Chapter 3

Social Surplus and Tractability

In this chapter we discuss the objective of social surplus. As we will see, ignoring computational tractability, the economics of designing mechanisms to maximize social surplus is relatively simple. There is a unique optimal mechanism, and that mechanism is a simple generalization of the second-price auction we have already discussed. Furthermore, is ex post IC and prior-free, i.e., it is not dependent on distributional assumptions. Unfortunately, there are many relevant settings where this optimal mechanism is computationally intractable. A mechanism that is computationally intractable, i.e., no computer could calculate the outcome of the mechanism in a reasonable amount of time, seems to violate 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 ex post IC approximation mechanisms.

Ex post IC and prior-free mechanisms may be too demanding for this setting. In fact, without making an assumptions on the setting 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 BIC 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.

3.1 Single-dimensional Settings

In our previous discussion of Bayes-Nash equilibrium we focused on the agents' incentives. Agents are single-dimensional, i.e., each has a single private value for receiving some abstract service and quasi-linear utility, i.e., the agent's utility is their value for the service less their payment. Here we formalize the designer's constraints and objectives.

Definition 3.1 A *general costs setting* is one where the designer must pay a cost $c(\mathbf{x})$ for the allocation produced.

Definition 3.2 A *general feasibility setting* is one where there is a feasibility constraint over the set of agents that can be simultaneously served.

Definition 3.3 A *downward-closed* feasibility constraint is one where subsets of feasible sets are feasible.

Of course, downward-closed feasibility settings are a special case of general feasibility settings which are a special case of general costs settings. We can express general feasibility settings as general costs settings were $c(\cdot) \in \{0, \infty\}$. We can similarly express downward-closed feasibility settings as the further restriction where $c(\mathbf{x}) = 0$ and $\mathbf{x}' \leq \mathbf{x}$ (i.e., for all i, $x'_i \leq x_i$) implies that $c(\mathbf{x}') = 0$. We will be aiming for general mechanism design results and the most general results will be the ones that hold in the most general settings. However, the more specific the setting the easier it will be to obtain positive results.

The two most fundamental designer objectives are social surplus, a.k.a., social welfare,[1] and profit. We focus on social surplus in this chapter and profit in subsequent chapters.

Definition 3.4 The *social surplus* of an allocation is the cumulative value of agents served less the service cost:

$$\text{Surplus}(\mathbf{v}, \mathbf{x}) = \sum\nolimits_i v_i x_i - c(\mathbf{x})$$

Definition 3.5 The *profit* of allocation and payments is the cumulative payment of agents less the service cost:

$$\text{Profit}(\mathbf{p}, \mathbf{x}) = \sum\nolimits_i p_i - c(\mathbf{x})$$

Single-item auctions and single-minded combinatorial auctions are special cases of downward-closed feasibility settings. Single-item auctions have

$$c(\mathbf{x}) = \begin{cases} 0 & \text{if } \sum_i x_i \leq 1 \\ \infty & \text{otherwise.} \end{cases}$$

In single-minded combinatorial auctions, recall, each agent has a bundle of desired items S_i and is only happy if they receive every item in the bundle. An agent is "served" if they

[1] A mechanism that optimizes social surplus is said to be *economically efficient* we will not use this terminology because of possible confusion with *computational efficiency*.

receive their bundle and "not served" otherwise.

$$c(\mathbf{x}) = \begin{cases} 0 & \text{if } \forall i, i' \; x_i = x_{i'} = 1 \Rightarrow S_i \cap S_{i'} = \emptyset \\ \infty & \text{otherwise.} \end{cases}$$

Path auctions are a special case of general feasibility settings. Recall in path auctions that we are trying to procure an s-t path in a graph $G = (V, E)$ where the agents correspond to the edges. We consider an agent to be served if their edge is in the selected path and not served otherwise. Notice that path auctions are not downward-closed. The set of served agents must correspond to an s-t path, a subset of this set does not and is therefore infeasible.

$$c(\mathbf{x}) = \begin{cases} 0 & \text{if served agents form an } s\text{-}t \text{ path in graph } G \\ \infty & \text{otherwise.} \end{cases}$$

We have yet to formally discuss any examples of general costs settings. One natural problem is that of a *multicast auction*. The story for this problem comes from live video steaming. Suppose we wish to stream live video to agents in a computer network. Because of the high-bandwidth nature of video streaming the content provider must lease the network links. Each link has a publicly known cost. To serve a set of agents, the designer must pay the cost of network links that connect each agent, located at different nodes in the network, to the "root", i.e., the origin of the multicast. The nature of multicast is that the messages need only be transmitted once on each edge to reach the agents. Therefore, the total cost to serve these agents is the minimum cost of the *multicast tree* that connects them. (Algorithmically this is a *weighted Steiner tree* problem, a computationally challenging variant of the *minimum spanning tree* problem.)

3.2 Optimal Mechanisms for Social Surplus

We now derive the optimal mechanism for social surplus. To do this we walk through a standard approach in mechanism design. We completely relax the Bayes-Nash equilibrium incentive constraints and ask and solve the remaining non-game-theoretic optimization question. We then verify that this algorithmic solution does not violate the incentive constraints. We conclude that the resulting mechanism is optimal.

The the non-game-theoretic optimization problem of maximizing surplus is that of finding $\mathbf{x}$ to maximize $\text{Surplus}(\mathbf{v}, \mathbf{x}) = \sum_i v_i x_i - c(\mathbf{x})$. Let OPT be an optimal algorithm for solving this problem. We will care about both the allocation that OPT selects, i.e., $\text{argmax}_{\mathbf{x}} \text{Surplus}(\mathbf{v}, \mathbf{x})$ and its surplus $\max_{\mathbf{x}} \text{Surplus}(\mathbf{v}, \mathbf{x})$. Where it is unambiguous we will use notation $\text{OPT}(\mathbf{v})$ to denote either of these quantities. Notice that the formulation of OPT has no mention of Bayes-Nash equilibrium incentive constraints.

We know from our characterization that the allocation rule of any BNE is monotone. Thus, the mechanism design problem of finding a BIC mechanism to maximize surplus has an added monotonicity constraint. Even though we did not impose a monotonicity constraint on OPT it is satisfied anyway.

Lemma 3.6 *For each agent i and all values of other agents $\mathbf{v}_{-i}$, the i's allocation rule in OPT is monotone in i's value v_i.*

Proof: Consider any agent i. There are two situations of interest. Either i is served by $\text{OPT}(\mathbf{v})$ or i is not served by $\text{OPT}(\mathbf{v})$. We write out the surplus of OPT in both of these cases.

Case 1: ($i \in \text{OPT}$)

$$\begin{aligned}\text{OPT}(\mathbf{v}) &= \max_{\mathbf{x}} \text{Surplus}(\mathbf{v}, \mathbf{x}) \\ &= v_i + \max_{\mathbf{x}_{-i}} \text{Surplus}((\mathbf{v}_{-i}, 0), (\mathbf{x}_{-i}, 1))\end{aligned}$$

Define $\text{OPT}_{-i}(\mathbf{v})$ as the second term on the right hand side. Thus,

$$\text{OPT}(\mathbf{v}) = v_i + \text{OPT}_{-i}(\mathbf{v}).$$

Notice that $\text{OPT}_{-i}(\mathbf{v})$ is not a function of v_i.

Case 2: ($i \notin \text{OPT}$)

$$\begin{aligned}\text{OPT}(\mathbf{v}) &= \max_{\mathbf{x}} \text{Surplus}(\mathbf{v}, \mathbf{x}) \\ &= \max_{\mathbf{x}_{-i}} \text{Surplus}((\mathbf{v}_{-i}, 0), (\mathbf{x}_{-i}, 0)).\end{aligned}$$

Define $\text{OPT}(\mathbf{v}_{-i})$ as the term on the right hand side. Thus,

$$\text{OPT}(\mathbf{v}) = \text{OPT}(\mathbf{v}_{-i}).$$

Notice that $\text{OPT}(\mathbf{v}_{-i})$ is not a function of v_i.

OPT chooses whether or not to allocate to agent i, and thus which of these cases we are in, so as to optimize the surplus. Therefore, OPT allocates to i whenever the surplus from Case 1 is greater than the surplus from Case 2. I.e., when

$$v_i + \text{OPT}_{-i}(\mathbf{v}) \geq \text{OPT}(\mathbf{v}_{-i}).$$

Solving for v_i we conclude that OPT allocates to i whenever

$$v_i \geq \text{OPT}(\mathbf{v}_{-i}) - \text{OPT}_{-i}(\mathbf{v}).$$

Notice that neither of the terms on the right hand side contain v_i. We conclude that agent i has a critical value $t_i = \text{OPT}(\mathbf{v}_{-i}) - \text{OPT}_{-i}(\mathbf{v})$ and when i's value is above this critical value i wins, and when below i loses. The allocation rule for i is a step function which is monotone. ■

Since OPT is monotone for each agent and all values of other agents it satisfies our strongest incentive constraint. With the appropriate payments (i.e., the "critical values") truthtelling is a dominant strategy equilibrium (Corollary 2.8). This mechanism is jointly credited to William Vickrey, Edward Clarke, and Theodore Groves.

Mechanism 3.1 *The* Vickrey-Clarke-Groves *(VCG) mechanism is:*

1. *Solicit and accept sealed bids* $\mathbf{b}$.
2. $\mathbf{x} = \text{OPT}(\mathbf{b})$, *and*
3. *for each* i, $p_i = \text{OPT}(\mathbf{b}_{-i}) - \text{OPT}_{-i}(\mathbf{b})$.

An intuitive description of $\text{OPT}(\mathbf{v}_{-i}) - \text{OPT}_{-i}(\mathbf{v})$ is the *externality* that i imposes on the other agents by being served. I.e., because i is served the other agents obtain total surplus $\text{OPT}_{-i}(\mathbf{v})$ instead of the surplus $OPT(\mathbf{v}_{-i})$ that they would have received if i was not served. Hence, we can interpret the VCG mechanism as serving agents to maximize the social surplus and charging each agent the externality imposed on the others.

By Theorem 2.7 and Lemma 3.6 we have the following theorem. The corollary follows from the optimality of OPT and the assumption that agents follow the dominant truthtelling strategy.

Theorem 3.7 *VCG is IC.*

Corollary 3.8 *VCG optimizes social surplus.*

The second-price path auction from Chapter 1 is simply an instantiation of the VCG mechanism where feasible outcomes $\mathbf{x}$ are s-t paths in a graph. The problem of computing a shortest path in a graph, i.e., the optimization problem of OPT, is easily solvable by Dijkstra's classic shortest paths algorithm. Unfortunately, as we will see next, not all auction problems are so easily solvable.

3.3 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 3.9 $\mathcal{P}$ is the class of problems that can be *solved* in polynomial time (in the "size of the input").

Consider the aforementioned shortest paths problem. The input to such a problem is the graph $G = (V, E)$ (on vertices V and edges E) with costs on edges, which we denote here by $-v_i$ for edge $i \in E$, and special vertices s and t.[2] The size of this input is $n = |E|$ as each agent must be represented. Dijkstra's algorithm for shortest paths runs in time $O(n \log n)$ which is polynomial.

[2]As we have defined values as the positive benefit an agent receives from being served, in cost problems the agents' values are negative.

Definition 3.10 $\mathcal{NP}$ 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 knows how to come up with a better algorithm than such an exhaustive search.

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 failing. The following approach allows one to leverage this collective failing to argue that a given problem X is unlikely to be polynomial-time solvable by a computer.

Definition 3.11 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 3.12 A problem X is $\mathcal{NP}$*-hard* if all problems $Y \in \mathcal{NP}$ reduce to it.

Definition 3.13 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.

3.4 Single-minded Combinatorial Auctions

Recall the example setting of single-minded combinatorial auctions. This is an important setting 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 3.14 *The single-minded combinatorial auction problem is $\mathcal{NP}$-complete.*

3.4.1 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.

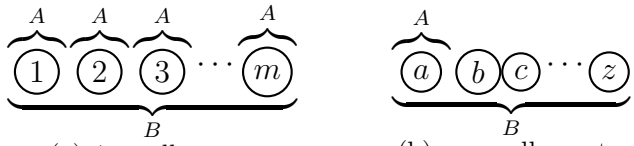
Algorithm 3.1 *A* static greedy algorithm *is*

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. Agent 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 3.1(b) 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 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., $\Omega(m)$.



(a) 1 small agent. (b) m small agents.

Figure 3.1: Challenge cases for greedy orderings as a function of value and bundle size.

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 the 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 3.1(a) 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 3.1.

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 3.1.

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

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 bidder. 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 single-minded combinatorial auctions by Daniel Lehmann, Liadan O'Callaghan, and Yoav Shoham.

Definition 3.15 The *Lehmann-O'Callaghan-Shoham* (LOS) algorithm is greedy by $v_i/\sqrt{|S_i|}$.

Theorem 3.16 *LOS is a $\sqrt{m}$-approximation algorithm (where m is the number of items).*

Proof: Let I be the set selected by LOS and I^* be the set selected by OPT. 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 LOS) by $v_i/\sqrt{|S_i|}$. For an agent $i^* \in I^*$ not to be served by LOS, 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 LOS, we charge it to itself, i.e., $F_{i^*} = \{i^*\}$. Notice that the sets F_i partition the agents I^* of OPT.

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

$$\begin{aligned}
\mathrm{OPT}(\mathbf{v}) &= \sum_{i^*} v_{i^*} = \sum_{i \in I} \sum_{i^* \in F_i} v_{i^*} && (3.1)\\
&\leq \sum_{i \in I} \tfrac{v_i}{\sqrt{|S_i|}} \sum_{i^* \in F_i} \sqrt{|S_{i^*}|} && (3.2)\\
&\leq \sum_{i \in I} \tfrac{v_i}{\sqrt{|S_i|}} \sum_{i^* \in F_i} \sqrt{m/|F_i|} && (3.3)\\
&= \sum_{i \in I} \tfrac{v_i}{\sqrt{|S_i|}} \sqrt{m|F_i|} && (3.4)\\
&\leq \sum_{i \in I} v_i \sqrt{m} = \sqrt{m} \cdot \mathrm{LOS}(\mathbf{v}). && (3.5)
\end{aligned}$$

Line (3.1) follows because F_i partition I^*. Line (3.2) 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 (3.3) 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 (3.4). Finally, line (3.5) 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. ■

The as witnessed by the theorem above, the LOS 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 3.17 *Under standard complexity-theoretic assumptions,*[3] *no polynomial time algorithm gives an $o(\sqrt{m})$-approximation to weighted set packing.*

3.4.2 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 VCG formula 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 LOS 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 3.1(b) with $A = 1$ and $B = \sqrt{m} + \epsilon$). On such an input LOS accepts only agent $m+1$. However, when computing the payment with the VCG-like formula $p_{m+1} = \text{LOS}(\mathbf{v}_{-(m+1)}) - \text{LOS}_{-(m+1)}(\mathbf{v})$ 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 VCG in 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 LOS algorithm no attempt was made to obtain monotonicity, it is satisfied anyway.

Lemma 3.18 *For each agent i and all values of other agents $\mathbf{v}_{-i}$, the i's allocation rule in LOS is monotone in i's value v_i.*

Proof: It suffices to show that if i with value v_i is served by LOS on $\mathbf{v}$ and i increases their 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 their bid they 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 of i is considered earlier. Therefore, i will still be served with a higher bid. ■

The above proof shows that there is a critical value t_i for i and if $v_i > t_i$ then i is served. It is easy to identify this critical value by simulating LOS on $\mathbf{v}_{-i}$. Let i' be the earliest agent

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

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 LOS. Therefore i's critical value is the t_i for which $v_i = t_i$ would tie agent i' in the ordering. I.e., $t_i = v_{i'}\sqrt{|S_i|}/\sqrt{|S_{i'}|}$.

We conclude by formally giving the LOS mechanism and the theorem and corollary that describe its incentives and performance.

Mechanism 3.2 *The* Lehmann-O'Callaghan-Shoham *(LOS) mechanism is:*

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

Theorem 3.19 *LOS is IC.*

Corollary 3.20 *LOS is a* $\sqrt{m}$*-approximation mechanism.*

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 3.1 When are approximation algorithms monotone?

Question 3.2 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 3.3 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 3.18).

Theorem 3.21 *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.*