Algorithmic Mechanism Design

Through the lens of Multi-unit auctions

Noam Nisan [*]

January 21, 2014

Abstract

Mechanism Design is a sub-field of game theory that aims to *design games* whose equilibria have desired properties such as achieving high efficiency or high revenue. *Algorithmic Mechanism Design* is a subfield that lies on the border of Mechanism Design and Computer Science and deals with Mechanism Design in algorithmically-complex scenarios that are often found in computational settings such as the Internet.

The central challenge in Algorithmic Mechanism Design is the tension between the computational constraints and the game-theoretic ones. This survey demonstrates both the tension and ways of addressing it by focusing on a single simple problem: multi-unit auctions. A variety of issues will be discussed: representation, computational hardness, communication, convexity, approximations, VCG mechanisms and their generalizations, single-parameter settings vs. multi-parameter settings, and the power of randomness.

1 Introduction

As the Internet was becoming ubiquitous in the 1990's it became increasingly clear that its understanding requires combining insights from Computer Science with those from Game Theory and Economic Theory. On one hand, much economic activity was moving to the Internet, and on the other hand many computational systems started involving computers with different owners and goals. Computational implementations of familiar economic institutions such as catalogs, markets, auctions, or stores opened up unprecedented new opportunities – and challenges – in terms of scale, size, speed, and complexity.

[*]Microsoft Research and Hebrew University of Jerusalem. Research partially supported by a grant from the Israeli Science Foundation.

As familiar computerized protocols were adapted to the Internet environment, the economic aspects of the resulting cooperation and competition needed to be increasingly taken into account. A new field of study for handling such combinations of computerized and economic considerations was born, a field which is often termed *Algorithmic Game Theory*. A textbook for this field is Nisan et al. [2007].

Perhaps the most natural sub-field of economics to be combined with computational considerations is that of *Mechanism Design*. The field of Mechanism Design aims to *design* economic institutions as to achieve various economic goals, most notably revenue and social welfare. Combining the points of view of social choice and of game theory, it aims to design games whose outcome will be as the game designer desires – whenever rational players engage in them. The field was initiated by the seminal, Nobel prize winning, work of Vickrey [1961], with an extensive amount of further work done over the last half century. As argued in Roth [2003], the field of mechanism design may be viewed as *Engineering*, aiming to create *new* "mechanisms" using its tools of the trade.

In the mid 1990's researchers from several disciplines recognized that mechanism design could be relevant to computation over the Internet. Examples of influential discussions along these lines include "Economic mechanism design for computerized agents" written by an economist (Varian [1995]), "Rules of encounter: designing conventions for automated negotiation among computers" written by Artificial Intelligence researchers (Rosenschein and Zlotkin [1994]), and "Pricing in computer networks: Reshaping the research agenda" written by computer network researchers (Shenkar et al. [1996]). By the end of the millennium, formal theoretical models that combine mechanism design with computation were introduced in Nisan and Ronen [2001] where the term *Algorithmic Mechanism Design* was coined.

Before we dive into the special challenges of the *combination* of "Algorithmic" with "Mechanism Design", let us briefly present the basic point of view of each of these two disciplines separately. The field of Mechanism Design considers a set of *rational players*, where "rational" is defined to be – as usual in economic theory – utility-maximizing. A *social outcome* that affects all these players needs to be chosen by a *planner* who does *not have full information*, information needed for determining the "preferred" social outcome. The planner's goal is to design a mechanism – a protocol – that when played by the rational agents results in an equilibrium that satisfies the planner's goals.

In classical Computer Science there are no utilities. A problem describes an input-output relationship and the goal is to produce the correct output for every given input. The difficulty is that of computation: we would like to *efficiently* find the correct output, where efficiency is measured in terms of computational resources: mostly computation time, but also memory, communication, etc. Much work in computer science focuses on distributed problems, where the input and output are distributed among multiple computers who must communicate with each other. Such work on distributed systems became very central as the Internet became the default computational platform.

The usual treatment of the distributed processors in Computer Science was that they were either "obedient" – programmed and controlled by the designer-engineer – or that they were "faulty" – in which case we have no control over their behavior within the bounds of our fault models. However, for many Internet applications, none of these points of view seemed to capture the essence of the situation. What seems to be the essence is that the computers are simply trying to further the goals of their owners. While such behavior is the bread and butter of economics and game-theory, it was certainly new for Computer Scientists. Hence the *rational* modeling of participants was adopted into computational settings and Mechanism Design was embraced as a paradigm for the design of distributed computational protocols over the Internet.

2 Algorithmic Mechanism Design and This Survey

2.1 The Field of Algorithmic Mechanism Design

By now, the field called Algorithmic Mechanism Design is quite large and varied. Moreover, the dividing lines between it and "classical" economic fields of Mechanism Design, Market Design, and Auction Design are very blurry. Some dividing line can be put according to whether complexity and algorithms play a crucial role in the problem. We can say that a problem is part of Algorithmic Mechanism Design whenever we have an economic design challenge where a central essence of the problem is the multitude of "input pieces" or "outcome possibilities". In other words, whenever some combinatorial complexity is a key issue interacting with incentive issues. Thus, for example, while auctioning a single item is the paradigmatic problem of Mechanism Design, auctioning *multiple* (related) items would be the paradigmatic problem of Algorithmic Mechanism Design. Market or auction design problems whose domain of application is computerized (e.g. ad auctions) most usually fall into this category and considered to be part of Algorithmic Mechanism Design. Finally, as the theory related to such scenarios has developed, some "approaches" to Mechanism Design that are characteristic to Computer Scientists have emerged, in particular a preference to "worst-case" analysis and a willingness to accept approximate solutions.

This survey will not attempt to give a general overview of the field of Mechanism Design, for which surveys can be found in textbooks such as Mas-Collel et al. [1995] or Osborne and Rubistein [1994] or in the more comprehensive books Krishna [2002] and

Milgrom [2004]. This survey will not even attempt covering the whole field of Algorithmic Mechanism Design, a nearly impossible task due to its size, ill-defined borders and rapid pace of change. Instead this survey will focus on the single key issue driving the field: *the clash between algorithmic considerations and incentive constraints.* To this effect, we will be limiting ourselves to the following choices:

1. **Private Values:** We will only be talking about scenarios where each participant has full information about his own value for each possible outcome. In contrast, a significant amount of classical Mechanism Design (see Krishna [2002], Milgrom [2004]) as well as some recent Algorithmic Mechanism Design (e.g. Roughgarden and Talgam-Cohen [2013]) considers situations where bidders have only partial knowledge about their values.

2. **Quasi-linear Utilities:** We will assume that bidders' values can be measured in monetary units (money) that can be transferred to and from the auctioneer. This assumption of quasi-linearity is quite common in general mechanism design, although a very large body of work on matching markets does not require this assumption, as do some other works in "Algorithmic Mechanism Design without Money" (see e.g. Schummer and Vohra [2007] and Procaccia and Tennenholtz [2009]).

3. **Risk Neutrality:** We will assume that all participants are risk neutral. There is of course much work in Mechanism Design that concerns effects of risk aversion.

4. **Focus on Efficiency:** The two central goals of both Mechanism Design and Algorithmic Mechanism Design are revenue and efficiency (social welfare). In this survey we will completely ignore all revenue issues and exclusively consider the goal of efficiency. See Hartline and Karlin [2007] and Hartline [2012] for surveys of work in Algorithmic Mechanism Design that is focused on Revenue.

5. **Dominant Strategy Analysis:** The standard way of modeling lack of information in economics (and most scientific fields) is using a Bayesian prior, and looking at probabilities and expected values over the possible "states of the world". In Computer Science, however, there is a general mistrust in our ability to correctly capture real world distributions, and the standard approach is to use worst-case analysis. While being more limited, worst case analysis is more robust as well as usually technically simpler. In our context, the worst-case point of view favored by Computer Scientists translates to a focus on dominant strategy equilibria that

makes no distributional assumptions[1]. While much of the classical work in Mechanism Design is in the Bayesian setting, this survey will stick to the simpler dominant strategy analysis. For a survey of the growing body of recent work in Algorithmic Mechanism Design that uses Bayesian notions or intermediate ones, see Hartline [2012].

All these limiting choices are made as to allow us to focus on the central issue in the field of Algorithmic Mechanism Design, the very basic clash between incentives and computation. We should perhaps say explicitly that we consider the algorithmic issues to include many facets of the complexity of implementation like communication or description costs.

2.2 Our Example: Multi-unit Auctions

Even after we make all the restrictions specified above, and after deciding to focus on the clash between incentives and computation, we are still left with quite a large body of work which is hard to survey in an organized manner. The choice we have made is to take a single representative problem and demonstrate all the basic issues just on it. The problem we have chosen for this is "Multi-unit Auctions" – an auction of a multiple homogenous goods. Beyond its importance as of itself, it allows us to go over many of the common themes in Algorithmic Mechanism Design, covering issues that arise when addressing many other problems such as the paradigmatic *combinatorial auctions* covered from various aspects in Cramton et al. [2006], Milgrom [2004], Blumrosen and Nisan [2007], or de Vries and Vohra [2003]. There are two main reasons for the choice of this specific problem. The first is its relative combinatorial simplicity: almost all the issues that we deal with are technically simpler in the case of multi-unit auctions than they are in most other problems, and thus allow us to focus in the cleanest way on the basic concepts. The second reason is personal: the author has simply worked much on this problem and thus finds it easier to demonstrate general concepts with it. The reader should be warned however that this problem is not the most striking one in practical terms: the computational issues become a real world bottleneck only for very large numbers of items; this is in contrast to other applications such as combinatorial auctions for which the computational bottlenecks are biting starting with the modest numbers of items that occur in most practical settings.

So here is the multi-unit auction problem:

[1]Or, more generally, ex-post Nash equilibria, which in independent private-value settings is essentially equivalent to dominant strategy equilibria (see Nisan [2007]).

- We have m identical indivisible units of a single good to be allocated among n strategic players also called *bidders*.
- Each bidder i has a privately known *valuation function* $v_i : \{0, ..., m\} \to \Re^+$, where $v_i(k)$ is the value that this bidder has for receiving k units of the good. We assume that $v_i(0) = 0$ and free disposal: $v_i(k) \le v_i(k+1)$ (for all $0 \le k < m$).
- Our goal (as the auctioneer) is to allocate the units among the bidders in a way that optimizes *social welfare*: each bidder i gets m_i units and we aim to maximize $\sum_i v_i(m_i)$, where the feasibility constraint is that $\sum_i m_i \le m$.

Our goal is to develop a *computationally-efficient mechanism* for this problem. *Efficiency* in the computational sense here should imply a protocol that can be realistically implemented even for large numbers of bidders and goods. There may be different quantitative notions of computational-efficiency, but we will see that (for our problem as well as many others) the answer will be quite robust to the exact choice. A *Mechanism* here means a protocol that ensures that the required outcome is achieved *when the bidders act rationally.* Again, as mentioned in the problem description, the required outcome is the socially-efficient allocation of resources.

There are basically three types of difficulties we need to overcome here:

1. **Representation:** The representation of the "input" to this problem, i.e. the vector of bidders' valuations, contains m numbers. We are interested in large values of m, where communicating or handling such a large amount of data is infeasible. We will thus need to cleverly find more succinct representations.
2. **Algorithmic:** Even for the simplest types of valuations, it turns outs that just computing the optimal allocation is computationally intractable. This will lead us to considering also relaxations of our economic efficiency goals, settling for *approximate* optimization.
3. **Strategic:** In order to achieve our social efficiency goal, the auctioneer would need to know the valuations of the bidders. However, the bidders' valuations are *private information.* So how can the auctioneer access them?

Combinatorial Auctions and Beyond

As we present the issues for our problem of multi-unit auctions, we will also shortly point out the corresponding issues for combinatorial auctions. In a *combinatorial auction* we are selling m *heterogenous* items to n bidders. The valuation function of each bidder thus needs to assign a value for each *subset* of the items and so is captured by $2^m - 1$

numbers $v_i(S)$ for every non empty set S of the items (we assume $v(\emptyset) = 0$.) Free disposal is usually assumed, i.e. $S \subseteq T$ implies $v_i(S) \leq v_i(T)$. The first goal considered is, as in the multi-unit auction case, the socially-efficient allocation of resources. These same three issues are the ones that arise in the treatment of combinatorial auctions. As we go along we will point out some early or central references, but for more information on these issues in the context of combinatorial auctions, the reader is referred to the surveys Cramton et al. [2006] or Blumrosen and Nisan [2007]. We will sometimes also very shortly mention the situation for other algorithmic mechanism design problems.

2.3 Where are we Going?

Before taking the journey into the analysis of our problem, let us first take a high-level view of the map of this coming journey.

In the first part of the journey – sections 3 and 4 – we will just be laying the foundations for our representational and algorithmic treatment. These are far from trivial due to the large "input" size. We start this part of the journey by describing the two basic approaches to handling such "large inputs": via *bidding languages* and via *queries*, the most general of which are usefully viewed in terms of *communication.* We then continue by looking at algorithms for the problem, starting with one that it is not computationally-efficient. While for *convex settings* a computationally-efficient algorithm can be designed, we demonstrate that this is not possible in general. We present two different *intractability proofs* of this last fact, that correspond to the choices made in input representation. At this point we reach the culmination of the algorithmic part of the trip, a computationally-efficient algorithm for *approximating* the optimal allocation arbitrarily well.

Up to this point we have not really touched any strategic or incentive issues, and this is what we do in the second part of the journey – section 5 – where we study the introduction of *incentives* that will motivate rational bidders to interact with the algorithms of the first part in a way that will lead to the desired results. We start by presenting the classic *VCG mechanisms* which would solve the problem, had there not been any computational concerns. We then show how this solution clashes with the computational constraints, and study in detail the subclass of *Maximum-in-Range Mechanisms* where no such clash exists. For the special *single parameter* case we present a good non-VCG mechanism, and then discuss the difficulties in obtaining such for general *multi parameter* settings. We conclude by demonstrating that *randomization* in Mechanisms can help.

3 Representation

Any process that solves a problem, must certainly get access to the data specifying the problem requirements. In our case this is to the valuation functions of the n players. Now, each valuation function v is specified by m numbers, $v(1), v(2), ..., v(m)$. In some cases, where m is of moderate size, these numbers can just be listed, and we have a full representation of the problem (consisting of mn real numbers), but in other cases we may want to consider huge values of m, in which case a full listing of m numbers is infeasible. Economic examples of this flavor may include auctions of bonds or other financial instruments, where often billions of identical units are sold. Examples in computerized systems may include auctions for communication bandwidth in a computer network where we may imagine selling the billions of communication packets over a communication link in a single second or selling millions of online ad slots of a certain type. This case, of a large value of m, is the one that we deal with here. Alternatively, and essentially equivalently, one may wish to model a huge number of identical items as a single infinitely divisible good, in which case m would just correspond to the precision that we can handle of specifying the fractions of the good.

Before we continue, let us get one hurdle out of the way: the representation of real numbers. In principle we would need infinite precision to represent even a single real number. From such a point of view we can never implement anything dealing with real numbers over any computer system. In practice this is never an issue: the standard 64-bit precision in a standard personal computer is more than enough for representing the world GDP in micro-penny precision. The usual formal treatment in Computer Science is to allow finite representations in the form of rational numbers and count the number of bits in the representation as part of the input size. Almost equivalently, and as we do here, is to ignore the issue and assume that real numbers are represented directly at the required precision. Usually, and as is the case here, this merely modifies all complexity calculations by a factor that is the number of bits of precision that we require, a modification that does not matter significantly.

So how can we succinctly represent a valuation when m is too large? The simple answer is that we can't: in an information-theoretic sense the valuation cannot be compressed to less than m numbers.[2] However, there are two useful approaches for such a representation: the first one is by means of fixing a "bidding language" – a formalism for succinctly representing "useful" types of valuations, and the second is by means of

[2]One may always encode an arbitrary finite number of real numbers by a single real number since $\Re^m$ and $\Re$ have the same cardinality. This however completely breaks down in any finite representation. When working formally in a model of direct representation of real numbers, some technical dimension-preserving condition is taken to rule out such unrealistic encodings. See e.g. Nisan and Segal [2006].

a "queries" that specify the type of interactions possible with the valuation. We will describe these two approaches below.

3.1 Bidding Languages

The first approach for specifying valuations is to fix some syntactic formalism for doing so. Such a formalism is a "language" where we need to specify the syntax as well as the semantics, the latter being a mapping from strings of characters to valuations. Specifying languages is a standard concept in Computer Science, and the usual tradeoff is between the dual goals of expressiveness and simplicity. On one hand, we would like to be able to *succinctly* express as many as possible "interesting" types of valuations, and on the other hand we want to make sure that it is "easy" to handle valuations expressed in such a language. Here "ease" may have multiple meanings, both regarding conceptual ease for humans and regarding technical ease for computers. Here are three common bidding languages, in increasing order of expressiveness:

1. **Single Minded Bids:** This language allows only representing "step functions", valuations of the form $v(k) = 0$ for $k < k^*$ and $v(k) = w^*$ for $k \geq k^*$. Clearly to represent such a valuation we only need to specify two numbers: k^* and w^*. Clearly, also, this is a very limited class of valuations.

2. **Step Functions:** This language allows specifying an arbitrary sequence of pairs $(k_1, w_1), (k_2, w_2), \ldots, (k_t, w_t)$ with $0 < k_1 < k_2 \cdots < k_t$ and $0 < w_1 < w_2 < \cdots < w_t$. In this formalism $v(k) = w_j$ for the maximum j such that $k \geq k_j$. Thus, for example, the bid $((2,7),(5,23))$ would give a value of \$0 to 0 or 1 items, a value of \$7 to 2, 3, or 4 items, and a value of \$23 to 5 or more items. Every valuation can be represented this way, but most valuations will take length m. Simple ones – ones that have only a few "steps" – will be succinctly represented.

3. **Piece-Wise Linear:** This language allows specifying a sequence of marginal values rather than of values. Specifically, a sequence $(k_1, p_1), (k_2, p_2), \ldots, (k_t, p_t)$ with $0 < k_1 < k_2 \cdots < k_t$ and $p_j \geq 0$ for all j. In this formalism p_j is the marginal value of item k for $k_j \leq k < k_{j+1}$. Thus $v(k) = \sum_{l=1}^{k} u_l$ with $u_l = p_j$ for the largest j such that $l \geq k_j$. In this representation, the bid $((2,7),(5,23))$ would give a value of \$7 to 1 item, \$14 to 2 items, \$37 to 3 items, \$60 to 4 items, and \$83 to 5 or more items.

Languages can often be compared to each other in terms of their expressiveness. It is clear that the Step Function language is more expressive than the Single Minded Bid language since it contains the latter as a special case but may represent a larger class of valuations. However, we may also say that the Piece-wise Linear language is more expressive than the Step Function language: every valuation expressed as a Step Function $((k_1, w_1)...(k_t, w_t))$ can be expressed also in the Piece-sise linear form by converting each step (k_j, w_j) into two marginal values: $(k_j, w_j - w_{j-1})$ and $(k_j + 1, 0)$. This only mildly increases the size of expressing the valuation (by a factor of 2), but a similar mild increase would not be possible had we wanted the opposite emulation. The Piece-wise Linear bid $(m, 1)$ which represents the valuation $v(k) = k$ may be represented as a Step Function but only by listing all possible m steps (k, k), blowing up the size of the representation by an unacceptable factor of m.

The choice of a language is somewhat arbitrary and there are various reasonable ways to strike the expressiveness vs. simplicity tradeoff, depending on the application. For maximum bite, we would like to prove positive results for the strongest language that we can handle, while proving negative results for the weakest language for which they still hold. This will be the case for the results we show in this chapter.

Of course, one may easily think of even more expressive (and less simple) languages such as allowing Piece-wise quadratic functions, or arbitrary expressions, etc. The extreme expressive language in this sense is to allow the expression of an arbitrary computation. To be concrete, one may take a general purpose programming language and allow expressing a valuation v by a program in this programming language. While it would seem that it would not be easy to manipulate valuations presented in such general form, we will see below that even this is not hopeless, and in fact there is much we can do even with such "black box" access that provides the value $v(k)$ for any given "query" k.

Combinatorial Auctions

Taking the wider perspective of combinatorial auctions, there is very large space of possible bidding languages, striking various balances between simplicity and power. The simplest "Single Minded Bid" language, allows the specification of a value for a single target subset of the items. More complicated variants allow various ways of combining such simple bids. A survey of bidding languages for combinatorial auctions appears in Cramton et al. [2006], Chapter 9.

3.2 Query Access to the Valuations

A more abstract way of representing the input valuations is by treating them as opaque "black boxes". In this approach we do not concern ourselves with how the valuations are represented but rather only with the interface they provide to the algorithm or mechanism. This has the advantage of leaving significant freedom in the actual representation of the valuation, decoupling the algorithm design from the representation issue. Of course, the choice of interface – the set of "queries" that the black box needs to answer – is important here. If this interface is too weak then we will have a hard time constructing computationally-efficient algorithms using this weak set of queries while if the set of allowed queries is too strong then we will have difficulties in implementing them with realistic representations. Never the less, even un-realistically strong sets of queries are interesting for proving "lower bounds" – limits on what computationally-efficient algorithms or mechanisms may achieve: any limitation of mechanisms or algorithms that are allowed to use even strong unrealistic queries will certainly apply also to mechanisms that only use more realistic ones. In general, one usually attempts using the strongest possible set of queries when proving "lower bounds" and attempts using the weakest possible set of queries when proving "upper bounds" – actual algorithms or mechanisms. This is what we will also do here, use the realistic and weak "value queries" for upper bounds and use the unrealistically strong and general "communication queries" for lower bounds.

3.2.1 Value Queries

A *value query* to a valuation $v : \{0, \ldots, m\} \rightarrow \Re_+$ gets a number of items k and simply returns the value $v(k)$.

The requirement that a certain bidding language be able to answer value queries essentially means that it has an effective implementation. I.e. that there is a computationally-efficient algorithm that takes as input is a valuation represented in the language as well as a query k and produces as output the value $v(k)$. That the algorithm is computationally-efficient means that it runs in time that is polynomial in its input size, i.e. in the *representation length*. Note that all the bidding languages suggested in the previous section can indeed answer such queries easily.

Take for instance the strongest one suggested above, the Piece-wise Linear bidding language. Given a valuation v represented in this language as $((k_1, p_1), (k_2, p_2), \ldots, (k_t, p_t))$ and a query k, how can we efficiently compute the value of $v(k)$? Directly applying the definition of Piece-wise Linear valuations requires summing the k marginal values, an operation whose running time may be significantly larger than the input size that is proportional to t (rather than k which may be as big as the number of items m). However, a quick inspection will reveal that one can replace the repeated addition of equal marginal values by multiplication, calculating $v(k) = \sum_{j=1}^{l-1} k_j p_j + (k - \sum_{j=1}^{l-1} k_j) \cdot p_l$, where l is the largest integer such that $\sum_{j=1}^{l-1} k_j \leq k$. This now takes time that is polynomial in the input size, as required.

3.2.2 General Communication Queries

The strongest imaginable set of queries to a valuation is to simply allow all possible queries. This means that we could allow all functions on the set of valuations.

Even in this general setting two key restrictions remain: the first is that a query may only address a single valuation function and may not combine several. Thus for example a query "find the odd k that maximized $v_i(k) - k^2$" would be allowed, but "find the k that maximizes $v_1(k) + v_2(m - k)$" can not be a single query since it combines two separate valuations. The second is that the reply from a query needs to be small, we shouldn't for example be able to ask for an extremely long string that encodes the whole valuation. To be completely precise we should require that the reply to a query is always a single bit (i.e. a yes/no answer), but equivalently we can allow arbitrary answers, counting the number of answer bits instead of the number of queries.

When thinking about an algorithm or mechanism that makes such general queries to the valuations, it is best to think about it in the completely equivalent model of *communication*. In such a model the algorithm is really a protocol for communication between the valuation black boxes which we now think of as the bidders themselves. An algorithmic query to a valuation is interpreted as the protocol specifying that the appropriate bidder communicate (to the algorithm) the answer to the query – an answer that obviously depends on the valuation that the bidder has, but not on any other information (beyond the state of the protocol). Under this interpretation, the total number of queries that an algorithm makes is exactly the total number of communicated "numbers" made by the protocol.

Combinatorial Auctions

For combinatorial auctions, there are a variety of natural specific query models, the simplest of which is that of value queries, and the strongest of which is the communication model. Another very natural query model is that of "demand queries", where a query specifies a price vector $(p_1, ..., p_m)$ and the response is the demanded set of items under these prices, i.e. a set S that maximizes the utility $v(S) - \sum_{i \in S} p_i$. A systematic discussion of query models for combinatorial auctions appears in Blumrosen and Nisan [2010], while a discussion of communication appears in Nisan and Segal [2006].

4 Algorithms

By now we have only discussed how our "input" may be represented or accessed by the algorithm or mechanism. At this point we can start discussing the algorithmic challenge of optimally allocating the items:

The Allocation Problem

Input: The sequence of valuations of the players: $v_1, \ldots, v_n$.
Output: An allocation $m_1, \ldots, m_n$ with $\sum_{i=1}^{n} m_i \leq m$ that maximizes $\sum_{i=1}^{n} v_i(m_i)$.

Combinatorial Auctions

For the case of combinatorial auctions, the valuations v_i provide a value for each subset of the items, and the required output is an allocation $(S_1 ... S_n)$ that maximizes $\sum_i v_i(S_i)$, where the constraint is that $S_i \cap S_j = \emptyset$ for $i \neq j$.

4.1 Algorithmic Efficiency

Our basic requirement from the algorithm is that it be "computationally-efficient". Specifically, for every input we would want the algorithm to provide an output "quickly". As usual, the running time is a function of the input size parameters n and m. Also as usual, the running time may depend on the precise definition of the model of computation, but this dependence is very minor, and the running time will be almost the same under all reasonable definitions. Thus, as usual, we will just formally require that the running time is polynomial in the *input size*. In all cases in this survey a polynomial running time will actually have reasonably small degrees and small hidden constants while a non-polynomial running time will actually be exponential.

In case that we are dealing with the input presented in a certain bidding language the interpretation of this is the completely standard one: we want an algorithm whose running time is polynomial in the sum of lengths of the representations of the input valuations. In the case where the input valuations are given by query access to "black boxes" there is no explicit representation of the input, so what we will require is that the running time is polynomial in the numbers of bidders, n, and the number of bits required to represent a basic quantity in the system: $\log m$ bits to represent a number of items, and the maximum number of bits of precision of a value, denoted by t.

Let us reflect further on why we do not view a running time that is polynomial in m as computationally-efficient. First, consider that the black box query model is intended to give an abstract way of considering a wide class of representations. For none of these representations do we envision its size to be "trivial" – of an order of magnitude similar to m – but rather we expect it to be significantly smaller, and thus a run time that is polynomial in m will not be polynomial in the input size under this representation. Second, consider the query model as an abstraction of some protocol between the players. A run time that is polynomial in m allows sending complete valuations, completely loosing the point of the abstraction. Third, it is not uncommon to have multi-unit auctions of millions or billions of items (e.g. treasury bonds or web ad impressions). While all of the algorithms and mechanisms presented below that have a run time that is polynomial in $\log m$ and n can be practically applied in these cases, algorithms whose run time is polynomial in m will probably be too slow.

Let us start with a *non-computationally-efficient* but still not-trivial dynamic programming[3] algorithm for solving this problem. We will present the algorithm in the value-queries model. Presenting it in this weak model ensures that it applies also to all the other models presented above. The basic idea is to use dynamic programming to fill up a table that for any $0 \leq i \leq n$ and $0 \leq k \leq m$ holds the optimal total value that can be achieved by allocating k items among the first i bidders. An entry in this table can be filled using the values of previously filled entries, and this information suffices for actually reconstructing the optimal allocation itself.

```
General Allocation Algorithm

  1. Fill the $(n+1)*(m+1)$ table $s$, where $s(i,k)$ is the maximum value achievable
     by allocating $k$ items among the first $i$ bidders:
       (a) For all $0 \leq k \leq m$: $s(0,k) = 0$
       (b) For all $0 \leq i \leq n$: $s(i,0) = 0$
       (c) For $0 < i \leq n$ and $0 < k \leq m$: $s(i,k) = max_{0 \leq j \leq k}[v_i(j) + s(i-1,k-j)]$
  2. The total value of the optimal allocation is now stored in $s(n,m)$
  3. To calculate the $m_i$'s themselves, start with $k = m$ and for $i = n$ down to 1 do:
       (a) Let $m_i$ be the value of $j$ that achieved $s(i,k) = v_i(j) + s(i-1,k-j)$
       (b) $k = k - m_i$
```

The running time of this algorithm is $O(nm^2)$: the calculation of each of the mn cells in the table s requires going over at most m different possibilities. Note that while the dependence of the running time on n is acceptable, the dependence on m is not so since we required a polynomial dependence on $\log m$. Thus, we do not consider this algorithm to be computationally-efficient and indeed it would likely not be practical when handling even as few as a million items. We have thus proved:

[3]Dynamic Programming is a common algorithmic paradigm that builds up a table of partial results, each of which is computed using previous ones.

Theorem 1 *There exists an algorithm for the multi-unit allocation problem (in any of the models we considered) whose running time is polynomial in n and m (rather than in $\log m$).*

Combinatorial Auctions

In the case of combinatorial auctions, the valuations hold information that is exponential in m. In this case computationally-efficient algorithms are allowed a running time that is polynomial in both n and m, rather than grow exponentially in m. Mirroring the case for multi-unit auctions, it is not difficult to come up with a dynamic programming algorithm for combinatorial auctions whose running time is polynomial in n and 2^m.

4.2 Downward Sloping Valuations

At this point, it would perhaps be a good idea to see an example of a computationally-efficient algorithm. We can provide such an algorithm for the special case of "downward sloping valuations", those that exhibit diminishing marginal values, i.e. that satisfy $v_i(k+1) - v_i(k) \leq v_i(k) - v_i(k-1)$ for all bidders i and number of items $1 \leq k \leq m-1$. This is the case of a "convex economy", a case which turns out to be algorithmically easy. A simple intuitive algorithm will allocate the m items one after another in a greedy fashion, at each point giving it to the player that has highest marginal value for it (given what it was already allocated). As stated, this algorithm takes time that increases linearly in m rather than in $\log m$ as desired, but its simple greedy nature allows us to use binary search[4] to convert it to a computationally-efficient one.

The idea here is that with downward sloping valuations we have a market equilibrium: a clearing price p, and an allocation $(m_1, ..., m_n)$ with $\sum m_i = m$ that has the property that for each i, m_i is i's demand under price p, i.e. that $v_i(m_i) - v_i(m_{i-1}) \geq p > v_i(m_{i+1}) - v_i(m_i)$ which implies that $v_i(m_i) - m_i p \geq v_i(k) - kp$ for all k. *The First Welfare Theorem* in this context states that such a market equilibrium must maximize $\sum_i v_i(m_i)$, since for any other allocation $(k_1, ..., k_n)$ with $\sum k_i \leq m$ we have that $\sum_i v_i(m_i) - mp = \sum_i (v_i(m_i) - m_i p) \geq \sum_i (v_i(k_i) - k_i p) \geq \sum_i v_i(k_i) - mp$.

Algorithmically, given a potential price p, we can calculate players' demands using binary search to find the point where the marginal value decreases below p. This allows us to calculate the total demand for a price p, determining whether it is too low or too high, and thus search for the right price p using binary search. For clarity of exposition we will assume below that all values of items are distinct, $v_i(k) \neq v_{i'}(k')$ whenever $(i,k) \neq (i',k')$.

[4]Binary Search is a common algorithmic paradigm used to find a target value within a given range by repeatedly splitting into two sub-ranges and determining which one contains the desired target. The key point is that this process requires only time that is only logarithmic in the range size.

(This is without loss of generality since we can always add values $\epsilon_{i,k}$ to $v_i(k)$ to achieve this.) This assumption also makes it evident that a clearing price p exists: for the proof imagine starting with price $p = 0$ and then increasing p gradually. Since the total demand starts high and decreases by a single unit at each price-point $v_i(k)$, at some point it will reach exactly m units and the market will clear.

Allocation Algorithm for Downward Sloping Valuations

1. Using binary search, find a clearing price p in the range $[0, V]$, where $V = max_i[v_i(1)]$:
   (a) For each $1 \leq i \leq n$, use binary search over the range $\{0, 1, ..., m\}$, to find m_i such that $v_i(m_i) - v_i(m_{i-1}) \geq p > v_i(m_{i+1}) - v_i(m_i)$
   (b) If $\sum_i m_i > m$ then p is too low; if $\sum_i m_i < m$ then p is too high, otherwise we have found the right p.
2. The optimal allocation is given by $(m_1, ..., m_n)$.

The important point is that this algorithm is computationally-efficient: the binary search for p requires only t rounds, where t is the number of bits of precision used to represent a value. Similarly the binary search for m_i requires only $\log m$ rounds. We have thus shown:

Theorem 2 *There exists an algorithm for the multi-unit allocation problem among downward sloping valuations (in any of the models we considered) whose running time is polynomial in n, $\log m$, and the number of bits of precision.*

Combinatorial Auctions

In the case of combinatorial auctions, there is a corresponding class of "convex" valuations, called "substitutes" valuations. For this class, equilibrium prices are known to exists (Gul and Stacchetti [1999]) and the optimal allocation among such valuations can be found computationally-efficiently (Gul and Stacchetti [2000], Blumrosen and Nisan [2010]).

4.3 Intractability

Now the natural question is whether one can get a computationally-efficient algorithm even for the general case, not just the downward sloping one. It turns out that it is quite easy to show that this is impossible in the valuequeries model.

Consider such a hypothetical algorithm even just for the special case of two players. For any two valuations v_1 and v_2 it outputs the optimal allocation (m_1, m_2). Now let us follow this algorithm when it accepts as input the two linear valuations $v_1(k) = k$ and $v_2(k) = k$ for all k. The algorithm keeps querying the two valuations at various values and for each query k gets as a response the value k. After some limited amount of queries it must produce an answer (m_1, m_2). Now the critical thing is that if the algorithm was computationally-efficient and thus in particular made less than $2m - 2$ queries then it had to produce the answer without querying some value (even excluding m_1, m_2 which it may or may not have queried). Without loss of generality let us say that it never queried $v_1(z)$ (with $z \notin \{m_1, m_2\}$). Now we may consider a different input where we increase the value of $v_1(z)$ by one, i.e. $v'_1(k) = k$ except for $v'_1(z) = z + 1$. Note that this v'_1 is still a proper valuation, as monotonicity was maintained. Note also that now the unique optimal allocation is $(z, m - z)$ which gives value $v'_1(z) + v_2(m - z) = m + 1$, while any other allocation only gives a total value of m. Our poor algorithm, however, has no way to know that we are now in the case of (v'_1, v_2) rather than (v_1, v_2) since the only difference between the two cases is for the query z, a query which it did not make, so it will provide the same answer (m_1, m_2) as before, an answer which is not optimal. We have thus shown:

Theorem 3 *Every algorithm for the multi-unit allocation problem, in the value-queries model, needs to make at least $2m - 2$ queries for some input.*

This argument concludes the impossibility result for the weakest black-box model, the one that only allows value queries. It is already quite interesting, exhibiting a significant computational gap between the "convex economy" case where a computationally-efficient algorithm exists and the general case for which it does not. It turns out that the same impossibility result is true even in stronger (and more interesting) models. We now bring two different intractability results, one for the case where the valuations are given in a bidding language – even the weakest possible such language, and the other one for query models – even for the strongest one that allows communication queries.

4.3.1 NP-completeness

Let us consider the simplest possible bidding language, that of Single Minded Bids. In this language each valuation v_i is represented as (k_i, p_i) with the meaning that for $k < k^*$ we have $v_i(k) = 0$ and for $k \geq k_i$ we have $v_i(k) = p_i$. The optimal allocation will thus need to choose some subset of the players $S \subseteq \{1 \ldots n\}$ which it would satisfy. This would give total value $\sum_{i \in S} p_i$ with the constraint that $\sum_{i \in S} k_i \leq m$. This optimization problem is a very well known problem called the knapsack problem, and it is one of a family of problems known to be "NP-complete". In particular, no computationally-efficient algorithm exists for solving the knapsack problem unless "P=NP", which is generally conjectured not to be the case (the standard reference is still Garey and Johnson [1979]). It follows that no computationally-efficient algorithm exists for our problem when the input valuations are presented in any of the bidding languages mentioned above all of which contain the single-minded one as special cases. We thus have:

Theorem 4 *Assuming $P \neq NP$, there is no polynomial-time algorithm for the multi-unit allocation problem, in any of the bidding language models.*

We should mention that this intractability result does rely on the unproven assumption of $P \neq NP$. On the other hand the allocation problem – for any effective bidding language – is clearly an optimization variant of an NP problem. Thus, if it turns out that $P = NP$ then a computationally-efficient allocation algorithm does exists.

4.3.2 Communication Complexity

When the valuations are given in any effective bidding language, our allocation problem is a "normal" computational problem and we cannot hope for unconditional answers as our problem is equivalent to the $P = NP$ problem. However, when the input to our problem is given as black boxes, we are really looking for algorithms whose running time should be sub-linear in "true" input size, and for this we have techniques for proving unconditional impossibility results. In particular we have already seen that there is no sub-linear algorithm that uses only value queries to access the input valuations. Here we extend this result and show that it holds even when allowing arbitrary queries to each of the input valuations, i.e. in the communication model. The "proper way" of proving this is by using the well known results from the theory of communication complexity (surveyed in Kushilevitz and Nisan [1997]) and deducing from them impossibilities in this situation (as was done in Nisan and Segal [2006]). Here we give, however, a self-contained treatment that uses the simplest technique from the field of communication complexity ("fooling sets") directly on our problem.

Let us consider the following set of valuations: for every set $S \subseteq \{1, \dots m-1,\}$ let $v_S(k) = k + 1_{k \in S}$, i.e. for $k \notin S$ we have $v_S(k) = k$ while for $k \in S$ we get one more, $v_S(k) = k+1$. Now let us consider a two-player situation when player 1 gets v_S and player 2 gets the "dual" valuation v_{S^*}, where $S^* = \{0 < k < m | m-k \notin S\}$. The dual valuation was defined so that any allocation between v_S and v_{S^*} would have only value exactly $v_S(k) + v_{S^*}(m-k) = m$. Now there are 2^{m-1} possible pairs of input valuations (v_S, v_{S^*}) that we are considering, and the main point of the proof will show that for no two of these pairs can the algorithm see exactly the same sequence of answers from the two black-boxes (i.e., in terms of communication, exactly the same sequence of bits communicated between the two parties). It follows that the total number of possible sequences of answers is at least 2^{m-1} and thus at least one of them requires at least $m-1$ bits to write down. As each answer to a query is "short" – say at most t bits, for some small t – the total number of queries made by the algorithm must be at least $(m-1)/t$ for at least one of these input pairs (in fact, for most of them), and this is a lower bound on the number of queries made by the algorithm (communication protocol) and thus on its running time.

We are left with showing that indeed for no two pairs of valuations (v_S, v_{S^*}) and (v_T, v_{T^*}) with $S \neq T$ the algorithm cannot receive the same sequence of answers. The main observation is that, had this been the case, then the algorithm would also see the same sequence of answers on the "mixed' pairs (v_S, v_{T^*}) and (v_T, v_{S^*}). Consider the algorithm when running on the input pair (v_S, v_{T^*}). It starts by making some query, a query which is also the first query it made on both (v_S, v_{S^*}) and (v_T, v_{T^*}). If it queries the first player then it gets the same answer as it did on input (v_S, v_{S^*}) (since it has the same valuation for the first player) while if it queries the second player then it gets the same answer as in (v_T, v_{T^*}). In both cases it gets the same information as in the original cases (v_S, v_{S^*}) and (v_T, v_{T^*}) and thus continues to make the next query exactly as it has made the next query in those cases. Again, it must get the same answer for this new query, and as it continues getting the same answers and thus making the same queries as in the original cases, it never observes a deviation from these two cases and thus ends up with the same final outcome as it did in the original cases. The same is true for the other mixed pair (v_T, v_{S^*}) for which it gives again the same final outcome. This however contradicts the optimality of the algorithm since an outcome allocation that is optimal for (v_S, v_{T^*}) is never optimal for (v_T, v_{S^*}). To see this fact just note that since $S \neq T$ there exists some k in the symmetric difference, say, wlog, $k \in S - T$. In this case notice that $v_S(k) + v_{T^*}(m-k) = m+1$ and thus the optimal allocation would give at least (actually, exactly) this total value. However any k which gives an optimal allocation for (v_S, v_{T^*}), i.e., $v_S(k) + v_{T^*}(m-k) = m+1$ would give a sub-optimal allocation for (v_T, v_{S^*}), since $v_S(k) + v_{S^*}(m-k) + v_T(k) + v_{T^*}(m-k) = 2m$, and thus we would have $v_T(k) + v_{S^*}(m-k) = m-1$. We have thus shown:

Theorem 5 *(Nisan and Segal [2006]) Every algorithm for the multi-unit allocation problem, in any query model, needs to ask queries whose total answer length is at least $m-1$ in the worst case.*

The argument presented above also holds in the stronger model where the algorithm is given "for free" the optimal allocation and only needs to *verify* it, a model known as "non-deterministic protocols". This can be viewed as a lower bound on the "number of prices" needed to support an equilibrium. Informally, and taking a very broad generalization, a generalized equilibrium specifies some "generalized pricing information" for a market situation, information that allows each player to *independently of the other players* decide on the set of goods that is allocated to him. The above argument then actually provides a lower bound on the number "generalized prices" required in any general notion of equilibrium. See Nisan and Segal [2006] for more details.

Combinatorial Auctions

Going back again to combinatorial auctions, similar NP-hardness results are known for bidding language models (Sandholm [1999]) as well as query and communication models (Nisan and Segal [2006]), showing that a running time that is polynomial in m is not possible.

4.4 Approximation

At this point we seem to have very strong and general impossibility results for algorithms attempting to find the optimal allocation. However, it turns out that if we just slightly relax the optimality requirement and settle for "approximate optimality" then we are in much better shape. Specifically, an *approximation scheme* for an optimization problem is an algorithm that receives, in addition to the input of the optimization problem, a parameter ϵ which specifies how close to optimal do we want to solution to be. For a maximization optimization problem (like our allocation problem) this would mean a solution whose value is at least $1 - \epsilon$ times the value of the optimal solution. It turns out that this is possible to do, computationally-efficiently, for our problem using dynamic programming.

The basic idea is that there exists an optimal algorithm whose running time is polynomial in the *range of values*. (This is a different algorithm than the one presented in section 4.1 whose running time was polynomial in the number of items m.) Our approximation algorithm will thus truncate all values $v_i(k)$ so that they become small integer multiples of some δ and run this optimal algorithm on the truncated values. Choosing $\delta = \epsilon V/n$ where $V = max_i[v_i(m)]$ strikes the following balance: on one hand, all values are truncated to an integer multiple $w\delta$ for an integer $0 \leq w \leq n/\epsilon$, and so the range of the total value is at most the sum of n such terms, i.e. at most n^2/ϵ. On the other hand, the total additive error that we make is at most $n\delta \leq \epsilon V$. Since V is certainly a lower bound on the total value of the optimal allocation this implies that the fraction of total value that we loose is at most $(\epsilon V)/V = \epsilon$ as required.

The allocation algorithm among the truncated valuations uses dynamic programming, and fills a $(n+1) * (W+1)$-size table K, where $W = n^2/\epsilon$. The entry $K(i, w)$ holds the minimum number of items that, when allocated optimally between the first i players, yields total value of at least $w\delta$. Each entry in the table can be computed efficiently from the previous ones, and once the table is filled we can recover the actual allocation.

```
Approximation Scheme

1. Fix $\delta = \epsilon V/n$ where $V = max_i[v_i(m)]$, and fix $W = n^2/\epsilon$.

2. Fill an $(n+1) * (W+1)$ table $K$, where $K(i,w)$ holds the minimum number
   of items that, when allocated optimally between the first $i$ players, yields total
   value of at least $w\delta$:

   (a) For all $0 \leq i \leq n$: $K(i,0) = 0$
   (b) For all $w = 1, \ldots, W$: $K(0,w) = \infty$
   (c) For all $i = 1, \ldots, n$ and $w = 1, \ldots, W$ compute $K(i,w) = min_{0 \leq j \leq w} val(j)$,
       for $val(j)$ defined by:
       i. Use binary search to find minimum number of items $k$ such that
          $v_i(k) \geq j\delta$
       ii. $val(j) = k + K(i-1, w-j)$

3. Let $w$ be the maximum index such that $K(n,w) \leq m$

4. For $i$ going from $n$ down to 1:

   (a) Let $j$ be so that $K(i,w) = val(j)$ (as computed above)
   (b) Let $m_i$ be the minimum value of $k$ such that $v_i(k) \geq j\delta$
   (c) Let w = w - j
```

Our algorithm's running time is dominated by the time required to fill the table K which is of size $n \times n^2/\epsilon$ where filling each entry in the table requires going over all possible n/ϵ values j and for each performing a binary search that takes $O(\log m)$ queries. Thus the total running time is polynomial in the input size as well as in the precision parameter ϵ which often called a fully polynomial time approximation scheme. For most realistic optimization purposes such an arbitrarily good approximation is essentially as good as an optimal solution. We have thus obtained:

Theorem 6 *There exists an algorithm for the approximate multi-unit allocation problem (in any of the models we considered) whose running time is polynomial in* n, $\log m$, *and*

ϵ^{-1}. *It produces an allocation* $(m_1, ..., m_n)$ *that satisfies* $\sum_i v_i(m_i) \geq (1-\epsilon)\sum_i v_i(opt_i)$, *where* $(opt_1, ..., opt_n)$ *is the optimal allocation.*

Combinatorial Auctions

When going to the wider class of combinatorial auctions, such a good approximation scheme is not possible. The best possible polynomial time approximations loose a factor of $O(\sqrt{m})$ and no better is possible. Significant work has been done on obtaining improved approximation factors for various sub-classes of valuations. See Blumrosen and Nisan [2007] for survey and references.

5 Payments, Incentives, and Mechanisms

Up to this point we have treated the bidders in the auction as doing no more than providing their inputs to the allocation algorithm. A realistic point of view would consider them as economic agents that act rationally to maximize their own profits. To formalize such a setting we would need to make assumption about what exactly is it that the players are aiming to maximize. The simplest, most natural, such model will having bidder i rationally maximize his net utility: $u_i = v_i(m_i) - p_i$ where m_i is the number of units he is allocated and p_i is the price that he is asked to pay for these items. Notice how we already went beyond the pure algorithmic question that only involved the allocation $(m_1...m_n)$ and went into an economic setting which explicitly deals with the payments $(p_1...p_n)$ for these items. The assumption that bidders make payments p_i and that bidders' utilities (which each aims to maximize) are the simple differences $u_i = v_i(m_i) - p_i$ (rather than some more complicated function of m_i and p_i) is often called "quasi-linear utilities" and is essentially equivalent to the existence of a common currency that allows comparison and transfer of utilities between players.

At this point we will apply the basic notions of Mechanism Design and define a *(direct revelation) mechanism* for our setting: this is just an algorithm that is adorned with payment information. In our case the input to a mechanism will be the n-tuple of valuations $(v_1, ..., v_n)$ and the output will consist of both the allocation $(m_1, ..., m_n)$ and the vector of payments $(p_1, ..., p_m)$. So here is the strategic setup that we are considering: the auctioneer gets to specify a mechanism, which is public knowledge ("power of commitment".) Each bidder i *privately* knows his valuation functions v_i. The bidders then interact with the mechanism transmitting to it their "bids" and are hence put in a game whose outcome is the resulting allocation and payment. In this game the players are acting strategically as to optimize their resulting utility. In particular the "bids" that they will report to the mechanism need not necessarily be their true valuations but rather whatever optimizes their utility. The *equilibrium* outcome of a given mechanism for the input valuations $(v_1, ..., v_n)$ will be its allocation $(m_1, ..., m_n)$ and payments $(p_1, ..., p_n)$ as calculated for the equilibrium bids (rather than as calculated for the true valuations.)

At this point we need to further specify our equilibrium notion which will of course depend on our informational assumptions: what do players know about each other's valuation and behavior. There are two main notions in the literature here. The Bayesian setup assumes that bidder's valuations v_i come from some known distribution F on valuation tuples. The players (and mechanism designer) have common knowledge of F and of each others' rationality, but each player i only knows the realization of his own v_i and not the realization of the other v_j's. The second notion is more robust as well as technically simpler, and makes a very strong requirement: that the equilibrium be robust to any information that players may have. This is the definition that we will use here.

Notations: A mechanism $M = (m(\cdot), p(\cdot))$ accepts as input an n-tuple of valuations $(v_1, ..., v_n)$ and outputs an allocation $(m_1, ..., m_n)$ and payments $(p_1, ..., p_n)$. Both the allocation and payments are functions of the n-tuple of v_i's and are denoted using $m_i(v_1, ..., v_n)$ and $p_i(v_1, ..., v_i)$. We will also use the abbreviation (v_{-i}, w_i) to denote the n-tuple of valuations where i's valuation is w_i and j's valuation for $j \neq i$ is v_j.

Definition: We say that a mechanism $M = (m(\cdot), p(\cdot))$ is *truthful* (in dominant strategies) if for every $(v_1, ..., v_n)$, every i, and every $\tilde{v}_i$ we have that $u_i \geq u'_i$ where $u_i = v_i(m_i(v_i, v_{-i})) - p_i(v_i, v_{-i})$ and $u'_i = v_i(m_i(\tilde{v}_i, v_{-i})) - p_i(\tilde{v}_i, v_{-i})$. Truthful mechanisms are equivalently called *strategy-proof* or *incentive compatible*.

What this means is that in no case will a player have any motivation to report to the mechanism anything other than his true valuation v_i; any other bid $\tilde{v}_i$ that he may attempt to report instead can never give him higher utility. For such mechanisms we may realistically assume that all bidders will indeed report their private valuations truthfully to the mechanism who may then come up with a good outcome.

This requirement may seem quite excessive and even unrealistic. First, the informational assumptions are quite strict: players strategic behavior does not depend on any information about the others, i.e., strategically all players have dominant strategies. Second, this dominant strategy is not arbitrary but is to report their true valuations. While the first assumption has significant bite and is what gives the high robustness to this notion relative to Bayesian notions mentioned above, the second point turns out to be without loss of generality. One can convert any mechanism with dominant strategies to an equivalent incentive compatible one by internalizing the dominant strategies into the mechanism. This is also true for general mechanisms with arbitrary protocols (not just getting the v_i's as input): as long as dominant strategies exist they can be incorporated into the mechanism, with the result being a truthful mechanism; this is known as the "revelation principle".

5.1 Vickrey-Clarke-Groves Mechanisms

The basic general positive result of mechanism design applies to our situation. This result, originally suggested by Vickrey [1961] and then generalized further by Clarke [1971] and Groves [1973], elucidates the following simple idea: if we charge each participant an amount that is equal to his *externality* on the others then we effectively align all users' interests with the total social welfare, and thus ensure that they behave truthfully as long as our mechanism maximizes social welfare. Specifically, here is this mechanism applied to Multi-unit auctions:

VCG Mechanism

1. Each Player reports a valuation $\tilde{v}_i$.
2. The mechanism chooses the allocation $(m_1, ..., m_n)$ that maximizes $\sum_j \tilde{v}_j(m_j)$ and outputs it.
3. For each player i:
   (a) The mechanism finds the allocation $(m'_1, ..., m'_n)$ that maximizes $\sum_{j \neq i} \tilde{v}_j(m'_j)$.
   (b) Player i pays $p_i = \sum_{j \neq i} \tilde{v}_j(m'_j) - \sum_{j \neq i} \tilde{v}_j(m_j)$.

The main property of this mechanism is that it is incentive compatible. To see this, first notice that the term $\sum_{j \neq i} \tilde{v}_j(m'_j)$ in the calculation of p_i has no strategic implications since its value does not depend in any way on the valuation $\tilde{v}_i$ reported by i and thus it does not effect in any way i's incentives (for any fixed value of v_{-i}). So to analyze i's strategic behavior we can simply assume that the first term is 0, i.e. that he is getting *paid* the amount $\sum_{j \neq i} \tilde{v}_j(m_j)$ and thus trying to maximize $v_i(m_i) + \sum_{j \neq i} \tilde{v}_j(m_j)$. However notice that as long as i indeed reports $\tilde{v}_i = v_i$ the mechanism is performing exactly this optimization! Thus i can not do better than by being truthful. The reader may wonder why is the term $\sum_{j \neq i} \tilde{v}_j(m'_j)$ used at all in the calculation of payments since, if we only care about truthfulness, it can be eliminated or replaced by an arbitrary expression that does not depend on v_i. However, this term is usually added as it ensures that the payments are always "in the right ballpark", satisfying "no positive transfers", $p_i \geq 0$, as well as "individual rationality", $v_i - p_i \geq 0$. We have thus proved:

Theorem 7 *(Vickrey [1961], Groves [1973]) There exists a truthful mechanism for the multi-unit allocation problem.*

The central problem with this mechanism is its computational efficiency. The allocation algorithm it uses is supposed to find the optimal allocation, and this we have already seen is impossible to do computationally in an efficient way. Now, for the special case of downward sloping valuations we did see, in section 4.2, a computationally-efficient optimal allocation algorithm, and thus for this special case we get a truthful computationally-efficient mechanism. This is the case that was dealt with in the seminal paper Vickrey [1961]. This however does not hold for the general problem. The basic difficulty in the field of Algorithmic Mechanism Design is to *simultaneously* achieve both incentive compatibility and computational efficiency. We will now continue trying to do so for multi-unit auctions.

5.2 The Clash Between Approximation and Incentives

As we have already seen in section 4.3 that no computationally-efficient algorithm can always find the optimal allocation, adding yet another requirement – that of truthfulness – would seem totally hopeless. However, we have also seen in section 4.4 that one may computationally-efficiently obtain approximately optimal allocations, and we could similarly hope for arbitrarily good *truthful approximation mechanisms.* Specifically, we would desire a truthful computationally-efficient mechanism that for every given $\epsilon > 0$ produces an allocation whose total value is at least $1 - \epsilon$ fraction of the optimal one. We may even settle for weaker approximation guarantees, settling for an allocation whose total value is fraction α, for as large as possible fixed value of α.

The natural approach to this goal would be to simply replace in the previous algorithm the step of "find an allocation that maximizes $\sum_j \tilde{v}_j(m_j)$" by "find an allocation that *approximately* maximizes $\sum_j \tilde{v}_j(m_j)$". This we would do for all $n + 1$ optimizations problems encountered by the mechanism: the global optimization to determine the allocation, as well as the n personalized ones, needed to compute the payments of the n bidders. We can call such a mechanism *VCG-based.* It would seem natural to assume that this would maintain truthfulness, but this is not the case.

Let us present a simple example of an approximation algorithm where if we use its results instead of the optimal ones we would lose truthfulness. Consider the allocation of two items between two players, both of whose valuations are: $v(1) = 1.9$ and $v(2) = 3.1$. The optimal allocation would allocate one item to each bidder, giving each a value of 1.9, and requesting from each a payment of $1.2 = 3.1 - 1.9$ which is the difference for the other between getting a single item or both. Now consider the case where we use an approximation algorithm that first truncates the values down to integer values and then finds the optimal allocation. Let us first consider the case where both players bid truthfully, $(1.9, 3.1)$. The approximation algorithm would truncate these values to $v(1) = 1$ and $v(2) = 3$, for which the optimal allocation is to give both items to one of the players. His payment will be the loss of the other player from not getting both items which is 3.1, giving our player zero (real) utility. Had our player been non-truthful bidding $\tilde{v}(1) = \tilde{v}(2) = 3$ then the approximation algorithm would choose the allocation that assigns a single item to each player, charging our non-truthful one $1.2 = 3.1 - 1.9$ giving him a net positive utility. (This calculation used the reported (unrounded) values for calculations of payments; using the rounded values will not restore truthfulness either.)

This example is in no way unusual. Not only is such a truncation algorithm essentially the approximation algorithm of section 4.4, but also it turns out that essentially *every* approximation algorithm will exhibit this non-truthfulness if we just plug it into the VCG payment calculation (Nisan and Ronen [2007], Dobzinski and Nisan [2007]). There is a small, almost trivial, set of approximation algorithms for which this is not the case and we study them in the next subsection.

Combinatorial Auctions and Beyond

Taking a wider look at the whole field of Algorithmic Game Theory, this clash between incentives and computation, is a main occurring theme. For many other algorithmic mechanism design problems we have reasonable approximation algorithms even though finding the optimal solution is not computationally feasible. In all of these cases, we are not able to apply the VCG technique as to obtain a computationally-efficient truthful mechanism. The central problem of this form is combinatorial auctions that has been the focal point such investigations since the early Lehmann et al. [2002]. There is yet another well-studied type of problems in algorithmic mechanism design where the main challenge is the lack of applicability of VCG mechanisms. These are problems where the optimization goal itself is not the maximization of social welfare, but rather some other computational goal. A central problem of this form is that of machine scheduling where the bidders correspond to machines that can execute jobs for a fee and a standard goal is the minimization of the the "makespan" – time that the last job is finished (Nisan and Ronen [2001]).

At this point one may naturally attempt to deal with the non-truthfulness that approximation algorithms cause by slightly relaxing the notion of truthfulness. One approach would be to require only "approximate truthfulness" in some sense. A major problem with this approach is that players' utilities and payments are *differences* between quantities. A good approximation of each quantity does not imply a good approximation of the difference.[5] Another relaxation approach would be to notice that lying is worthwhile only if it improves the overall allocation, something which is supposedly computationally hard. However, it is not completely clear how to define such "bounded-computational"-truthfulness, sensibly quantifying over different running times of different algorithms on different inputs. None of these approaches seems to be "clean" enough in general, so we will continue with the basic definition of exact truthfulness and continue studying whether it may be achieved.

5.3 Maximum in Range Mechanisms

The optimal VCG mechanism requires maximization of $\sum_i v_i(m_i)$ over all possible allocations $(m_1, ..., m_n)$. One can think about a "restricted" version of VCG which does not aim to maximize over all possible allocations but rather only over some pre-specified range of possible allocations. The smaller the range, the easier, potentially, it will be to optimize over it, but the less good will the best allocation in this limited range be. To take an extreme case, we could limit ourself in advance to bundle all the m items. This means that we will only maximize over the range of n possible allocations that have $m_i = m$ for a single agent i and $m_j = 0$ for all other j. This optimization is easy to do computationally (just check which i has the largest $v_i(m)$), but clearly it may only recover a $1/n$ fraction of possible welfare (e.g., when each bidder is willing to pay 1 for a single item, but no more than 1 even if he gets all items).

Mechanisms that take this approach are called Maximum-in-Range: such a mechanism fixes in advance a range of possible allocations; completely optimizes over this range; and then uses a VCG-like payments where everything is computed just over this range. The important point is that such mechanisms will always be truthful: the proof of truthfulness of the usual VCG mechanisms as given above directly applies also to every VCG-based mechanism. To see this, note the truthfulness of VCG mechanism was due to the fact that it aligned the utility of each player with that of the social welfare. This remains, and nothing that a bidder can do will take the result outside the range of allocations over which the mechanism already optimized. As mentioned, these turn out to be that the *only* approximation mechanisms that become truthful when simply using the VCG payment rule.

While the family of VCG-based mechanisms seems to be a trivial extension of VCG mechanisms we can still get non-trivial mileage from it. The trick would be to carefully

[5]An example of this phenomena exists in the recent FCC "incentive auctions" for buying back Spectrum from TV broadcasters. This auction requires sophisticated optimization and while the FCC's optimization programs were able to provably reach 97% efficiency, when these were plugged into the VCG formula, nonsensical results were obtained, including e.g. negative payments.

choose the range so that on one hand it is simple enough so we can effectively optimize over it, and on the other hand is rich enough so that optimizing over it guarantees some non-trivial level of approximation. For multi-unit auctions such a range will be obtained by pre-bundling the items into a small number of "bundles". For simplicity of presentation we will assume without loss of generality that m is an integer multiple of n^2:

Approximation Mechanism

1. Split the m items into n^2 equi-sized bundles of size m/n^2 each
2. Run the optimal VCG mechanism with the items being the n^2 bundles, using the general allocation algorithm from section 4.1

Note that this mechanism is Maximum-in-Range and hence truthful since be are completely optimizing over allocations that correspond to the pre-bundling. The point is that this mechanism is also computationally-efficient since it is just an auction with n^2 items (the bundles), which we have already seen in section 4.1 can be done in time that is polynomial in the number of items, which is now just polynomial in n.

The quality of allocations produced by this mechanism is that they always have at least 1/2 the total value of the optimal one: $\sum_i v_i(m_i) \geq \frac{1}{2}\sum_i v_i(m'_i)$, where $(m_1, ..., m_n)$ is the optimal allocation that respects this bundling (which is the one the mechanism produces) and $(m'_1, ..., m'_n)$ is any other allocation. To see this, start with the optimal allocation $(m'_1, ..., m'_n)$ and consider the bidder i that got the largest number of items, $m'_i \geq m/n$. Now there are two possibilities. If at least half the total value comes just from this i, $v_i(m'_i) \geq \sum_{j \neq i} v_j(m'_j)$, then clearly the allocation that gives everything to him (which is part of our restricted optimization range) gets at least $\frac{1}{2}\sum_i v_i(m'_i)$ so our allocation is only better. If on the other hand $v_i(m'_i) < \sum_{j \neq i} v_j(m'_j)$ then we can take away i's allocation, and use these $m'_i \geq m/n$ items to complete the allocations of all other bidders to exact multiples of m/n^2 (which may require at most m/n^2 for each bidder), now resulting in a bundling allocation that is in our restricted optimization range but with value of at least $\sum_{j \neq i} v_j(m'_j) \geq \frac{1}{2}\sum_i v_i(m'_i)$ which is thus a lower bound on $\sum_i v_i(m_i)$ as needed. We thus have:

Theorem 8 *(Dobzinski and Nisan [2007]) There exists a truthful mechanism for the 2-approximate multi-unit allocation problem (in any of the models we considered) whose running time is polynomial in n and $\log m$. It produces an allocation $(m_1, ..., m_n)$ that satisfies $\sum_i v_i(m_i) \geq (\sum_i v_i(opt_i))/2$, where $(opt_1, ..., opt_n)$ is the optimal allocation.*

So we have obtained a truthful computationally-efficient 2-approximation mechanism. This is still far from the arbitrarily good approximation that is possible when we just take into account one of the two constraints of efficient computation or of truthfulness. Can we do better? We will now see that no Maximum-in-Range mechanism (with any range) can get a better approximation ratio. We will prove the impossibility of a better approximation even for the special case of two players for which we have already seen in section 4.3 that finding the exactly optimal allocation is intractable even in the general queries (communication) model.

Consider an algorithm that guarantees strictly better than 1/2 of the maximum possible value. If the range is complete, i.e. contains all possible allocations $(m_1, m - m_1)$ then we are doing perfect optimization which as we have seen is hard, and thus the algorithm is not computationally-efficient. Otherwise, our range is missing at least one possible outcome $(m_1, m - m_1)$. Consider the input valuations given by the single minded bids where the first player is willing to pay 1 for at least m_1 items and the second is willing to pay 1 for at least $m - m_1$ items. Clearly the optimal allocation for this case would be $(m_1, m - m_1)$ with $v_1(m_1) + v_2(m - m_1) = 2$. However, this allocation is not in our range, so our maximum-in-Range mechanism will output a different allocation, and all other allocations get value of at most 1, which is exactly half of optimal. This argument assumes a black-box query model and, in fact, Dobzinski and Nisan [2007] also show that better approximations are possible in the bidding language models discussed above. We have thus shown:

Theorem 9 *(Dobzinski and Nisan [2007]) There is no Maximum-in-Range mechanism for approximate multi-unit allocation (in any query model) whose running time is polynomial in n and $\log m$ and always produces an allocation $(m_1, ..., m_n)$ that satisfies $\sum_i v_i(m_i) > (\sum_i v_i(opt_i))/2$, where $(opt_1, ..., opt_n)$ is the optimal allocation.*

So, as we have reached the limits of the VCG technique, are there other new techniques for the construction of truthful mechanisms? Can any of these techniques give an efficiently computable mechanism that gets a better approximation ratio?

Combinatorial Auctions

When looking at the wider class of combinatorial auctions, even the algorithmic problem can not be approximated arbitrarily well. As significant algorithmic work has been done on obtaining various approximation factors for various sub-classes of valuations, for each of these subclasses, the corresponding challenge is to find a truthful mechanism that achieves similar approximation ratios. Maximum-in-Range mechanisms achieve various partial results in this respect – see Blumrosen and Nisan [2007] for survey and references.

5.4 Single Parameter Mechanisms

There is one class of situations where classical Mechanism Design provides another, non-VCG, technique for constructing truthful mechanisms. This is the case where buyers' private valuations are *single dimensional.* For the single dimensional case, Myerson [1981] essentially gives a complete characterization of incentive compatible mechanisms. Our presentation here essentially utilizes these results, but slightly differs from the classic treatment in two respects. First, we will limit our discussion to deterministic mechanisms, as we have done so far. Second, our scenario considers *Single-Minded bids*, which are really slightly more than single dimensional as they contain an additional discrete parameter (the desired number of items).

So, in this section we restrict ourselves to bidders whose valuations are Single Minded: each bidder i desires a specific number of items k_i and is willing to pay v_i if he gets at least k_i items (and gets value zero otherwise). This is a very limited class of valuations for which we have no representation problems. However, in this section we will rely on their *strategic simplicity* which is beyond the representational one. In particular, the results in this section do not apply when we allow even double-minded bidders that are still simple in the representational sense.

A bidder that is single-minded may either lose or win the auction – winning gives value v_i and losing gives value 0. We can always assume without loss of generality that a mechanism never allocate to any winner i more than exactly the $\tilde{k_i}$ items it requested and never allocates anything to a losing bidder. As a normalization, we may also assume without loss of generality that losing bidders always pay 0. Under these conventions it turns out that, when bidders are limited to being Single-Minded, truthful mechanisms are exactly the ones that satisfy the following two properties:

1. **Monotonicity:** If a player wins with bid (k_i, v_i) then it will also win with any bid which offers at least as much money for at most as many items. I.e. i will still win if the other bidders do not change their bids and i changes his bid to some (k'_i, v'_i) with $k'_i \leq k_i$ and $v'_i \geq v_i$.

2. **Critical Payment:** The payment of a winning bid (k_i, v_i) is the smallest value needed in order to win k_i items, i.e. the infimum of v'_i such that (k_i, v'_i) is still a winning bid, when the other bidders do not change their bids.

Let us see why these are sufficient conditions: First let us see why for a bidder whose real valuation is (k_i, v_i) bidding a quantity $k'_i \neq k_i$ is always dominated by k_i. If $k'_i < k_i$ then even if i wins he gets only $k'_i < k_i$ items and thus his value from winning is 0 and thus his utility is not positive. On the other hand, a bid $k'_i > k_i$ can not help him since monotonicity implies that anything that he wins with bid k'_i he could also win with k_i and furthermore more, given the the critical payment condition, the payment in the latter case is never higher than in the former case. We now show that misreporting v_i is not beneficial either. If v_i is winning and paying the critical value $p_i \leq v_i$ then every possible bid $v'_i \geq p_i$ is still winning and still paying the same amount p_i leading to the same utility as when bidding v_i. A bid $v'_i < p_i$ in this case will lose, getting utility 0, which is certainly not better than for bidding v_i. If, on the other hand, v_i is a loosing bid, and the smallest winning bid is $p_i \geq v_i$ then bidding less than p_i will still lose, and bidding at least p_i will win but will yield negative utility.

This characterization is quite fruitful: first, it decouples the algorithmic issue from the strategic one: algorithmically we only need to devise a good *monotone* approximation algorithm. Incentive compatibility will follow once we add the critical payment rule. Second, it opens up a wide design space of monotone algorithms. At first, the limitation of being monotone does not seem to be too challenging as it actually is hard to imagine any useful algorithmic idea that is not monotone. Then, at second thought, one finds that any interesting algorithmic idea attempted breaks monotonicity. Finally, one learns how to be quite careful in the design of algorithms and ensure monotonicity. Let us look at these three stages as they relate to our problem.

Let us start with the approximation algorithm presented in section 4.4 and consider it in our single-minded setting of inputs (k_i, v_i). This gives the following mechanism:

Single-Minded Additive Approximation Mechanism

1. Let $\epsilon > 0$ be the required approximation level, and let δ be a truncation precision parameter
2. Truncate each v_i to $j\delta$ where $0 \leq j \leq n/\epsilon$ is an integer
3. Find the optimal allocation among the truncated v_i's, using the algorithm from section 4.4
4. Charge the critical payments

It is not difficult to see that for each fixed ϵ and δ, the δ-scale approximation algorithm is indeed monotone: the truncation is clearly a monotone operation, and the optimal solution is clearly monotone. However, what should the choice of δ be? If we just take a fixed choice of δ then we do get an *additive* approximation error of $n\delta$, however this will not translate to any approximation *ratio* since for very small valuations we may lose everything. In our original description of the algorithm in section 4.4 our choice was $\delta = \epsilon V/n$ where $V = max_i v_i$. This choice strikes a balance ensuring on one hand that an additive error of $n\delta$ which the truncation causes translates to an approximation factor of $1-\epsilon$ and on the other hand keeps the range of rounded values small (i.e. n/ϵ) enough for computational efficiency. However, this choice of δ is not monotone: it is possible that a bidder who has the largest bid $v_i = V$ is a winner, but when he slightly increases his bid, then δ will also increase, leading to a different rounding in which he loses. Consider selling $m = 12$ items to $n = 7$ bidders: each of three bidder are willing to each pay \$7 for 4 items, while each of four bidders are willing to pay \$5.9 for three items. Now for $\epsilon = 1$ we get $\delta = \$1$ and the first group of three bidders all win. Had one of the first group of bidders slightly increased his bid to \$7.1, then δ would increase slightly above 1, and the new rounding would favor the second group of bidders over the first group, breaking monotonicity hence implying that the mechanism is not truthful.

It is not at all clear whether there is any monotone way of choosing δ in a way that will maintain both the approximation ratio and the computational efficiency. The solution proposed in Briest et al. [2011] is to simultaneously try multiple values of δ as all different orders of scale:

Single-Minded Approximation Mechanism:

1. Let $\epsilon > 0$ be the required approximation level.
2. For all $\delta = 2^t$ where t is an integer (possibly negative) with $\epsilon V/(2n^2) \le \delta \le V$, where $V = max_i v_i$ do:
   (a) Run the Additive Approximation Algorithm with precision parameter δ obtaining a "δ-scale allocation"
3. Choose the δ-scale allocation that achieved the highest value (as counted with the truncated values) to determine the final allocation
4. For each winner i in the final allocation: Find, using binary search, the lowest value that would still have bidder i win when the others' values are unchanged, and set this critical value as i's payment

It would seem that we have only made things worse: previously we had a single δ that depended on V, now we have a range of such δ's. However, the main observation is that in fact the range of δ's does not depend on V: for $\delta > V$ the truncation would anyway result in rounding everything to 0's, while for $\delta < \epsilon V/(2n^2)$, the δ-scale approximation will truncate each v_i to at most $V/(2n)$, yielding a total value of at at most $V/2$, which is less than that obtained for the largest δ in the range checked by allocating to the highest bidder. Thus to strategically analyze this mechanism we can imagine that we really tried *all* (infinitely many) possible values of $\delta = 2^t$ for integer t and chose the best allocation encountered. This imaginary infinite search would yield exactly the same outcome as our finite one does. However, it is clear that in this imaginary mechanism the choice of each δ does not depend in any way on the bids so we maintain monotonicity.

Or do we? While it is true that for every single δ we have a monotone algorithm, it does not follow that an algorithm that runs several monotone algorithms and then chooses the best among them is monotone itself. It could be that a winner, that wins when the optimum was achieved via a certain value of δ, increases his bid, causing a different value of δ to be the winning choice, a value of δ in which our original bidder loses. However, in our case it is possible to show that this does not happen: a nice property of the additive approximation is that the only way that the value of a player influences the solution quality is by winning or not. Thus if an increase in v_i causes a new value of δ to achieve the highest value then v_i must be winning for the new value of δ and thus global monotonicity is maintained. The same is true for manipulations in the requested number of items k_i. We have thus obtained:

Theorem 10 *(Briest et al. [2011]) There exists a truthful mechanism for the approximate multi-unit allocation problem among single-minded bidders whose running time is polynomial in n, $\log m$, and ϵ^{-1}. It produces an allocation $(m_1, ..., m_n)$ that satisfies $\sum_i v_i(m_i) \geq (1-\epsilon)\sum_i v_i(opt_i)$, where $(opt_1, ..., opt_n)$ is the optimal allocation.*

So we have shown a truthful polynomial time approximation mechanism for the single-minded case. What about general valuations?

Combinatorial Auctions and Beyond

A similar phenomena appears in many others algorithmic mechanism design problems, where good approximation mechanisms are known for single-parameter settings. This was demonstrated effectively in the context of combinatorial auctions in Lehmann et al. [2002] where a truthful $O(\sqrt{m})$-approximation mechanism was designed, matching the algorithmic possibilities. Such mechanisms can be applied also to problems that do not maximize social welfare, as has been done, for example in the context of scheduling in early work of Archer and Tardos [2001].

5.5 Multi-Parameter Mechanisms Beyond VCG?

Once we have seen that for the strategically simple case of single-minded bidders we can design arbitrarily good computationally-efficient truthful approximation mechanisms it is natural to try to get the same type of result for general bidders. Is that possible? While this is still open, it is known that truthfulness implies rather severe constraints in multi-parameter settings. This section will demonstrate these type of constraints by proving a partial impossibility result. We will show that a truthful mechanism for $n = 2$ players that is required to *always allocate all items* which obtains an approximation factor that is strictly better than 2 requires exponentially many queries, in the value queries model. So let us fix such a mechanism $M = (m_1(\cdot), m_2(\cdot), p_1(\cdot), p_2(\cdot))$, where m_i denotes the number of items allocated to player i, where we always have $m_1 + m_2 = m$.

We start by deducing a necessary condition on the allocation function itself $(m_1(\cdot), m_2(\cdot))$ (independently of the payment functions). This condition will then allow us to analyze the behavior of truthful mechanisms and bound their power. Our analysis starts with stating the basic structure of payments induced by fixing one of the two input valuations and varying only the other.

The Taxation Principle: For every fixed value of v_2, there exists prices $\{p_a\}$ for $0 \leq a \leq m$ such that $p_1(v_1, v_2) = p_a$ whenever $a = m_1(v_1, v_2)$ (we will denote $p_a = \infty$ if there is no v_1 with $a = m_1(v_1, v_2)$). Moreover, $m_1(v_1, v_2) = argmax_a\{v_1(a) - p_a\}$.

The taxation principle states that, once v_2 is fixed, the prices that the first bidder face may only depend on the allocation it gets rather than arbitrarily on his full valuation. Moreover, under this pricing rule, his allocation maximizes his utility. Otherwise, if $m_1(v_1, v_2) = m_1(v_1', v_2)$ but $p_1(v_1, v_2) > p_1(v_1', v_2)$ then clearly a bidder with valuation v_1 will gain by misreporting v_1'. Similarly if $m_1(v_1, v_2) \neq argmax_a\{v_1(a) - p_a\} = m_1(v_1', m_2)$ then a a bidder with valuation v_1 will gain by misreporting v_1'.

Let us now consider a situation where, for some fixed value of v_2, when the first bidder changes his valuation from v_1 to v_1' then his allocation changes from a to a'. By the taxation principle this means that $v_1(a) - p_a \geq v_1(a') - p_{a'}$ while $v_1'(a') - p_{a'} \geq v_1'(a) - p_a$. Combining these two inequalities allows us to cancel p_a and $p_{a'}$, showing that the allocation rule of a truthful mechanism satisfies the following property, a property that also turns out to be sufficient (Lavi et al. [2003]):

Weak Monotonicity: The allocation function $m_1(\cdot)$ is called *weakly monotone* if whenever $m_1(v_1, v_2) = a \neq a' = m_1(v_1', v_2)$ then we have that $v_1(a) - v_1'(a) \geq v_1(a') - v_1'(a')$.

The heart of our proof will use this property to show that for some fixed value of v_2, the range of $m_1(\cdot, v_2)$ is full, i.e. for every $0 \le a \le m$ there exists v_1 such that $m_1(v_1, v_2) = a$ and, moreover, each of these v_1's is strictly monotone, $0 < v_1(1) < \cdots < v_1(m)$. Using the taxation principle, this in particular implies $p_0 < p_1 < \cdots < p_m < \infty$. This allows to use the following adversary argument. Fix v_2 and our adversary will choose v_1 adaptively, as it answers value queries, as follows: for each query a it will answer $v_1(a) = p_a - p_0$. Note that as long as the algorithm is missing even a single query $0 < a \le m$, it is still possible that for the missing a, we have that $v_1(a) > p_a - p_0$ which would give strictly higher utility than all previous queries, and so the truthful mechanism may need to output it. Thus in order for the algorithm to be truthful it must query $v_1(a)$ for all m possible values of a.

We will choose v_2 to be the linear valuation $v_2(k) = 2k$ (there is nothing special about this valuation and any other strictly monotone one will do as well). For every $0 \le a \le m$ we will exhibit a valuation v_1 such that for the input (v_1, v_2), the output must be $m_1 = a$ and $m_2 = m - a$. We will argue this by starting with some other valuations (v_1', v_2') for which we can know the output and "modifying" them to become the desired (v_1, v_2) using weak monotonicity to control how the results may change.

Let us denote by $1_{\ge a}$ the valuation that gives value ϵ for each item with a large bonus of 1 for at least a items (i.e. $1_{\ge a}(k) = k\epsilon$ for $k < a$ and $1_{\ge a}(k) = 1 + k\epsilon$ for $k \ge a$) and consider the allocation on the input $v_1' = 1_{\ge a}$ and $v_2' = 1_{\ge m-a}$. The fact that the mechanism is a strictly better than 2 approximation implies that for this input $m_1 = a$ and $m_2 = m - a$.

We now consider a "stretched" version v_1 of v_1' where for $k < a$ we define $v_1(k) = v_1'(k)$, for $k > a$ we define $v_1(k) = c + v_1'(k)$ and $v_1(a) = c + v_1'(a) + \epsilon/2$, where c is a large constant. We claim that for input (v_1, v_2') our mechanism must still produce the allocation $m_1 = a$ and $m_2 = m - a$. This is true due to weak monotonicity as in our construction $v_1(a) - v_1'(a) \le v_1(a') - v_1'(a')$ only holds for $a' = a$ (notice that the roles of a and a' are switched from their roles in the definition).

Let us now consider the output of our mechanism on the input (v_1, v_2). Since c is large, to maintain an approximation ratio the first player must be allocated at least a items and thus the second must be allocated at most $m - a$ items. Can the second player be allocated strictly less items, $m - a'$ for $a' > a$? If so, then using the weak monotonicity of m_2 for the fixed choice of v_1, we have that $v_2(m-a) - v_2'(m-a) \le v_2(m-a') - v_2'(m-a')$ which in our construction only holds for $a' \le a$. Thus $m_2 = m - a$ and $m_1 = a$ as required, completing the proof of:

Theorem 11 *(Lavi et al. [2003]) Every truthful mechanism for approximate multi-unit allocation with two players that always allocates all items, $m_1 + m_2 = m$, and for some fixed $\alpha > 1/2$ always satisfies $v_1(m_1) + v_2(m_2) \ge \alpha(v_1(opt_1) + v_2(opt_2))$, where (opt_1, opt_2)*

is the optimal allocation, requires at least m queries in the value queries model.

So how "biting" are the constraints induced by truthfulness? Can we extend this theorem to hold without the requirement of always allocating all items? This is still not known. To gain a perspective about what we may be able to achieve, it will be useful to consider a strong characterization result due to Roberts [1979] that holds in a more general mechanism design setting. In this abstract non-restricted setting there are n agents that have valuations that assign an arbitrary values to possible abstract outcomes in the set A, i.e. $v_i : A \to \Re$. The mechanism in such a case has to choose a single outcome $a \in A$ (as well as payments) in a truthful way. It is not difficult to verify that the VCG mechanism that we have seen in section 5.1 directly generalizes to this setting yielding a truthful mechanism whose outcome maximizes $\sum_i v_i(a)$ over all $a \in A$. It is also not difficult to generalize the VCG mechanism by assigning weights $\alpha_i \geq 0$ to the players as well as weights β_a to the outcomes, obtaining a truthful mechanism that maximizes $\beta_a + \sum_i \alpha_i v_i(a)$ over all $a \in A$. A mechanism whose output always maximizes this expression (for some choice of $\{\alpha_i\}, \{\beta_a\}$) is called an *affine maximizer*. The payment rule to make an affine maximizer truthful is given by a natural weighted variant of the VCG payments. Roberts' theorem states that these are the *only* truthful mechanisms for the general unrestricted mechanism design problem when there are at least three outcomes. This is in stark contrast to the single-parameter case where Myerson's technique provides many truthful non-affine-maximizing (non-VCG) mechanisms.

Now back to our setting of multi-unit auctions. The case of Single-Minded bidders is close to being single-parameter and as we saw in section 5.4 there are good, non-VCG, mechanisms for this case. The more general case, where bidders have more general valuations is far from being single-parameter and we basically lack any techniques that go beyond (weighted versions of) VCG, which are very limited in their power. For example, as shown in section 5.3, they cannot guarantee, in polynomial time, more than half of the social welfare. (It is not difficult to generalize the proof there to weighted versions of VCG, i.e. to any affine maximizer.) Had Roberts' theorem applied to multi-unit auctions and not just to the unrestricted case then we would have a general impossibility result of going beyond (weighted) VCG. However, the setting of multi-unit auctions is more restricted in two respects. First, we are assuming free disposal (v_i is monotone) and, second, we are implicitly assuming no externalities. (v_i is only a function of m_i rather than of the full allocation). The case analyzed here of two players where all items must always be allocated, voids the "no externalities" restriction and becomes sufficiently close to the unrestricted case as to allow Lavi et al. [2003] to extend (a version of) the Roberts characterization to it. On the other hand, without this restriction, several non-VCG truthful mechanisms are known for multi-unit auctions, including some that guarantee more than half of the social welfare (Dobzinski and Nisan [2011]), although not in polynomial time. An impossibility result is given in Dobzinski and Nisan [2011] for the subclass of "scalable mechanisms" – those where multiplying all values by the same constant factor does not change the allocation.

The general question remains open: are there any *useful* truthful non-VCG mechanisms for multi-unit auctions? This is open for most interpretations of "useful", including the one we embrace in this survey: computationally-efficient and guaranteeing strictly more than half the optimal social welfare[6].

Combinatorial Auctions and Beyond

The basic question of existence of "useful" non-VCG computationally-efficient truthful combinatorial auctions is open for general valuations as well as for most interesting subclasses. There is one case where a truthful non-VCG mechanism is known (Bartal et al. [2003]), as well one case where impossibility results are known (Dobzinski [2011], Dobzinski and Vondrák [2012]). A similar situation holds for other mechanism design problems where again essentially no non-VCG mechanisms are known for multi-parameter settings. For a problem that is "close" to being unrestricted, there is a strong characterization extending Roberts (Papadimitriou et al. [2008]). There is also a restricted impossibility results for scheduling due to Ashlagi et al. [2009].

5.6 Randomization

We now slightly widen our scope of mechanisms and allow also randomized ones. We will see that these indeed do provide us with increased capabilities. A randomized mechanism will still accept as input the valuations of the players but will now output a *distribution over allocations and payments*. A randomized mechanism will be called truthful (in expectation) if for all v_i, v_{-i}, v_i' we have that $E[v_i(M_i) - P_i] \geq E[v_i(M_i') - P_i']$ where M_i and P_i are the random variables indicating the number of items allocated to i and the payment of i when he bids v_i, and M_i' and P_i' are these random variables when he bids v_i'. We will concentrate on a specific class of randomized mechanisms that first *deterministically* choose a *distribution* on allocations, and a payment and then just output a random realization of an allocation from the chosen distribution.

We can now generalize the idea of maximum in range mechanisms. We can think of a mechanism that pre-decides on a range of *distributions on allocations*. For any such possible distribution D on allocations $(m_1 ... m_n)$, we can look at the expected value of each v_i on this distribution: $v_i(D) = E_{(m_1,...,m_n)\sim D}[v_i(m_i)]$. From this point of view we

[6]It is probably good to remark that if we agree to relax our notion of equilibria to Bayesian, then a multitude of computationally-efficient Bayesian-truthful mechanisms exist (e.g. Hartline et al. [2011]).

can look at any such distribution as a new possible allocation. We will call a randomized mechanism *maximum in distributional range* if it always maximizes the expected social value over such a pre-chosen range of distributions, i.e. always chooses the distribution that maximizes $\sum_i v_i(D) = E_{(m_1...m_n)\sim D}[\sum_i v_i(m_i)]$ over all D in the range. Clearly we can apply the VCG payment rule to such a mechanism by simply viewing the range of distributions as a new range of "deterministic" outcomes, this way obtaining a randomized truthful (in expectation) mechanism.

Can such mechanisms be helpful? In particular can we build a maximum-in-distributional-range mechanism that provides a good approximation while being computationally-efficient? The approximation ratio of a randomized mechanism is defined to be the worst case ratio obtained over all possible inputs, where for each input we measure ratio between the expected social welfare produced and the optimal social welfare for this input. As shown by Dobzinski and Dughmi [2009], it turns out that the answer is yes. In fact, a very basic randomized range already suffices for this. The distributions we will use in our range will all have support size of 2: the empty allocation ($m_i = 0$ for all i) and a single other allocation $(m_1, ..., m_n)$. We will have such a distribution for every possible allocation $(m_1, ..., m_n)$; we will define a "penalty probability" $0 \le z(m_1, ..., m_n) < \epsilon$ for each $(m_1, ..., m_n)$, and will choose the empty allocation with probability $z(m_1, ..., m_n)$ and allocate m_i to each i with probability $1 - z(m_1, ..., m_n)$. The expected social welfare of such a distribution is simply $(1 - z(m_1, ..., m_n)) \sum_i v_i(m_i)$, and it is this expression that our algorithm maximizes (over all $(m_1, ..., m_n)$.) It should immediately be clear that this allocation will also approximately maximize the non-penalized social welfare, i.e. $\sum_i v_i(m_i) \ge (1 - \epsilon) \sum_i v_i(m'_i)$ for every other allocation $(m'_1...m'_n)$. The wonder is that we will be able to define these penalties in a way that will allow the computationally-efficient maximization of $(1 - z(m_1, ..., m_n)) \sum_i v_i(m_i)$ over all allocations.

The idea is to penalize "non-roundness". Let us define the weight of an integer m_i, $w(m_i)$, to be the largest power of two that divides m_i (thus the weight of 8 is 3, the weight of 9 is 0, and the weight of 10 is 1). Let us also define the weight of an allocation to be the sum of the weights of the m_i's, $w(m_1, ..., m_n) = \sum_i w(m_i)$. Our penalty function will be defined as $z(m_1, ..., m_n) = (C - w(m_1, ..., m_n))\delta$, where C is a fixed upper bound on the weight of any allocation ($C = n \log m$) and δ chosen to be small enough so that z is in the required range, ($\delta = \epsilon/C$). The point is that these penalties for "non-round" allocations will suffice to significantly trim down the search space.

Let us look at a single player: do we need to consider all m possible values of m_i to allocate to him or can we significantly trim down the search space and consider only a smaller number of possible m_i's? Consider two possible values $m_i < m'_i$ with $w(m_i) > w(m'_i)$. The point is that if their values are close, $v_i(m_i)/v_i(m'_i) > (1 - \delta)$, then m'_i can clearly be trimmed from the search space since the tiny addition to $v()$ when increasing m_i to m'_i can not offset the loss of welfare due to the decrease in $w_i()$ (which increases z). It turns out that this suffices for the construction of an computationally-efficient algorithm.

Maximum-in-Distributed-Range Mechanism

1. For each player i, compute a list $List_i$ of possible allocations m_i to player i by calling $List_i = List_i(0, \log m)$:

   For an integer $0 \le t \le \log m$, and $0 \le L \le m$ with $w(L) \le t$, $LIST_i(L,t)$, will output the list of all non-trimmable values of m_i in the range $L < m_i \le L + 2^t$:

   (a) If $t = 0$ then return the singleton $\{L+1\}$.

   (b) If $v_i(L)/v_i(L+2^{t-1}) < 1-\delta$ then return $LIST_i(L, t-1)$ concatenated with $LIST_i(L+2^{t-1}, t-1)$.

   (c) Else return $LIST_i(L+2^{t-1}, t-1)$.

2. Find the allocation $(m_1, ..., m_n)$ that maximizes $(1-z(m_1...m_n))\sum_i v_i(m_i)$, over all $m_i \in List_i$.

3. With probability $z(m_1, ..., m_n)$ output the empty allocation, otherwise output $(m_1, ..., m_n)$.

4. Compute VCG payments for each bidder i by re-running the optimization without i.

The point is that the subroutine LIST branches to two recursive calls only if $v(L)/v(L+2^{t-1}) < 1-\delta$, otherwise it only makes a single recursive call. In terms of correctness this is OK since if the gap between $v(L)$ and $v(L+2^{t-1})$ is too small then we have seen that the whole range $L < m_i \le L + 2^{t-1}$ can be trimmed away. To bound the running time, we will bound the number of invocations of LIST where a branch can be made, and will do that for each fixed value of the $1 + \log m$ possible levels t's. Since the subranges on which $LIST$ is called at level t are a partition, and since a branch is undertaken only if there is a $(1-\delta)$-gap between the value of v_i at the two end points of the subrange, then there can be at most $O(T/\delta)$ invocations of LIST that branch at any given level t of the recursion, where T is the number of bits of precision in the representation of v_i. Thus the total number of recursive invocations of $LIST_i$ is bounded by $O(T \log m/\delta)$ giving a total polynomial bound on the running time as well as on the length of $LIST_i$.

If we just worry about information considerations, i.e. the number of queries made, then we are already done: Finding the optimal allocation only requires querying the values of $v_i(m_i)$ only for these polynomially-many $m_i \in LIST_i$. To completely specify a polynomial time algorithm we need to explicitly describe how to use this polynomial-length information to find the optimal allocation. For the case of a constant number of players, this is immediate by exhaustive search over all possible combinations. The general case is more complex and appears in Dobzinski and Dughmi [2009]. We have thus obtained:

Theorem 12 *(Dobzinski and Dughmi [2009]) There exists a randomized truthful (in expectation) mechanism for the approximate multi-unit allocation problem (in any of the models we considered) whose running time is polynomial in n, $\log m$, and ϵ^{-1}. It produces an allocation $(m_1, ..., m_n)$ that satisfies $E[\sum_i v_i(m_i)] \geq (1-\epsilon)\sum_i v_i(opt_i)$, where $E[]$ denotes expectation over the random choices of the mechanism and $(opt_1, ..., opt_n)$ is the optimal allocation.*

A slightly stronger result in terms of truthfulness is also known (Vöcking [2012]). This mechanism randomly chooses among deterministic truthful mechanism for the problem. The guarantee on the quality of the output is still probabilistic, but the truthfulness holds universally for each random choice rather than only in expectation.

Combinatorial Auctions and Beyond

Looking at the wider picture of Algorithmic Mechanism Design, the power of randomized mechanisms is a general phenomena and there are many cases where we have randomized mechanisms that are better than the deterministic ones. This, in particular, is so for combinatorial auctions where a randomized mechanism is known that achieves an $O(\sqrt{m})$-approximation ratio which matches the ratio possible due to algorithmic constraints alone (Dobzinski et al. [2006]). Similarly, for many subclasses of valuations we have randomized mechanisms whose approximation ratios are much better than those of the known truthful deterministic ones. See Blumrosen and Nisan [2007] for references.

6 Conclusion

The challenge of designing good truthful efficiently-computable (deterministic) mechanisms for welfare maximization in multi-unit auctions remains open.

The same is true for most other interesting algorithmic mechanism design problems such as combinatorial auctions.

Acknowledgements

I would like to thank Moshe Babaioff, Shahar Dobzinski, Michal Feldman and Renato Paes Leme for extensive comments on a previous draft. I would like to thank Shahar for the proof of theorem 11.

References

Aaron Archer and Eva Tardos. Truthful mechanisms for one-parameter agents. In *FOCS'01*, 2001.

Itai Ashlagi, Shahar Dobzinski, and Ron Lavi. An optimal lower bound for anonymous scheduling mechanisms. In *ACM Conference on Electronic Commerce*, pages 169–176, 2009.

Yair Bartal, Rica Gonen, and Noam Nisan. Incentive compatible multi unit combinatorial auctions. In *TARK*, 2003.

L. Blumrosen and N. Nisan. Combinatorial auctions (a survey). In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, 2007.

L. Blumrosen and N. Nisan. On the computational power of demand queries. *SIAM Journal on Computing*, 39(4):1372–1391, 2010.

P. Briest, P. Krysta, and B. Vcking. Approximation techniques for utilitarian mechanism design. *SIAM Journal on Computing*, 40(6):1587–1622, 2011.

E. H. Clarke. Multipart pricing of public goods. *Public Choice*, pages 17–33, 1971.

P. Cramton, Y. Shoham, and R. Steinberg (Editors). *Combinatorial Auctions.* MIT Press, 2006.

Sven de Vries and Rakesh Vohra. Combinatorial auctions: A survey. *INFORMS jounral of computing*, 15(3):284–309, 2003.

Shahar Dobzinski. An impossibility result for truthful combinatorial auctions with submodular valuations. In *STOC*, pages 139–148, 2011.

Shahar Dobzinski and Shaddin Dughmi. On the power of randomization in algorithmic mechanism design. In *FOCS*, pages 505–514, 2009.

Shahar Dobzinski and Noam Nisan. Mechanisms for multi-unit auctions. In *Proceedings of the 8th ACM conference on Electronic commerce*, EC '07, pages 346–351, 2007.

Shahar Dobzinski and Noam Nisan. Multi-unit auctions: beyond roberts. In *Proceedings of the 12th ACM conference on Electronic commerce*, EC '11, pages 233–242, 2011.

Shahar Dobzinski and Jan Vondrák. The computational complexity of truthfulness in combinatorial auctions. In *ACM Conference on Electronic Commerce*, pages 405–422, 2012.

Shahar Dobzinski, Noam Nisan, and Michael Schapira. Truthful randomized mechanisms for combinatorial auctions. In *STOC*, 2006.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

T. Groves. Incentives in teams. *Econometrica*, pages 617–631, 1973.

Faruk Gul and Ennio Stacchetti. Walrasian equilibrium with gross substitutes. *Journal of Economic Theory*, 87:95 – 124, 1999.

Faruk Gul and Ennio Stacchetti. The english auction with differentiated commodities. *Journal of Economic Theory*, 92(3):66 – 95, 2000.

Jason Hartline. *Approximation in Economic Design.* Lecture Notes, 2012.

Jason D. Hartline and Anna R. Karlin. Profit maximization in mechanism design. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory.* Cambridge University Press, 2007.

Jason D Hartline, Robert Kleinberg, and Azarakhsh Malekian. Bayesian incentive compatibility via matchings. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 734–747, 2011.

V. Krishna. *Auction theory.* Academic press, 2002.

Eyal Kushilevitz and Noam Nisan. *Communication Complexity.* Cambridge University Press, 1997.

Ron Lavi, Ahuva Mu'alem, and Noam Nisan. Towards a characterization of truthful combinatorial auctions. In *FOCS*, 2003.

Daniel Lehmann, Liadan Ita O'Callaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auctions. In *JACM 49(5)*, pages 577–602, Sept. 2002.

A. Mas-Collel, W. Whinston, and J. Green. *Microeconomic Theory.* Oxford university press, 1995.

Paul Milgrom. *Putting auction theory to work.* Cambridge University Press, 2004.

R. B. Myerson. Optimal auction design. *Mathematics of Operations Research*, 6(1):58–73, 1981.

Noam Nisan. Introduction to mechanism design (for computer scientists). In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory.* Cambridge University Press, 2007.

Noam Nisan and Amir Ronen. Algorithmic mechanism design. *Games and Economic Behaviour*, 35:166 – 196, 2001. A preliminary version appeared in STOC 1999.

Noam Nisan and Amir Ronen. Computationally feasible vcg mechanisms. *J. Artif. Int. Res.*, 29(1):19–47, May 2007. ISSN 1076-9757.

Noam Nisan and Ilya Segal. The communication requirements of efficient allocations and supporting prices. *Journal of Economic Theory*, 129(1):192–224, 2006.

Noam Nisan, Tim Roughgarden, Eva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory.* Cambridge University Press, New York, NY, USA, 2007. ISBN 0521872820.

M. J. Osborne and A. Rubistein. *A Course in Game Theory.* MIT press, 1994.

Christos H. Papadimitriou, Michael Schapira, and Yaron Singer. On the hardness of being truthful. In *FOCS*, pages 250–259, 2008.

A. D. Procaccia and M. Tennenholtz. Approximate mechanism design without money. In *Proc. of 10th EC*, pages 177–186, 2009.

Kevin Roberts. The characterization of implementable choise rules. In Jean-Jacques Laffont, editor, *Aggregation and Revelation of Preferences. Papers presented at the first European Summer Workshop of the Economic Society*, pages 321–349. North-Holland, 1979.

Jeffrey S. Rosenschein and Gilad Zlotkin. *Rules of Encounter: Designing Conventions for Automated Negotiation Among Computers.* MIT Press, 1994.

Alvin E. Roth. The economist as engineer: Game theory, experimentation, and computation as tools for design economics. *Econometrica*, 70(4):1341–1378, 2003.

Tim Roughgarden and Inbal Talgam-Cohen. Optimal and near-optimal mechanism design with interdependent values. In *Proceedings of the fourteenth ACM conference on Electronic commerce*, pages 767–784, 2013.

Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions. In *IJCAI-99*, 1999.

James Schummer and Rakesh V. Vohra. Mechanism design without money. In N. Nisan, T. Roughgarden, E. Tardos, and V. Vazirani, editors, *Algorithmic Game Theory*. Cambridge University Press, 2007.

S. Shenkar, Clark D. E., and Hertzog S. Pricing in computer networks: Reshaping the research agenda. *ACM Computational Comm. Review*, pages 19–43, 1996.

Hal R. Varian. Economic mechanism design for computerized agents. In *Proceedings of the First Usenix Conference on Electronic Commerce*, New York, July 1995.

W. Vickrey. Counterspeculation, auctions and competitive sealed tenders. *Journal of Finance*, pages 8–37, 1961.

Berthold Vöcking. A universally-truthful approximation scheme for multi-unit auctions. In *Proceedings of the Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '12, pages 846–855, 2012.