3.5 Bayesian Algorithm and Mechanism Design

In the preceding section we saw that worst case approximation factors for tractable algorithms can may be so large that they do not distinguish between good algorithms and bad ones. We also noted that mechanisms must satisfy an additional requirement beyond just having good performance; the allocation rule must also be monotone. For the example of combinatorial auctions we were luck and our approximation algorithm was monotone. Beyond greedy algorithms, such luck is the exception rather than the rule. Indeed, the entanglement of the monotonicity constraint with the original approximate optimization problem that the designer faces suggests that approximation mechanism design, from a computational point of view, could be more difficult than approximation algorithm design.

Imagine a realistic setting where a designer wishes to design a mechanism for some setting where worst-case approximation guarantees do not provide practical guidance in selecting among algorithms and mechanisms. Without some foreknowledge of the setting, improving beyond the guarantees of worst-case approximation algorithms is impossible. Therefore, let us assume that our designer has access to a representative data set. The designer might then attempt to design a good algorithm for this data set. Such an algorithm would have good performance on average over the data set. In fact, in most applied areas of computer science this methodological paradigm for algorithm design is prevalent.

Algorithm design in such a statistical setting is a bit of an art; however, the topic of this text is mechanism design not algorithm design. So let us assume that this algorithmic design problem is solved. Our mechanism design challenge is then to reduce the mechanism design problem to this algorithm design problem, i.e., to show that any algorithm, with access to the true private values of the agents, can be turned into a mechanisms, where the agents may strategize, and in equilibrium the outcome of the mechanism is as good as that of the algorithm. Such a result would completely disentangle the incentive constraints from the algorithmic constraints. Notice that the VCG approach solves this mechanism design problem for an optimal algorithm designer; here we solve it for an *ad hoc* algorithm designer.

This statistical setting fits neatly into the context of Bayesian mechanism design that we have already discussed. The main result of this section is the constructive proof of the following theorem.

Theorem 3.22 *For any single-parameter agent setting, any product distribution* $\mathbf{F}$*, and any algorithm* $\mathcal{A}$*, there is a BIC mechanism* $\bar{\mathcal{A}}$ *satisfying* $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}\big[\bar{\mathcal{A}}(\mathbf{v})\big] \geq \mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}(\mathbf{v})]$.

3.5.1 Monotonization

Let us focus on a single agent i. Our algorithm, has allocation rule $x_i(v_i) = \mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[x_i(\mathbf{v}) \mid v_i]$. Recall, this is the probability that we allocate to i when i has value v_i and the other agents' values are drawn from the distribution. If $x_i(\cdot)$ is monotone non-decreasing then there exist a payment rule, via the payment identity in the BIC characterization (Corollary 2.15), such that truthtelling is a best response for i (assuming others also truthtell). If $x_i(\cdot)$ is non-monotone then there is no such payment rule. Therefore the challenge before us is the

potential non-monotonicity of $x_i(\cdot)$. Our goal will be to construct an $\bar{x}_i(\cdot)$ from $x_i(\cdot)$ with the following properties.

1. (monotonicity) $\bar{x}_i(\cdot)$ is monotone non-decreasing.
2. (surplus preservation) $\mathbf{E}_{v_i \sim F_i}[v_i \bar{x}_i(v_i)] \geq \mathbf{E}_{v_i \sim F_i}[v_i x_i(v_i)]$.
3. (locality) No other agent j can tell whether we run $x_i(v_i)$ or $\bar{x}_i(v_i)$.

That the first two conditions are needed is intuitively clear. Monotonicity is required for BIC. Surplus preservation is required if our construction is to not harm our objective. The requirement of localized is more subtle; however, notice that if no other agent can tell whether we are running $x_i(\cdot)$ or $\bar{x}_i(\cdot)$ then we can independently apply this construction to each agent.

Assume that the distribution functions F_i are continuous over their support. This assumption implies that $F_i(z) = \mathbf{Pr}[v_i \leq z]$ is a continuous, monotone function and its inverse $F_i^{-1}(q)$ is well defined. Therefore the cumulative distribution function and its inverse, $F_i(\cdot)$ and $F_i^{-1}(\cdot)$, enable transformation back and fourth between *valuation space* and *probability space*. Notice that while the values may range over $[0, \infty)$ the probabilities are always on $[0, 1]$. Furthermore, the distribution of i's value in probability space is always uniform. This simplification makes Bayesian mechanism design problems much more intuitive when considered in probability space.

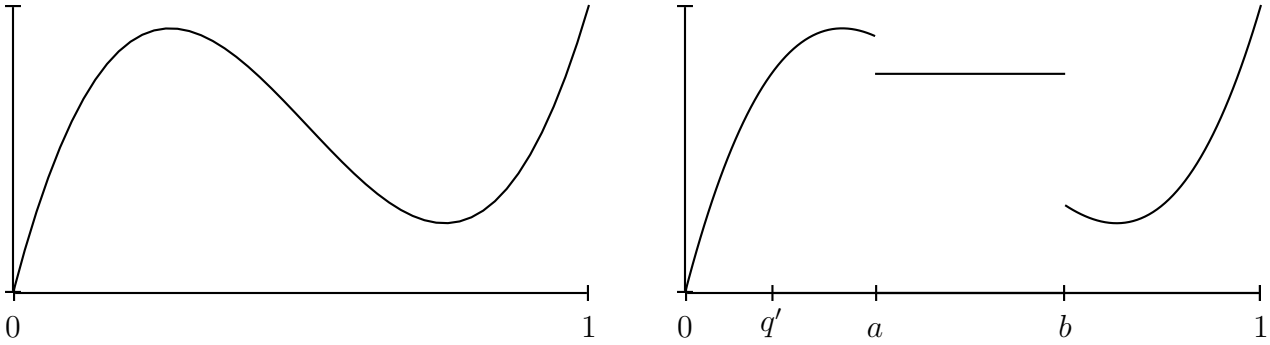
We will focus for the sake of exposition on a two agent case and name the agents Alice and Bob. Our goal will be to monotonize Alice's allocation rule without affecting Bob's allocation rule. Our discussion will focus on Alice and for notational cleanliness we will drop subscripts.

Alice has value $v \sim F$ and faces non-monotone allocation rule $x(v)$. Let $q = F(v)$ represent Alice's value in probability space and let $g(q) = x(F^{-1}(q))$ represent Alice's allocation rule in probability space. Since $F(\cdot)$ monotone, $x(\cdot)$ is monotone if and only if $g(\cdot)$. Therefore we focus on monotonizing $g(\cdot)$.

Resampling and Locality

Notice first, if the allocation rule for Alice is non-monotone over some interval $[a, b]$ then one way to make it monotone in this interval is to treat her the exact same way regardless of where her value lies within this interval. This would result in a constant allocation in the interval and a constant allocation is non-decreasing, as desired. There are many ways to do this, for instance, if $q \in [a, b]$ we can run $g(q')$ instead of $g(q)$. (Back in valuation space, this can be implemented by ignoring Alice's value v and inputing $v' = F^{-1}(q')$ into the algorithm instead.) Unfortunately, if we did this, Bob would notice. The distribution of Alice's input would no longer be F, for instance it would have no mass on interval (a, b) and a point mass on q'. See Figure 3.2(b).

Notice second, there is a very natural way to fix the above construction to leave the distribution of Alice's input to the algorithm unchanged. Instead of the arbitrary choice of inputing q' into the algorithm we can resample the distribution F on interval $[a, b]$. In

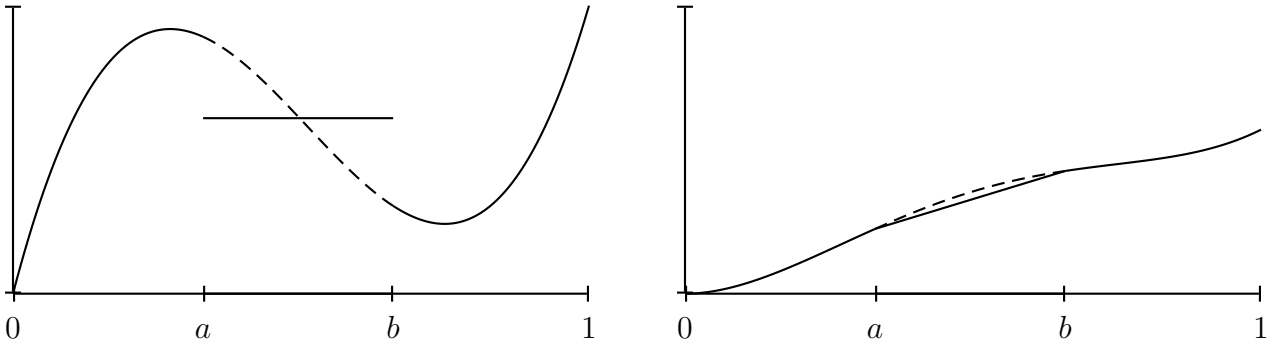


(a) allocation rule $g(q)$. (b) allocation rule $g'(q)$.

Figure 3.2: The allocation rule $g(q)$ and $g'(q)$ constructed by running $g(q')$ when $q \in [a, b]$.

probability space this is corresponds precisely with uniformly picking q' from $[a, b]$. Formally, the proposed transformation is the following. If $q \in [a, b]$ then resample $q' \sim U[a, b]$. If $q \notin [a, b]$ then simply set $q' = q$. Now run $g(q')$ instead of $g(q)$, i.e., simulate the algorithm with input $v' = F^{-1}(q')$ in place of Alice's original value. Let $g'(q)$ denote the allocation rule of the simulation as a function of Alice's original value probability. Notice that for $q \notin [a, b]$ we have $g'(q) = g(q)$. For $q \in [a, b]$, Alice receives the average allocation probability for the interval $[a, b]$, i.e, $\frac{1}{b-a} \int_a^b g(r)dr$. See Figure 3.3(a).

Notice third, this approach is likely to improve the expected surplus of agent i. Suppose $g(q)$ is decreasing on $[a, b]$ then this monotonization approach is just shifting allocation probability mass from low values to higher values.



(a) allocation rules $g(q)$ and $g'(q)$. (b) cumulative allocation rules $G(q)$ and $G'(q)$.

Figure 3.3: The allocation rule $g(q)$ (dashed) and $g'(q)$ (solid) constructed by drawing $q' \sim U[a, b]$ and running $g(q')$ when $q \in [a, b]$. Corresponding cumulative allocation rules $G(q)$ (dashed) and $G'(q)$ (solid).

Interval Selection and Monotonicity

The allocation rule $g'(\cdot)$ constructed in this fashion, while monotone over $[a, b]$, may still fail to be monotone. Though intuitively it should be clear that the resampling process can replace a non-monotone interval of the allocation rule with a constant one, we still need to ensure that the final allocation rule is monotone. Of special note is the potential discontinuity of the allocation rule at the end-points of the interval.

Notice first, that $g'(q)$ is monotone if and only if its integral, $G'(q) = \int_0^q g'(r)dr$, is convex. This fact follows because the derivative of a convex function is monotone. We will refer to $G'(q)$ and $G(q)$ (defined identically for $g(q)$) as *cumulative allocation rules*.

Notice second, the implication for $G'(q)$ of the resampling procedure on $G(q)$. Consider some $q \leq a$ as $g(q) = g'(q)$, clearly $G(q) = G'(q)$. In particular $G(a) = G'(a)$. Now calculate $G(b)$ and $G'(b)$. These are equal to $G(a)$ plus the integral of the respective allocation rules on $[a, b]$. Of course $g'(q)$ is constant on $[a, b]$ and equal, by definition, to $\frac{1}{b-a}\int_a^b g(r)dr$. The integral of a constant function is simply the value of the function times the length of the interval. Therefore $\int_a^b g'(r)dr = \int_a^b g(r)dr$. We conclude that $G(b) = G(b')$. Therefore, for all $q \geq b$, $G(q) = G'(q)$ as $g(q) = g'(q)$ for all such q. Thus, $G(q)$ and $G'(q)$ are identical on $[0, a]$ and $[b, 1]$. Of course $g'(q)$ is a constant function on $[a, b]$ so therefore its integral is a linear function; therefore, it must be the linear function that connects $(a, G(a))$ to $(b, G(b))$ with a straight line. See Figure 3.3(b).

Notice third, our choice of interval can be arbitrary and will always simply replace an interval of $G(q)$ with a straight line. Let $\bar{G}(\cdot)$ be the convex hull of $G(\cdot)$. Let k be the number of contiguous subintervals of $[0, 1]$ for which $\bar{G}(q) \neq G(q)$ and let I_j be the jth interval. Let $\mathcal{I} = (I_1, \ldots, I_k)$. The following resampling procedure implements cumulative allocation rule $\bar{G}(\cdot)$. If $q \in I_j \in \mathcal{I}$ then resample $\bar{q} \sim U[I_j]$ (the uniform distribution on I_j), otherwise set $\bar{q} = q$. This is implemented by running the algorithm on the value corresponding to $\bar{q}$, i.e, $\bar{v} = F^{-1}(\bar{q})$. The resulting allocation rule $\bar{g}(q)$ is $\frac{d\bar{G}(q)}{dq}$ which, by the convexity of $\bar{G}(\cdot)$, is monotone. See Figure 3.4.

Surplus Preservation

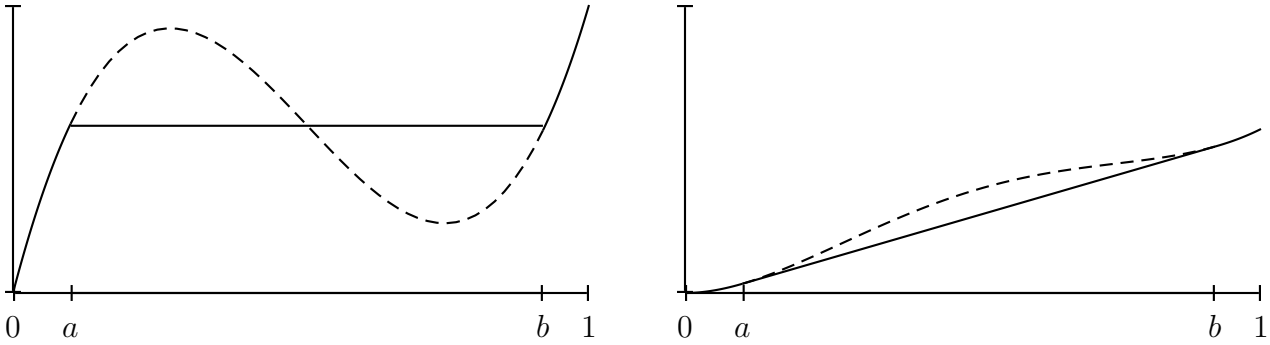
Notice that $\bar{G}(\cdot)$, as the convex hull of $G(\cdot)$, satisfies $\bar{G}(q) \leq G(q)$. Furthermore $\bar{G}(1) = G(1)$. Therefore $\bar{G}(1) - \bar{G}(q) \geq G(1) - G(q)$. Intuitively these final two quantities are the conditional probability of allocating to Alice given that Alice's value probability is at least q under allocation rules $\bar{g}(\cdot)$ and $g(\cdot)$. As this holds for all q, the probability of allocating to higher values in $\bar{g}(\cdot)$ is at least that of $g(\cdot)$. We leave the formal proof as an exercise.

Reduction

The general reduction from BIC mechanism design to algorithm design is the following.

Mechanism 3.3 *Construct the BIC mechanism $\bar{\mathcal{A}}$ from $\mathcal{A}$ as follows.*

1. *For each agent i, identify intervals of non-monotonicity $\mathcal{I}_i$ by taking the convex hull of the cumulative allocation rule (in probability space).*



(a) allocation rule $\bar{g}(q)$.

(b) cumulative allocation rule $\bar{G}(q)$.

Figure 3.4: The allocation rule $\bar{g}(q)$ (solid) and cumulative allocation rule $\bar{G}(q)$ (solid) constructed by taking the convex hull of $G(q)$ (dashed). Interval $I = [a, b]$ is defined as $\{q : \bar{G}(q) \neq G(q)\}$.

2. *For each agent i, if $v_i \in I \in \mathcal{I}_i$ resample $\bar{v}_i \sim F_i[I]$ otherwise set $\bar{v}_i \leftarrow v_i$. (Here $F_i[I]$ denotes the conditional distribution of $v_i \in I$ for F_i.)*

3. $\bar{\mathbf{x}} \leftarrow \mathcal{A}(\bar{\mathbf{v}})$.

4. $\bar{\mathbf{p}} \leftarrow$ *payments from payment identity for* $\bar{\mathbf{x}}(\cdot)$.

This mechanism satisfies our requirements of monotonicity, surplus preservation, and locality. These lemmas follow directly from the construction and we will not provide further proof. Theorem 3.22 directly follows.

Lemma 3.23 *The construction of $\bar{\mathcal{A}}$ from $\mathcal{A}$ is monotone.*

Lemma 3.24 *The construction of $\bar{\mathcal{A}}$ from $\mathcal{A}$ is localized.*

Lemma 3.25 *The construction of $\bar{\mathcal{A}}$ from $\mathcal{A}$ is surplus preserving.*

3.5.2 Blackbox Computation

It should be immediately clear that the reduction given in the preceding section relies on incredibly strong assumptions about our ability to obtain closed form expressions for the allocation rule of the algorithm and perform calculus on these expressions. In fact theoretical analysis of the statistical properties of algorithms on random instances is exceptionally difficult and this is one of the reasons that theoretical analysis of algorithms is almost entirely done in the worst case. Therefore, it is unlikely these assumptions hold in practice.

Suppose instead we cannot do such a theoretical analysis but we can make blackbox queries to the algorithm and we can sample from the distribution. While we omit all the details from this text, it is possible get accurate enough estimates of the allocation rule that the aforementioned resampling procedure can be approximated arbitrarily precisely. Because

this approach is statistical, it may fail to result in absolute monotonicity. However, we can take the convex combination of the resulting allocation rule with a blatantly monotone one to fix these potentially small non-monotonicities.

Naturally, such an approach may lose surplus over the original algorithm because if the small errors it makes. Nonetheless, we can make this loss arbitrarily small. For convenience in expressing the theorem the valuation distribution is normalized over $[0, h]$. The following theorem results.

Theorem 3.26 *For any n-agent single-parameter agent setting, any product distribution $\mathbf{F}$ over $[0, h]^n$, any algorithm $\mathcal{A}$, and any ϵ, there is a BIC mechanism $\bar{\mathcal{A}}_\epsilon$ satisfying $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}\left[\bar{\mathcal{A}}_\epsilon(\mathbf{v})\right] \geq \mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}(\mathbf{v})] - \epsilon$. Furthermore if $\mathcal{A}$ is polynomial time in n, then $\bar{\mathcal{A}}_\epsilon$ polynomial time in n, $1/\epsilon$, and $\log h$.*

While this construction seems to be a great success, it is important to note where it fails. Bayesian incentive compatibility is a weaker notion of incentive compatibility than ex post incentive compatibility. Notable, our construction only gives a monotone allocation rule in expectation when other agents' values are drawn from the distribution. It is not, therefore, a dominant strategy for the agents to bid truthfully. So while we have verified that BIC mechanism design is computationally equivalent to Bayesian algorithm design. Are these both computationally equivalent to ex post IC mechanism design? We leave with two open questions.

Open Question 1 Is there a blackbox reduction for converting any algorithm into a ex post IC mechanism while preserving its performance, e.g., in a Bayesian sense.

Open Question 2 Is existence of an algorithm that meet a particular performance criterion (e.g., Bayesian or worst-case approximation) enough to guarantee that there is an ex post IC mechanism that meets the same criterion.

Notice the subtle difference in the two questions. The first is stronger and more likely to be answered in the negative.

3.5.3 Payment Computation

Recall that the payment identity requires that a monotone allocation rule $x_i(v_i)$ be accompanied by a payment rule $p_i(v_i) = v_i x_i(v_i) - \int_0^{v_i} x_i(z)dz$. At first glance, this appears to require having access to the functional form of the allocation rule. Again, such a requirement is unlikely to be satisfied. This problem, however, is much easier than the monotonization discussed in the previous section because the payment rule only must satisfy the payment identity in expectation. We show how to do this with only two backbox calls to the algorithm.

We compute a random variable P_i with expectation $\mathbf{E}[P_i] = p_i(v_i)$. We will do this for the two parts of the payment identity separately.

Algorithm 3.2 *The* blackbox payment algorithm *for $\mathcal{A}$ computes payment P_i for agent i with value v_i as follows:*

1. $X_i \leftarrow \begin{cases} 1 & \textit{if } i \textit{ wins in } x_i(v_i) \\ 0 & \textit{otherwise.} \end{cases}$

   I.e., X_i is an indicator variable *for whether i wins or not when running $\mathcal{A}(\mathbf{v})$.*

2. $Z_i \sim U[0, v_i]$ *(drawn at random).*

3. $Y_i \leftarrow \begin{cases} 1 & \textit{if } i \textit{ wins in } x_i(Z_i) \\ 0 & \textit{otherwise.} \end{cases}$

   I.e., Y_i is an indicator variable for whether i would win or not when simulating the algorithm on $\mathcal{A}(\mathbf{v}_{-i}, Z_i)$.

4. $P_i \leftarrow v_i(X_i - Y_i)$.

We first note that this payment rule is *individually rational* in the following strong sense. For any realization $\mathbf{v}$ of agent values and any randomization in the algorithm and payment computation, the utilities of all agents are non-negative. This is clear because the utility of an agent is their value minus their payment, i.e., $v_i X_i - P_i = v_i Y_i$. Since $Y_i \in \{0, 1\}$, this utility is always non-negative. Oddly, this payment rule may result in a losing agent being paid, i.e., there may be *positive transfers*. This is because the random variables X_i and Y_i are independent. We may instantiate $X_i = 0$ and $Y_i = 1$. Agent i then loses and has a payment of $-v_i$, i.e., i is paid v_i.

Lemma 3.27 *The blackbox payment computation algorithm satisfies $\mathbf{E}[P_i] = p_i(v_i)$.*

Proof: This follows from linearity of expectation and the definition of expectation in the following routine calculation. First calculate $\mathbf{E}[Y_i]$ notating the probability density function for Z_i is $f_{Z_i}(z) = 1/v_i$ for $Z_i \sim U[0, v_i]$.

$$\begin{aligned} \mathbf{E}[Y_i] &= \int_0^{v_i} x_i(z) f_{Z_i}(z) dz \\ &= \tfrac{1}{v_i} \int_0^{v_i} x_i(z) dz. \end{aligned}$$

Now we calculate $\mathbf{E}[P_i]$ as,

$$\begin{aligned} \mathbf{E}[P_i] &= \mathbf{E}[v_i X_i] - \mathbf{E}[v_i Y_i] \\ &= v_i x_i(v_i) - v_i \mathbf{E}[Y_i] \\ &= v_i x_i(v_i) - \int_0^{v_i} x_i(z) dz. \end{aligned}$$

■

There is a slightly more complicated construction that has no positive transfers (regardless of randomization) but we leave its construction as an exercise. As a hint, suppose that

we repeatedly simulated the algorithm $\mathcal{A}$ on $\mathbf{v}$ until it allocated to i for a second time. Let T_i be a random variable for how many times we had to repeat the algorithm. T_i is a geometric random variable with failure probability $x_i(v_i)$. Therefore $\mathbf{E}[T_i] = \frac{1}{x_i(v_i)}$. Now consider $P_i = v_i X_i(1 - T_i Y_i)$.

3.6 Computational Overhead of Payments

The focus of this chapter has been on ascertaining the extent to which mechanism design is, computationally, more difficult than algorithm design. Positive results, such as our reduction from BIC mechanism design to algorithm design enable designers a generic approach for the computer implementation of a mechanism.

We conclude this chapter with a esoteric-seeming question that has important practical consequences. Notice that when turning a monotone algorithm, either by VCG payments $p_i = \text{OPT}(\mathbf{v}_{-i}) - \text{OPT}_{-i}(\mathbf{v})$, or by our blackbox payment calculation of random variable P_i. The number of calls to the algorithm is $n + 1$, one call to compute the outcome and an additional call for each agent to compute that agent's payment. The question is then, from a computational point of view, whether n times more computation must be performed to run a mechanism, than it takes to just compute its outcome.

A more practically relevant viewpoint on this question is whether repeatability of the algorithm is necessary. We know from our BIC characterization that any mechanism must be monotone. However, approaches described thus far for calculating payment have required that the algorithm be repeatable by simulation as well.

This repeatability requirement poses severe challenges in some practical contexts. Consider an online advertising problem where there is a set of advertisers (agents) who each have an ad to be shown on an Internet web page. An advertiser is "served" if their ad is clicked on. Each advertiser i has a private value v_i for each click they receive. Each ad i has a *click-through rate* c_i, i.e., a probability that the ad will be clicked if it is shown. If the mechanism designer knew these click-through rates in advance, the surplus maximizing rule would be to show the advertiser with the highest $v_i c_i$. Unfortunately, these click-through rates are often unknown to the mechanism designer and the advertisers. The mechanism can attempt to use any of a number of learning algorithm to learn advertiser click-through rates as it shows ads. Most of these learning algorithms are in fact monotone, meaning the higher i's value v_i the more clicks i will receive in expectation. Unfortunately it is difficult to turn these learning algorithms into incentive compatible mechanisms because there is no way to go back in time and see what would have happened if a different advertiser had been shown. What is needed here is a way to design a mechanism from a monotone algorithm with only a single call to the algorithm.

3.6.1 Communication Complexity Lower Bound

For single-dimensional agent problems with special structure (i.e., on the cost function of the designer, $c(\cdot)$) it is possible to design an algorithm that computes payments at the same time

as it computes the allocation with no significant extra computational effort. For instance, for optimizations based on linear programming duality, a topic we will not cover here, the *dual variables* are often exactly the payments required for the VCG mechanism.

It is likely that such a result does not hold generically. However, the conclusion of our discussion of computational tractability earlier in this chapter was that proving lower bounds on computational requirements is exceptionally difficult. We therefore analyze a related question, namely the *communication complexity* of an allocation rule versus its associated payment rules. To analyze communication complexity we imagine that each of our n agents has their private input and they collectively want to compute some desired function. Each agent may perform an unlimited amount of local computation and can broadcast any information simultaneously to all other agents. The challenge is to come up with a protocol the agents can follow so that at the end of the protocol all agents know the value of the function and the total number of bits broadcast is minimized. In this exercise, agents are assumed to be non-strategic and will follow the chosen protocol precisely.

As an example consider a *public good* with cost C. In this setting we must either serve all agents or none of them. If we serve all of them we incur the cost of C. Our objective is to maximize social surplus so clearly we wish to serve the agents whenever $\sum_i v_i \geq C$. This is a single-parameter agent setting with cost function given by

$$c(\mathbf{x}) = \begin{cases} C & \text{if } \sum_i x_i = n \\ 0 & \text{if } \sum_i x_i = 0 \\ \infty & \text{otherwise.} \end{cases}$$

Notice that it is infeasible to serve one agent and not another. This problem arises naturally. Assume the government is considering whether or not to build a bridge. It costs C to build the bridge. Naturally the government only wants to build the bridge if the total value of the bridge to the people exceeds the cost. Of course, if the bridge is built then everyone can use it.

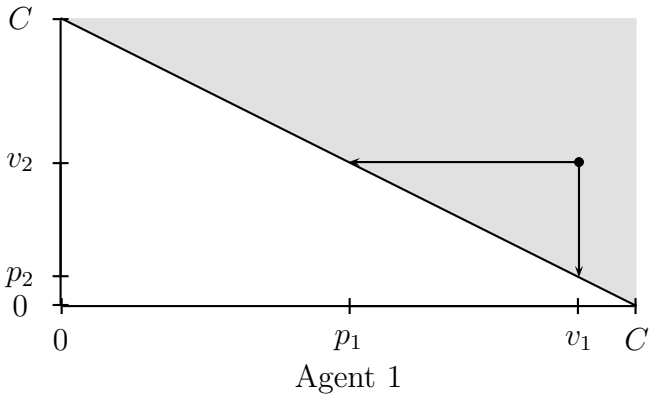


Figure 3.5: The region of allocation (gray) for the public project, as a function of agent 1's value (x-axis) and agent 2's value (y-axis), and the payments $\mathbf{p} = (p_1, p_2)$ for valuation profile $\mathbf{v} = (v_1, v_2)$.

Consider the special case of the above problem with two agents, each with an integral value ranging between 0 and C. Let $k = \log C$ be the number of bits necessary to represent each agent's value in binary. To compute the surplus maximizing outcome, i.e., whether to allocate to both agents or neither of them, the following protocol can be employed:

1. Agent 1 broadcasts their k bit value v_1.

2. Agent 2 determines whether $v_1 + v_2 \geq C$ and broadcasts 1 if it is and 0 otherwise.

Notice that the total number of bits broadcast by this protocol is $k + 1$ and both parties learned the desired outcome.

Now suppose we also wish to communication protocol to compute the incentive compatible payments for this monotone allocation rule wherein both agents must learn p_1 and p_2. How many bits of communication are required? Notice that in the case that we serve the agents, $p_1 = C - v_2$ and $p_2 = C - v_1$. Importantly, there is a unique $\mathbf{p}$ for each unique $\mathbf{v}$ that is served. There are $C^2/2$ such payment vectors in total. The broadcast bits must uniquely determine which of these $\mathbf{p}$ is the correct outcome. Given such a large number of payment vectors the most succinct representation would be to number them and write the index of the desired payment vector in binary. This takes a number of bits that is logarithmic in the number of payment vectors possible. In our case this is $\log(C^2/2)$. Of course, $C = 2^k$ so the number of bits is $2k-1$. Agent 1 has k bits, but the other $k-1$ bits should be communicated from Agent 2, and vice versa. Therefore a total of $2k - 2$ bits must be communicated for both agents to learn $\mathbf{p}$.

We conclude that in this two-agent setting about twice as many bits are required to compute payments than to compute the outcome. We summarize this in the following lemma.

Lemma 3.28 *There exists a two-agent single-parameter agent setting where the communication complexity of computing payments is twice that of computing the allocation.*

The above two agent example can be generalized to an n-agent setting where the communication complexity of payments is n times more than that for computing the allocation. The single-parameter agent problem that exhibits such a separation is contrived, i.e., there is no natural economic interpretation for it. The real challenge in this construction is construction of a setting where the allocation can be computed with very low communication. We omit the proof and construction of the setting.

Theorem 3.29 *There exists an n-agent single-parameter agent setting there the communication complexity of computing payments is n times that of computing the allocation.*

The conclusion of the above discussion on the communication complexity of computing payments is that there are settings where payments are a linear factor harder to compute. If we wish to take any algorithm and construct payments we can expect that the construction, in worst-case, will require a linear factor more work. For example, it may by invoking the algorithm $n + 1$ times as is done in the constructions previously discussed. Notice, however,

that this lower bound leaves open the possibility that subtle changes to the allocation rule might permit payments to be calculated without such computational overhead.

3.6.2 Fast Payments via Approximation

The main result of this section is to describe a procedure for taking any monotone algorithm and altering it in a way that does not significantly decrease its surplus and enables payments to be computed implicitly from one single, original invocation of the algorithm.

The presentation in this text will focus on the Bayesian setting as the approach and result are the most relevant for application to any algorithm. (There is a similar approach, that we do not discuss, for achieving the same flavor of result for worst-case approximation algorithms.) Therefore, assume as we did in preceding sections that agents' values are distributed according to a product distribution that is continuous on its support, i.e., each agent's density function is strictly positive.

Two main ideas underlie this approach. Recall that the blackbox payment computation drew $Z_i \sim U[0, v_i]$ and defined Y_i as an indicator for $x_i(Z_i)$. The expected value of this Y_i is then related to $\int_0^{v_i} x_i(z)dz$. The first idea is that there is nothing special about the uniform distribution; we can do the exact same estimation with any continuous distribution, e.g., with F_i. The second idea is that if with some small probability ϵ we redraw $V_i' \sim F_i$ and input that into the algorithm instead of v_i then this changes the allocation rule in an easy to describe way. It multiplies it by $(1-\epsilon)$ and adds a constant $\mathbf{E}[x_i(V_i')]$. For the purpose of computing payments, which is a function of the integral of the allocation rule, adding a constant to it has no effect.

Notice in the definition of the implicit payment mechanism, below, that the algorithm $\mathcal{A}$ is only called once.

Mechanism 3.4 *For algorithm $\mathcal{A}$, distribution $\mathbf{F}$, and parameter $\epsilon > 0$, the* implicit payment mechanism $\mathcal{A}'_\epsilon$ *is*

1. *for all i:*
   (a) *Draw $Z \sim F_i$*
   (b) $V_i' \leftarrow \begin{cases} v_i & \text{with probability } 1-\epsilon \\ Z & \text{otherwise.} \end{cases}$
2. $\mathbf{X}' \leftarrow \mathcal{A}(\mathbf{V}')$.
3. *for all i:* $P_i' \leftarrow \begin{cases} v_i X_i' & \text{if } V_i' = v_i \\ -\frac{1-\epsilon}{\epsilon} \cdot \frac{X_i'}{f_i(V_i')} & \text{if } V_i' < v_i \\ 0 & \text{otherwise.} \end{cases}$

As you can see, the implicit payment mechanism sometimes calls $\mathcal{A}$ on v_i into and sometimes it draws a completely new value V_i' and inputs that to $\mathcal{A}$ instead. The actual payments

are a bit strange. If it inputs i's original value and serves i then i is charged their value. If it inputs a value less than i's original value and serves i then i is given a very large rebate, as ϵ and $f(V_i')$ should be considered very small. If we do not serve i then i pays nothing. Furthermore, if the implicit payment mechanism inputs a value greater than i's value and allocate to i then i also pays nothing.

Such a strange payment computation warrants a discussion as to the point of this exercise in the first place; it seems like such a crazy payment rule would completely impractical. Recall, though, that our search for a BIC mechanism was one of existence. We wanted to know if there existed a mechanism in our model (in this case, with a single call to the algorithm) with good BNE. We have verified that. In fact, BIC mechanisms are often not practical. The omitted step is in finding a practical mechanism with good BNE, and this means undoing the revelation principle to ask what other more natural mechanism might have the same BNE but also satisfy whatever additional practicality constraints we may require. This final step of mechanism design is often overlooked, and it is because designing non-BIC mechanisms with good properties is difficult.

With such practical considerations aside, we now prove that monotonicity of $\mathcal{A}$ (in a Bayesian sense) is enough to ensure that $\mathcal{A}'_\epsilon$ is BIC. Furthermore, we show that the expected surplus of $\mathcal{A}'_\epsilon$ is close to that of $\mathcal{A}$.

Lemma 3.30 *For $\mathbf{v} \sim \mathbf{F}$, if the allocation rule $x_i(\cdot)$ for $\mathcal{A}$ is monotone, then the allocation rule $x'_i(\cdot)$ for $\mathcal{A}'_\epsilon$ is monotone.*

Proof: From each agent i's perspective we have not changed the distribution of other agents' values. Furthermore, $\mathbf{x}'(\mathbf{v}) = (1-\epsilon)\cdot\mathbf{x}(\mathbf{v}) + \epsilon\cdot\mathbf{E}_{\mathbf{V}'}[\mathbf{x}(\mathbf{V}')]$, i.e., we have scaled the algorithms allocation rule by $(1-\epsilon)$ and added a constant. Therefore, if $\mathcal{A}$ was monotone before then $\mathcal{A}'_\epsilon$ is monotone now. (Furthermore, if $\mathcal{A}$ was monotone in an ex post sense before then the implicit payment mechanism is monotone in an ex post sense now.) ■

Lemma 3.31 *For agent i with value v_i, $\mathbf{E}[P'_i] = p'_i(v_i)$ satisfying the payment identity.*

Proof: Since our allocation rule for agent i is $x'_i(v_i) = (1-\epsilon)\cdot x_i(v_i) + \epsilon\cdot\mathbf{E}[x_i(V'_i)]$ where the final term is a constant. We must show that $\mathbf{E}[P_i] = (1-\epsilon)p_i(v_i) = p'_i(v_i)$.

Define indicator variable A for the event that $V'_i = v_i$ and B for the event $V'_i < v_i$. With these definitions, $P_i = v_i X'_i A - \frac{1-\epsilon}{\epsilon}\cdot\frac{X'_i B}{f(V'_i)}$. The expectation of the first term is easy to analize:

$$\begin{aligned}\mathbf{E}[v_i X'_i A] &= \mathbf{Pr}[A=1]\cdot\mathbf{E}[v_i X'_i \mid A=1]\\ &= (1-\epsilon)v_i x_i(v_i).\end{aligned}$$

This is exactly as desired. Now we turn to the second term. For convenience we ignore the constants until the end.

$$\begin{aligned}\mathbf{E}[X'_i B/f_i(V'_i)] &= \mathbf{Pr}[B=1]\cdot\mathbf{E}[X'_i B/f_i(V'_i) \mid B=1]\\ &\quad + \mathbf{Pr}[B=0]\cdot\mathbf{E}[X'_i B/f_i(V'_i) \mid B=0]\\ &= \epsilon F_i(v_i)\cdot\mathbf{E}[X'_i/f_i(V'_i) \mid B=1]\,.\end{aligned}$$

Notice $B = 1$ implies that we drawn $V_i' \sim F_i[0, v_i]$ which has density function $f_i(z)/F_i(v_i)$. Thus, we continue our calculation as,

$$\begin{aligned} &= \epsilon F_i(v_i) \cdot \int_0^{v_i} \frac{x_i(z)}{f_i(z)} \cdot \frac{f_i(z)}{F_i(v_i)} dz \\ &= \epsilon \int_0^{v_i} x_i(z) dz. \end{aligned}$$

Combining this with the constant multiplier $\frac{1-\epsilon}{\epsilon}$ and the calculation of the first term, we conclude that $\mathbf{E}[P_i] = (1-\epsilon)p_i(v_i) = p_i'(v_i)$ as desired. ■

From the two lemmas above we conclude that $\mathcal{A}_\epsilon'$ is BIC. To discuss the performance of $\mathcal{A}_\epsilon'$ (i.e., surplus) we consider two of our general single-dimensional agent settings separately. We consider the general costs setting because it is the most general and the general feasibility setting because it is easier and thus permits a nicer performance bound.

Lemma 3.32 *For any distribution* $\mathbf{F}$ *and any general feasibility setting,* $\mathbf{E}[\mathcal{A}_\epsilon'(\mathbf{v})] \geq (1-\epsilon) \cdot \mathbf{E}[\mathcal{A}(\mathbf{v})]$.

Proof: This follows from considering each agent separately and using linearity of expectation. Our expected surplus from i is

$$\begin{aligned} \mathbf{E}_{v_i}[v_i x_i'(v_i)] &= (1-\epsilon) \cdot \mathbf{E}_{v_i}[v_i x_i(v_i)] + \epsilon \cdot \mathbf{E}_{v_i}[v_i] \cdot \mathbf{E}_{v_i}[x_i(v_i)] \\ &\geq (1-\epsilon) \cdot \mathbf{E}_{v_i}[v_i x_i(v_i)] \end{aligned}$$

The final step follows because the second term on the right-hand side is always non-negative and thus can be dropped. ■

Lemma 3.33 *For any distribution* $\mathbf{F}$ *and any* n*-agent general costs setting,* $\mathbf{E}[\mathcal{A}_\epsilon'(\mathbf{v})] \geq \mathbf{E}[\mathcal{A}(\mathbf{v})] - \epsilon h n$ *where* h *is an upper bound on any agent's value.*

Proof: This proof follows by noting that the algorithm always runs on an input that is random from the given distribution $\mathbf{F}$. Therefore, the expected costs are the same, i.e., $\mathbf{E}_{\mathbf{v}}[c(\mathbf{x}'(\mathbf{v}))] = \mathbf{E}_{\mathbf{v}}[c(\mathbf{x}(\mathbf{v}))]$. However, the expected total value to the agents is decreased relative to the original algorithm. Our expected surplus from i is

$$\begin{aligned} \mathbf{E}_{v_i}[v_i x_i'(v_i)] &\geq (1-\epsilon) \cdot \mathbf{E}_{v_i}[v_i x_i(v_i)] \\ &\geq (1-\epsilon) \cdot \mathbf{E}_{v_i}[v_i x_i(v_i)] + \epsilon \cdot \mathbf{E}_{v_i}[v_i x_i(v_i)] - \epsilon h \\ &= \mathbf{E}_{v_i}[v_i x_i(v_i)] - \epsilon h \end{aligned}$$

The first step is the same as before, we can then add the two terms on the right-hand side because the negative one is higher magnitude than the positive one. The final result follows from summing over all agents and the expect costs. ■

The following theorems follow directly from the lemmas above.

Theorem 3.34 *For any single-dimensional agent setting with general costs, any product distribution* $\mathbf{F}$ *over* $[0, h]^n$*, any algorithm* $\mathcal{A}$*, and any* $\epsilon > 0$*, the implicit payment mechanism* $\mathcal{A}'_\epsilon$ *is BIC and satisfies* $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}'_\epsilon(\mathbf{v})] \geq \mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}(\mathbf{v})] - \epsilon hn$*.*

Theorem 3.35 *For any single-dimensional agent setting with general feasibility constraints, any product distribution* $\mathbf{F}$*, any algorithm* $\mathcal{A}$*, and any* $\epsilon > 0$*, the implicit payment mechanism* $\mathcal{A}'_\epsilon$ *is BIC and satisfies* $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}'_\epsilon(\mathbf{v})] \geq (1-\epsilon)\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}(\mathbf{v})]$*.*

We conclude this section by answering our opening question. If we are willing to sacrifice an arbitrarily small amount of the surplus, we do not require algorithms to be repeatable to turn them into mechanisms. Our only requirement is monotonicity.

3.7 Notes

Richard Karp pioneered the use of $\mathcal{NP}$-completeness reductions to show that a number of relevant combinatorial optimization problems, including set packing, are $\mathcal{NP}$-complete [12]. Lehmann, O'Callaghan, and Shoham [13] and Nisan and Ronen [15] introduced the concept of approximation mechanisms. The single-minded combinatorial auction problem is an iconic single-dimensional agent mechanism design problem. Multicast auctions were introduced by Feigenbaum et al. [9].

There are several reductions from approximation mechanism design to approximation algorithm design (in single-dimensional settings). Briest, Krista, and Vocking [7] show that any polynomial time approximation scheme (PTAS) can be converted into an (ex post) incentive compatible mechanisms. The reduction from BIC mechanism design to Bayesian algorithm design that was described in this text was given by Hartline and Lucier [10]. For the special case of single-minded (and a generalization that is not strictly single-dimensional) combinatorial auctions Babaioff, Lavi, and Pavlov [5] give a general reduction that obtains a $O(c \log h)$ approximation mechanism from any c-approximation algorithm where h is an upper-bound on the value of any agent.

The unbiased estimator payment computation was given by Archer et al. [1]. The communication complexity lower bound for computing payments is given by Babaioff, Blumrosen, Naor, and Shapira [3]. Babaioff, Kleinberg, and Slivkins [4] developed the approximation technique that permits payments to be computed with one call to the allocation rule (i.e., the algorithm). This result enables good mechanisms in settings where the algorithm cannot be repeated. For instance, in online auction settings such as the one independently considered by Babaioff, Sharma, and Slivkins [6] and Devanur and Kakade [8].