Chapter 8

Computational Tractability

In many relevant environments optimal mechanisms are computationally intractable. A mechanism that is computationally intractable, i.e., where no computer could calculate the outcome of the mechanism in a reasonable amount of time, seems to violates even the loosest interpretation of our general desideratum of simplicity.

We will try to address this computational intractability by considering approximation. In particular we will look for an approximation mechanism, one that is guaranteed to achieve a performance that is close to the optimal, intractable, mechanism's performance. A first issue that arises in this approach is that approximation algorithms are not generally compatible with mechanism design. The one approach we have discussed thus far, following from generalization of the second-price auction, fails to generically convert approximation algorithms into dominant-strategy incentive-compatible approximation mechanisms.

Dominant-strategy incentive-compatible and prior-free mechanisms may be too demanding for this setting. In fact, without making an assumptions on the environment the approximation factor of worst case algorithms can be provably too far from optimal to be practically relevant. We therefore turn to Bayesian mechanism design and approximation. Here we give a reduction from BIC mechanism design to algorithm design. This reduction is a simple procedure that converts any algorithm into a Bayesian incentive-compatible mechanism without compromising its expected social surplus. If the algorithm is is tractable then the resulting mechanism is too. We conclude that under the BIC implementation concept incentive constraints and tractability constraints can be completely disentangled and any good algorithm or heuristic can be converted into a mechanism.

8.1 Tractability

Our philosophy for mechanism design is that a mechanism that is not computationally tractable is not a valid solution to a mechanism design problem. To make this criterion formal we review the most fundamental concepts from computational complexity. Readers are encouraged to explore these topics in greater detail outside this text.

Definition 8.1. *$\mathcal{P}$ is the class of problems that can be* solved *in polynomial time (in the* *"size of the input").*

Recall, from Chapter 4 the greedy algorithm for optimizing independent sets in a matroid. This algorithm sorts the agents by value, and then greedily, in this order, tries to add the agents to an independent set. Sorting takes $O(n \log n)$ and checking independence is usually fairly easy. Therefore, this algorithm runs in polynomial time.

Definition 8.2. *$\mathcal{NP}$ is the class of problem that can be* verified *in polynomial time.*

$\mathcal{NP}$ stands for *non-deterministic polynomial time* in that problems in this class can be solved by "guessing" the solution and then verifying that indeed the solution is correct. Of course, non-deterministic computers that can guess the correct solution do not exist. A real computer could of course simulate this process by iterating over all possible solutions and checking to see if the solution is valid. Unfortunately, nobody whether there is an algorithm that improves significantly on exhaustive search. Exhaustive search, for most problems, requires exponential runtime and is therefore considered intractable.

Consider the problem of verifying whether a given solution to the single-minded combinatorial auction problem has surplus at least V. If we were given an outcome $\mathbf{x}$ that for which $\text{Surplus}(\mathbf{v}, \mathbf{x}) \geq V$ it would be quite simple to verify. First, we would verify whether it is feasible by checking all i and i' with $x_i = x_{i'} = 1$ (i.e., all pairs of served agents) that $S_i \cap S_{i'} = \emptyset$ (i.e., their bundles do not overlap). Second, we would calculate the total welfare $\sum_i v_i x_i$ to ensure that it is at least V. The total runtime of such a verification procedure is $O(n^2)$.

While the field of computer science has failed to determine whether or not $\mathcal{NP}$ problems can be solved in polynomial time or not, it has managed to come to terms with this failure. The following approach allows one to leverage this collective failure to argue that a given problem X is unlikely to be polynomial-time solvable by a computer.

Definition 8.3. *A problem Y* reduces *(in polynomial time) to a problem X if we can solve any instance y of Y with a polynomial number (in the size of y) of basic computational steps and queries to a blackbox that solves instances of X.*

Definition 8.4. *A problem X is $\mathcal{NP}$-*hard *if all problems $Y \in \mathcal{NP}$ reduce to it.*

Definition 8.5. *A problem X is $\mathcal{NP}$-*complete *if $X \in \mathcal{NP}$ and X is $\mathcal{NP}$-hard.*

The point of these definitions is this. Many computer scientists have spent many years trying to solve $\mathcal{NP}$-complete problems and failed. When one shows a new problem X is $\mathcal{NP}$-hard, one is showing that if this problem can be solved, then so can all $\mathcal{NP}$ problems, even the infamously difficult ones. While showing that a problem cannot be solved in polynomial time is quite difficult, showing that it is $\mathcal{NP}$-hard is usually quite easy (if it is true). Therefore it is quite possible to show, for some new problem X, that under the assumption that $\mathcal{NP}$-hard problems cannot be solved in polynomial time (i.e., $\mathcal{NP} \neq \mathcal{P}$), that X cannot be solved in polynomial time.

We will make the standard assumption that $\mathcal{NP} \neq \mathcal{P}$ which implies that $\mathcal{NP}$-hard problems are computationally intractable.

8.2 Single-minded Combinatorial Auctions

Consider the example environment of single-minded combinatorial auctions. This environment is important as it is a special case of more general auction settings such as the FCC spectrum auctions (for selling radio-frequency broadcast rights to to cellular phone companies) and sponsored search auctions (for selling advertisements to be show along side on search-results page of Internet search engines). In single-minded combinatorial auctions each agent i has a value v_i for receiving a bundle S_i of m distinct items. Of course each item can be allocated to at most one agent so the intersection of the desired bundles of all pairs of served agents must not intersect.

The optimization problem of single-minded combinatorial auctions, also known as *weighted set packing*, is intractable. We state but do not prove this result here.

Theorem 8.6. *The single-minded combinatorial auction problem is $\mathcal{NP}$-complete.*

8.2.1 Approximation Algorithms

When optimally solving a problem is $\mathcal{NP}$-hard the standard approach from the field of algorithms is to obtain a polynomial time approximation algorithm, i.e., an algorithm that guarantees in worst-case to output a solution that is within a prespecified factor of the optimal solution.

As a first step at finding an approximation algorithm it is often helpful to look at simple-minded approaches that fail to give good approximations. The simplest algorithmic design paradigm is that of *static greedy algorithms*. Static greedy algorithms for general feasibility settings follow the following basic framework.

Algorithm 8.1. *A* static greedy algorithm *is*

1. *Sort the agents by some prespecified criterion.*
2. $\mathbf{x} \leftarrow \mathbf{0}$ *(the null assignment).*
3. *For each agent i (in this sorted order),*
   if $(\mathbf{x}_{-i}, 1)$ is feasible, $x_i \leftarrow 1$.
   (I.e., serve i if i can be served along side previously served agents.)
4. *Output* $\mathbf{x}$.

The first failed approach to consider is *greedy by value*, i.e., the prespecified sorting criterion in the static greedy template above is by agent values v_i. This algorithm is bad because it is an $\Omega(m)$-approximation on the following $n = m + 1$ agent input. Agents i, for $0 \leq i \leq m$, have $S_i = \{i\}$ and $v_i = 1$; agent $m + 1$ has $v_{m+1} = 1 + \epsilon$ and demands the grand bundle $S_{m+1} = \{1, \ldots, m\}$ (for some small $\epsilon > 0$). See Figure 8.1(a) with $A = 1$ and $B = 1 + \epsilon$. Greedy-by-value orders agent $m + 1$ first, this agent is feasible and therefore served. All remaining agents are infeasible after agent $m + 1$ is served. Therefore, the

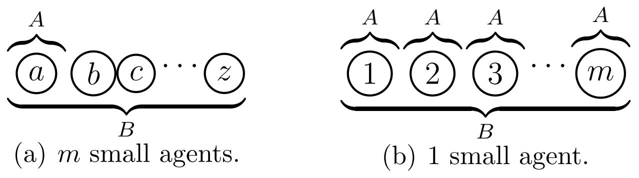


Figure 8.1: Challenge cases for greedy orderings as a function of value and bundle size.

algorithm serves only this one agent and has surplus $1 + \epsilon$. Of course OPT serves the m small agents for a total surplus of m. The approximation factor of greedy-by-value is the ratio of these two performances, i.e., $\Omega(m)$.

Obviously what went wrong in greedy-by-value is that we gave preference to an agent with large demand who then blocked a large number of mutually-compatible small-demand agents. We can compensate for this by instead sorting by value-per-item, i.e., $v_i/|S_i|$. *Greedy by value-per-item* also fails on the following $n = 2$ agent input. Agents 1 has $S_1 = \{1\}$ and $v_1 = 1 + \epsilon$ and agent 2 has $v_2 = m$ demands the grand bundle $S_2 = \{1, \ldots, m\}$. See Figure 8.1(b) with $A = 1+\epsilon$ and $B = m$. Greedy-by-value-per-item orders agent 1 first, this agent is feasible and therefore served. Agent 2 is infeasible after agent 1 is served. Therefore, the algorithm serves only agent 1 and has surplus $1 + \epsilon$. Of course OPT serves agent 2 and has surplus of m. The approximation factor of greedy-by-value-per-item is the ratio of these two performances, i.e., $\Omega(m)$.

The flaw with this second algorithm is that it makes the opposite mistake of the first algorithm; it undervalues large-demand agents. While we correctly realized that we need to trade off value for size, we have only considered extremal examples of this trade-off. To get a better idea for this trade-off, consider the cases of a single large-demand agent and either m small-demand agents or 1 small-demand agent. We will leave the values of the two kinds of agents as variables A for the small-demand agent(s) and B for the large-demand agent. Assume, as in our previous examples, that $mA > B > A$. These settings are depicted in Figure 8.1.

Notice that any greedy algorithm that orders by some function of value and size will either prefer A-valued or B-valued agents in both cases. The A-preferred algorithm has surplus Am in the m-small-agent case and surplus A in the 1-small-agent case. The B-preferred algorithm has surplus B in both cases. OPT, on the other hand, has surplus mA in the m-small-agent case and surplus B in the 1-small-agent case. Therefore, the worst-case ratio for A-preferred is B/A (achieved in the 1-small-agent case), and the worst-case ratio for B-preferred is mA/B (achieved in the m-small-agent case). These performances and worst-case ratios are summarized in Table 8.1.

If we are to use the static greedy algorithm design paradigm we need to minimize the worst-case ratio. The approach suggested by the analysis of the above cases would be trade off A versus B to equalize the worst-case ratios, i.e., when $B/A = mA/B$. Here m was a stand-in for the size of the large-demand agent. Therefore, the greedy algorithm that this suggests is to order the agents by $v_i/\sqrt{|S_i|}$. This algorithm was first proposed for use in

	m small agents	1 small agent	worst-case ratio
OPT	mA	B	(n.a.)
A-preferred	mA	A	B/A
B-preferred	B	B	mA/B

Table 8.1: Performances of A- and B-preferred greedy algorithms and their ratios to OPT.

single-minded combinatorial auctions by Daniel Lehmann, Liadan O'Callaghan, and Yoav Shoham and is often referred to as the LOS algorithm.

Algorithm 8.2. *Sort the gents by value-per-square-root-size, i.e.,* $v_i/\sqrt{|S_i|}$*, and serve them greedily while supplies last.*

Theorem 8.7. *The greedy by value-per-square-root-size algorithm is a* $\sqrt{m}$*-approximation algorithm (where* m *is the number of items).*

Proof. Let APX represent the greedy by value-per-square-root-size algorithm and its surplus; let REF represent the optimal algorithm and its surplus. Let I be the set selected by APX and I^* be the set selected by REF. We will proceed with a *charging argument* to show that if $i \in I$ blocks some set of agents $F_i \subset I^*$ then the total value of the blocked agents is not too large relative to the value of agent i.

Consider the agents sorted (as in APX) by $v_i/\sqrt{|S_i|}$. For an agent $i^* \in I^*$ not to be served by APX, it must be that at the time it is considered by LOS, another agent i has already been selected that *blocks* i^*, i.e., the demand sets S_i and S_{i^*} have non-empty intersection. Intuitively we will charge i with the loss from not accepting this i^*. We define F_i as the set of all $i^* \in I^*$ that are charged to i as described above. Of special note, if $i^* \in I$, i.e., it was not yet blocked when considered by APX, we charge it to itself, i.e., $F_{i^*} = \{i^*\}$. Notice that the sets F_i partition the agents I^* of REF.

The theorem follows from the inequalities below. Explanations of each non-trivial step are given afterwards.

$$\text{REF} = \sum_{i^*} v_{i^*} = \sum_{i \in I} \sum_{i^* \in F_i} v_{i^*} \tag{8.1}$$

$$\leq \sum_{i \in I} \frac{v_i}{\sqrt{|S_i|}} \sum_{i^* \in F_i} \sqrt{|S_{i^*}|} \tag{8.2}$$

$$\leq \sum_{i \in I} \frac{v_i}{\sqrt{|S_i|}} \sum_{i^* \in F_i} \sqrt{m/|F_i|} \tag{8.3}$$

$$= \sum_{i \in I} \frac{v_i}{\sqrt{|S_i|}} \sqrt{m|F_i|} \tag{8.4}$$

$$\leq \sum_{i \in I} v_i \sqrt{m} = \sqrt{m} \cdot \text{APX}\,. \tag{8.5}$$

Line (8.1) follows because F_i partition I^*. Line (8.2) follows because $i^* \in F_i$ implies that i precedes i^* in the greedy ordering and therefore $v_i^* \leq v_i\sqrt{|S_{i^*}|}/\sqrt{|S_i|}$. Line (8.3) follows because the demand sets S_{i^*} of $i^* \in F_i$ are disjoint (because they are a subset of I^* which is feasible and therefore disjoint). Thus we can bound $\sum_{i^* \in F_i} |S_{i^*}| \leq m$. The square-root function is concave and the sum of a concave function is maximized when each term is equal, i.e., when $S_{i^*} = m/|F_i|$. Therefore, $\sum_{i^* \in F_i} \sqrt{|S_{i^*}|} \leq \sum_{i^* \in F_i} \sqrt{m/|F_i|} = \sqrt{m|F_i|}$. This last equality gives line (8.4). Finally, line (8.5) follows because $|F_i| \leq |S_i|$ which holds because each $i^* \in F_i$ is disjoint but blocked by i because each contains some demanded item in S_i. Thus, S_i contains at least $|F_i|$ distinct items. $\square$

The as witnessed by the theorem above, the greedy by value-per-square-root-size algorithm gives a non-trivial approximation factor. A $\sqrt{m}$-approximation, though, hardly seems appealing. Unfortunately, it is unlikely that there is a polynomial time algorithm with better worst-case approximation factor, but we do not provide proof in this text.

Theorem 8.8. *Under standard complexity-theoretic assumptions,*[1] *no polynomial time algorithm gives an $o(\sqrt{m})$-approximation to weighted set packing.*

8.2.2 Approximation Mechanisms

Now that we have approximated the single-minded combinatorial auction problem without incentive constraints, we need add these constraints back in and see whether we can derive a $\sqrt{m}$-approximation mechanism.

We first note that we cannot simply use the formula for externalities from the surplus maximization mechanism for payments when we replace the optimal algorithm OPT with some approximation algorithm $\mathcal{A}$. I.e., $\mathbf{x} = \mathcal{A}(\mathbf{v})$ and $p_i = \mathcal{A}(\mathbf{v}_{-i}) - \mathcal{A}_{-i}(\mathbf{v})$ is not incentive compatible. An example demonstrating this with the greedy algorithm can be seen by m agents each i demanding the singleton bundle $S_i = \{i\}$ with value $v_i = 1$ and a final agent $m+1$ demanding the grand bundle $S_{m+1} = \{1, \ldots, m\}$ with value $\sqrt{m}+\epsilon$ (See Figure 8.1(a) with $A = 1$ and $B = \sqrt{m} + \epsilon$). On such an input the greedy algorithm APX accepts only agent $m + 1$. However, when computing the payment with the externality formula $p_{m+1} = \text{APX}(\mathbf{v}_{-(m+1)}) - \text{APX}_{-(m+1)}(\mathbf{v})$ we get $p_{m+1} = m$. This payment is higher than agent $m + 1$'s value and the resulting mechanism is clearly not incentive compatible.

Mirroring our derivation of the monotonicity of the surplus maximization mechanism in Chapter 3 Section 3.2, the BNE characterization requires each agent's allocation rule be monotone, therefore any incentive compatible mechanism must be monotone. Even though, in our derivation of the greedy algorithm no attempt was made to obtain monotonicity, it is satisfied anyway.

Lemma 8.9. *For each agent i and all values of other agents $\mathbf{v}_{-i}$, the i's allocation rule in the greedy by value-per-square-root-size algorithm is monotone in i's value v_i.*

[1] I.e., assuming that $\mathcal{NP}$-complete problems cannot be solved in polynomial time by a randomized algorithm.

Proof. It suffices to show that if i with value v_i is served by the algorithm on $\mathbf{v}$ and i increases her bid to $b_i > v_i$ then they will continue to be served. Notice that the set of available items is non-increasing as each agent is considered. If i increases her bid she will be considered only earlier in the greedy order. Since items S_i were available when i is originally considered, they will certainly be available if i is considered earlier. Therefore, i will still be served with a higher bid. □

The above proof shows that there is a critical value τ_i for i and if $v_i > \tau_i$ then i is served. It is easy to identify this critical value by simulating the algorithm on $\mathbf{v}_{-i}$. Let i' be the earliest agent in the simulation to demand and receive an item from S_i. Notice that if i comes after i' then i will not be served because S_i will no longer be completely available. However, if i comes before i' then i can and will be served by the algorithm. Therefore i's critical value is the τ_i for which $v_i = \tau_i$ would tie agent i' in the ordering. I.e., $\tau_i = v_{i'}\sqrt{|S_i|}/\sqrt{|S_{i'}|}$.

We conclude by formally giving the approximation mechanism induced by the greedy by value-per-square-root-size algorithm and the theorem and corollary that describe its incentives and performance.

Mechanism 8.1. *The greedy by value-per-square-root-size mechanism is:*

1. *Solicit and accept sealed bids* $\mathbf{b}$.
2. $\mathbf{x} = GVPSS(\mathbf{b})$, *and*
3. $\mathbf{p} =$ *critical values for GVPSS on* $\mathbf{b}$,

where GVPSS denotes the greedy by value-per-square-root-size algorithm.

Theorem 8.10. *The greedy by value-per-square-root-size mechanism is dominant strategy incentive compatible.*

Corollary 8.11. *The greedy by value-per-square-root-size mechanism gives a $\sqrt{m}$-approximation to the optimal social surplus in dominant strategy equilibrium.*

At this point it is important to note that again we have gotten lucky in that we attempted to approximate our social surplus objective without incentive constraints and the approximation algorithm we derived just happened to be monotone, which is all that is necessary for the mechanism with that allocation and the appropriate payments to be incentive compatible. If we had been less fortunate and our approximation algorithm not been monotone, it could not have been turned into a mechanism so simply. We conclude this section with three important questions.

Question 8.1. *When are approximation algorithms monotone?*

Question 8.2. *When an approximation algorithm is not monotone, is it possible to derive from it a monotone algorithm that does not sacrifice any of the original algorithm's performance?*

Question 8.3. *For real practical mechanism design where there are no good approximation algorithms, what can we do?*

To get a hint at the answer to the first of these questions, we note that the important property of the LOS algorithm that implied its monotonicity was the greedy ordering. We conclude that any static greedy algorithm that orders agents as a monotone function of their value is monotone. The proof of this theorem is identical to that for LOS (Lemma 8.9).

Theorem 8.12. *For any set of n monotone non-decreasing functions $f_1(\cdot), \ldots, f_n(\cdot)$ the static greedy algorithm that sorts the agents in a non-increasing order of $f_i(v_i)$ is monotone.*

8.3 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 lucky 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 environment where worst-case approximation guarantees do not provide practical guidance in selecting among algorithms and mechanisms. Without some foreknowledge of the environment, 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 approach of the surplus maximization mechanism (Chapter 3) approach solves this mechanism design problem for an optimal algorithm designer; here we solve it for an *ad hoc* algorithm designer.

The statistical environment discussed above is well suited to the Bayesian mechanism design approach that has underlied most of the discussion in this text. The main result of this section is the polynomial-time constructive proof of the following theorem.

Theorem 8.13. *For any single-dimensional agent environment, any product distribution* $\mathbf{F}$*, and any algorithm* $\mathcal{A}$*, there is a BIC mechanism* $\bar{\mathcal{A}}$ *satisfying* $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}\big[\bar{\mathcal{A}}(\mathbf{v})\big] \geq \mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}(\mathbf{v})]$.

8.3.1 Monotonization

Let $\mathbf{x}(\mathbf{v})$ denote the allocation produced by the algorithm on input $\mathbf{v}$. For agent i the algorithms interim allocation rule is $x_i(v_i) = \mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[x_i(\mathbf{v}) \mid v_i]$. 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 2.16), 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 $\bar{x}_i(v_i)$.

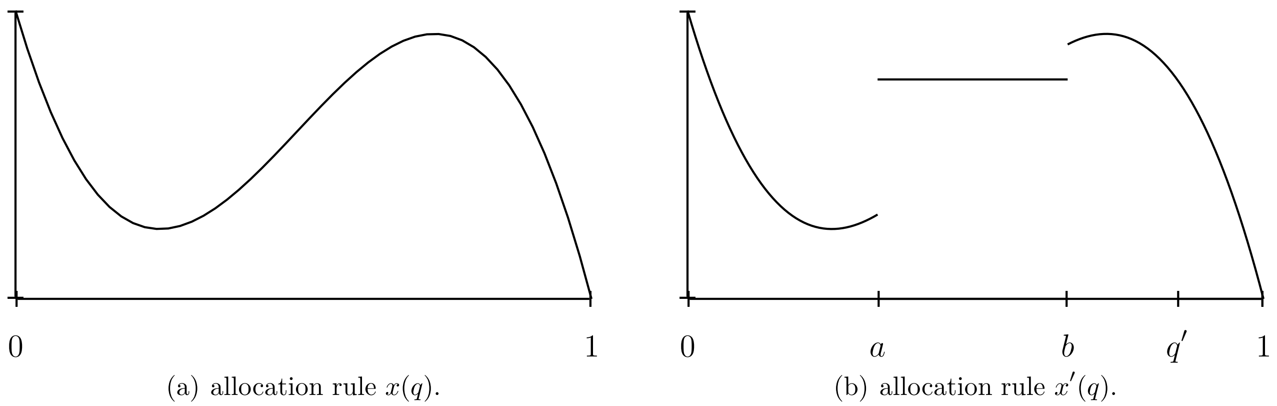
That the first two conditions are needed is intuitively clear. Monotonicity is required for Bayesian incentive compatibility. Surplus preservation is required if our construction is to not harm our objective. The requirement of locality is more subtle; however, notice that if no other agent can tell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

We will assume that the distribution F_i is continuous on its support and we will consider the allocation rule in quantile-space (cf. Chapter 3 Section 3.3.1). The transformation from value space to quantile space, recall, is given by $q_i = 1 - F_i(v_i)$. We denote the value associated with a given quantile as $v_i(q_i) = F_i^{-1}(1-q_i)$ and we express the allocation rule in quantile space as $x_i(q_i) = x_i(v_i(q_i))$. Notice that the quantile of agent i is drawn uniformly from $[0, 1]$.

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's quantile is $q \sim U[0, 1]$, her value is $v(q)$, her allocation rule (in quantile space) is $x(q)$. We assume that $x(\cdot)$ is not monotone non-increasing and focus on monotonizing it.

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 $x(q')$ instead of $x(q)$. (Back in valuation space, this can be implemented by ignoring Alice's value v and inputing $v' = v(q')$ 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 8.2(b).



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

Figure 8.2: The allocation rule $x(q)$ and $x'(q)$ constructed by running $x(q')$ when $q \in [a, b]$.

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 quantile 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 $x(q')$ instead of $x(q)$, i.e., simulate the algorithm with input $v' = v(q')$ in place of Alice's original value. Let $x'(q)$ 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 $x'(q) = x(q)$. For $q \in [a, b]$, Alice receives the average allocation probability for the interval $[a, b]$, i.e, $\frac{1}{b-a} \int_a^b x(r)\, dr$. See Figure 8.3(a).

Notice third, this approach is likely to improve Alice's expected surplus. Suppose $x(q)$ is increasing on $[a, b]$, meaning higher values are less likely to be allocated than low values, then this monotonization approach is just shifting allocation probability mass from low values to higher values.

Interval Selection and Monotonicity

The allocation rule $x'(\cdot)$ 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 $x'(q)$ is monotone if and only if its integral, $X'(q) = \int_0^q x'(r)dr$, is convex. We will refer to $X'(q)$ and $X(q)$ (defined identically for $x(q)$) as *cumulative allocation rules.*

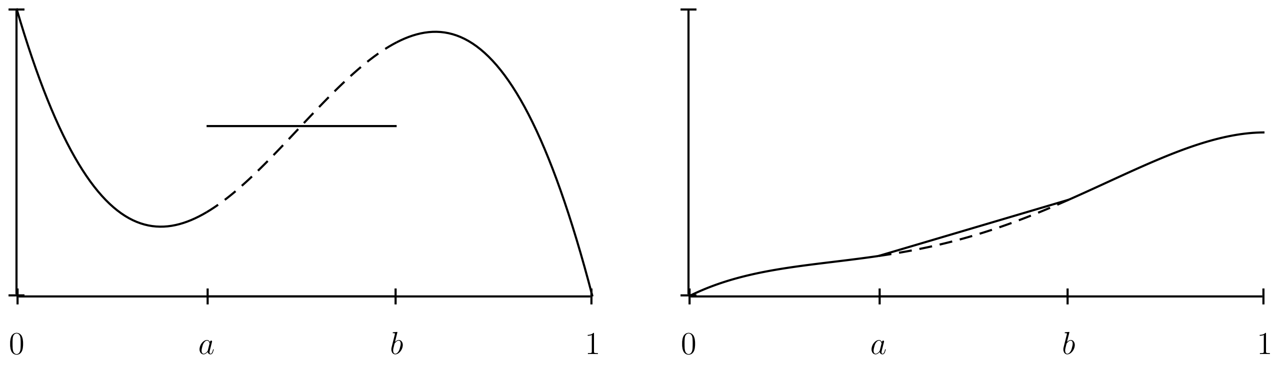


Figure 8.3: The allocation rule $x(q)$ (dashed) and $x'(q)$ (solid) constructed by drawing $q' \sim U[a, b]$ and running $x(q')$ when $q \in [a, b]$. Corresponding cumulative allocation rules $X(q)$ (dashed) and $X'(q)$ (solid).

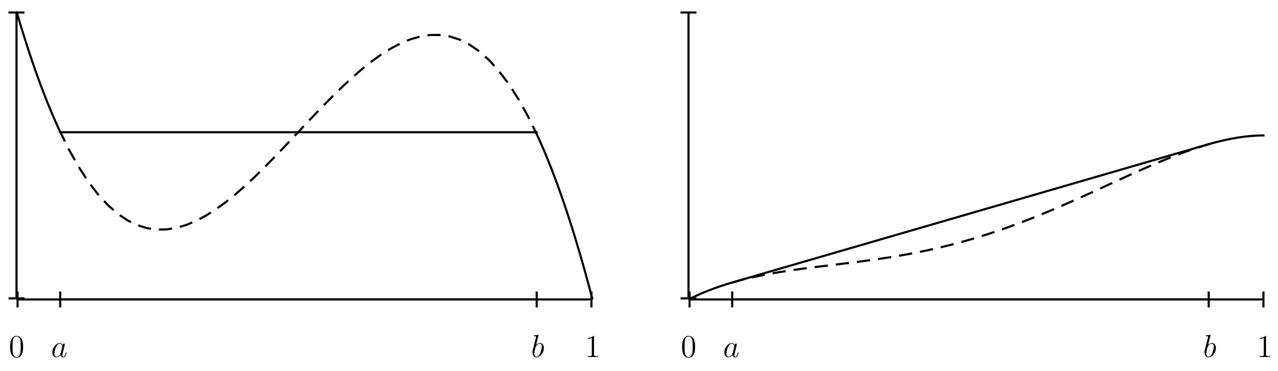
Notice second, the implication for $X'(q)$ of the resampling procedure on $X(q)$. Consider some $q \leq a$ as $x(q) = x'(q)$, clearly $X(q) = X'(q)$. In particular $X(a) = X'(a)$. Now calculate $X(b)$ and $X'(b)$. These are equal to $X(a)$ plus the integral of the respective allocation rules on $[a, b]$. Of course $x'(q)$ is constant on $[a, b]$ and equal, by definition, to $\frac{1}{b-a} \int_a^b x(r)dr$. The integral of a constant function is simply the value of the function times the length of the interval. Therefore $\int_a^b x'(r)dr = \int_a^b x(r)dr$. We conclude that $X(b) = X(b')$. Therefore, for all $q \geq b$, $X(q) = X'(q)$ as $x(q) = x'(q)$ for all such q. Thus, $X(q)$ and $X'(q)$ are identical on $[0, a]$ and $[b, 1]$. Of course $x'(q)$ is a constant function on $[a, b]$ so therefore its integral is a linear function; therefore, it must be the linear function that connects $(a, X(a))$ to $(b, X(b))$ with a straight line. See Figure 8.3(b).

Notice third, our choice of interval can be arbitrary and will always simply replace an interval of $X(q)$ with a straight line. Let $\bar{X}(\cdot)$ be the smallest concave function that upper bounds $X(\cdot)$. Let k be the number of contiguous subintervals of $[0, 1]$ for which $\bar{X}(q) \neq X(q)$ and let I_j be the jth interval. Let $\mathcal{I} = (I_1, \ldots, I_k)$. The following resampling procedure implements cumulative allocation rule $\bar{X}(\cdot)$. 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, $\bar{v} = v(\bar{q})$. The resulting allocation rule $\bar{x}(q)$ is $\frac{d\bar{X}(q)}{dq}$ which, by the convexity of $\bar{X}(\cdot)$, is monotone. See Figure 8.4.

Surplus Preservation

Notice that $\bar{X}(\cdot)$, as the smallest concave function that upper bounds $X(\cdot)$, satisfies $\bar{X}(q) \geq X(q)$. This dominance has the following interpretation: higher values have receive higher service probability. We formalize this argument in the following lemma.

Lemma 8.14. *For $x(\cdot)$ and $\bar{x}(\cdot)$ (as defined in the construction above),* $\mathbf{E}[v(q)\bar{x}(q)] \geq \mathbf{E}[v(q)x(q)]$.



(a) allocation rule $\bar{x}(q)$. (b) cumulative allocation rule $\bar{X}(q)$.

Figure 8.4: The allocation rule $\bar{x}(q)$ (solid) and cumulative allocation rule $\bar{X}(q)$ (solid) constructed by taking the convex hull of $X(q)$ (dashed). Interval $I = [a, b]$ is defined as $\{q : \bar{X}(q) \neq X(q)\}$.

Proof. We show that $\mathbf{E}[v(q)(\bar{x}(q) - x(q))]$ is non-negative.

$$\begin{aligned}\mathbf{E}[v(q)(\bar{x}(q) - x(q))] &= \int_0^q v(q)(\bar{x}(q) - x(q))\,dq \\ &= \Big[v(q)(\bar{X}(q) - X(q))\Big]_0^1 - \int_0^1 v'(q)(\bar{X}(q) - X(q))\,dq.\end{aligned}$$

The second line follows from integrating by parts. Of course: $v(\cdot)$ is decreasing so $v'(\cdot)$ is non-positive, $\bar{X}(q) = X(q)$ for $q \in \{0, 1\}$, and $\bar{X}(q) - X(q)$ is non-negative; therefore, in the second line above the first term is zero and the second term is non-negative. □

Reduction

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

Mechanism 8.2. *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 quantile space).*
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 8.13 directly follows.

Lemma 8.15. *The construction of $\bar{\mathcal{A}}$ from $\mathcal{A}$ is monotone.*

Lemma 8.16. *The construction of $\bar{\mathcal{A}}$ from $\mathcal{A}$ is local.*

Lemma 8.17. *The construction of $\bar{\mathcal{A}}$ from $\mathcal{A}$ is surplus preserving.*

8.3.2 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 8.18. *For any n-agent single-dimensional agent environment, 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 $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}\big[\bar{\mathcal{A}}_\epsilon(\mathbf{v})\big] \geq \mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}(\mathbf{v})] - \epsilon$. 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 dominant strategy incentive compatibility. Notably, the 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 DSIC mechanism design? This question has been partially answered in the negative; the conclusion being that there is loss in restricting attention to dominant strategy mechanisms in computationally bounded environments.

8.3.3 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 blackbox 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 8.3. *The* blackbox payment algorithm *for $\mathcal{A}$ computes payment P_i for agent i with value v_i as follows:*

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

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

   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 her value minus her 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 8.19. *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,

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

This construction was slightly odd because a loser may be paid a positive transfer. This can be partially addressed. There is a slightly more complicated construction wherein all losers have payments identically equal to zero, but winners may be paid to receive service. We leave the construction as an exercise.

8.4 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 externality payments $p_i = \mathrm{OPT}(\mathbf{v}_{-i}) - \mathrm{OPT}_{-i}(\mathbf{v})$ 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. One 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 allocation.

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 her ad is clicked on. Each advertiser i has a private value v_i for each click she receives. 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. Many 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.

8.4.1 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 surplus maximization mechanism.

It is likely that such a result does not hold generally. 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 a private input and the agents 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 that 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. This is a single-dimensional agent environment 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. For the objective of social surplus, clearly we wish to serve the agents whenever $\sum_i v_i \geq C$.

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 her 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.

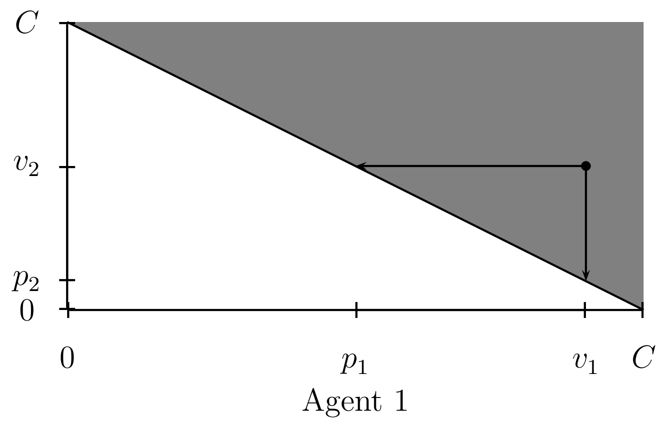


Figure 8.5: As a function of agent 1's value (x-axis) and agent 2's value (y-axis) and the the region of allocation (gray) of the surplus maximizing mechanism for the public project with cost C is depicted. Valuation profile $\mathbf{v} = (v_1, v_2)$ is depicted by a point and the payments $\mathbf{p} = (p_1, p_2)$ can be calculated by following the accompanying arrows.

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 the 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 environment about twice as many bits are required to compute payments than to compute the outcome. We summarize this in the following lemma.

Lemma 8.20. *There exists a two-agent single-dimensional agent environment 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 environment where the communication complexity of payments is n times more than that for computing the allocation. The single-dimensional agent problem that exhibits such a separation is contrived, i.e., there is no natural economic interpretation for it. The real challenge in this generalization is in determining an environment where the allocation can be computed with very low communication. We omit the proof and construction.

Theorem 8.21. *There exists an n-agent single-dimensional agent environment where 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 environments 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, by invoking the algorithm $n+1$ times as is done by the payment algorithms previously discussed. Notice, however, that the above lower bound leaves open the possibility that subtle changes to the allocation rule might permit payments to be calculated without such computational overhead.

8.4.2 Implicit Payments

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 the outcome and all payments to be computed implicitly from a single 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 without a prior distribution.) 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 $\mathbf{E}\big[x_i(V_i')\big]$. 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 8.3. *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) $V_i' \leftarrow \begin{cases} v_i & \textit{with probability } 1-\epsilon \\ Z & \textit{otherwise.} \end{cases}$

2. $\mathbf{X}' \leftarrow \mathcal{A}(\mathbf{V}')$.

3. for all i: $P'_i \leftarrow \begin{cases} v_i X'_i & \textit{if } V'_i = v_i \\ -\frac{1-\epsilon}{\epsilon} \cdot \frac{X'_i}{f_i(V'_i)} & \textit{if } V'_i < v_i \\ 0 & \textit{otherwise.} \end{cases}$

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 her 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 8.22. *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, $\mathbf{x}'(\mathbf{v}) = (1-\epsilon)\cdot\mathbf{x}(\mathbf{v}) + \epsilon\cdot\mathbf{E}_{\mathbf{V}'}\big[\mathbf{x}(\mathbf{V}')\big]$, i.e., we have scaled the algorithms allocation rule by $(1-\epsilon)$ and added a constant. Therefore, relative to prior distribution F, if $\mathcal{A}$ was monotone before then $\mathcal{A}'_\epsilon$ is monotone now. (Furthermore, if $\mathcal{A}$ was monotone in an dominant strategy sense before then the implicit payment mechanism is monotone in an dominant strategy sense now.) □

Lemma 8.23. *For agent i with value* v_i*,* $\mathbf{E}\big[P'_i\big] = p'_i(v_i)$ *satisfying the payment identity.*

Proof. Since our allocation rule for agent i is $x'_i(v_i) = (1-\epsilon)\cdot x_i(v_i) + \epsilon \cdot \mathbf{E}\big[x_i(V'_i)\big]$ 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 analyze:

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

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

$$\begin{aligned}\mathbf{E}\big[X_i' B/f_i(V_i')\big] &= \mathbf{Pr}[B=1]\cdot\mathbf{E}\big[X_i'/f_i(V_i') \mid B=1\big]\\ &= \epsilon F_i(v_i)\cdot\mathbf{E}\big[X_i'/f_i(V_i') \mid B=1\big].\end{aligned}$$

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. $\square$

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 environments separately. We consider the general costs environment because it is the most general and the general feasibility environment because it is easier and thus permits a nicer performance bound.

Lemma 8.24. *For any distribution* $\mathbf{F}$ *and any general feasibility setting,* $\mathbf{E}\big[\mathcal{A}_\epsilon'(\mathbf{v})\big] \geq (1-\epsilon)\cdot\mathbf{E}[\mathcal{A}(\mathbf{v})]$.

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

$$\begin{aligned}\mathbf{E}_{v_i}\big[v_i x_i'(v_i)\big] &= (1-\epsilon)\cdot\mathbf{E}_{v_i}[v_i x_i(v_i)] + \epsilon\cdot\mathbf{E}_{v_i}[v_i]\cdot\mathbf{E}_{v_i}[x_i(v_i)]\\ &\geq (1-\epsilon)\cdot\mathbf{E}_{v_i}[v_i x_i(v_i)] \qquad (8.6)\end{aligned}$$

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

Lemma 8.25. *For any distribution* $\mathbf{F}$ *and any* n*-agent general costs setting,* $\mathbf{E}\big[\mathcal{A}_\epsilon'(\mathbf{v})\big] \geq \mathbf{E}[\mathcal{A}(\mathbf{v})] - \epsilon h n$ *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., $\mathbf{E}_{\mathbf{v}}\big[c(\mathbf{x}'(\mathbf{v}))\big] = \mathbf{E}_{\mathbf{v}}[c(\mathbf{x}(\mathbf{v}))]$. However, the expected total value to the agents is decreased relative to the original algorithm. Our expected surplus from i is

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

The first step follows from equation (8.6), 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 8.26. *For any single-dimensional agent environment with general costs, any product distribution $\mathbf{F}$ over $[0,h]^n$, any Bayesian monotone algorithm $\mathcal{A}$, and any $\epsilon > 0$, the implicit payment mechanism $\mathcal{A}'_\epsilon$ is BIC and satisfies $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}\big[\mathcal{A}'_\epsilon(\mathbf{v})\big] \geq \mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}(\mathbf{v})] - \epsilon hn$.*

Theorem 8.27. *For any single-dimensional agent environment with general feasibility constraints, any product distribution $\mathbf{F}$, any Bayesian monotone algorithm $\mathcal{A}$, and any $\epsilon > 0$, the implicit payment mechanism $\mathcal{A}'_\epsilon$ is BIC and satisfies $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}\big[\mathcal{A}'_\epsilon(\mathbf{v})\big] \geq (1-\epsilon)\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[\mathcal{A}(\mathbf{v})]$.*

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.

Exercises

8.1 Section 8.3 we showed that we can turn a non-monotone algorithm for any single-dimensional agent environment into a BIC mechanism that has at least the same expected social surplus. Consider another important objective in computer systems: *makespan*. Suppose we have n machines and m jobs. Each machine i is a selfish agent with privately known slowness-parameter $|v_i|$ and each job j has an intrinsic length w_j. The time it takes i to perform job j is $|v_i w_j|$ and we view this as a cost to agent i.[2] The load on machine i for a set of jobs J is $|v_i| \sum_{j\in J} w_j$. A scheduling is a partitioning of jobs among the machines. Let J_i be the jobs assigned to machine i. The makespan of this scheduling is the maximum load of any machine, i.e., $\max_i |v_i| \sum_{j\in J_i} w_j$. The goal of a scheduling algorithm is to minimize the makespan. (Makespan is important outside of computer systems as "minimizing the maximum load" is related to fairness.)

[2]We are putting these quantities in absolute values because if the private value represents a cost, it is most consistent with the course notation to view v_i as negative.

This is a single-dimensional agent environment, though the outcome for each agent i is not a binary $x_i \in \{0,1\}$. Instead if i is allocated jobs J_i then $x_i = \sum_{j \in J_i} w_j$. Of course $x_i(v_i)$ is, as usual, $\mathbf{E}_{\mathbf{v}\sim\mathbf{F}}[x_i(\mathbf{v}) \mid v_i]$. The agent's cost for such an outcome is $|v_i x_i(v_i)|$.

Show that the method of Mechanism 8.2 for monotonizing a non-monotone algorithm fails to preserve the expected makespan. (Hint: All you need to do is come up with an instance and a non-monotone algorithm for which expected makespan increases when we apply the transformation. Do not worry about the values being negative nor about the allocation being non-binary; these aspects of the problem do not drive the non-monotonicity result.)

8.2 The blackbox payment algorithm (Algorithm 8.3) sometimes makes positive transfers to agents. Give a different algorithm for calculating payments with the correct expectation where (a) winners never pay more than their value and (b) the payment to (and from) losers is always identically zero. The number of calls your payment algorithm makes to the allocation algorithm should be at most a constant (in expectation).

8.3 Dominant strategy incentive compatible mechanisms can be combined.

(a) Consider the following algorithm:

- Simulate greedy by value (i.e., sorting by v_i).
- Simulate greedy by value-per-item (i.e., sorting by $v_i/|S_i|$).
- Output whichever solution has higher surplus.

Prove that the resulting algorithm is monotone.

(b) We say a deterministic algorithm is *independent of irrelevant alternatives* when

$$\mathbf{x}(\mathbf{v}_{-i}, v_i) = \mathbf{x}(\mathbf{v}_{-i}, v_i') \qquad \text{iff} \qquad x_i(\mathbf{v}_{-i}, v_i) = x_i(\mathbf{v}_{-i}, v_i'),$$

i.e., the outcome is invariant on the value of a winner or loser. Prove that the algorithm $\mathcal{A}$ that runs k deterministic monotone independent-of-irrelevant-alternatives algorithms $\mathcal{A}_1 \ldots, \mathcal{A}_k$ and then outputs the solution of the one with the highest surplus is itself monotone.

8.4 Consider the following knapsack problem: each agent has a private value v_i for having an object with publicly known size w_i inserted into a knapsack. The knapsack has capacity C. Any set of agents can be served if all of their objects fit simultaneously in the knapsack. We denote an instance of this problem by the tuple $(\mathbf{v}, \mathbf{w}, C)$. Notice that this is a single-dimensional agent environment with cost function:

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

The knapsack problem is $\mathcal{NP}$-complete; however, very good approximation algorithms exist. In fact there is a *polynomial time approximation scheme* (PTAS). A PTAS is an family of algorithms parameterized by $\epsilon > 0$ where $\mathcal{A}_\epsilon$ is a $(1+\epsilon)$-approximation runs in polynomial time in n, the number of agents, and $1/\epsilon$.

PTASs are often constructed from pseudo-polynomial time algorithms. Let V be an upper bound on the value of any agent, i.e., $V \geq v_i$ for all i, and assume all agent values are integers. For the integer-valued knapsack problem there is an algorithm with runtime $O(n^2V)$. (Constructing this algorithm is non-trivial. Try to do it on your own, or look it up in an algorithms text book.) Notice that this is not fully polynomial time as V is a number that is part of the input to the algorithm. The "size" of V is the amount of space it takes to write it down. We ordinarily write numbers on a computer in binary (or by hand, in decimal) which therefore has size $\log V$. An algorithm with runtime polynomial in V is exponential in $\log V$ and, therefore, not considered polynomial time. It is *pseudo-polynomial time.*

A pseudo-polynomial time algorithm, $\mathcal{A}$, for surplus maximization in the integer values setting can be turned into a PTAS $\mathcal{A}_\epsilon$ for the general values setting by rounding. The construction:

- $v'_i \leftarrow v_i$ rounded up to the nearest multiple of $V\epsilon/n$.
- $v''_i \leftarrow v'_i n/(V\epsilon)$, an integer between 0 and n/ϵ.
- Simulate $\mathcal{A}(\mathbf{v}'', \mathbf{w}, C)$, the integer-valued pseudo-polynomial time algorithm.
- Simulate the algorithm that allocates only to the highest valued agent.
- Output whichever solution has higher surplus.

Notice the following about this algorithm. First, its runtime is $O(n^3/\epsilon)$ when applied to the $O(n^2V)$ pseudo-polynomial time algorithm discussed above. Second, it is a $(1+\epsilon)$-approximation if we set $V = \max_i v_i$. (This second observation involves a several line argument. Work it out yourselves or look it up in an algorithms text book.)

In this question we will investigate the incentives of this problem which arise because we do not know a good choice of V in advance.

(a) Suppose we are given some V. Prove that $\mathcal{A}_\epsilon$, for any ϵ, is monotone.

(b) Suppose we do not know V in advance. A logical choice would be to choose $V = \max_i v_i$ and then run $\mathcal{A}_\epsilon$. Prove that this combined algorithm is not monotone.

(c) For any given ϵ, derive an DSIC $(1+\epsilon)$-approximation mechanism from any integer-valued pseudo-polynomial time algorithm. Your algorithm should run in polynomial time in n and $1/\epsilon$. (Hint: The hard part is not knowing V.)

Chapter Notes

Richard Karp (1972) 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. Lehmann et al. (2002) and Nisan and Ronen (2001) introduced the concept of computationally tractable approximation mechanisms. The single-minded combinatorial auction problem is an iconic single-dimensional agent mechanism design problem.

There are several reductions from approximation mechanism design to approximation algorithm design (in single-dimensional settings). Briest et al. (2005) show that any polynomial time approximation scheme (PTAS) can be converted into an (dominant strategy) 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 (2010); Hartline et al. (2011) improve on the basic approach. For the special case of single-minded (and a generalization that is not strictly single-dimensional) combinatorial auctions Babaioff et al. (2009a) 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.

Several of the above results have extensions to environments with multi-dimensional agent preferences. Specifically, Dughmi and Roughgarden (2010) show that in multi-dimensional additive-value packing environments any PTAS algorithm can be converted into a PTAS mechanism. The result is via a connection between smoothed analysis and PTASes: A PTAS can be viewed as optimally solving a perturbed problem instance and any optimal algorithm is incentive compatible via the standard approach. Hartline et al. (2011) and Bei and Huang (2011) generalized the construction of Hartline and Lucier (2010) to reduce BIC mechanism design to algorithm design in general multi-dimensional environments. Notably, the approach is brute-force in the type space of each agent.

The unbiased estimator payment computation was given by Archer et al. (2003). The communication complexity lower bound for computing payments is given by Babaioff et al. (2008). Babaioff et al. (2010) 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 environments where the algorithm cannot be repeated, for instance, in online auction settings such as the one independently considered by Babaioff et al. (2009b) and Devanur and Kakade (2009).