe that bid 1 is marginal in the budget and both goods, while bid 2 is marginal in the goods but must spend its entire budget. The search tree that we explore is shown in the figure.

Figure 1: Search tree