

4600 Report

Algorithmic Mechanism Design

Qianhui Wang, u7070170

November 8, 2022

Background

Mechanism Design is a field in economics and game theory that studies how we can design social mechanisms to achieve some desired outcomes. The usual objectives concern maximising overall social welfare, or minimising overall social cost. From the perspective of complexity theory, these are therefore optimisation problems. One important feature in classical mechanism design is the rationality of players. The central idea is that we could model a social choice scenario as a non-cooperative game among the participants. While each player has a set of strategies for decision making, their goal is to always maximise their self benefits, by playing their best strategy. Under such a light, we could expect that players potentially abuse the mechanism, for example by deceiving, and gain at the expense of overall welfare. This is a situation we aim to avoid, and hence, one of our objectives would be to remove the inherent properties from the mechanism that allows such undesired behaviours.

Putting aside the strategic intensity, increasingly we recognise that more computational considerations emerge in mechanism design. Such a merge with classical computer science is a result from the digitalisation of economic activities, where problem size and complexity scales up to an unprecedented level. These problems, whose central characteristics are often "computational hardness" and "input and outcome combinatorial complexity", require extensive analysis from a theoretical computer science point of view; taking the economic perspective alone would miss practically important concerns such as how much computational resources is required, and how efficient we could find the optimal, or approximately optimal, solutions. There is also a reverse interest in incorporating strategic considerations into computer science as well, since computers ultimately have rational end users with different objectives, and thus the game-theoretic notion of cooperation and competition also apply to computer systems and networks in general. Hence, the term "Algorithmic Mechanism Design" refers to designing mechanisms that balance players' economic incentives and the algorithm's computational requirements.

Mechanism Design vs Computer Science

To analyse a problem from both the strategic and computational points of view, we first recognise that there is inherent difference in the analysis approaches of mechanism design and of computer science, and therefore we need to make choices in the following analysis.

Mechanism Design	Computer Science
Bayesian Distribution	Black-Box
Average-case analysis	Worst-case analysis
Exactness	Approximation
No efficiency concern	Memory, Time, Communication Complexity
Dominant Strategy	No utility

Table 1: Different Characteristics

First, in Mechanism Design, a problem's input is explicitly modeled by Bayesian probabilities based on

real-world data, such as using random variables and expected values to perform average case studies; there is also an exclusive interest in finding exact solutions for the optimisation problem, and as such, there is no computational efficiency concerns which is one most important study in Computer Science. However, as in game theory, we use a well-defined model to evaluate how well a player's strategy is, i.e. through utility functions, and one central goal is to achieve "dominant strategy" equilibrium with the designed mechanism.

On the other hand, a Computer Science approach prefers abstraction over details, and would model the social mechanism as entirely a black box that, given any input, produces the corresponding outcome. Hence, there is no assumption about the input distributions, and the designed mechanism is said to be more robust and uses less technical definitions. With this approach, we no longer have the average definitions, but we can study how "bad" a solution could be in the worst case, where the badness is defined in terms of asymptotic behaviours, usually with the big-O notation. In addition, as the field of Algorithmic Mechanism Design mainly looks at NP-hard optimisation problems, it is generally impossible to find the optimal solution within tractable time (polynomial time in terms of the input size). The Computer Science approach would trade optimality for efficiency, by accepting approximate solutions. This also suggests that one major focus is on reducing computational requirements, i.e. how much memory, time and communication is required in the worst case. Nevertheless, the standard problem-solving algorithm is nothing more than an objective mechanism that defines a input-output relation according to the given specification. There is no subjective influence from the outside environment (the players) that interacts with the system (that might affect how the system works), and thus we lack a proper definition of utility and strategic dominance.

Definition 1 (Strategic Dominance) *For a given player, strategy A dominates strategy B if choosing A gives at least good outcomes as B, no matter how the player's opponents play.*

A game outcome is in dominant strategy equilibrium if all players play their own dominant strategies. The outcome is stable, or an equilibrium outcome, since no player could gain more by changing strategy, there is no incentive to change. A dominant strategy equilibrium is also a nash equilibrium.

A nash equilibrium is a more general term that describes the state of the game where no players could gain by changing one's own strategy, assuming all other players' strategies remain unchanged. A nash equilibrium is not necessarily the dominant strategy equilibrium, but the reverse is true.

We combine the core ideas of both approaches, and choose the following model for our analysis:

- Algorithmic model
 1. We look at algorithms that both produce exact solution and approximate solutions, and the how close the approximation is to optimality.
 2. We discuss computational efficiencies, in terms of time and communications.
 3. We assume no input distributions, and look at worst-case computational constraints.
- Strategic model
 1. We assume players have private valuation functions $v_i : \mathbb{O} \rightarrow \mathbb{R}$, that defines his own values for each possible game state.

2. We assume quasilinear utility. In economics, quasi-linearity refers to functions that are linear in one argument. In Mechanism Design, quasilinear utility is the difference of a player's value associated to the game outcome and the payment incurred. $u_i = v_i(o) - p_i$, where $o \in \mathcal{O}$.

With these choices, we demonstrate with a case study on Multi-Unit Auction the two central goals of Algorithmic Mechanism Design: incentive compatibility and algorithmic efficiency, and the difficulty present in achieving both at the same time.

Case Study : Multi-Unit Auction

Definition 2 (*Multi-Unit Auction*)

- Given m identical items,
- and n bidders each with a private valuation function $v_i : \{0, \dots, m\} \rightarrow \mathbb{R}^+$,
- find an allocation $(m_i)_{i=1}^n$ that maximises social welfare $\sum_i v_i(m_i)$,
- with the constraint that $\sum m_i \leq m$

The valuation function v_i specifies a player's value (or willingness to pay) for a number of items he is allocated. Thus, the equation $v_i(m_i) = p_i$ means that a player's value (or is willing to pay) p_i if allocated m_i items. In addition, $v_i(0) = 0$ and $v_i(k) \leq v_i(k+1)$ for any $k \geq 0$. (monotone valuation)

We would consider the problem in three layers : Efficiency models (Representation and Data Access), Algorithm, and Strategy. First, recognising that in reality the number of items m could be large in scale of millions or billions, we require a succinct way to represent the valuations as binary strings in a finite-storage computer. However, by succinct representation model, we imply a tradeoff with expressiveness. That is, we generally could not express all valuations with a formal succinct representation. On the other hand, we would like to evaluate our mechanism with arbitrary valuations, we would therefore look at a second black-box data access model and define a new approach to evaluate algorithmic efficiency. Second, based on the analysis model chosen, and the respective efficiency definitions, we would evaluate the algorithms that produce exact solutions to the maximisation problem in terms of algorithmic efficiency; and with either a succinct representation model, or an expressive data access model, we would conclude with a general intractability results. Therefore, we need to look at approximation algorithms and evaluate the closeness of approximation and efficiency at the same time. Third, as the algorithms take as inputs valuations from bidders of subjective interests, the values could be misreported and undermines the goal of maximising social welfare. Hence, to discuss truthfulness of the input valuations, we need to incorporate payment terms and utility functions as a strategic model into the problem.

Formally, our desired mechanism is a social choice function $f : V_1 \times V_2 \times \dots \times V_n \rightarrow A$, where V_i is the set of possible valuation functions of player i , and A is the possible allocation outcomes for the problem; and a set of payment functions p_1, \dots, p_n , where $p_i : V_1 \times V_2 \times \dots \times V_n \rightarrow \mathbb{R}$ is a function of the input valuations of all players.

1 Efficiency

Before we could evaluate an algorithm's efficiency, we first need a proper definition of it. Under the classic notion, efficiency is defined in terms of the size of input representation. Following this definition, we would look at ways to represent the valuation functions, mainly through constructing formal languages. On the other hand, efficiency can be evaluated in terms of the number of value queries or the length of answer bits. Under this definition, we are not concerned with the input representation, but rather how data is accessed. We therefore have two models of efficiency, the classic representation model and the new data access model; and we would describe the incentives and usage cases for each definition.

1.1 Representation Model by Language Formalism

Since the input to the social choice function consists of n valuation functions $V_1 \times V_2 \times \dots \times V_n$, and each $v_i : \{0, \dots, m\} \rightarrow \mathbb{R}^+$ specifies a player's value of a total m possible allocations. The full representation has nm real numbers in total. Without loss of generality, we can assume m is large. Hence, the full listing of nm numbers is in general impractical for any finite-storage computer, and thus, we require more succinct representations. From an information-theoretic view, it is impossible to reduce the representation size without a loss of information. However, as it is not harmful to trim down the full representation to only capturing its essential and useful information, we could model it by using formal languages to succinctly express an useful amount of information we need.

Bidding Language

The way to describe a language is by specifying the formal syntax and semantics. The syntax, in this case, is just some chosen mathematical functions that describe the compressed valuation rules, and these functions would be encoded as binary strings as inputs to our mechanism. On the other hand, the semantics is the obvious translation from the encoded rules into numerical values for a bidder's allocation.

In this problem, we look at three types of valuation functions. In an increasing order of expressiveness,

- Single-step function.

$$v(k) = \begin{cases} 0 & \text{for } k < k^* \\ w^* & \text{for } k \geq k^* \end{cases}$$

The original paper coins this as "Singled Minded Bids". This means that the player only specifies one single bid, i.e. w^* number of items, and the corresponding value he is willing to pay k^* . The semantics for the valuation rule is that for an allocation beyond w^* items, the player has value k^* , and for any allocation below w^* items, his value is 0. Even though the language appears simplistic and restrictive, it satisfies the need for succinctness since only 2 values (k^*, w^*) are needed to express the full valuation. However, as the general rule of thumb for any language formalisation is to aim for a balance between succinctness and expressiveness, this language lose too much information as typical real-world valuations have m different values (i.e. m steps) to represent.

- Step functions in general (or pieewise constant function)

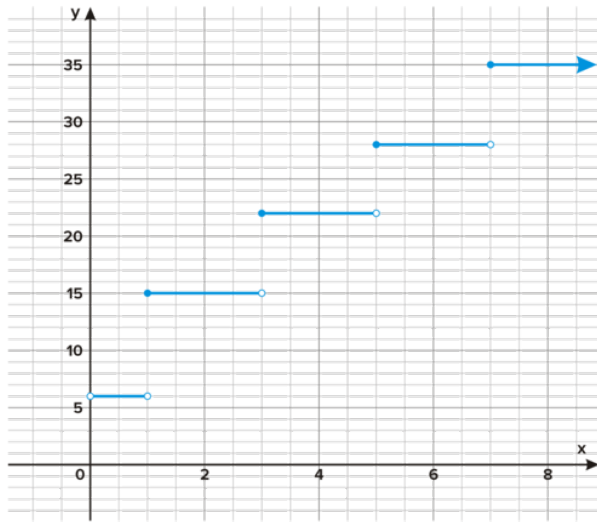
$$v(k) = \begin{cases} 0 & \text{for } k < k_1 \\ w_1 & \text{for } k_1 \leq k < k_2 \\ w_2 & \text{for } k_2 \leq k < k_3 \\ \dots & \\ w_t & \text{for } k_t \leq k \end{cases}$$

This language is a generalisation of the previous single-step language, as it now allows players to specify multiple bids and the corresponding values for each bid. The semantics for this valuation rule is that for an allocation k , the player's value corresponds to the highest "step" w_j such that $k \geq k_j$. For example (Fig 1(a)), if $k = 6$, then the bidder has value $v(k) = 28$; and $k = 8$, $v(k) = 35$. Syntatically, this language satisfies succinctness since we specify t "(bids, values)" pairs, and $t \ll m$ is a self-controlled value. In terms of expressiveness, it can recover valuations with at most t distinct values, but could not do so for most valuations of close to m distinct values.

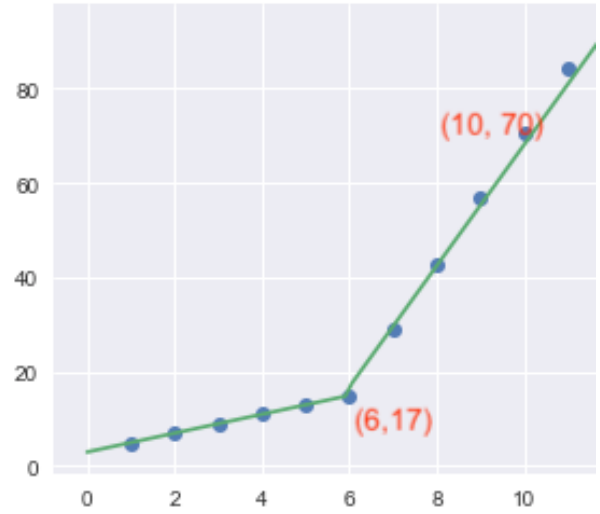
- Pieewise linear function.

$$u(k) = \begin{cases} 0 & \text{for } k < k_1 \\ p_1 & \text{for } k_1 \leq k < k_2 \\ p_2 & \text{for } k_2 \leq k < k_3 \\ \dots & \\ p_t & \text{for } k_t \leq k \end{cases}$$

Syntatically, the same step function applies to this language; however the semantics is different. The function $u(k)$ now does not specify a value for a total k number of allocated items, but rather, the k^{th} single allocated item. This implies that to obtain the original valuation $v(k)$, we need to sum the partial values $v(k) = \sum_i^k u(i)$. The partial value for the single k^{th} item $u(i)$ is still the highest step p_j for $j \leq k$. For the same example in Fig 1(a), if $k = 6$, then $u(k) = 28$ becomes the partial value for the 6^{th} allocated item. In terms of expressiveness, since the same step function now represents partial values, we have more freedom in terms of being able to obtain m distinct values, rather than only t in the previous language. Graphically, we could perceive the partial values $\{p_j\}_{j=1}^t$ as the gradient of a linear function within the same interval. Fig 1(b) represents the original valuation $v(k)$ after summing the partial values. The partial values p_j 's can be obtained from the slopes, i.e. $p_1 = \frac{17-0}{6-0}$ for $k < 6$, $p_2 = \frac{70-17}{10-6}$ for $6 \leq k < 10$. Notice that since each interval is a monotonic increasing linear function, we now can express m distinct values for m allocations. This means that conversion of this language into a step function would require $O(m)$ size of explosion, thus pieewise linear is strictly more expressive than pieewise constant. However, we are not free to express any valuations, but it provides at least an acceptable level of balance between simplicity and expressiveness.



(a) Step(Pieewise Constant)



(b) Pieewise Linear

Figure 1: Bidding Languages

Since one of our analysis approach (stated in Section 1) is to assume no input distributions and allow any form of valuations, we would like to know if it is viable to formalise any valuations as languages that are as expressive as possible. Such languages might take the form of more complex functions, e.g. piecewise quadratic function, any random functions, or even with the complete logic expressed in the form of a computer program. However, with complex formal rules encoded as inputs to our mechanism, it would be less easy for the algorithm to decode, process and retrieve back the original valuations. This would thus violate our efficiency requirement of being polynomial time within the size of the representation. Therefore, the main usage case for this representation model is whenever the valuations can be captured by a straightforward syntax that is easy for data retrieval.

To allow arbitrary valuations in analysis, we would therefore define a new data access model, which abstracts away from the input encodings, and exclusively looks at how efficient our designed mechanism, or firstly algorithm, is with data access. This implies a new definition of efficiency as well.

1.2 Data Access Model

Data access refers to a black-box process where given a query like "what is bidder i 's value when he is allocated k items", the model returns back the desired answer $v_i(k)$, by some (simple or complex) interaction with the underlying input representation. By splitting the form of interactions into a simple and a complex classes, we have two ways to define the amount of time taken in data access.

First, if each query iteration is simple, i.e. takes only polynomial time in its underlying representation, then it satisfies the classic efficiency notion, so we define an algorithm's runtime by just counting the number of queries to make; Second, if we allow any complex interactions that could go beyond the polynomial time constraints, then we shall completely ignore the interaction, and lower-bound our algorithm's runtime by the time taken for reading the answer i.e. counting the number of answer bits. The first approach is called the value-query model, and the second the general communication model.

The value-query model requires an effective interaction interface to the underlying representation, i.e. a polynomial time Turing Machine (notion in Theory of Computation) that takes as input the valuation representation and a query, and produces the answer on the tape. As an illustration, the piecewise linear language has an effective implementation.

Since value-query requires a realistic implementation of the interaction logic, we say it provides an upper-bound for an algorithm on the most amount of time it takes; while the general communication model is not concerned with interaction time at all, we use it to provide a lower-bound analysis on the least amount of time an algorithm would take.

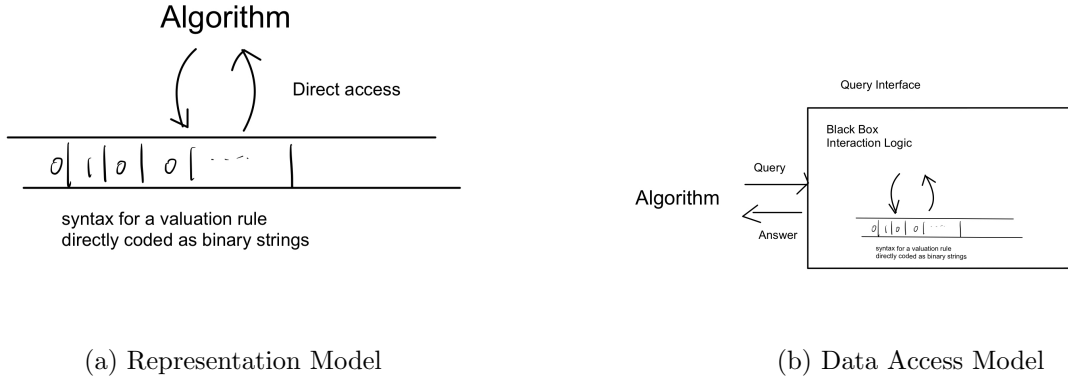


Figure 2: Bidding Languages

1.3 Formal Definition of Efficiency

Having these two models properly defined, we can now specify the mathematical meaning of efficiency.

Definition 3 *Efficiency*

- **Representation Model** An efficient algorithm is one that takes polynomial time in terms of the total size of input representation. For a simple bidding language, there is n valuations, each represented by $t \ll m$ "bid-value" pairs, we therefore require polynomial time in terms of
 - n , for n valuations;
 - $\log m$, for the maximum bid is m , and each bid is in its binary representation;
 - $t = \log(\max_i v_i(m))$, i.e. number of bits of precision, for the maximum value of a bid is $\max_i v_i(m)$, and each value is in its binary representation.
- **Data Access Model** An efficient algorithm is one that has total number of queries or number of answer bits
 - polynomial in terms of n , for n valuations;
 - sublinear in terms of m , e.g. $\log m, \ln m, \sqrt{m}$. Sublinearity is required since polynomial in m allows communication of the entire valuation, contradicting with the goal of finding succinct representation above.

2 Algorithm

Having the proper definition of efficiency, we discuss how, from a pure algorithmic perspective, that finding exact solution efficiently is hard. Thus, to satisfy this core constraint, we will relax ourselves to the approximate solutions, and study an arbitrarily good efficient algorithm for Multi-Unit Auction.

2.1 General Allocation Algorithms

The general allocation algorithm produces the optimal solution to the maximisation problem, by the concept of dynamic programming.

Definition 4 *Dynamic Programming (DP) is an optimisation technique applicable to the class of problems with optimal substructure, i.e. a problem can be solved optimally by breaking it into simpler subproblems, and recursively finding the optimal solutions to the subproblems.*

$$DP = \text{Divide-and-Conquer} + \text{Optimal Substructure}$$

DP for Multi-unit Auction

- **Subproblem**

Given a subset of the first $i \leq n$ players, and the first $k \leq m$ items,

Find the optimal allocation (m_1, m_2, \dots, m_i) s.t. $\sum_{j=1}^i v_j(m_j)$ is maximised,

with the constraint $\sum_{j=1}^i m_j \leq k$

- **Optimal substructure**

Let $s[i, k]$ be the value of the optimal solution to a subproblem of i players and k items. We have the relation

$$s[i, k] = \max_{0 \leq j \leq k} \{v_i(j) + s[i-1, k-j]\}$$

To find the optimal value, there are $k+1$ possible allocations, i.e. allocating $j = 0$ to $j = k$ items to the i^{th} player, and $(k-j)$ items to the first $i-1$ players. To obtain the optimal value for each allocation, we just need to sum the value from the i^{th} player $v_i(j)$ and the optimal value of a corresponding subproblem, then choose the best allocation out of these $k+1$ possibilities.

Algorithm 1 General Allocation Algorithm ($\{v_i\}_{i=1}^n$)

- Construct a dynamic programming table of $(1+n) * (1+m)$ size, where row i represent the first i players, and column k represent the first k items. Cell $s[i, k]$ stores the value of the optimal solution to a problem instance of i players and k items.
- - $s[0, k] = 0$, for allocating to 0 players gives 0 value.
 - $s[i, 0] = 0$, for allocating 0 items gives no value.
 - Fill the table by the optimal substructure

$$s[i, k] = \max_{0 \leq j \leq k} \{v_i(j) + s[i-1, k-j]\} \quad (*)$$

for $0 < i \leq n$ and $0 < k \leq m$.

#bidders\ items	0	1	2	...	m-1	m
0						
1						
2						
...						
n						




Figure 3: DP table

Backtracking

For the purpose of retrieving the optimal allocation, when solving each subproblem $s[i, k]$, we need to store the allocation m_i for player i that achieves such an optimal value, i.e.

$$m_i = \arg \max_j \{v_i(j) + s[i - 1, k - j]\}$$

Then to recover the optimal allocation m_1, m_2, \dots, m_n , we look at the last cell $s[n, m]$ that corresponds to the original problem with n players and m items.

Algorithm 2 Backtracking routine

Let $k = m, i = n$.

for $i = n; i > 0; i --$ **do**

- Let $m_i = j$ such that $s[i, k] = v_i(j) + s[i - 1, k - j]$
 - $k = k - m_i$
-

Using the representation model, we see that this algorithm is not computationally efficient, since it requires computing a total $(n + 1)(m + 1) = O(nm)$ number of subproblems, and each subproblem needs to make at most $O(m)$ comparisons in step (*). Hence, the runtime scales as $O(nm^2)$, in contradiction with the requirement that the runtime being polynomial in $\log m$.

2.2 Special case: Downward Sloping Valuations

Still, not all hope is lost. There is a special case under which we could obtain an efficient algorithm that produces the optimal solution. This is the case when the players' valuations are "Downward Sloping", i.e. the partial values u_i 's are monotonically decreasing. Mathematically,

$$u_i(k) \leq u_i(k-1) \text{ for } \forall 1 \leq k \leq m$$

The benefits of a downward sloping valuation is that it exhibits both optimality and efficiency properties.

Optimality

Claim 5 *There is a "clearing price" p with the property that*

- *by allocating $m_i = j$ items such that $u_i(j) \geq p > u_i(j+1)$ i.e. the partial value for the j^{th} allocated item exceeding the clearing price p , and that of the $j+1^{\text{th}}$ item strictly less than p .*
- *this allocation maximises $\sum_i v_i(m_i)$,*
- *and satisfies $\sum m_i = m$.*

Proof: Let $m_i = j$ with $u_i(j) \geq p > u_i(j+1)$. First we show that for all k ,

$$v_i(m_i) - m_i p \geq v_i(k) - kp \tag{L1}$$

By downward sloping, we have

$$u_i(k) - p \begin{cases} \geq 0 & \text{for } k \leq m_i \\ < 0 & \text{for } k > m_i \end{cases} \begin{matrix} (1) \\ (2) \end{matrix}$$

- if $k \leq m_i$.

$$\begin{aligned} v_i(m_i) - m_i p &= \sum_{j=1}^{m_i} (u_i(j) - p) \\ &= \sum_{j=1}^k (u_i(j) - p) + \sum_{j=k}^{m_i} (u_i(j) - p) \\ &\geq \sum_{j=1}^k (u_i(j) - p) = v_i(k) - kp \end{aligned}$$

since the second term $\sum_{j=k}^{m_i} (u_i(j) - p) \geq 0$ by (1).

- if $k > m_i$.

$$\begin{aligned}
v_i(k) - kp &= \sum_{j=1}^k (u_i(j) - p) \\
&= \sum_{j=1}^{m_i} (u_i(j) - p) + \sum_{j=m_i+1}^k (u_i(j) - p) \\
&< \sum_{j=1}^{m_i} (u_i(j) - p) = v_i(m_i) - m_i p
\end{aligned}$$

since the second term $\sum_{j=m_i+1}^k (u_i(j) - p) < 0$ by (2).

To prove that $(m_i)_{i=1}^n$ with $\sum_i m_i = m$ is the optimal solution, consider any other allocation $(k_i)_{i=1}^n$ with $\sum_i k_i \leq m$.

- By L1, we have $\sum_i (v_i(m_i) - m_i p) \geq \sum_i (v_i(k_i) - k_i p)$.
- Then since $\sum_i k_i \leq m$, $\sum_i (v_i(k_i) - k_i p) \geq \sum_i v_i(k_i) - mp$.
- On the other hand, $\sum_i m_i = m$, so $\sum_i (v_i(m_i) - m_i p) = \sum_i v_i(m_i) - mp$.
- Combining the inequalities, we have $\sum_i v_i(m_i) - mp \geq \sum_i v_i(k_i) - mp$. Hence, $\sum_i v_i(m_i) \geq \sum_i v_i(k_i)$. $(m_i)_{i=1}^n$ is the optimal solution.

■

In economics, this is a "market equilibrium". Essentially, p can be deemed as a fixed payment on each allocated item. For an item of a value exceeding the payment, the player would have a net gain from this allocation, hence he would wish to bid for this item; conversely, if payment exceeds an item's value, there is no incentive to bid. Since in downward sloping valuations, an item's value decreases as allocation amount increases, the clearing price p defines a player's allocation "demand", which is m_i . Thus, our task is to find the correct p such that the property $\sum_i m_i = m$ holds.

Efficiency

The trick with efficiency is that the m partial values can be viewed as a sorted decreasing array, hence given a fixed clearing price p , we can use binary search to find the demand m_i for each player. This only produces a complexity that scales with $O(\log m)$, satisfying the efficiency constraints.

We can use the same binary search trick to find the correct p . Due to the same reason that each partial valuation is monotonic decreasing, the corresponding $\sum_i m_i$ would be monotonic, hence binary search is applicable to find the right point that $\sum_i m_i = m$ holds. Notice that the search space for p is $[0, \max_i u_i(1)]$, which is upper bounded by the maximum value of a single item across the bidders (the first item is the most valuable because of downward sloping!). This search scope is reasonable since setting p beyond the maximum value would mean all players demanding nothing. Therefore, the complexity for doing so is $O(\log \max_i u_i(1))$, which is exactly the number of bits of precision t to represent a value. Hence, this falls within our efficiency requirements as well.

Algorithm 3 Allocation Algorithm for Downward Sloping Valuations

Let $lb = 0$, and $ub = \max_i u_i(1)$,

Loop

- Use binary search to find a clearing price p over the range $[lb, ub]$,
- For all players, use binary search over the range $\{0, 1, \dots, m\}$ to find m_i such that $u_i(m_i) \geq p > u_i(m_i + 1)$
- If $\sum_i m_i < p$, $ub = \frac{lb+ub}{2}$ since price is too high
- Else if $\sum_i m_i > p$, $lb = \frac{lb+ub}{2}$ since price is too low
- Else exit loop since we have found the right p

The optimal allocation is $(m_i)_{i=1}^n$ under this p .

2.3 Intractability

Despite that we have a special case where we could achieve both optimality and efficiency, in general, optimality would mean intractability. We use both models of efficiency (representation and data access) to prove the intractability results.

2.3.1 Representation

We could show that intractability holds for even the least expressive language. This naturally imply an impossible result for the more expressive ones.

The language we look at is the "Single-Minded Bid", the single-step function. The proof uses a reduction from the multi-unit auction problem with the single-step valuations to the Knapsack problem. Since Knapsack is NP-complete, and assuming that $P \neq NP$, then we know we cannot achieve polynomial runtime with even the simplest valuations.

Theorem 6 *Assuming $P \neq NP$, there is no polynomial-time algorithm for the multi-unit auction problem in the bidding language model.*

Proof: Reduction

- player i with valuation $v_i(k) = \begin{cases} 0 & \text{for } k < k^* \\ p & \text{for } k \geq k^* \end{cases}$ = item i with value p , weight k^* for Knapsack
- Allocating k^* items with value p to player i = Packing item i with value p and weight k^*
- m units of auction item = knapsack capacity is m
- Hence, allocating no more than m items = total weight of packed items no more than m
- Maximising total value of players = Maximising total value of packed items

■

2.3.2 Data Access Model

Value-Query

Efficiency in the value-query model has the requirement that the number of queries be sublinear in terms of m . We would show by contradiction using the simplest two-player scenario that this is impossible to achieve.

Theorem 7 *Every algorithm in the value-query model needs to make $O(m)$ queries for some problem instance.*

Proof: The specific problem instance we consider is a two-player scenario where both players have the same linear valuations $v_1(k) = v_2(k) = k$ for $0 \leq k \leq m$. Suppose we have an efficient algorithm that given the input (v_1, v_2) produces the optimal allocation m_1, m_2 within a runtime sublinear in m . Notice that for this special choice of valuations, the optimal allocation always has a total value of m , i.e. $v_1(k) + v_2(m - k) = k + (m - k) = m$.

However, the sublinear runtime in the value-query model would mean that the algorithm makes a number of queries less than $2m$, and there is at least a value $v_1(z)$ that it does not query. Now by changing $v_1(z) = z + 1$, we still maintain that the valuation is monotonic increasing, and thus this is a valid input to our algorithm. We would therefore expect the algorithm to produce the new optimal allocation $(m'_1 = z, m'_2 = m - z)$, since it gives a total value of $m + 1 > m$. However, since our algorithm never queries $v_1(z)$, it would still produce the original allocation with a total value of m , and thus we obtain a contradiction. We therefore conclude that any algorithm for the multi-unit auction would require $O(m)$ runtime for this two-player problem instance. ■

Communication-Query The same $O(m)$ result holds for the communication model. Here we would just count the number of answer bits to define the runtime.

Theorem 8 *Every algorithm in the communication model needs to ask queries whose total answer length has at least $m - 1$ bits for some problem instance*

Proof: The contradiction proof uses a set of dual valuations that always produce allocation of total value m as well. Consider a set $U = \{1, \dots, m - 1\}$, let $S \subseteq U$. We define

$$v_S(k) = \begin{cases} k & \text{if } k \notin S \\ k + 1 & \text{if } k \in S \end{cases} \quad v_S^*(k) = \begin{cases} k & \text{if } k \notin S \\ k - 1 & \text{if } k \in S \end{cases}$$

such that an (optimal) allocation $v_S(k) + v_S(m - k) = m$.

Notice there are 2^{m-1} such pairs of valuations as there are 2^{m-1} possible subsets S . The central trick is to show that these pairs of inputs would never produce the same answer sequences.

Suppose there exists two pairs of inputs (v_S, v_S^*) and (v_T, v_T^*) that produce the same answer sequences. This implies that a query to $v_{S/T}$ has the same answer as a query to v_{S^*/T^*} . Hence, the algorithm would produce the same answer sequences for the mixed inputs (v_S, v_T^*) and (v_T, v_S^*) as well. Since $S \neq T$, there exists some $k \in S$ but $k \notin T$. Then, the optimal solution for (v_S, v_T^*) has a total value

$$v_S(k) + v_T^*(m - k) = k + 1 + m - k = m + 1$$

However, this solution is not optimal for (v_T, v_S^*) ,

$$v_T(k) + v_S^*(m - k) = k + m - k - 1 = m - 1$$

Hence, we arrive at a contradiction, and it is not possible that (v_S, v_S^*) and (v_T, v_T^*) produce the same answer sequence.

Since we have 2^{m-1} such different answer sequences, we must use $m - 1$ bits to represent these different possibilities. Hence, there must be some answer sequence that has at least $m - 1$ bits. This provides a lower bound on the runtime of the algorithm. ■

2.4 Approximation

We have proved that it is in general not possible to achieve optimality within polynomial runtime. Hence, to achieve efficiency, we would relax our conditions and look at the approximation algorithms that have different level of closeness to optimality.

Definition 9 (Approximation) *Let $A(I)$ be an approximation solution by a polynomial-time algorithm A , $Opt(I)$ be the optimal solution.*

$$\text{maximisation approximation ratio } \alpha = \min \frac{f(A(I))}{f(Opt(I))}$$

where $f(\cdot)$ is the objective function of the maximisation problem.

Definition 10 (α -approximation algorithm) *For a maximisation problem, an α -approximation algorithm always returns a solution $A(I)$ such that*

$$\alpha * f(A(I)) \leq f(Opt(I))$$

with $\alpha < 1$.

Definition 11 (Fully polynomial-time approximation scheme) *A maximisation problem admits a fully polynomial-time approximation scheme (FPTAS) if, given any $0 < \epsilon < 1$, the algorithm always returns a solution $A(I)$ such that*

$$(1 - \epsilon) * f(A(I)) \leq f(Opt(I))$$

in polynomial time in terms of $\frac{1}{\epsilon}$

Truncation Approximation Scheme

By modifying the General Allocation Algorithm, we could obtain a FPTAS for the multi-unit auction problem. Recall that the General Allocation Algorithm fails to achieve efficiency because there are m columns in the DP table, i.e. nm subproblems, and each subproblem takes at most m comparisons. The trick for an approximation scheme is therefore to reduce both the the number of columns in the Dynamic Programming table and the number of comparisons for a single subproblem.

The approach we take is to truncate the valuations such that the number of different values we can take is expressed in some form in $\frac{n}{\epsilon}$, rather than m . Then we shall exploit this property to define a different subproblem such that both the column number and comparison number scale in terms of $\frac{n}{\epsilon}$.

The truncation scheme that allows us to precisely do so is:

- Set truncation precision $\delta = \frac{\epsilon V}{n}$, where $V = \max_i v_i(m)$ is the maximum value across the valuations;
- Truncate each value into some integer multiple $w\delta$, where $0 \leq w \leq \frac{V}{\delta} = \frac{V}{\epsilon V/n} = \frac{n}{\epsilon}$.

Notice now that the total value across n bidders is capped by the integer multiple $nV = \frac{n^2}{\epsilon}\delta$. Thus, we could define our new subproblem based on the total value achieved by an optimal allocation, and the column size of the DP table would become $\frac{n^2}{\epsilon}$; Also, the number of different values is at most $\frac{n}{\epsilon}$, which means that we restricted the comparison number for a single subproblem to $O(\frac{n}{\epsilon})$ as well.

- **Subproblem**

Given the first $i \leq n$ players, and a constraint on the total value $w\delta \leq \frac{n^2}{\epsilon}\delta$,

Find the minimum number of items s.t. when optimally allocated among the i players, yield a total value $\geq w\delta$

- **Optimal substructure**

Let $K[i, w]$ be the minimum number of items obtained for a subproblem of i players and total value constraint $w\delta$. We have the relation

$$K[i, w] = \min_{0 \leq j \leq \frac{n}{\epsilon}} \{k + K[i - 1, w - j]\}$$

where k is the minimum number of items such that $v_i(k) \geq j\delta$.

We have $\frac{n}{\epsilon}$ possibilities to satisfy a total value constraint, i.e. requiring $j\delta$ value contributed by the i^{th} player, and $(w - j)\delta$ value from the first $i - 1$ players. Notice that $0 \leq j \leq \frac{n}{\epsilon}$ since the value contributed by a single player is no more than the maximum value $V = \frac{n}{\epsilon}\delta$.

To find the minimum number of items k required from the i^{th} player, we can just use a binary search over its valuation (since the valuation is monotonic). This gives us a complexity factor of $O(\log m)$. The minimum number of items required from the first $i - 1$ players is stored in $K[i - 1, w - j]$. Thus, the total minimum is a comparison out of these $\frac{n}{\epsilon}$ possibilities, each taking $O(\log m)$ time to compute.

Algorithm 4 Truncation Approximation Algorithm ($\{v_i\}_{i=1}^n$ after truncation)

- Construct a dynamic programming table of $(1 + n) * (1 + \frac{n^2}{\epsilon})$ size, where row i represent the first i players, and column w represent the total value constraint $w\delta$. Cell $K[i, w]$ stores the minimum number of items s.t. when optimally allocated among the i players, yield a total value $\geq w\delta$.
- - $K[0, w] = \infty$, since it is impossible to obtain a total value of $w\delta$ from 0 players.
 - $K[i, 0] = 0$, since we require 0 items to obtain a total value of 0.
 - Fill the table by the optimal substructure for $0 < i \leq n$ and $0 < w \leq \frac{n^2}{\epsilon}$

$$K[i, w] = \min_{0 \leq j \leq \frac{n}{\epsilon}} \{k + K[i - 1, w - j]\}$$

where k is the minimum number of items such that $v_i(k) \geq j\delta$.

Use binary search to find such a k .

#bidders/ total value constraint	0	1	n^2/e
0					
1			look at previous n/e cells in the row above		
...				(i, w)	
n					

Figure 4: DP table

Backtracking

Notice that the solution to the original problem has a maximal total value such that the number of allocated items $\leq m$. This corresponds to a subproblem of n players and a maximal total value constraint $w\delta$ such that the minimum number of items required $\leq m$. In the table, this is the cell at row n with maximum index w such that $K[n, w] \leq m$. To obtain the corresponding allocation, notice that for each subproblem, we have computed the optimal partition of value $j\delta$ that comes from player i and the corresponding number of items k required such that $v_i(k) \geq j\delta$. Hence, we shall just set $m_i = k$, and backtrack for one player a time.

Algorithm 5 Backtracking routine

Let $i = n$, w be the minimum index such that $K[n, w] \leq m$.

for $i = n$; $i > 0$; $i --$ **do**

- Let $j = \arg \min_{0 \leq j \leq \frac{n}{e}} \{k + K[i - 1, w - j]\}$
 - Let $m_i = k$ such that $v_i(k) \geq j\delta$
 - $w = w - j$
-

This algorithm is efficient since there is in total $n * \frac{n^2}{e}$ subproblems, each subproblem takes $O(\frac{n}{e} \log m)$ time. Moreover, it is $1 - \epsilon$ approximation scheme, since by our choice of truncation precision $\delta = \frac{\epsilon V}{n}$, we have a total additive error at most $n\delta = \epsilon V$. Now notice that the optimal value is lower bounded by V , hence the fraction of total value loss is at most $\frac{\epsilon V}{V} = \epsilon$. This means that the approximated value $f(A)$ satisfies $(1 - \epsilon) * f(Opt) \leq f(A) < f(Opt)$. Hence, combining both polynomial time and $(1 - \epsilon)$ approximation ratio, we know that the truncation approximation scheme is a FPTAS for the Multi-Unit Auction problem.

3 Truthfulness

As mentioned, the two core ideas in Algorithmic Mechanism Design is efficiency and truthfulness. While we have considered exclusively how to ensure efficiency from the algorithmic perspective, we now add the strategic model into the scene, and consider how to ensure incentive compatibility of a mechanism.

To define the strategic model, we need two new notions, payment and utility. Payment p_i for a player is the price he is asked to pay upon an allocated m_i items. Utility (quasilinear) $U_i = v_i(m_i) - p_i$ is the difference between the value from an allocation and its corresponding payment. The incentive of a player is then to maximise his utility by choosing a dominant strategy. In the case of Multi-Unit Auction, players' strategy would be to report their valuations in a way that his own utility is maximised. In other words, the valuations can be false reported, and the allocation from the designed mechanism could potentially be not optimised with respect to the true valuations. Hence, we require that we design mechanism in such a way that players have no incentive to lie, i.e. their dominant strategies is being truthful. Can we always achieve this? By the "Revelation Principle" in economics, as long as dominant strategies exist, one can design an incentive compatible mechanism by internalising the dominant strategies.

Definition 12 *A mechanism $M = (f(\cdot), p(\cdot))$ is truthful, or incentive compatible, if we have for all player i , the utility from any false valuation \hat{v}_i is never higher than the utility from true valuation v_i .*

$$U_i = v_i(f_i(v_i, v_{-i})) - p_i(v_i, v_{-i}) \geq U'_i = v_i(f_i(\hat{v}_i, v_{-i})) - p_i(\hat{v}_i, v_{-i})$$

where $f : V_1 \times V_2 \times \dots \times V_n \rightarrow A$ is the social choice function that outputs the allocation,
 (v_i, v_{-i}) is the strategy profile that contains a strategy for each player,
 $p_i : V_1 \times V_2 \times \dots \times V_n \rightarrow \mathbb{R}$ is the payment function for player i .

3.1 Vickrey-Clarke-Groves Mechanisms

We first see a famous truthful mechanism in economics, without considerations for efficiency. This VCG mechanism ensures truthfulness by using a payment rule that charges each player his "externality" on others. In economics, externality refers to the indirect cost that imposes on others that are not involved in an economic activity. In the context of auction, it refers to a bidder's harm caused to others by participation alone, regardless of reporting a true or false valuation. This implies that the payment would be independent of his valuation, and aligns the utility with the social welfare. Hence, as long as the mechanism maximises the social welfare, there would be no incentive for lying.

Algorithm 6 VCG Mechanism $(\{v_i\}_{i=1}^n)$

Output allocation (m_1, m_2, \dots, m_n) that maximises the social welfare $\sum_i v_i(m_i)$

For each player i :

- The mechanism finds the allocation $(m'_1, m'_2, \dots, m'_n)$ that maximises the value of all other players $\sum_{j \neq i} v_j(m'_j)$
 - Charge player i by $p_i = \sum_{j \neq i} v_j(m'_j) - \sum_{j \neq i} v_j(m_j)$
-

Proof:[Truthfulness] The payment term ensures that each player's incentive to maximise his utility is aligned with the social welfare since the utility now becomes

$$u_i = v_i(m_i) - p_i = v_i(m_i) + \sum_{j \neq i} v_j(m_j) - \sum_{j \neq i} v_j(m'_j)$$

The player cannot adjust the term $\sum_{j \neq i} v_j(m'_j)$ since it is independent of valuation v_i . Now notice that the mechanism optimises $v_i(m_i) + \sum_{j \neq i} v_j(m_j)$ based on the valuations (v_i, v_{-i}) players give, and produces the

corresponding allocation $\{m_i\}_{i=1}^n$. From a different perspective, it means that any other valuations (v'_i, v_{-i}) would produce no better value than (v_i, v_{-i}) on this allocation $\{m_i\}_{i=1}^n$. Hence, if a player misreports v_i to the mechanism and obtains an output (m_i, m_{-i}) , the true valuation v'_i would give a suboptimal utility term that is no better than what the mechanism optimises for him, i.e.

$$v'_i(m_i) + \sum_{j \neq i} v_j(m_j) \leq v_i(m_i) + \sum_{j \neq i} v_j(m_j)$$

Hence, the player has no incentive for lying. ■

However, the VCG mechanism is not efficient, since to ensure incentive compatibility with social welfare, we need to truly optimise the total welfare, which means a use of the $O(nm^2)$ General Allocation Algorithm. Then, as we have seen earlier that approximation can yield good efficiency, the problem then becomes "can we relax the condition of optimality while still ensuring truthfulness".

Claim 13 *If we simply replace the General Allocation Algorithm by an approximation scheme, the VCG mechanism would not remain truthful.*

Proof: Consider the Truncation Approximation Scheme. On a scenario with two players and two items, and the same valuation $v(1) = 1.9$, $v(2) = 3.1$, the truncated valuation would be $v(1) = 1$, $v(2) = 3$. The optimal allocation would be two items for one player (say player 1), and none for the other (player 2), yielding a total value of 2. The utility for player 2 is $3.1 - 3.1 = 0$. However, if player 2 misreports $v(1) = v(2) = 3.1$, the optimal allocation under the truncation scheme would become one for each player, yielding a total value of 4. The new utility for player 2 is $3.1 - 1.9 = 1.2 > 0$. Hence, player 2 would have an incentive to lie. The VCG mechanism with an approximation scheme is no longer truthful. ■

3.2 Maximum-in-Range

The reason why simply replacing the optimisation algorithm with an approximation scheme fails with VCG is that, truthfulness requires the mechanism to consider all possible allocations for the valuations given. Hence, the trick would be not to compress the valuations within the mechanism, but do so before running the mechanism, i.e. giving a compressed version of the valuations (in some form of n) to the mechanism to ensure efficiency, and use the optimisation algorithm within to ensure truthfulness.

The way to do this is to bundle the m items such that total number of bundles is expressed in some form of n . The range cannot be too small for a good approximation ratio. Without loss of generality, we can choose to split m into n^2 bundles, each of size $\frac{m}{n^2}$. This ensures a constant approximation ratio of $\frac{1}{2}$.

Claim 14 *The approximation ratio for the Maximum-in-Range Mechanism with $\frac{m}{n^2}$ bundling is $\frac{1}{2}$. There is no Maximum-in-Range Mechanism with approximation ratio better than $\frac{1}{2}$.*

Proof: To see why the mechanism with $\frac{m}{n^2}$ bundling produces at least half of the total value of the optimal allocation, consider the optimal allocation $(m'_i)_{i=1}^n$, and the player i who is allocated the most number of items, i.e. $m_i \geq \frac{m}{n}$.

- If at least half the total value comes from player i , i.e. $v_i(m'_i) \geq \frac{1}{2} \sum_i v_i(m'_i)$

With the $\frac{m}{n^2}$ bundling, we can take everything from all other players and allocate solely to player i alone. This guarantees that the new allocation $(m_i, m_{-i}) = (m, 0)$ has a total value $\sum_i v_i(m_i) = v_i(m_i) \geq v_i(m'_i) \geq \frac{1}{2} \sum_i v_i(m'_i)$. The Maximum-in-Range mechanism would have optimised over this allocation and could only have better total value.

- If less than half the total value comes from player i , i.e. $v_i(m'_i) < \frac{1}{2} \sum_i v_i(m'_i)$

In other words, $\sum_{j \neq i} v_j(m'_j) \geq \frac{1}{2} \sum_i v_i(m'_i)$. Hence, we can take everything from player i and allocate to all other players while ensuring that each unit of allocation satisfies $\frac{n^2}{m}$. This allocation would be considered by the Maximum-in-Range mechanism, hence we are sure the optimal total value $\geq \frac{1}{2} \sum_i v_i(m'_i)$

The reason why the mechanism cannot have better than $\frac{1}{2}$ approximation ratio is that it only looks at a subset of all allocations. Hence, in a two player scenario, there would be at least one allocation, say $(m_1, m - m_1)$, that the mechanism does not consider. Now if the problem input is

$$v_1(k) = \begin{cases} 1 & \text{if } k \geq m_1 \\ 0 & \text{otherwise} \end{cases} \quad v_2(k) = \begin{cases} 1 & \text{if } k \geq m - m_1 \\ 0 & \text{otherwise} \end{cases}$$

the mechanism would not consider the optimal solution which is exactly $(m_1, m - m_1)$ that has a total value of 2. All other solutions that are possibly returned by the mechanism has only a total value of 1. Hence, the approximation ratio is upper bounded by the worst case input which gives an $\alpha = \frac{1}{2}$. ■

3.3 Single Parameter Mechanism

Apart from the VCG-based techniques, there is a non-VCG technique that provides truthful mechanism. This only applies when the private valuation of each bidder is single-dimensional. Here the closest representation to being single-dimensional is the "single-minded bid" (that each bidder only specifies a single bid and its corresponding value (k_i^*, p_i)). Recall mathematically, it means

$$v_i(k) = \begin{cases} 0 & \text{for } k < k_i^* \\ w_i & \text{for } k \geq k_i^* \end{cases}$$

As in the reduction proof for intractability, we assume that the mechanism exactly allocates either 0 or k_i^* items, giving a value of 0 or w_i respectively. Using game-theoretic terminologies, we say a bidder wins if he is allocated k_i^* items, or lose if 0. Also, we can assume that a loss incurs no payment.

Properties for Truthfulness

Under the single parameter settings, the necessary and sufficient conditions for a mechanism to be truthful is that it satisfies these two properties:

- **Monotonicity** : That a player wins with bid (k_i, v_i) will also win with $k'_i < k_i$ and $v'_i > v_i$, assuming other players do not change their bids.
- **Critical payment** : The payment for a winning bid (k_i, v_i) is the smallest value needed to win for this k_i amount of items.

Proof: With monotonicity and critical payment, we show why reporting the true valuation (k_i^*, w_i) is the dominant strategy.

- In terms of the bidding amount, k_i^* dominates $k_i \neq k_i^*$.
 - If reporting $k_i < k_i^*$, then a win with k_i gives $v_i(k_i) = 0$, but payment $p_i > 0$, yielding a utility $U_i = v_i(k_i) - p_i < 0$.
 - If reporting $k_i > k_i^*$, then by monotonicity, a win with k_i also means a win with k_i^* , by the single step valuation, $v_i(k_i) = v_i(k_i^*)$; by critical payment, payment for a smaller number of items is never higher than a larger amount, hence the utility for misreporting is never higher than being truthful.
- In terms of the value, w_i dominates $w'_i \neq w_i$.
 - If w_i is winning, then the critical value $p_i \leq w_i$. By critical payment, misreporting any $w'_i \geq p_i$ also wins and pays the same amount p_i , yielding the same utility; misreporting $w'_i < p_i$ means a loss, giving a 0 utility.
 - If w_i is losing, then the critical value $p_i > w_i$. By critical payment, misreporting any $w'_i < p_i$ also loses, yielding the same 0 utility; misreporting $w'_i \geq p_i$ wins but $U_i = w_i - p_i < 0$.

■

That truthfulness only requires these two properties provide us the benefits that algorithmically, we only need to ensure monotonicity; once we add the critical payment rule, truthfulness is guaranteed. Hence, our task is to find a monotonic approximation algorithm with a good approximation ratio and efficiency.

FPTAS for Single-Parameter Mechanism

The first approximation scheme that has both an arbitrary good approximation ratio and efficiency is the Truncation Approximation Scheme. Recall that it ensures approximation ratio $(1 - \epsilon)$ by capping the loss of value due to truncation by $\epsilon \max v_i$; and ensures efficiency by restricting the number of different values across all valuations to $\frac{n}{\epsilon}$. Hence, we would do similarly for the mechanism that has the additional truthfulness consideration.

Algorithm 7 Single-Minded Approximation Mechanism

Let $\epsilon > 0$ be the required approximation level, and δ be the truncation precision

Truncate each value to an integer multiple $j\delta$ where $0 \leq j \leq \frac{n}{\epsilon}$

Apply the Truncation Approximation Algorithm

Charge the critical payment

Now we consider truthfulness. Notice that the truncation operation is indeed monotonic, since it does not change the properties of the valuations or the allocation. However, the problem arises when we discover that the truncation precision it uses depends on the maximal value among the valuations, i.e. $\delta = \frac{\epsilon \max v_i}{n}$. This means that if we want to check the monotonicity principle, by increasing the maximal value, the truncation precision can change in such a way that an original winning bid does not ensure winning anymore.

The solution would be to make δ independent of the player's valuations, i.e. to consider all δ possible and pick the one that returns the highest total value after truncation. However, it is not possible to try an infinite number of δ , we would therefore need to restrict the search to only useful ones for the purpose of efficiency. The range we choose is $\frac{\epsilon \max v_i}{2n^2} \leq \delta \leq \max v_i$. For $\delta > \max v_i$, all values are truncated to 0, which is trivial; for $\delta < \frac{\epsilon \max v_i}{2n^2}$, each value would be truncated to at most the maximal integer multiple $\frac{n}{\epsilon} * \delta = \frac{n}{\epsilon} * \frac{\epsilon \max v_i}{2n^2} = \frac{\max v_i}{2n}$, giving a total value at most $\frac{\max v_i}{2}$, which is even smaller than the value obtained from using the original $\delta = \frac{\epsilon \max v_i}{n}$.

In addition, we need to select a finite number of δ 's out of this chosen range. The clever choice is by setting δ as an integer power of 2. This gives the benefit that there would only be at most $\log \max v_i$ cases to consider, giving a complexity factor of $O(\log \max v_i) = O(t)$, exactly one of the polynomial time parameter, the number of precision bits to represent a value.

Now notice that the approach of maximising among different δ 's is indeed a monotonic operation, particularly because our valuation is "single-minded bid". The property of the single parameter settings is that a bidder can only affect the final total value by either winning or not. Hence, if we check the monotonicity principle, by increasing the value of a winning bid, and assuming other players do not change; if this results in a different δ that gives a higher total value, then it must be that it is the bidder that changes

his valuation contributes to this increase in the total value. Hence, he must be winning in the new *delta* allocation. Hence, we keep the mechanism monotonic.

Algorithm 8 Single-Minded Approximation Mechanism

Let $\epsilon > 0$ be the required approximation level, and δ be the truncation precision

for $\delta = 2^t$, such that $\frac{\epsilon \max v_i}{2n^2} \leq \delta \leq V$ **do**

 Truncate each value to an integer multiple $j\delta$ where $0 \leq j \leq \frac{n}{\epsilon}$

 Apply the Truncation Approximation Algorithm to obtain a " δ -scale allocation"

Choose the " δ -scale allocation" that achieves the highest total truncated value

For each winner i in the final allocation, charge the critical payment by using binary search over the values of the n bidders, find the minimum value that keeps the bidder i winning (the largest value $v_{j \neq i}$ smaller than v_i)

Since the Truncation Approximation Algorithm is FPTAS, and we only increase the complexity by a factor of $O(t)$, where t is the number of precision bits to represent a value, the Single-Minded Approximation Algorithm is still FPTAS; to charge the critical payment for player i , there is only $O(n \log n)$ operations by performing at most n times of binary search, hence the Single-Minded Approximation Scheme is FPTAS.

3.4 Randomisation

Apart from the deterministic mechanisms, we can also introduce randomness into our design scope. We need to redefine some notions for the randomised mechanism.

Definition 15 (Randomised Multi-Unit Auction) *A randomised mechanism $(f(\cdot), p(\cdot))$*

- *is a social choice function $f : V_1 \times V_2 \times \dots \times V_n \rightarrow A \sim$ that takes as input the valuations of the players $(v_i)_{i=1}^n$ and outputs a distribution D over allocations that maximises the expected social value $\sum_i v_i(D) = \mathbb{E}_{(m_i)_{i=1}^n \sim D}[\sum_i v_i(m_i)]$*
- *and a set of payment functions $(p_i)_{i=1}^n : V_1 \times V_2 \times \dots \times V_n \rightarrow \mathbb{R}$*

A randomised mechanism is truthful (in expectation) if the expected utility from misreporting is no better than the expected utility from reporting the true valuation, i.e.

$$\mathbb{E}[v_i(M_i) - P_i] \geq \mathbb{E}[v'_i(M_i) - P'_i]$$

where v_i is the true valuation, M_i and P_i are random variables of the allocation and payment for player i .

The approximation ratio of a randomised mechanism is

$$\alpha = \min \frac{\mathbb{E}[f(A(I))]}{f(\text{Opt}(I))}$$

To extend from determinism to randomisation while ensuring all three properties 1. efficiency, 2. truthfulness and 3. a good approximation ratio, we adapt the deterministic Maximum-in-Range mechanisms to the

randomised Maximum-in-Distributed-Range mechanism. It turns out that the Maximum-in-Distributed-Range is a FPTAS for randomised multi-unit auction problem. The idea here is to specify a compressed set of distributions on allocations to ensure efficiency and good approximation, and run the VCG mechanism on it to ensure truthfulness.

The difficulty lies in constructing the distributions such that we can reduce the search space while ensuring a good approximation ratio. First, it is proven that a randomisation range of 2 is sufficient. Hence, for each allocation $(m_i)_{i=1}^n$, we choose to randomise over two range : $(m_i)_{i=1}^n$ and (0) , the empty allocation. We would define the corresponding distribution $(1 - z(m_i)_{i=1}^n, z(m_i)_{i=1}^n)$ where z is a "penalty" probability function that depends on $(m_i)_{i=1}^n$ (For clarity, sometimes z is used in place of $z(m_i)_{i=1}^n$).

Notice that the mechanism maximises the expected social welfare which now becomes

$$\mathbb{E}_{(m_i)_{i=1}^n \sim (1-z, z)} \left[\sum_i v_i(m_i) \right] = (1 - z) \sum_i v_i(m_i) + z * 0 = (1 - z) \sum_i v_i(m_i)$$

To ensure a $1 - \epsilon$ approximation ratio, we set $0 \leq z < \epsilon$ for all $(m_i)_{i=1}^n$. Since our mechanism maximises over $(1 - z) \sum_i v_i(m_i)$ for all possible allocations, it is clear that for an allocation $(m_i)_{i=1}^n$, we have $\sum_i v_i(m_i) \geq (1 - z) \sum_i v_i(m_i) \geq (1 - z') \sum_i v_i(m'_i) \geq (1 - \epsilon) \sum_i v_i(m'_i)$ for all $(m'_i)_{i=1}^n$, which holds for the optimal solution as well.

To ensure that we can trim down the search space on allocations, we would define z such that it penalises allocation m_i that is not a power of 2. Let a weight of an allocation $w(m_i)$ be the largest power of 2 that divides m_i , i.e. $w(m_i) = \log m_i$ where $m_i \% 2 == 0$. Let the total weight of an allocation be the sum of the individual weights $w(m_i)_{i=1}^n = \sum_i w(m_i)$. Define the penalty function

$$z = (C - \sum_i w(m_i)) * \delta$$

where $C = n \log m$, the maximum weight of any allocation;

and $\delta = \frac{\epsilon}{C}$, a scaling factor to ensure that z is a valid probability within the range of $[0, 1]$

With this penalty function, we can trim the search space by the following rule:

- For two allocations $m_i < m'_i$, with $w(m_i) > w(m'_i)$
- If $\frac{v_i(m_i)}{v_i(m'_i)} > 1 - \delta$,
- then m'_i can be trimmed

The reason can be found by looking at the expected social welfare

$$(1 - z) \sum_i v_i(m_i) = (1 - C\delta + \delta \sum_i w(m_i)) * \sum_i v_i(m_i)$$

The potential gain from an increase in v_i is $(1 - z)v_i(m'_i) = (1 - z) * \frac{\delta}{1 - \delta} v_i(m_i)$; but the loss from a decrease in $w(m_i)$ is multiplied for all players $v_{j \neq i}$. We say the total gain cannot offset the loss.

Hence, we can define a recursive subroutine $LIST_i(L, t)$ that each time looks at the search space in half, and trims away the lower half if the above rule is satisfied. The subroutine takes as input a start point for

search $0 \leq L \leq m$, and a search scope (as a weight) $0 \leq t \leq \log m$ and returns all non-trimmable values of allocation m_i within the range $L < m_i \leq L + 2^t$:

Algorithm 9 Allocation subroutine $LIST_i(L, t)$

- If $t = 0$, then return the non-trimmable singleton $\{L + 1\}$.
 - If $\frac{v_i(L)}{v_i(L+2^{t-1})} < 1 - \delta$, return $LIST_i(L, t - 1)$ concatenated with $LIST_i(L + 2^{t-1}, t - 1)$.
which means if the increase in value from an allocation L to an allocation $L + 2^{t-1}$ is large, then we cannot trim away values in between, so we perform recursion on both halves, with a smaller search scope $t - 1$.
 - Else if $\frac{v_i(L)}{v_i(L+2^{t-1})} > 1 - \delta$, return $LIST_i(L + 2^{t-1}, t - 1)$.
which means if the increase is small, then we can trim away all allocations $L < m_i \leq L + 2^{t-1}$ and look at value from $L + 2^{t-1}$ above. Hence, we only search the upper half with the smaller search scope $t - 1$
-

Since the subroutine breaks the search space $0 \leq m_i \leq m$ into half each time, the number of recursion level is bound by $O(\log m)$. At a level t , we would only make two recursive calls if only the $v_i(L)$ and $v_i(L + 2^{t-1})$ has a $(1 - \delta)$ gap, and the values for comparison are all in the power of 2, hence there could be at most $O(\frac{\log \max v_i}{\delta})$ number of branches at a level t . Thus the runtime is in total bounded by $O(\log m * \frac{\log \max v_i}{\delta})$, where $\log \max v_i$ is the number of precision bits to represent a value. Hence, the subroutine satisfies the efficiency requirements.

Combining the search space reduction technique with the Maximum-in-Range Mechanism, we therefore have the complete Maximum-in-Distributed-Range Mechanism that is both efficient and truthful, and has a arbitrarily good approximation ratio (even better than the constant $\frac{1}{2}$ approximation ratio of Maximum-in-Range).

Algorithm 10 Maximum-in-Distributed-Range Mechanism

For each player i , compute a list $List_i$ of all possible allocations to player i by calling the subroutine $LIST_i(0, \log m)$.

Use the General Allocation Algorithm to find the allocation $(m_i)_{i=1}^n$ that maximises the expected social welfare $(1 - z) \sum_i v_i(m_i)$

Calculate probability $z = (C - \sum_i w(m_i)) * \delta$.

With probability z , output the empty allocation, otherwise $(m_i)_{i=1}^n$.

Compute VCG payment for each bidder i by:

Find allocation $(m'_i)_{i=1}^n$ that optimises the expected welfare without player i , i.e. $(1 - z) \sum_{j \neq i} v_j(m'_j)$, and charge $p_i = \sum_{j \neq i} v_j(m'_j) - \sum_{j \neq i} v_j(m_j)$

Reference

Noam Nisan (2014) *Algorithmic Mechanism Design, Through the lens of Multi-unit auctions*

END.