Consensus Mechanisms: Anytime Strategyproof Mechanisms for Combinatorial Auctions

A Thesis presented

by

Edward William Naim

to

Computer Science

in partial fulfillment of the honors requirements

for the degree of

Bachelor of Arts

Harvard College

Cambridge, Massachusetts

April 6, 2004

Acknowledgements

I would like to thank Professor David Parkes, my Computer Science advisor, for the tremendous amount of time and brainpower he invested in helping me make this thesis a reality. Undoubtedly, writing my thesis has been one of the most rewarding academic experiences of my undergraduate career, and I owe most of that to him. I could not have asked for more from my advisor.

I also would like to thank Professor Alvin Roth, my advisor from the Economics department. My conversations with him enabled me to view the ideas I was exploring from a different perspective and pushed me to take my thoughts to the next level. I firmly believe that the intense, challenging discussions I had with him strengthened my thesis significantly.

Table of Contents

Chapter 1: Introduction

1.1 Mechanism Design

Mechanism design is a subfield of microeconomic theory and game theory that focuses on solving distributed system-wide optimization problems among a collection of independent, self-interested agents who have private preference information [Parkes, 2001]. Each agent that interacts with a mechanism is assumed to have a strategy which dictates how the agent will act in every feasible state of the world. From a high level perspective, a mechanism defines the strategies that are available to each agent and provides a mapping from agent strategies to an outcome. Thus a mechanism can be thought of as an object that contains two items, a set of feasible agent strategies and a function that implements an outcome rule based on the actual strategies chosen by the agents. An instance of a mechanism is simply the mechanism running with a specific set of agent strategies chosen from the set of feasible strategies.

Although typically the mechanism designer himself does not have control over the specific strategies that agents choose, he aims to create rules that allow the mechanism to achieve particular economic goals, such as allocative efficiency, without knowing *a priori* the specific strategies that agents will choose, only the set of feasible strategies. In a distributed system setting, the mechanism designer usually attempts to massage the distributed problem he is examining into one in which agents will be best off, given their strategies, by acting in a way that implements system-wide goals, such as efficiency. Designing mechanisms to attack tough distributed problems, such as the allocation of

computing resources across a distributed network, routing responsibility in a sensor network [Greenberg and Naim, 2004], or the creation of a work plan among a large collection of independent agents, is becoming increasingly commonplace. In fact, Ng et al. argues that system designers must explicitly address the self-interest of individual parties in the new era of distributed networks if these systems are to succeed [Ng et al., 2000].

While traditional mechanism design has focused almost exclusively on the game-theoretic properties of mechanisms, computational mechanism design examines not only economics issues but also issues of computational tractability. Although the traditional mechanism designer, ignoring computational considerations, attempts to provide economic guarantees about a mechanism, the computational mechanism designer strives also for computational guarantees, and he thus is concerned not only with economic considerations but also with issues of computational complexity and algorithm design. To illustrate the distinction between the two fundamentally different aspects of mechanism study, Table 1 lists a few examples of economics and computational results that are often presented in research.

Table 1: Examples of Mechanism Design Results

Economics Results	*Computational Results*
The mechanism is allocatively efficient, for it chooses an allocation such that no other allocation can provide higher value across all agents.	The allocation algorithm employed by the mechanism can be reduced to a certain problem which we know to be NP-complete.
The mechanism employs a winner determination rule that makes it every agent's value-maximizing strategy to bid truthfully.	We can use a different winner determination algorithm in this mechanism in order to reduce its runtime to one that is linear in its input.

Table 1: Examples of Mechanism Design Results

Among all efficient and interim individually rational mechanisms, the Vickrey-Clarke-Groves mechanism maximizes the expected transfers from agents.	We can place bounds on the approximation ratio, compared to an optimal algorithm's solution in this Vickrey-Clarke-Groves-based mechanism, that this algorithm provides.
This mechanism is budget balanced, so it does not need external funding.	No algorithm can implement this mechanism and have a worst-case computational complexity that is linear in its input.

It is unfortunate that a great deal of research focuses on one of the two approaches to mechanism design while ignoring the other. Focusing solely on the economic properties of a mechanism is dangerous because it often leads to designs which are computationally infeasible for real-world use. Similarly, because achieving computational goals, such as tractability, often requires the designer to weaken some of the mechanism's game-theoretic properties, it is risky to attempt to improve a mechanism's computational properties without considering the economic ramifications of the optimizations. Since designers need to find the right trade-off of economic goals for computational efficiency, the study or construction of a mechanism needs to be tackled with the tools of both economics and computer science; this dual approach is the foundation of computational mechanism design.

1.2 Anytime Strategyproofness

One of the key challenges in computational mechanism design is to design a mechanism that is both *strategyproof* (it is the dominant strategy of every agent to submit bids that truthfully represent the agent's values for goods) and tractable. By tractable, we mean

that the algorithms the mechanism implements in determining an instance's outcome are tractable.

Mechanisms that implement combinatorial auctions have received a great deal of attention lately. Unlike traditional auctions, in which agents express values for and are rewarded individual goods, combinatorial auctions allow agents to bid on collections, or *bundles*, of goods. Combinatorial auctions are powerful because they enable bidders to express complementary preferences. For example, a bidder may value good A alone at $1 and good B alone at $1, but may value them together at $20. He would not want to bid $10 on each of them separately, for he has no guarantee that if he receives one he will receive the other. Yet if he could express to the auctioneer that he values them together at $20, another bidder that values each at $2 a piece and together at $5 would not win the two goods over him. This expression of complementaries is impossible with standard single-good auctions. Traditionally, combinatorial auctions implement a *winner determination* algorithm, which dictates how the goods are to be partitioned, and who is to receive each partition. Unfortunately, determining the optimal allocation is NP-complete [Vries and Vohra, 2001].

Virtually all previous work on reducing the computational complexity of optimally solving winner determination while maintaining strategyproofness has approached the problem by assuming restricted domains of agent preferences. In the realm of combinatorial auctions, for example, Lehmann et. al. examines the special case of truthful *single-minded bidders*, bidders who have a positive value for only one bundle. Mu'alem and Nisan propose techniques for constructing computable truthful mechanisms in the single-

minded bidder case where the mechanism can verify that the bundle on which each agent places a bid is truly the one that agent is interested in, even though the mechanism does not know the value the agent holds for the bundle. Archer et. al. proposes an approximation that is truthful but again assumes single-mindedness. Bartal et al. moves beyond single-mindedness, but relies on the assumption that there are several units of each good, and each bidder desires only a small number of units [Parkes and Schoenebeck, 2004].

Assuming single-mindedness, or restricting the domain of agent preferences in some other way, greatly reduces the power and usefulness of combinatorial auctions. Clearly, there is room for a mechanism that is able to make guarantees about truthfulness and has good computational properties but does not make strong assumptions about agent preferences or some other aspect of the system. Since determining the optimal winner is NP-complete, if a mechanism is to be tractable yet not make restrictions on the input domain, it will need to provide an approximation to the optimal solution. Ideally, the auctioneer should be able to control the quality of the approximation, deciding what is an appropriate level of trade-off between solution quality and run time.

We propose an anytime, strategyproof (with high probability) mechanism that tractably approximates solutions to the winner determination problem. By *anytime* we mean that if the mechanism is stopped at any point, it will be able to return an allocation of goods to agents; the longer the mechanism runs, the weakly better the approximation gets. By *strategyproof*, we mean that each agent has proper incentive to elicit its true values in all bids it makes. *Strategyproof with high probability* implies that with high probability each agent has incentive to elicit true values. Our mechanism does not have the strong lim-

itations on agent preferences that are imposed by the aforementioned mechanisms, and it does not require the designer to anticipate the complexity of the problem instance. All previous works that have proposed anytime approximation mechanisms, many of which are based on local search, are unfortunately not strategyproof. A contemporaneous work by Parkes and Schoenebeck proposes GrowRange, an anytime strategyproof mechanism; however, our mechanism is fundamentally different from GrowRange, and we explain this difference in Chapter 4.

We have written a robust, comprehensive combinatorial auction environment to analyze our mechanism experimentally, and we present the promising results that the experiments have shown. In particular, the results show that the restrictions placed on the mechanism's search process to make it strategyproof do not cripple its ability to provide high-quality approximations quickly, and that its performance is on par with other non-strategyproof approximation mechanisms that have been proposed.

1.3 Main Results

In this thesis, we present a number of theoretical and experimental results, which are summarized here:

1: Theoretically, our mechanism is strategyproof with a probability that can be fine-tuned by the entity running it. A trade-off exists between this probability and approximation quality. One factor external to the mechanism and specific to the domain in which the auction is run, a parameter we refer to as ρ, affects the

approximation quality without influencing the strategyproofness probability. This trade-off and the significance of ρ are confirmed experimentally.

2: Theoretically, if our mechanism is strategyproof with probability 0, it reduces to random walk hill-climbing. If it is strategyproof with probability 1, it reduces to random walk. In between these two extremes is where our mechanism can shine. These two flavors of random walk are confirmed experimentally.

3: Experimentally, we show that the restrictions placed on the mechanism which allow it to be strategyproof with high probability do not cripple its ability to provide high-quality approximations quickly, and that its performance is on par with other non-strategyproof approximation mechanisms that have been proposed.

4: Experimentally, we show that values of ρ which are suitable for many real-world domains allow the entity running the mechanism to choose both high probability of strategyproofness and good approximation quality.

1.4 Outline

Chapter 2 of this thesis explains concepts in detail from mechanism design that are relevant to our work. Chapter 3 presents an in-depth introduction to combinatorial auctions, and Chapter 4 discusses the anytime strategyproof property for which we aim. Chapter 5 presents the mechanism, and chapter 6 analyzes it theoretically. In chapter 7 we describe our experimental setup, and in chapter 8 we present the experimental results. Finally, Chapter 9 concludes the thesis and describes areas of future work.

Chapter 2: Mechanism Design

2.1 Introduction

Game theory is a way of modeling agent behavior in a system in which the agents interact with one another strategically. We provide here a brief discussion of terms and concepts from game theory that will be relevant to our later discussion. Please note that some of these definitions were adapted from [Parkes, 2001].

2.2 Basic Definitions

We begin with the concept of *type.* An agent's *type* defines its preferences over different outcomes of a game. More formally, we let $\theta_i \in \Theta_i$ define agent *i*'s type from the set of all possible types Θ_i that agent *i* can have. Intuitively, θ_i represents the preferences that an agent has for every outcome that a game could produce. An agent's utility for a particular outcome $o \in O$, where *O* is the set of all possible outcomes, can then be expressed by a utility function, which takes θ_i and *o* as parameters and returns a numerical value. Agent *i* prefers outcome o_1 over o_2 according to utility function *u* if $u(o_1, \theta_i) > u(o_2, \theta_i)$.

The actions that an agent performs during every point in the game are dictated by its strategy. We can define an agent's *strategy* as its complete plan of action for every possible state of the game. An agent's strategy is based on its type, so we can represent agent

i's strategy as $s_i(\theta_i)$. For example, in a single-item ascending auction setting, the state of the world for agent *i* can be represented by the tuple (*a*, *b*, *x*), where *a* represents the good for sale, *b* represents the current highest bid, and *x* is a boolean value indicating whether agent *i* is the agent that placed the current highest bid. Agent *i*'s strategy $s_i(\theta_i)$ would specify the bid *bid*(*a*, *b*, *x*) that the agent should place in every possible state of the auction. One strategy could be:

$$bid(a, b, x) = \begin{cases} b + 1 \text{ if } a = \text{Paul Pierce autograph and } x = \text{ false and } b + 1 < 100 \\ \text{NULL otherwise} \end{cases}$$

According to this strategy, the agent is only interested in bidding on one particular item, a Paul Pierce autograph, and the agent will bid up to $100 for that item. The set *s* of all strategies $s_i(\theta_i)$ that agents choose in a mechanism is commonly referred to as the mechanism's strategy profile; thus $s = \{s_1(\theta_1), ..., s_n(\theta_n)\}$, where *n* is the number of agents in the system.

A necessary part of any auction is the transfer of payments from agents to the mechanism in exchange for goods. We define the outcome space such that an outcome $o = (k, t_1, \ldots t_n)$ defines a choice $k(s) \in K$, where *K* is the set of possible choices, and a transfer $t_i(s)$, which is a real number, from each agent *i* to the mechanism, given strategy profile $s \in S$, where *S* is the set of all possible strategy profiles. The choice in an auction is typically the decision about which goods to allocate to which bidders. For example, in the combinatorial auction example described by Tables 2 and 3 in Chapter 3, the choice

rule might be that the allocation which maximizes value across all agents is the one that is chosen, meaning that Bidder 1 is awarded no goods and Bidder 2 is awarded goods A and B. The transfer rule might specify that each agent pay the mechanism the agent's value for the goods it is awarded, so Bidder 1 would pay the mechanism \$0 ($t_1 = \$0$)and Bidder 2 would pay the mechanism \$20 ($t_2 = \$20$).

As is common in mechanism design, we assume throughout this paper that agents have quasilinear preferences, meaning that $u_i(o, \theta_i) = v_i(k, \theta_i) - t_i$, where $v_i(k, \theta_i)$ is the valuation function for agent *i*. A *valuation function* represents the value that an agent holds for allocation *k* based on the agent's type.

Although we have been using the term "game" rather loosely up to this point, we can now define it formally. A *game* defines the complete set of actions that are available to agents, as well as a mapping from agent strategies to an outcome. Since a game maps the set of agent strategies to an outcome, the utility function for agent *i* that we defined before as $u_i(o \in O, \theta_i)$ can be rewritten as $u_i(s_1, s_2, \ldots, s_n, \theta_i)$, where *n* is the number of agents in the game and s_j represents the strategy of agent *j*. Thus the utility of each agent defines its preferences over its own strategy and the strategies of the other agents in the world, given its type θ_i.

As an example of how an agent's utility could depend on the strategies of other agents, consider a single-item auction consisting of two agents. Each agent's strategy is to bid \$1 over the current ask price if the agent values the good more than the current asking price, and not to place a bid otherwise. Agent 2 has value \$100 for the item, so its strategy

is to bid up to $100 for the item. Agent 1 has value v_1 for the item. Its utility, based on its own strategy, the strategy of the other agent, and its value for the item (which is encapsulated in θ_i), will thus be v_1 - 101 if $v_1 > 101$, and 0 otherwise. Clearly, then, its utility depends on its own strategy, the strategy of agent 2, and its type.

A mechanism defines both the set of strategies available to every agent and an outcome rule that is based on the strategies of the agents. More formally, a *mechanism* specifies the set of strategies $\{\Sigma_1, \ldots, \Sigma_n\}$ that are available to the agents, where Σ_i is the set of strategies available to agent i and there are n agents, and provides an *outcome rule* $g\colon \Sigma_1 \times \ldots \times \Sigma_n \rightarrow O$ that is a mapping from agent strategies to outcomes. Thus, fed the specific strategies of the agents, g would return an outcome that is based on the strategies and on the outcome rule. A mechanism can thus be represented as $M = (\Sigma_1, \ldots, \Sigma_n, g)$.

The outcome rule g that the mechanism implements is based on a *social choice function f*, which maps agent types to an outcome; that is, $f\colon \Theta_1 \times \ldots \times \Theta_n \rightarrow O$.

Although there are many possible social choice functions, the one most relevant to our discussion is that of *allocative efficiency*. A social choice function is said to be allocatively efficient in an auction environment if the total value of the resulting allocation across all agents cannot be larger with a different allocation. That is, if:

$$\sum_{i=1}^{n} v_i(x, \theta_i) \geq \sum_{i=1}^{n} v_i(x', \theta_i) \;\forall\; x' \in X,$$

where X is the set of all possible allocations, x is a member of X, and v_i is a function representing agent i's value for the allocation that is passed to the function, based on the type θ_i that is also passed to the value function.

The mechanism design problem is typically to structure the mechanism, by defining possible strategies and making sure that the outcome rules align with the strategies, in such a way that the social choice function is implemented regardless of agent self-interest. The mechanism asks agents for their types, and uses these reported types to generate an outcome that is consistent with the social choice function. It is critical to note that an agent does not necessarily have incentive to report its true type to the mechanism, and thus an agent may attempt to alter the outcome by intentionally deceiving the mechanism.

2.3 Direct Revelation Mechanisms

We focus on *direct-revelation* mechanisms, in which the only actions that agents perform are the submission of direct claims about their preferences to the mechanism. More formally, a direct-revelation mechanism restricts the strategy set Σ_i of each agent i to Θ_i, meaning that the strategy set of each agent becomes the set of possible preferences that an agent could submit. That is, the strategy of each agent i is to report type $\overline{\theta}_i \in \Theta_i$ based on actual type $\theta_i \in \Theta_i$; thus $\overline{\theta}_i = s(\theta_i)$. The mechanism has outcome rule $g\colon \Theta_1 \times \ldots \times \Theta_n \rightarrow O$, which maps from preferences to an outcome. Agents submit claims about their preferences, and g essentially takes the types of the agents, *as reported*

by the agents, and maps these to an outcome. That is, $g(\overline{\theta}_1 \ldots, \overline{\theta}_n) = o$, where $o \in O$. Note that the reported type $\overline{\theta}_i$ of agent *i* is not necessarily the agent's actual type θ_i.

The most common way in which agents make claims about their preferences to a mechanism is through bids. The traditional open-bid ascending-price auction, in which the auctioneer sets the initial current bid, agents continuously raise the current bid price until no agent is willing to raise it further, and the item is then awarded to the bidder who bid the current price, is an example of a direct-revelation mechanism. It is a direct-revelation mechanism because the only involvement that the agents have with the mechanism is the revelation of claims, via bids, about their preferences.

2.4 Strategyproof Mechanisms

A *truth-revealing* strategy is one such that $s(\theta_i) = \theta_i \; \forall \; \theta_i \in \Theta_i$. A direct-revelation mechanism is *strategyproof* if it is the dominant strategy of each agent to have a truth-revealing strategy; that is, for agent *i*,

$$u_i(g(\theta_i, \theta_{-i}), \theta_i) \geq u_i(g(\overline{\theta}_i, \theta_{-i}), \theta_i) \quad \forall \theta_{-i}, \forall \overline{\theta}_i \neq \theta_i,$$

where $\overline{\theta}_i$ is the type reported by agent *i* and θ_{-i} represents the types reported by all other agents. In words, an agent reporting a type other than its true type cannot have higher utility than that associated with reporting its true type. Since in an auction agents may have incentive to report types that are different from their true types, leading to an outcome that does not properly implement what the social choice function would have dictated if

agents' true types were reported, strategyproofness is an important property in many auction settings.

Perhaps the most well-known family of strategyproof auction mechanisms is the Groves family. A groves mechanism is defined by choice rule

$$k^*(\bar{\theta}) = \arg\max_{k \in K} \sum_i v_i(k, \bar{\theta}_i)$$

and transfer rules

$$t_i(\bar{\theta}) = h_i(\bar{\theta}_{-i}) - \sum_{j \neq i} v_j(k^*(\bar{\theta}), \bar{\theta}_j),$$

where h_i is an arbitrary function that is independent of the reported type of agent i. In words, the choice rule implements the choice that maximizes value across all agents, based on reported type information, and the transfer rule specifies that each agent i must pay the mechanism an amount equal to the reported value that all other agents hold for the choice subtracted from some arbitrary amount that is independent of agent i's reported type. Groves (1973) proved that mechanisms of this form are strategyproof and efficient, and Green and Laffont (1977) showed that Groves mechanisms are the only direct-revelation ones which are allocatively-efficient in a dominant strategy equilibrium.

A Vickrey-Clarke-Groves (VCG) mechanism is a Groves mechanism that defines the h_i function as

$$h_i(\bar{\theta}_{-i}) = \sum_{j \neq i} v_j(k^{-i}, \bar{\theta}_j),$$

where $k^{-i} = \max_k \sum_{j \neq i} v_j(k, \overline{\theta}_j)$. In words, h_i, when incorporated into the Groves transfer rule, is thus enforcing that the price agent i should pay is equal to the difference between the value of the best allocation across all other agents if agent *i* were not in the system and the value of the actual allocation across all other agents with agent *i* in the system. If agent *i* enters the system and wins goods, the total value of the system's outcome across all other agents is weakly reduced from the total value before the agent had entered, for if i wins goods then in essence the number of goods available to other agents is reduced. This possible reduction of available goods to other agents means that the total value of the outcome to them must be less than or equal to the value before agent *i* entered. This weak reduction of values for the other agents can be viewed as a transfer of value from them to agent *i,* and thus the VCG transfer rule dictates that agent *i* must pay the mechanism a payment equal to this transfer.

It is trivial to understand why a VCG mechanism is allocatively efficient, for the choice rule is in essence the definition of optimal efficiency. The proof of a VCG mechanism's strategyproofness is fairly straightforward and is omitted in the interest of space.

Chapter 3: Combinatorial auctions

3.1 Introduction

In this chapter we define combinatorial auctions, define the COMBINATORIAL AUCTION WINNER DETERMINATION problem and characterize its computational properties, and describe two primary classes of algorithms that attempt to approximate the optimal solution to a WINNER DETERMINATION problem.

3.2 Combinatorial Auctions and Winner Determination

Unlike traditional auctions, in which agents express values for and are rewarded individual goods, combinatorial auctions allow agents to express values for and win collections, or *bundles*, of goods. Combinatorial auctions are powerful in that they enable bidders to express complementary preferences. This ability to express complementarities often results in a more efficient allocation of goods to bidders. For example, suppose that a seller decides to hold an auction for two items with two bidders, and let us assume that the bidders have the values for goods described in Table 2.

Table 2: Agent Values

	Good A	Good B	Goods A and B Together
Bidder 1	$2	$2	$6
Bidder 2	$1	$1	$20

Table 3: System-Wide Allocation Values

Choice	Allocation	Value	Value′
1	bidder 1: {A} bidder 2: {B}	$3	$3
2	bidder 1: {B} bidder 2: {A}	$3	$3
3	bidder 1: {A, B} bidder 2: {}	$4	$6
4	bidder 1: {} bidder 2: {A, B}	$2	$20

In Table 3, Value represents what the mechanism would calculate as a choice's value if it did not have access to the last column in Table 2, and Value′ represents the value the mechanism would compute if it did have access to that last column. In an efficiency-maximizing auction, if agents could not express complementary preferences—that is, if the mechanism did not have access to the last column in Table 2—Bidder 1 would win both goods, for he would be willing to bid up to $2 for each, while Bidder 2 would only be willing to pay up to $1 for each. Choice 3 would be chosen. Clearly, though, the most economically efficient allocation corresponds to choice 4, in which both goods would be awarded to Bidder 2. Without the ability of bidders to express complementary preferences, the auctioneer is limited to the information in all but the last column of the two tables, and in this case is forced to incorrectly assume that the allocation of A and B to Bidder 1 is the most efficient one. Thus a combinatorial auction gives the auctioneer more useful information on the values of possible allocations than does a standard single-good auction.

More formally, in a combinatorial auction a seller has a set G of goods it wishes to sell, a set A of agents that can place bids on the goods, and for each agent $a \in A$ a valuation function V_a: $b \rightarrow \mathbb{R}_+$, where $b \subseteq G$. We assume *free disposal* of goods, meaning that, if S and T are bundles in which agent a is interested, $S \subseteq T \Rightarrow V_a(S) \leq V_a(T)$. B is the set of all bundles for which at least one agent is interested in bidding. We can view every bid on a bundle b as an agent's reported value $\overline{V_a(b)}$ for that bundle; a bidding language, then, is a formal way of expressing valuations. Accompanying well our assumption of free disposal, we assume in this thesis that our bidding language is an *XOR bidding language*, meaning that:

(1) a bidder can submit as many bids as he wishes, but is willing to obtain at most one of the bundles on which he has bid.

(2) for a bundle b, an agent's bid price must be at least the maximum value of his bid prices on all subsets of b on which he has also bid [Nisan, 2000].

A *partition* is a splitting of goods into non-overlapping bundles. We let B' be the set of all subsets of B whose members are disjoint; thus B' is essentially the set of all feasible partitions that contain only bundles in B. An *allocation* is an assignment of bundles in a partition to agents. If agents pay the mechanism a price equal to the reported values of the bundles they win, the revenue of an allocation $X \in B'$ is simply

$$\sum_{b \in X} \max_{a \in A}(\overline{V_a(b)}),$$

where A is the set of all agents. The auctioneer typically aims to find the member $X^* \in B'$ that maximizes this revenue. It is interesting to note that if we define revenue in this way, X^* is not only the partition that maximizes the auctioneer's income but also is the partition that maximizes total value across all agents. Finding X^* is commonly referred to as the COMBINATORIAL AUCTION WINNER DETERMINATION problem.

3.3 Solving the Winner Determination Problem

Putting incentives to one side for the moment, many approaches to solving WINNER DETERMINATION have been proposed. The most straightforward method of solving the problem is via an exhaustive search of the partition space; that is, the mechanism looks at every possible partition of goods into bundles and determines the one that has the maximal value across all agents. In an auction with goods A, B, and C, the partition space, assuming that all goods are to be sold, would consist of the partitions [{A}{B}{C}], [{AB}{C}], [{A}{BC}], [{AC}{B}], and [{ABC}], where the letters in the inner curly braces correspond to the goods in a bundle. An exhaustive search would visit each partition, assigning each a value corresponding to its calculated revenue (based on the bids that the agents submit for bundles within each partition). The partition with highest value that the search algorithm encounters is the one that is returned. Obviously, an exhaustive search is computationally poor, for it is searching a space in which the number of search items is exponential in the number of goods.

Unfortunately, in the worst case we cannot do much better computationally than exhaustive search if we hope to solve the WINNER DETERMINATION problem, for WINNER DETERMINATION is NP-complete, and thus any algorithms which attempt to solve it optimally will require time that is exponential in the number of goods in the worst case (assuming NP does not equal P). It is useful to note that the COMBINATORIAL AUCTION WINNER DETERMINATION problem is equivalent to the WEIGHTED SET PACKING problem. In the WEIGHTED SET PACKING problem, there are m objects and there exists a set S of sets. Each member s_i of S contains at most $k \leq m$ objects, and each s_i has associated with it a weight. The goal is to find the collection C of disjoint sets that maximizes total weight. WEIGHTED SET PACKING is NP-complete, and therefore COMBINATORIAL AUCTION WINNER DETERMINATION is also NP-complete [Vries and Vohra, 2001].

Although WINNER DETERMINATION is NP-complete, a number of algorithms have been proposed that work well in practice on particular distributions of bids. Sandholm et. al. proposed a structured search algorithm, a fast optimal algorithm called CABOB, which relies upon a depth-first branch-and-bound tree search technique that branches on bids, using decomposition techniques, upper and lower bounding, bid ordering heuristics, and a mix of structural observations [Sandholm et al., 2001]. Collins provided a fast extension to Sandholm's search algorithm [Collins, 2002].

Another set of algorithms uses integer programming to solve combinatorial auctions. [Andersson et al]. The WINNER DETERMINATION problem is modeled as an integer program which has an objective that is subject to a series of constraints. Typically, the objective is to determine the allocation that maximizes the sum of the values that the

agents hold for the allocation, subject to the constraints that no good can appear more than once in an allocation and that agents cannot hold fractions of a good.

Nevertheless, because COMBINATORIAL AUCTION WINNER DETERMINATION is NP-complete, we know that any mechanism which attempts to find optimally the *k* defined by Groves mechanisms will be NP-complete; thus it is not necessarily ideal to solve the problem optimally. Some researchers have proposed using local search methods, like simulated annealing and Tabu search, to provide approximation mechanisms [Hoos, 2000], [Collins, 2002], [Naim, 2003], mechanisms which implement an outcome that is an approximation to the optimal one by running an allocation algorithm that approximates the solution to WINNER DETERMINATION. A nice property about local search algorithms is that most are anytime; the algorithms can be stopped at any point and return the best solution, or partition, they have encountered so far. Another nice property is that the search direction is guided by what the mechanism has learned about the search space based on past bids, and thus can be expected, and has been shown, to perform better than a blind search. As Hoos and Boutilier explain, "[Stochastic local search] has been used in AI and operations research for many decision and optimization problems with great success, and has generally proven more successful than systemaic methods on a wide range of combinatorial problems...[it] can be applied with great success to the winner determination problem, finding high quality solutions much more quickly than systematic techniques and also finding optimal solutions" [Hoos and Boutilier, 2000].

3.4 Local Search and Casanova In Depth

Because our Consensus Mechanism is based on local search, in this section we present in detail our version of directed hill-climbing local search, and then we present in detail Casanova, the most widely known example of a published local search approximation mechanism. Unlike Casanova and other published local-search-based techniques, our local search runs a linear program in each step in order to determine the values of neighbors.

LP-Based Local Search

The local search we present depends on a linear program being solved at each step. In our version of directed hill-climbing local search, the mechanism begins with a random partition π, which we will refer to as the *current best partition*. Based on some notion of "neighbor" decided *a priori*, at each step in the local search the mechanism chooses *m* neighboring partitions of π. It elicits bids from each of the participating agents for every bundle found in the *m* neighboring partitions as well as for every bundle in π. Based on these bids, it then uses a linear program to compute the partition that has the highest total value across all agents, makes this partition the current best partition π, and then repeats the process. At any point, the mechanism can be stopped and will return the current best partition π (often a simple auction, like a VCG one, is then run on π to determine which bundles to allocate to whom and at what prices).

Below is pseudocode for a basic local search algorithm.

```
function LocalSearch(NumberNeighborsToConsider)
{
      π:=RandomPartition()
      m:= NumberNeighborsToConsider
      while(!stopped) {
            NeighborsList = GetNeighbors(π, m)
            π = LocalRoundWinner(NeighborsList)
      }
      return π
}
```

The `GetNeighbors` function takes as input the current best partition π and the number of neighbors to consider, and returns a random list of neighbors of π; this list also includes π itself. The `LocalRoundWinner` function takes as input a list of partitions and chooses a partition to be the winner for that round; in the case of our version of local search, it runs a linear program to determine optimally the partition in the current neighborhood that has the highest value across all agents according to the agents' bids. The linear program it uses is presented in section 5.4. The basic local search algorithm shown above can be easily extended to allow for more sophisticated local search techniques, like stochastic local search and tabu search; these extensions would simply be incorporated into the `LocalRoundWinner` and `GetNeighbors` functions, leaving the skeleton code above essentially unchanged.

By running a linear program at each step to optimally solve the winner determination problem on the set of partitions in the current neighborhood, our local search is using a technique called *maximal-in-range,* meaning that in each step it is selecting the partition that yields the optimal allocation, but is doing so only over the set of partitions currently in

the neighborhood. Thus in each step our local search solves the winner determination problem optimally, but only for a limited range.

Casanova

The idea of applying local search to combinatorial auctions for approximating solutions to the winner determination problem was proposed in 2000 by Hoos and Boutilier [Hoos and Boutilier, 2000]. Their Casanova algorithm has been shown to perform well under several different experimental settings.

The authors of Casanova define a neighbor of partition π to be one in which a bundle corresponding to a bid that is not satisfied by the current partition is added to the partition and all overlapping bundles are removed. By unsatisfied, we mean that the bundle is not in the current partition. Using this neighbor definition, a value *sc(b)* can be used to represent the change in revenue obtained by adding the bundle *b* to the current partition and removing all overlapping bundles. Casanova employs the notion of "revenue per good," meaning that these scores are normalized by the number of goods in the bundle; thus the scoring function is $score(b) = sc(b)/length(b)$. Revenue per good is commonly used to measure the quality of a bundle in search-based approaches to combinatorial auctions. Hoos and Boutilier also define the *age* of a bundle to be the number of steps since that bundle was last selected to be added to a candidate partition.

Casanova begins by collecting bids from every agent on the bundles in which each agent is interested. The search begins with an empty allocation, and at each step with probability *wp* selects a random unsatisfied bundle, and with probability 1 - *wp* selects an

unsatisfied bundle greedily by ranking all unsatisfied bundles by score. If a bundle is chosen greedily by ranked score, either the highest-ranked bundle b_h or the second-highest bundle b_s is chosen as follows: if $age(b_h) \geq age(b_s)$, then choose b_h; otherwise, with probability *np* choose b_s, and otherwise choose b_h. The search proceeds for *NumberSteps* steps and is restarted with an empty allocation every *NumberStepsUntilRetry* steps. At the end of its run, Casanova returns the best allocation found at any step of the algorithm.

Clearly, unlike the flavor of local search we have presented, Casanova does not solve a linear program in each iteration, and thus is not maximal-in-range.

Hoos and Boutilier's experimental results showed that local search, and in particular Casanova, was indeed a promising anytime technique. However, as described in the next section and in Chapter 4, the fundamental problem with local-search-based algorithms like Casanova is their susceptibility to manipulation by untruthful agents. We propose a mechanism that harnesses the power of local search, yet is resilient to unilateral manipulation by any single agent.

3.5 Addressing Incentives

Although the local search algorithms that have been proposed do cut back on the computational complexity inherent in the optimal algorithms, they must do so in return for the introduction of an approximation. The quality of this approximation is inherently tied to how well the mechanism searches the partition space, and this in turn depends on how well the agent bids guide the mechanism. Because the search direction is determined by

the bids that agents submit, agents may have incentive to unilaterally manipulate the direction of the search, and thus the outcome of the auction, by reporting false values for bundles. These local search algorithms do not provide proper incentives for agents to bid truthfully, and thus are not strategyproof. Because agents can mislead the mechanism, this absence of strategyproofness can have a devastating impact on approximation quality.

In recognition of the importance of strategyproofness, there has been a good amount of work on non-VCG-based strategyproof mechanisms that cut back on WINNER DETERMINATION's computational complexity by assuming restricted domains of agent preferences. These mechanisms can also be thought of as maximal-in-range approximation mechanisms because their solutions are optimal for the subset of actual partitions they consider; thus each solution is an approximation of the solution for the full set of partitions. For example, Lehmann et. al. examines the special case of truthful *single-minded bidders*, bidders who have a positive value for only one bundle. Mu'alem and Nisan propose techniques for constructing computable truthful mechanisms in the single-minded bidder case where the mechanism can verify that the bundle on which each agent places a bid is truly the one that the agent is interested in, even though the mechanism does not know the value the agent holds for the bundle. Archer et. al. proposes an approximation that is truthful but again assumes single-mindedness. Bartal et al. moves beyond single-mindedness, but relies on the assumption that there are several units of each good, and each bidder desires only a small number of units [Parkes and Schoenebeck, 2004]. Unfortunately, we believe that the restrictions these mechanisms place on agent preference domains are quite limiting.

Chapter 4: Anytime Strategyproofness

4.1 Introduction

In the previous chapter we described the two primary classes of approximation algorithms that currently exist. Algorithms of the first use local search's power to provide quality approximations. They are anytime but not strategyproof. The second are non-VCG mechanisms that are strategyproof, but place restrictions on agent preference domains that are quite limiting. We want the best of both worlds. We want a mechanism that has the good approximation and anytime properties of the first class, but with the strategyproofness that comes with the second class, but without the limitations of the second class. This thesis proposes such a mechanism, one that does not have the second class's restrictions on agent preferences, builds on the strengths of local-search-based algorithms to provide quality approximations, and is strategyproof.

However, before we describe our mechanism in Chapter 5, we devote this chapter to a further explanation of the critical property we strive for, anytime strategyproofness, and discuss why this property is so difficult to ensure in a local-search-based mechanism. This chapter also serves as a transition from the mechanisms of the previous chapter to our mechanism in the next chapter by presenting a contemporaneously-proposed anytime strategyproof mechanism, GrowRange, and highlighting the fundamental difference between our mechanism and GrowRange.

We define in this chapter what we mean by a mechanism that is anytime strategyproof, we explain why most approximation mechanisms based on local search are not

strategyproof, we describe GrowRange, and we describe what goals we have for the mechanism we are proposing.

4.2 Anytime Mechanisms

An *anytime* mechanism $M = (\Sigma_1, \ldots, \Sigma_n, g)$ is one in which the g function can be stopped at any point by the controller of the mechanism to return an outcome $o' \in O$. A desirable feature of an anytime mechanism is that at time t+1 the outcome returned by g is weakly better than the one returned at time t; most anytime mechanisms are constructed to have this property. For example, if the social choice function aims for allocative efficiency, an anytime mechanism should be designed such that the value of the outcome across all agents if g were stopped at time $t + 1$ is at least as high as, if not higher, than the value of the outcome if g were stopped at time t. Thus if we graph allocation value over time we should get a monotonic curve. It is possible that the solution returned by g after being stopped at a certain time is optimal.

An anytime approximation algorithm has the nice property that the entity running it can determine the appropriate trade-off between approximation quality and run time by choosing for how long to run the algorithm.

As discussed in the previous chapter, local search algorithms are well-suited to be anytime approximations for WINNER DETERMINATION. There have not been published results on the performance of maximal-in-range local search algorithms, because we present them for the first time here, but experimental results on non-LP-based local search

algorithms have shown the power that these algorithms have. They have been shown to produce quality results in most settings within a short amount of time relative to the time that the optimal solution takes [Hoos and Boutilier, 2000], [Naim, 2003].

4.3 Strategyproof Considerations

Although it has received attention because of its ability to quickly search the partition space, the idea of using pure local search techniques for solving or approximating the winner determination problem has a fundamental limitation; it ignores the issue of agent incentives. In reality, incentives need to be dealt with, for agents may not have incentive to bid their true values on certain bundles. Because in a local search the mechanism determines the value of each neighbor—and in turn determines the best neighbor—based on the values submitted by agents, an agent, by purposely submitting bids that do not represent its true values for certain bundles, can unilaterally manipulate the direction of the search, and thus the outcome of the mechanism.

The problem with such manipulation is that it can reduce the economic efficiency of the allocation that the mechanism returns. As an example, suppose there are three agents participating in a local-search-based auction. The mechanism is currently attempting to choose among three different neighboring partitions. Partition α contains bundles 1, 2, and 3, partition β contains bundles 4, 5, and 6, and partition γ contains bundles 5, 6, 7 and 8. Table 4 shows the actual values that the three agents have for each bundle. Based on the table, it is clear that partition α is the best across all agents, for it has a total value

of 35 (agent 1 is allocated bundle 3, agent 2 is allocated bundle 2, and agent 3 is allocated bundle 1). Yet agent 3 has more value for partition γ. Suppose agents 1 and 2 submit their true values for the bundles. Agent 3, however, is better off by lying to the mechanism. If he submits bids of 0 for all bundles except 8, and submits bids of 100 for 8, the mechanism will choose partition γ as the best partition. Suppose at this point the mechanism is stopped. It will thus return γ rather than α as the best partition encountered, resulting in an allocation that is 22 units lower than it would have been if agent 3 had been truthful. Although agent 3 is better off by manipulating the mechanism, the system as a whole is worse off. This type of manipulation can occur at any point in the local search, not just at the end, and can thus affect the direction that the local search takes.

Table 4

		Bundles							
		1	*2*	*3*	*4*	*5*	*6*	*7*	*8*
Agents	*1*	5	5	15	0	0	0	0	0
	2	2	10	5	0	0	0	0	0
	3	10	2	1	0	1	2	3	13

Clearly, the lack of strategyproofness that these mechanisms have limits their ability to discover allocations that truly are value-maximizing, for an agent can unilaterally fool the mechanism into pursuing partitions that make the agent better off but do not move the search in the direction of allocative efficiency.

4.4 GrowRange: A Strategyproof Anytime Mechanism

Aiming to propose an anytime approximation algorithm that is strategyproof with high probability and does not place limits on agent preferences, Parkes and Schoenebeck have contemporaneously proposed GrowRange, an anytime VCG-based mechanism that is based on the principle of maximal-in-range allocations [Parkes and Schoenebeck, 2004]. An allocation is said to be *maximal-in-range* for a set of bundles S if it is the best allocation of bundles in S to agents. We refer to the bundles in S as the *range* of bundles being considered. The GrowRange mechanism begins with a limited set, or range, of bundles, and determines the value $V(n)$ of the maximal-in-range allocation across all n agents. Then, for each agent i, it calculates the value $V(n/i)$ of the allocation that is maximal-in-range across all n agents except i. It then grows in parallel the partition associated with $V(n)$ and the partition associated with each $V(n/i)$ to include more bundles, updating the $V(n)$ and $V(n/i)$ values along the way. This process continues until the mechanism is stopped, at which point it returns the allocation that is currently maximal-in-range across all agents. The mechanism uses the $V(n/i)$ values to determine payments from the agents for the allocation.

The GrowRange mechanism can be thought of as a search-based algorithm that in each iteration grows the current search space and then finds the maximal-in-range solution within the new, augmented search space. Agents cannot manipulate the direction of the search—that is, the way in which the search space is augmented—because the search space is grown randomly. Agents could manipulate the mechanism by submitting untruthful bids that alter the time the mechanism takes to find the maximal-in-range solution in a

given round and thereby affecting the number of ranges that the mechanism visits in a given amount of time. However, Parkes and Schoenebeck address this problem through the use of a special class of functions called ρ-consensus functions, which make time-based manipulation infeasible with high probability. We too use ρ-consensus functions in the Consensus mechanism we propose, but not to tackle time manipulation, a problem our mechanism does not have, but rather to prevent unilateral misguidance of the mechanism by an agent.

Although both our mechanism and GrowRange are anytime and strategyproof with high probability, and although both mechanisms depend on maximal-in-range calculations, the GrowRange mechanism is fundamentally different from ours. The two employ drastically different algorithms, and comparing their algorithms is like comparing apples to oranges. Nevertheless, we can pinpoint a single fundamental difference between the high-level approaches that the two take: GrowRange lacks guidance in exploring its search space, while the Consensus algorithm is based on directed local search. GrowRange does not use information it has learned from agent bids when deciding which bundles to add to its range; instead, it randomly adds bundles. Although this lack of guidance helps the mechanism's strategyproof property, information extracted from agent bids can be a good heuristic in determining which bundles not in the current range might yield allocations with high value if added.

We propose a local-search-based mechanism that, like GrowRange, is anytime and strategyproof with high probability, but unlike GrowRange is able to have this strategyproof property while also using agent bids to guide the search direction. In addition, our

mechanism allows the entity running it to specify the probability with which it is strategyproof, enabling the appropriate level of trade-off between strategyproofness and efficiency to be specified at runtime. This last property gives the mechanism a good amount of robustness, for it allows it to have good performance across different auction domains that may have different optimal trade-off levels.

GrowRange is undoubtedly a novel mechanism that has produced impressive experimental results. We choose to build a mechanism that is similar to GrowRange in that it is anytime strategyproof and relies upon maximal-in-range calculations, but fundamentally takes a different approach from GrowRange by attempting to employ the power of local search to guide its maximal-in-range search.

Chapter 5: Consensus Mechanism

5.1 Introduction

In this chapter we discuss the details of our mechanism. Please note that for now incentives are ignored, and that in Chapter 6: Theoretical Analysis, we explore incentives and prove the probability with which the Consensus Mechanism described in this chapter is strategyproof.

5.2 ρ-consensus Functions

To achieve our goal of resilience to unilateral manipulation, we employ the use of ρ-consensus functions, functions which Goldberg and Hartline proposed in a different context [Goldberg and Hartline].

We say that g: $\mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$ is a ρ-consensus function at x if:

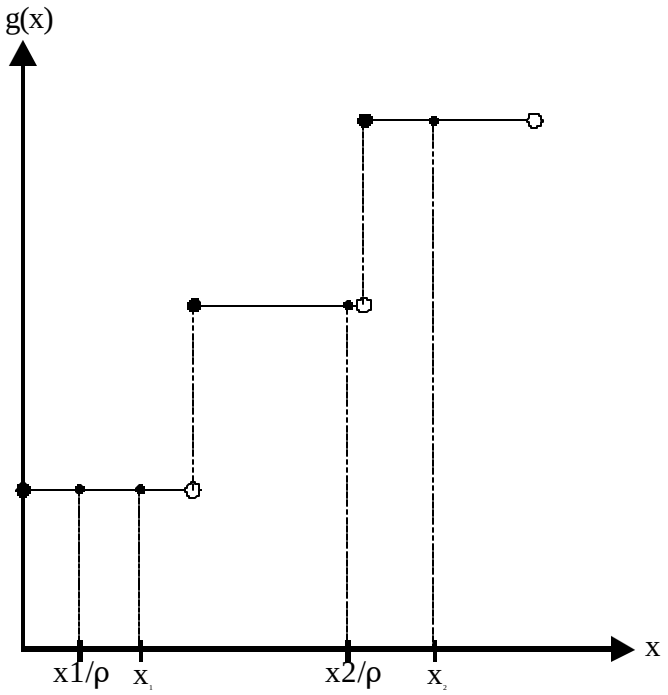
(1) $g(x) \geq x \ \forall \ x$

(2) $g(x) = g(x') \ \forall \ x'$ s.t. $\frac{x}{\rho} \leq x' \leq x$ for some $\rho > 1$.

Thus, as defined, a ρ-consensus function for a number x is essentially a step function that maps x to a value $g(x)$ that is (1) at least as large as x and that (2) is the same for all x' between $\frac{x}{\rho}$ and x. With this definition, if g is a ρ-consensus function for x, any input x' that is between x/ρ and x yields a $g(x')$ value that is independent of the exact

value of x'. Figure 1 illustrates this definition. In this figure, g is a ρ-consensus function for x_1 but not for x_2.

Figure 1: Graphical Representation of a ρ-consensus Function



5.3 High-Level Overview of Consensus Mechanism

From a high-level perspective, our anytime mechanism will employ a local search algorithm to explore the partition space. As described before, the primary drawback to the local search algorithms that have been proposed in the past is their susceptibility to agent manipulation; an agent can control the direction that the search takes by submitting false bids that alter the mechanism's evaluation of one or more neighbors that it is considering in a particular step. To avoid this manipulation, our mechanism, when evaluating a partic-

ular neighbor, will rely on a ρ-consensus function to map the value calculated for that neighbor based on agent bids to a numerical value that no single agent can influence with its bids alone. That is, the mechanism uses ρ-consensus functions in an attempt to map the total value $\overline{V_\pi} = V(\pi, \bar{\theta})$ calculated for a neighbor partition π based on agent bids to a value $V_\pi' = g(\overline{V_\pi})$ that no agent can affect unilaterally by changing $\overline{V_\pi}$. V_π' is then used by the mechanism as an estimate for the value of π. When the mechanism is attempting to evaluate partition π in a particular step of the search, if it is the case that for partition π no agent can unilaterally alter V_π' by changing $\overline{V_\pi}$, then there is said to be a consensus on the value of π, and V_π' is used as an estimate of the value. Otherwise, there is said to be no consensus on π's value and the mechanism ignores π. For the mechanism to perform well in its search, it is important that the entity running it choose a ρ that is suitable for the domain in which the auction is being run; the significance of the ρ value that is chosen, as well as considerations that must be taken into account in determining an appropriate value of ρ, are discussed later in this chapter and in our experimental results presentation in Chapter 8.

5.4 Details of Consensus Mechanism

In our combinatorial auction environment, we let $V(\pi) = V(\pi, \theta)$ be the total value of partition π for all the agents in the auction, and let $V_{-i}(\pi) = V(\pi, \theta_{-i})$ be the

total value of π without agent *i*. By *total value* of a partition we mean the value that results from allocating bundles in the partition to agents in the way that maximizes the sum of the values of the agents for the allocation. The reader can remind himself what "total value" means in our combinatorial auction setting by revisiting the example described by Tables 2 and 3 in Chapter 3. In this example, allocation 1, for instance, has total value \$3 and allocation 4 has total value \$20. We let $\overline{V(\pi)} = V(\pi, \bar{\theta})$ be the reported total value of partition π across all agents, and $\overline{V_{-i}(\pi)} = V(\pi, \overline{\theta_{-i}})$ be the reported value of π without agent *i*. For some $\rho > 1$, it must be the case that, for all agents *i* and all partitions π, $\frac{V(\pi)}{\rho} \leq V_{-i}(\pi) \leq V(\pi)$, assuming $V(\pi) \geq 0$ and $V_{-i}(\pi) \geq 0$. In other words, assuming that agents have bounded and positive values for bundles, there is some $\rho > 1$ such that no agent has values for the bundles in any π that would move the total value of π, with the agent in the system, by greater than a factor of ρ from the total value of π without the agent in the system.

Although later in this chapter we will make more precise the way in which we derive the neighbors of a partition, for the moment assume that the mechanism has a blackbox function which takes as input a partition and returns a list of *m* neighboring partitions. This list of *m* neighbors is a subset of the full set of neighbors that the partition has. Because the consensus mechanism is local-search-based, at each step in the search the mechanism must derive a list of neighbors of the highest-valued partition it has found thus far, and choose as the best partition the neighbor with the highest value (or keep the cur-

rent partition as the best one if it has higher value than the neighbors at which it has looked). If the mechanism needs to evaluate partition π in a particular step of the local search, it determines if g is a ρ-consensus function for π's value by checking that $g(\overline{V_{-i}(\pi)}) = g(\overline{V(\pi)})\ \forall i$. If g is in fact a ρ-consensus function for π's value, then the value associated with π is $g(\overline{V(\pi)})$. If g is not a ρ-consensus function for it, the mechanism ignores π in this step.

We now need to define g. We let $c = \rho^{\frac{1}{(1-q)}}$, where q is the desired probability of consensus and u is chosen uniformly from [0,1]. By "probability of consensus" we mean the probability that g will be a ρ-consensus for the value of a partition π chosen at random. Following Goldberg and Hartline, we define

$$g_{u,c}(x) = \min_{j \in Z}(c^{u+j}) \text{ s.t. } x \le c^{u+j}$$

to be our ρ-consensus function. In words, $g_{u,c}(x)$ chooses and returns the minimum value c^{u+j} such that j is an integer and $x \le c^{u+j}$.

We now give the pseudocode for our algorithm:

```
function Main(NumberNeighborsToConsider, rho, q)
{
      π = RandomPartition()
      π′ = NULL
      m = NumberNeighborsToConsider
      u = RandomDouble(0.0, 1.0)
      while (!stopped) {
            NeighborsList = GetNeighbors(π, m)
            π′ = LocalConsensusWinner(NeighborsList, u, rho, q)
            if (g(Value(π′),u,rho,q) >= g(Value(π),u,rho,q)) {
```

```
                  π = π′
            }
      }

      run VCG(π) and return resulting allocation
}

function LocalConsensusWinner(NeighborsList, u, rho, q)
{
      CurrentBestVal = 0
      ConsensusNeighbors = {}

      for each Neighbor in NeighborsList {

            // if there is a consensus on this lot
            if for all i (g(ValueWithoutI(Neighbor,i),u,rho,q) ==
                        g(Value(Neighbor),u,rho,q)) {
                  if (g(Value(Neighbor)) > CurrentBestValue) {
                        CurrentBestValue = g(Value(Neighbor))
                        ConsensusNeighbors.Clear()
                        ConsensusNeighbors.Add(Neighbor)
                  } else if (g(Value(Neighbor)) == CurrentBestVal){
                        ConsensusNeighbors.Add(Neighbor)
                  } else {
                        // discard the neighbor--do nothing with it
                  }
            }
      }
      if (ConsensusNeighbors == {}) {
            return random Neighbor from NeighborsList
      } else {
            return random Neighbor from ConsensusNeighbors
      }
}

function g(x, u, rho, q)
{
      c = rho ^ (1 / (1 - q))
      j = 0
      current_min = 0

      // keep going until find x <= c^(u+j), in which case return
      // c^(u+j)
      do {
            current_min = c^(u+j)
            j = j +1
```

```
        } while (current_min < x)

        return current_min
}
```

The `GetNeighbors` function returns a list of `m` random neighbors of the partition π that is passed to it. The `Main` function chooses `u` uniformly from [0,1], begins with a random partition, and repeatedly generates random neighbors of π, updating π in each loop to be what is returned by the `LocalConsensusWinner` function if what is returned has a higher consensus value than the current π. The `LocalConsensusWinner` function performs the brunt of the work. For each neighbor in the list that is passed to it, the function determines whether or not there is a ρ-consensus on the value of the neighbor. There is a ρ-consensus on the value of the neighbor if, for every agent i, `g` passed the neighbor value without i yields the same value; that is, $g(\overline{V_{-i}(\pi)}) = g(\overline{V(\pi)}) \ \forall i$. The `LocalConsensusWinner` returns the neighbor that has the highest consensus value, or, if two or more neighbors are tied for having the highest consensus value, the function breaks ties randomly. If none of the neighbors has a consensus value, the function returns a member of `NeighborsList` at random.

We assume that the `Value` function has access to agent bids. The value of a partition, based on the bids submitted by agents, is the revenue-maximizing allocation of bundles in the partition to agents. According to how we defined revenue in section 3.2, we know that the revenue-maximizing allocation is equivalent to the allocation that maximizes system-wide value. If we let B be the total number of bundles in the partition, let A be the total number of agents, let the binary variable x_{ij} represent whether or not agent i

gets good j, and let v_{ij} represent agent i's value for good j, the value of a partition is modelled by the following linear program:

$$\textbf{maximize} \quad \sum_{j=1}^{B} \sum_{i=1}^{A} x_{ij} v_{ij}$$

$$\textbf{subject to} \quad \sum_{i=1}^{A} x_{ij} \leq 1 \forall j \quad \textbf{(1)}$$

$$\sum_{j=1}^{B} x_{ij} \leq 1 \forall i \quad \textbf{(2)}$$

$$x_{ij} \in \{0, 1\} \quad \textbf{(3)}$$

The first constraint ensures that only one agent can hold good j. The second mandates that no agent can hold more than one bundle. The third specifies that x_{ij} is a binary variable. The `ValueWithoutI` function is the same as the `Value` function, except it computes the solution to the above linear program without agent i.

The `g` function simply returns the value that it maps input `x` to according to the definition $g_{u,c}(x) = \min_{j \in Z}(c^{u+j})$ s.t. $x \leq c^{u+j}$, where $c = \rho^{\frac{1}{(1-q)}}$. As illustrated by the pseudocode, `g`'s runtime is linear in its input `x`.

At the end of the `Main` function, we run a VCG auction on the partition that the mechanism has chosen. This VCG auction allocates the bundles in the partition chosen by the mechanism to agents and calculates the transfers required from agents to the mechanism. The details of VCG auctions are given in Chapter 2.

We choose to define a neighboring partition as one that is generated by the following process: A neighbor π' of partition π is generated by choosing a random bundle *b'* not in π, adding *b'* to the initially empty π', and then adding every bundle in π that does not overlap (has no overlapping goods) with *b'* to π'.

Chapter 6: Theoretical Analysis

6.1 Introduction

We aim in this chapter to analyze and prove the incentive properties of the consensus mechanism that we described in Chapter 5, and we also analyze the trade-offs that the entity running the consensus mechanism must make between strategyproofness and approximation quality.

6.2 Strategyproof Analysis

Although the previous chapter describes how the mechanism works, it may not be clear to the reader why with high probability this mechanism cannot be manipulated unilaterally. This chapter analyzes the properties of the mechanism that ensure its non-manipulability.

For convenience, we again state what it means for a function to be a ρ-consensus function for a number. We say that $g: \mathbb{R}_{\geq 0} \rightarrow \mathbb{R}$ is a ρ-consensus function for x if:

(1) $g(x) \geq x \ \forall \ x$

(2) $g(x) = g(x') \ \forall \ x'$ s.t. $\frac{x}{\rho} \leq x' \leq x$ for some $\rho > 1$.

It is important first to note that no function works as a consensus estimator for all x for the simple reason that if any such function did exist it would need to be constant to sat-

isfy the second criterion in the definition above, but the function cannot be constant according to the first criterion.

We define G_x to be the distribution from which our g functions, as we defined them, can be drawn.

Lemma 1

The probability that $g(x)$ drawn from the distribution G_x is a consensus for x is constant for all x; the probability is $1 - \log_c \rho$.

Proof taken directly from [Parkes and Schoenebeck, 2004]

We fix an x and integrate over the possible values of u that give us a ρ-consensus function. Without loss of generality, we assume that $\frac{x}{\rho} = c^j$ for some integer j.

Then $c^{j+u} \in \left[\frac{x}{\rho}, \frac{cx}{\rho}\right)$. It is important to note that $c^{j+u} \in \left[\frac{x}{\rho}, x\right)$ iff this function is not a ρ-consensus function.

$$\int_{c^{u+j} = \frac{x}{\rho}}^{c^{u+j} = x} \pi(u)du = \int_{u = \log_c \frac{x}{\rho} - j}^{u = \log_c x - j} \pi(u)du$$

$$= (\log_c x - j) - \left(\log_c \frac{x}{\rho} - j\right) = \log_c \rho \text{ . } \textbf{QED}.$$

Claim 1

If the value of u chosen at runtime is hidden from the agents, no agent can unilaterally affect whether or not a given partition π has a consensus on its value.

Proof

From Lemma 1, we know that if we choose our g function uniformly from G_x, the probability that there is a consensus on input x is constant for all x. If an agent does not know the value of u chosen by the mechanism, then it does not know which g function was chosen from G_x. Thus, from the agent's perspective, any value that is fed to the g function has an equal probability of being deemed a consensus value. Thus an agent cannot hope to affect whether there is a consensus on the reported value v' of partition π by changing the reported value to v'', for the probability that there is a consensus on v' is the same as the probability that there is a consensus on v''. **QED**.

Claim 1 implies that the mechanism can discard all partitions for which it is not the case that g is a consensus for $\overline{V(\pi)}$, yet still not be manipulable by agents. After all, since agents do not know the g function ahead of time (because u is chosen randomly at run time), they cannot purposely attempt to influence whether or not there is a consensus on the value of the neighbor by manipulating $\overline{V(\pi)}$, for the probability that the g function that is chosen by the mechanism is a consensus for $\overline{V(\pi)}$ is constant for all $\overline{V(\pi)}$.

Even though the proof of Claim 1 shows that an agent cannot influence whether or not there is a ρ-consensus on the reported value of a partition, it nevertheless is a bit difficult to understand conceptually how the ρ-consensus functions handle the case of an ill-intentioned agent that tries to manipulate the auction by bidding extreme values on certain bundles in a partition that the agent would like thrown out (and hence would like to force

the partition's value to not have a consensus). In understanding why the ρ-consensus functions are resilient to such manipulation, we must remember that the value ρ is chosen such that for all agents *i* and all partitions π, $\frac{V(\pi)}{\rho} \leq V_{-i}(\pi) \leq V(\pi)$. Recall that ρ is external to the mechanism and is chosen by the entity running the mechanism as a value appropriate for the domain in which the auction is run. For example, if the bids in an auction fall into a particular distribution of bids in which it is infeasible for the bids of a single agent to move the value of a partition by more than a factor of 2 from the value of the partition across all other agents, 2 would be a suitable value for ρ. We present in our experimental results chapter concrete examples of suitable ρ values for a variety of auction settings. Because ρ is chosen as appropriate for the domain in which the auction is run, agents are not able to bid extreme values that would yield a non-consensus, for a bid that could throw off whether a partition is a consensus or not would be infeasible according to the domain-specific value of ρ. The value of ρ that is fed to the mechanism should be chosen to make the bounds $\frac{V(\pi)}{\rho} \leq V_{-i}(\pi) \leq V(\pi)$ on $V_{-i}(\pi)$ as tight as possible.

Because *c* is simply a constant, note that the proof of Claim 1 is independent of the value of *c* that is chosen. Our mechanism is thus free to choose a *c*, so we decide to choose one that, building on the results of Claim 1, allows the probability of consensus to be a parameter on which *c* is based. Since the probability of consensus *q* is $1 - \log_c \rho$, we see that:

$$q = 1 - \log_c \rho$$

$$q - 1 = -\log_c \rho$$

$$1 - q = \log_c \rho$$

$$c^{1-q} = \rho$$

$$c = \rho^{\frac{1}{1-q}}.$$

Hence, for a given ρ, we can fine-tune c by adjusting the probability that g will be a ρ-consensus function for any input x.

Claim 2

When we have a consensus, the value $g(\overline{V(\pi)})$ cannot be unilaterally influenced by a single agent.

Proof

A consensus implies that for every agent i, $g(\overline{V_{-i}(\pi)}) = g(\overline{V(\pi)})$, meaning that no single agent could have submitted bids that influenced the consensus value $g(\overline{V(\pi)})$. **QED**.

Thus far we have analyzed the incentive properties of some individual components of our mechanism, but it is necessary at this point to put these individual pieces together to show what guarantees the mechanism as a whole can make about its strategyproofness.

Claim 3

The consensus mechanism is non-manipulable with probability $1 - (1 - q)^n$, where n is the number of neighbors that the mechanism considers in each iteration of the local search.

Proof

Our mechanism includes the `Main`, `LocalConsensusWinner`, `g`, `VCG`, and `GetNeighbors` functions. We will examine the manipulability claims we can make about the objects returned by each of these. We know that the allocation returned by the `VCG` function is non-manipulable because of the VCG auction's strategyproof property that we discussed in Chapter 2. The list of neighbors that the `GetNeighbors` function returns is not susceptible to manipulation because this function simply returns a list of random neighbors of the partition it is fed. Because the `g` function simply maps an input to an output, and because the value that the input is mapped to is based entirely on the `u`, `rho`, and `q` parameters, which no agent can affect, `g` is not susceptible to manipulation.

In our `LocalConsensusWinner` function, because the `u` value is chosen in the `Main` function at runtime and is not known to the agents, we know from Claim 1 that no agent can affect whether there is a consensus on a particular partition, so no agent can affect whether or not the algorithm passes the first `if` statement. If the code within the first `if` statement's body is executed, the only items that can be updated are the `CurrentBestValue` variable and `ConsensusNeighbors` array. However, the decisions about whether to update these two items and if so with what values are dictated by the value returned by the `g` function; so the only way an agent could alter these two objects is by altering the `g` value. Yet we know that there is a consensus value for the partition we are considering if we are executing the code in this `if` statement's body, and we know from Claim 2 that as a result the

g value cannot be manipulated; thus no agent can affect whether or not these two objects are updated and to what values they are updated. Since the only times in which we update the only two objects in the LocalConsensusWinner function, CurrentBestValue and ConsensusNeighbors, are within this first if statement, and since we now know that the decisions made within this if statement are not susceptible to manipulation, we know the values of CurrentBestValue and ConsensusNeighbors cannot be influenced by a single agent. The last if statement simply decides what to return based on these two objects, and since these two objects cannot be manipulated, the item returned by the LocalConsensusWinner function cannot be manipulated.

We now examine the Main function. The Main function repeatedly calls the LocalConsensusWinner function and compares the g value of the partition π' returned by that function with the g value of π. We know that as long as there is a consensus on the value fed to the g function that the output of the g function is not susceptible to manipulation. Thus, whenever the LocalConsensusWinner returns a π' for which there is a consensus on its value, the g value of π' will be non-manipulable, and thus the comparison that the Main function makes between the g values of π and π' is not susceptible to manipulation. However, if the LocalConsensusWinner function returns a partition π' that does not have a consensus value, then the comparison the Main function makes between the g values can be manipulated. We know that the only way in which the LocalConsen-

`susWinner` function would return a partition that does not have a consensus on its reported value is if no neighbor it considered had a consensus on its reported value, in which case `LocalConsensusWinner` returns a random neighbor that does not have a consensus. We know that with probability q a particular neighbor that is being considered will have a consensus on its reported value, and with probability $1 - q$ it will not have a consensus. Thus, if the `LocalConsensusWinner` function evaluates n neighbors, with probability $(1 - q)^n$ no neighbor will have a consensus on its reported value, so with probability $1 - (1 - q)^n$ it will not be the case that no neighbor has a consensus on its reported value (i.e., at least one neighbor will have a consensus on its reported value). Thus with probability $(1 - q)^n$ the comparison that the `Main` function makes can be manipulated. Thus with probability $(1 - q)^n$ an agent can change the direction of the search at a given step, so with probability $1 - (1 - q)^n$ no agent can change the direction of the search in a given step. Thus with probability $1 - (1 - q)^n$ the mechanism is non-manipulable. **QED**.

It follows from Claim 3 that only when $q = 1$ (which is impossible because this case would result in division by 0 when calculating c) is the mechanism completely manipulation-free. The reader may at this point wonder why it is not to the mechanism designer's advantage to set q as close as possible to 1 in order to have the mechanism be non-manipulable with as high probability as possible. We will see in the following section, however, that the value of q affects the quality of the approximation that the mecha-

nism returns, and that for this reason it is not always advantageous to set q as high as possible.

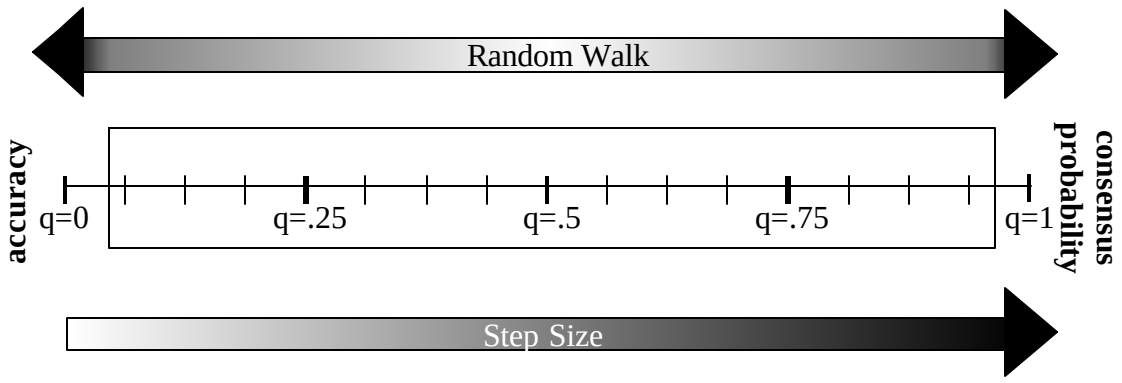
6.3 Performance Analysis

For a given ρ, c grows as the probability q of consensus grows. Thus as the probability that g will be a ρ-consensus function for a random value x increases, so do the step sizes in the step function g. Intuitively, this linkage between consensus probability and step size makes sense, for the larger the steps in g, the higher is the probability that the region between $\frac{x}{\rho}$ and x will fit on the same step. As c increases, though, the accuracy of the estimates that g produces decreases, for as step size increases, g maps more input values to the same step, meaning that the value returned by g is less meaningful in determining which partitions are more valuable. Although increasing the probability of consensus is good because it means that the mechanism discards fewer partitions and thus has a smaller chance of throwing away valuable partitions, increasing the accuracy is also good because it means that for the partitions that do have a consensus, the g function more accurately maps the partitions to their actual values, and thus there is a smaller chance that partition B, which is less valuable than partition A, is chosen over A because both mapped to the same value and B was randomly chosen instead of A.

It is interesting to observe that as we move toward the extreme ends of the spectrum that has accuracy on one end and consensus probability on the other, we in essence move toward a random walk algorithm. As accuracy falls (probability of consensus rises),

the algorithm moves closer to a random walk, for more partitions are mapped to the step which has the highest-valued partition, and thus more partitions are chosen at random by the consensus algorithm; in the extreme case, where there is only one step, the mechanism reduces to a random walk, for it simply chooses a random partition at each iteration in the search. Yet as accuracy increases (probability of consensus decreases), the algorithm again moves closer to a random walk, for the smaller step sizes mean that there is a lower chance of consensus, implying that valuable partitions have an increasing chance of being discarded and thus less valuable ones have a greater chance of being selected; in the extreme case, the steps are so small that because there is no consensus for any input x a partition is chosen at random in each iteration of the search, and again the algorithm reduces to random walk. Figure 2 illustrates our observations.

Figure 2



We note, though, that the flavor of random walk at the lefthand side of the spectrum is different from that at the righthand side. Even though with both $q = 0$ and $q = 1$ the `LocalConsensusWinner` function returns a random partition, there is a difference

between the two in what happens with this returned partition in the `Main` function. When the probability of consensus is 0, the *g* function is highly accurate, but when the probability is 1 it is perfectly inaccurate (in the sense that every input maps to the same output). Since in both cases the `Main` function will be able to find a consensus for neither π nor π', it will rely on the *g* value returned by the `LocalConsensusWinner` function to determine if the most recently returned partition is the best one seen so far. Since when $q = 1$ the *g* function returns the same value on every input, the `Main` function at each iteration simply selects the last partition randomly chosen by `LocalConsensusWinner` to be the best partition seen π, for `g(Value(`π'`),u,rho,q) >= g(Value(`π`),u,rho,q)` will always be true, and thus π' will always replace π. In essence, then, the algorithm is acting as a non-hill-climbing random walk, for at each step it makes a random partition the best one, regardless of how it compares value-wise to previously seen partitions. On the other hand, when $q = 0$, the performance of the consensus algorithm is essentially the same as random walk hill-climbing, for the highly accurate *g* function maps every input to a separate output such that if $x > x'$ then $g(x) > g(x')$; in essence, then, the algorithm is performing hill-climbing random walk. Because when $q = 1$ the information returned by *g*, which is based on the reported value of agents, is not used at all in determining which partition to make the current best, the non-hill-climbing random walk is completely non-manipulable. However, when $q = 0$, the algorithm completely reduces to hill-climbing random walk, which means that its entire decision on which partition to move to is based on the reported values of agents, for *g* maps every input to a unique output that maintains that if $x > x'$ then $g(x) > g(x')$. Thus, as we move from the left hand side of the spectrum to the right-

hand side, as we move from hill-climbing random walk to non-hill climbing random walk, we move from completely manipulable to completely non-manipulable, which is confirmed by our earlier observation that the algorithm is non-manipulable with probability $1 - (1 - q)^n$, where n is the number of neighbors the local search considers in each iteration.

This inherent tension between accuracy and probability of consensus means that it is important to determine an appropriate q for a given value of ρ: that is, a q that effectively balances accuracy and consensus probability, coming close to maximizing the value of the allocation that is returned by the mechanism while still yielding a high strategyproof probability. There is a nice middleground between the two ends of the spectrum that allows the mechanism to avoid the ignorance of random walk, and we show in our experimental results chapter the impact that varying q has on the search's success.

It is important to note that the value of ρ that is fed to the mechanism should be chosen so that ρ is as small as possible while maintaining that $\frac{V(\pi)}{\rho} \leq V_{-i}(\pi) \leq V(\pi)$. As ρ increases for a given q, the value of c gets larger, meaning that the accuracy of the estimate falls. However, c rising for a given q means we get a dip in accuracy without a compensating increase in consensus probability; the step sizes are larger, meaning that more inputs get mapped to the same g output, yet the consensus probability stays the same because the region between $\frac{x}{\rho}$ and x grows with the step size. Thus accuracy falls yet we do not get the benefits of an increase in consensus probability (we do not get a decrease in

our chances of discarding a valuable partition). Thus the larger that ρ gets, the closer we move to the ignorance of random walk, for accuracy is lower while all else stays the same. This negative effect of increasing ρ implies that ρ should be made as small as possible, given the realities of the domain in which the auction is run. We show in our experimental results the effect of increasing ρ while keeping all else the same.

Chapter 7: Experimental Setup

7.1 Introduction

Although we have shown that our consensus algorithm adequately addresses incentive issues, an important question we need to explore is how well the mechanism performs from the perspective of allocative efficiency. The main goal of shaping our mechanism's winner determination process in the form of a local search algorithm is that these algorithms, ignoring issues of agent incentives, are known to provide good approximations; yet we need to feel confident that the modifications and constraints we have added to basic local search through our use of consensus functions is not too limiting from an efficiency viewpoint. Even if the proposed mechanism has desirable incentive properties, it is quite useless if it does a poor job of approximating the optimal solution. For this reason, we built a robust combinatorial auction simulator which allowed us experimentally to evaluate our mechanism's efficiency.

7.2 Algorithms Used as Benchmarks

A study of the quality of our algorithm's approximations is quite meaningless unless we examine its quality in comparison to other algorithms' performance. We chose to implement three other local-search-based algorithms to use as benchmarks against our consensus approximation: LP-based directed hill climbing, LP-based random walk hill climbing, and Casanova. Pseudocode together with a description of the directed hill

climbing algorithm was given earlier (directed hill climbing was earlier referred to as basic hill climbing or basic local search). A description of Casanova was also given earlier. The random walk hill climbing mechanism simply starts off with a random partition π and determines its value across all agents based on agent bids by running the linear program we described earlier. It then chooses a random partition π' and determines its value across all agents based on bids by running the linear program. If it is the case that π' has a higher reported value than π, then π' becomes π. Otherwise π stays the same. A new π' is then chosen at random, and the process repeats. This simple process continues until the algorithm is stopped.

We chose LP-based directed hill climbing because it is essentially identical to our algorithm, except it does not employ consensus functions and is not strategyproof. We can thus view directed hill climbing as our algorithm but with the "incentive-compatible switch" turned off. We selected LP-based random walk hill climbing because its lack of direction helps illustrate the value of guidance in a local search's productivity; because our algorithm has tendencies toward random walk as q goes to 0 and q goes to 1, we want to show what our algorithm reducing to random walk—and thus losing all guidance—would mean. Finally, we decided to use Casanova as a benchmark because it is the most widely-known published example of an approximation mechanism based on local search.

In our simulation, the mechanism has access to the true values of participating agents. We must keep in mind that because none of the three benchmark algorithms is strategyproof, yet in our experiments we are intrinsically assuming truthful bidding by giving the mechanism access to agents' true values, we expect the benchmark algorithms

to give their best-case approximations. In general, the approximations of a non-strategyproof mechanism in a truthful setting will be better than in a non-truthful setting, because the search direction is guided by more "accurate" information. That is, agents are not attempting to manipulate the mechanism by feeding it information that is false. Because the search is directed by more accurate information, it is expected to perform better; in essence this desire for accurate information is the primary reason that strategyproofness is such a desirable property. Since we are comparing the consensus algorithm's performance to the best-case performance of the other algorithms, we are setting a high standard for our algorithm.

Because our simulation tracks efficiency, it is necessary to calculate the optimal solution for any problem on which the algorithms are run; our code thus runs a mixed integer program which optimally solves the winner determination problem.

7.3 Inputs to Simulation

In addition to specifying which, if not all, of the four algorithms should be run in a particular experiment, the user is able to specify problem types on which the algorithms are run, fine-tune parameters that are fed to the algorithms, indicate distributions in which agent valuation functions fall, and dictate halting conditions. All inputs fed to the simulation are stored in input files. To ensure that the simulation could support different file formats, all input data is parsed by a "manager" program which sends input parameters to the actual simulator. All data is output to files; the data from each algorithm is stored in a sep-

arate file, in a directory specific to the experimental run. Table 7 in the appendix lists what can be specified at run time.

7.4 How the Value of a Partition is Determined

For the directed hill climbing local search, random walk, and consensus algorithms, we need to determine the values of partitions in each search step; to do this, it is necessary to solve the linear program presented in Section 5.4. To solve this linear program, we used the mixed integer programming functionality of CPLEX, a commercial linear program solver (http://www.cplex.com). We also used CPLEX to solve every auction optimally; recall that we need to solve each auction optimally in order to determine the efficiency of the allocations returned by the approximation algorithms.

7.5 Agent Valuation Functions

We abstractly think of the values an agent places on bundles as a valuation function; on input *b*, where *b* is a bundle, an agent's valuation function will return 0 if the agent is not interested in the bundle, or will return the value the agent holds for the bundle if it is interested. Our simulation needs to construct valuation functions for each agent.

The number of bundles in which each agent is interested is a parameter passed to the simulator. The specific bundles in which each agent is interested, and the values that each agent places on such bundles, are dependent on the chosen distribution. We employ

the four distributions that Sandholm [Sandholm, 1999] proposed as appropriate for running combinatorial auction experiments, allowing the user to specify m and α:

- *Random*: For each bundle, pick the number of goods randomly from {1, 2, ..., m}. Randomly choose that many goods. Pick the value of the bundle randomly from [0,1].
- *Weighted random*: Like random, but pick the value between 0 and the number of goods in the bundle.
- *Uniform*: Draw the same number of randomly chosen goods for each bundle. Pick the value of each bundle from [0,1].
- *Decay*: Start off with one random good. Then repeatedly add a new random good with probability α until a good is not added or the bundle includes all m goods. Pick the value of each bundle between 0 and the number of goods in the bundle.

We should note one subtlety inherent in constructing agent valuation functions. Care must be taken to ensure that if an agent values a particular bundle b at value v, all other bundles in which it is interested that are a superset of b (i.e. contain all the goods that b contains) must have a value of at least v. Our simulator uses a combination of hash tables and lists of linked lists to ensure that this constraint does not have too significant an impact on the performance of valuation function construction.

7.6 Experimental Details

We define an instance of an auction to be a set of agents, goods, and valuation functions, independent of the actual winner determination procedure that is run. In each

trial run, the simulator begins by generating an auction instance, and then feeds this same instance to each of the four winner determination algorithms; thus the algorithms are compared on the same problems.

The simulator was written in C++ and run on a Windows XP Professional PC with a 2.52 GHz Pentium IV processor and 512 MB of RAM.

A tremendous amount of time was spent optimizing the mechanism code and the code for each of the algorithms so that the experimental results would be as true to the inherent characteristics of the algorithms as possible. A large amount of data is stored in efficient lookup structures like hash tables and linked lists of arrays, and significant portions of the simulator were rewritten several times to improve performance.

The simulator source code spans forty-four files, and due to its enormous size is not included with this document. Interested parties may contact the author if they would like access to the source.

Chapter 8: Results

8.1 Introduction

Our goal in experimentally evaluating our mechanism was to compare the performance of the consensus-based approximation mechanism to non-strategyproof anytime approximation mechanisms that are also based on local search. As described in the previous chapter, the three other algorithms to which we compare ours are LP-based directed hill-climbing, LP-based random-walk hill-climbing, and Casanova. We compared in five problem sizes the performance of the four winner determination algorithms across the four distributions. The five problem sizes we chose are listed in Table 5.

Table 5

Agents	Goods	Bundles	distribution *m*	Run Time (s)
80	160	320	5	30
80	320	480	5	115
100	200	600	5	60
40	80	480	5	30
10	100	600	15	30

We attempted through this selection of problems to cover a broad range of problem sizes: auctions with lots of agents, ones with few agents, ones with lots of goods, ones with few goods, ones with lots of bundles, and so on. The first four problems were ones chosen by Parkes and Schoenebeck in their GrowRange paper [Parkes and Schoenebeck, 2004]. The run time for each problem was chosen based on test results; whenever we felt it was useful

to expand our time window, we did so. Each of the problems was run separately on each of the four valuation distributions. Unless stated otherwise, all data is for ten runs and we used a ρ value of 1.1 and a q value that was determined experimentally to be near-optimal.

We also aimed in our experiments to evaluate intrinsic aspects of our approximation mechanism. We thus chose to show the effects that different ρ values have on efficiency, the effect of varying q for a given ρ, the factor across the four distributions by which a single agent who bids maximally can unilaterally change the value of the optimal allocation for a given problem size (to show what are appropriate values of ρ), and how the marginal effect of a single agent is influenced by the number of agents in the system.

We summarize some of the conclusions we derive from our experimental results:

- The restrictions placed on the mechanism which allow it to be strategyproof with high probability do not cripple its ability to provide high-quality approximations quickly. We see this is the case by observing our mechanism's performance relative to the three non-strategyproof mechanisms on different problem sizes across the four distributions.
- We see that values of ρ which are suitable for many real-world domains are sufficiently low to allow the entity running the mechanism to choose both high probability of strategyproofness and good approximation quality. We also examine the robustness of our determination of an appropriate ρ value by showing how the number of agents in the system affects this value.

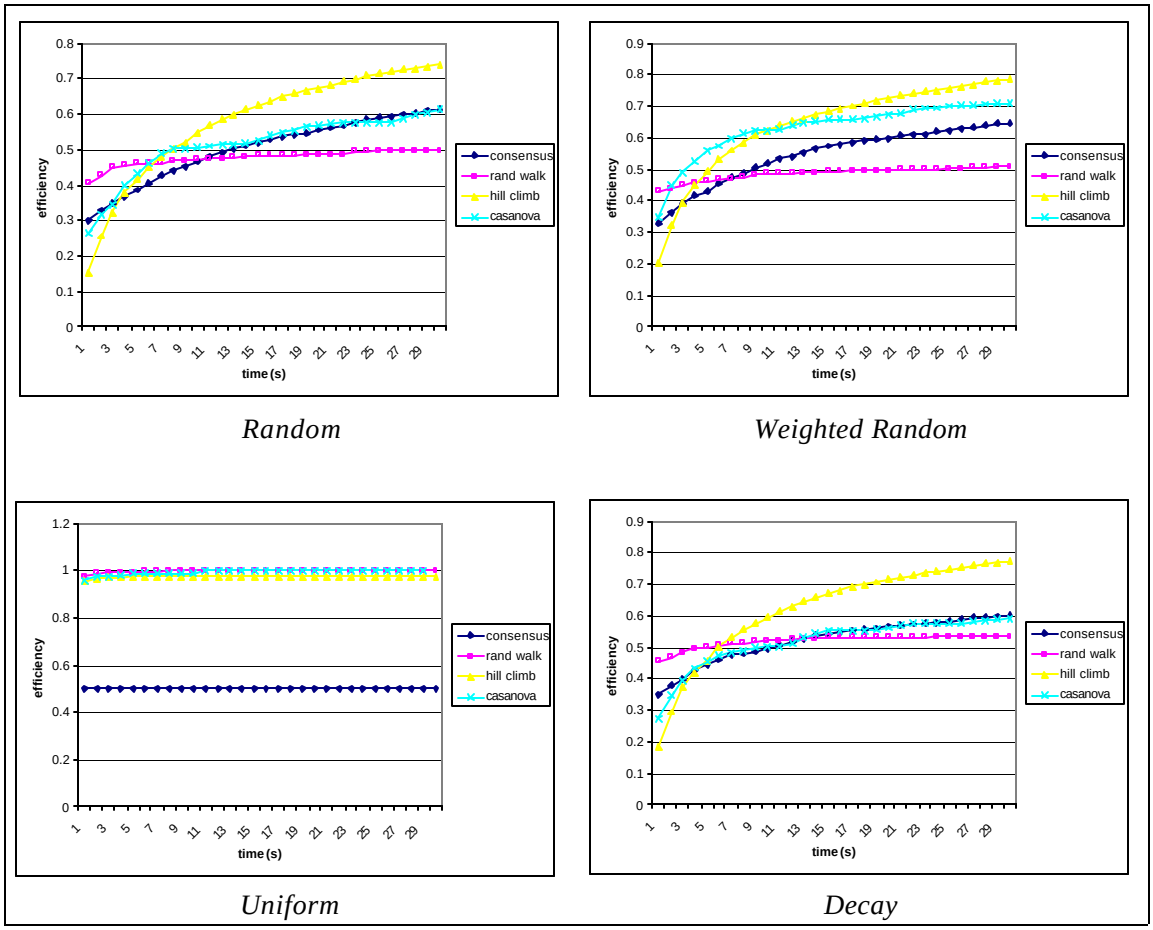
- We see first-hand the trade-off between consensus probability and approximation quality as we observe how efficiency varies with q.
- We observe that a lower ρ does indeed result in a better approximation.
- We see that as ρ increases the optimal value of q falls.
- We see that our theoretical observation that $q = 0$ results in random walk hill-climbing and $q = 1$ results in random walk holds true experimentally.

8.2 Comparison with Other Local Search Algorithms

80 agents, 160 goods, 320 bundles

Figure 3 shows the relative efficiency over time of the four algorithms for problems with 80 agents, 160 goods, and 320 bundles. The results in the random and weighted random distributions are fairly comparable. In both, at the end of the 30 seconds directed hill-climbing has found an allocation with higher efficiency than the three other algorithms. However, the results confirm that, as suggested by the Hoos and Boutilier paper, Casanova at first does a better job than directed hill-climbing, only being surpassed by hill-climbing after around 8 seconds. This result indicates that Casanova is better able than directed hill-climbing to find a good approximation quickly. Random walk surprisingly is able to outperform the other algorithms within the first few seconds, but improves little as time progresses. The recorded performance of random walk suggests that its inherent randomness allows for the quick discovery of a better allocation than the others, because it is able to jump in one step to a point on the hill, whereas the directed local searches must

Figure 3: 80 agents, 160 goods, 320 bundles



step-by-step climb the hill. Nevertheless, as the results show, the lack of direction inherent in random walk means that it is unable to benefit from the lucky jumps it takes by continuing up a hill on which it has found a good allocation; although its random nature allows it to get "lucky" quickly, this luck takes the algorithm only so far, for it is unable to use the knowledge it acquired in guessing luckily to find a better neighboring partition.

What is most relevant to our discussion is the performance of the Consensus algorithm. Surprisingly, in the random and weighted random distributions, the consensus algorithm performs rather well relative to the non-strategyproof directed hill-climbing and Casanova algorithms. In fact, in the random distribution, the Casanova and Consensus

algorithms are essentially neck-and-neck. The fact that Consensus performs well relative to Casanova and directed hill-climbing is surprising because these two, as a result of their ignorance of incentives, are able to "use" fully all of the information presented to them by agent bids, whereas the consensus algorithm must throw out some of this information. The results thus favorably show that the fact that the consensus algorithm throws out some information is not detrimental to its performance; the impact of this lesser amount of information was the primary concern we had on the consensus algorithm's performance, and therefore the results help alleviate some of our concerns.

The decay distribution produced results pretty similar to those of the random distribution.

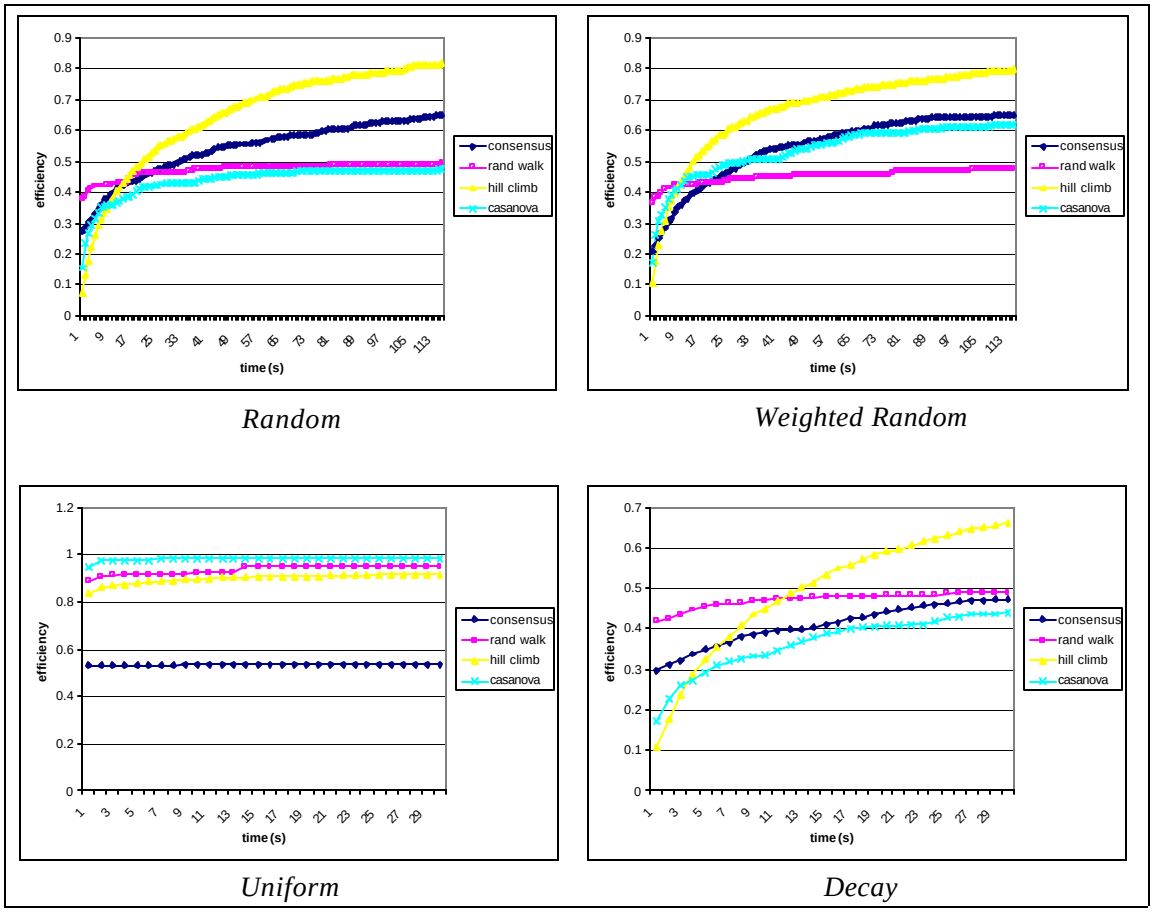
The most striking results are in the uniform distribution. While the three non-strategyproof algorithms consistently performed remarkably well, the consensus algorithm's efficiency near .5 speaks to the high variability of its results across the individual auction instances. It is easy to understand why the other three did well. In a uniform distribution, all bundles are of the same size. That means that, unlike in other distributions, no bundle's value is inherently tied to the value of a subset or superset bundle, for no bundle is a subset of another. This means that the value of every bundle is completely random and independent of the value of any other bundle. As a result, the values of all partitions are independent and uniformly distributed between [0, optimal value], and as a result it would be expected that directed hill-climbing local searches will act randomly, for in essence neighbor values are truly random. It is thus unsurprising that directed hill climbing and Casanova performed comparably to random walk hill-climbing. Random walk hill-climb-

ing would be expected to perform well because the uniform distribution of partition values, coupled with the fact that there are many partitions with 320 bundles of size 5 and 160 goods, means that there are many partitions with high values, and as a result a random walk will eventually stumble upon one of these iterations (in fact, within 10 iterations it should be expected to find at least one partition with a value within .9 of optimal). Because there are many iterations within one second of run time, and thus many partitions are visited within a second, it is thus unsurprising that random walk (and directed hill-climbing and Casanova, which are in this case reduced to random walk) consistently found good partitions within a second. In the uniform case, though, consensus's incentive-compatible constraints are limiting, as illustrated by the results, for they prevent the consensus algorithm from reducing to random walk as in the directed hill-climbing and Casanova. We noted before that as the value of q becomes 0 or 1, the consensus algorithm's performance more closely matches that of random walk, for in these two cases the consensus algorithm is explicitly forced to perform like random walk; however, we did not graph its performance when q is near 0 or near 1 because we already know what the results of these two cases would be and thus did not feel that they were interesting to graph.

80 agents, 320 goods, 480 bundles

Figure 4 graphs the average performance of the four algorithms on problems with 80 agents, 320 goods, and 480 bundles. In the random, weighted random, uniform, and decay distributions, the relative performance of the consensus, directed hill-climbing, and

Figure 4: 80 agents, 320 goods, 480 bundles



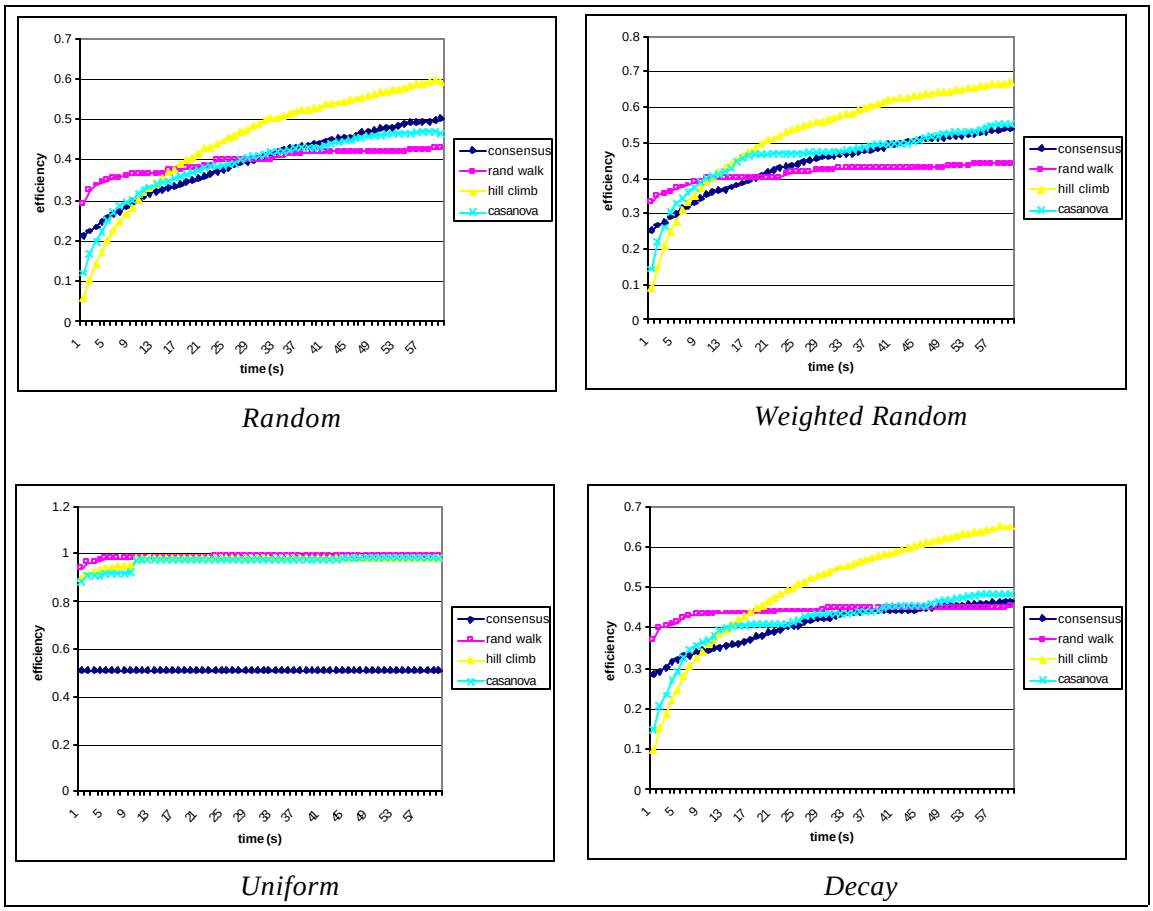
random walk algorithms is similar to the performance in problems with 80 agents, 160 goods, and 320 bundles. However, Casanova is not performing relatively as well as in the previous problem on the random, weighted random, and decay distributions, suggesting that as the number of goods rises relative to the number of bundles, and thus as the number of feasible partitions relative to the number of bundles rises (because a smaller percentage of bundles overlap), Casanova's performance takes a dip. It is important to note that these algorithms take longer than in the previous problem to find partitions of comparable efficiency, probably because of this increase in the number of feasible partitions. In general, a larger search space means that on average a particular search algorithm will take longer to

find a specific partition previous algorithms; in our domain, this means it will take longer to find a partition of high value.

100 agents, 200 goods, 600 bundles

Figure 5 shows the results of the four algorithms with 100 agents, 200 goods, and 600 bundles. In this problem, there are fewer goods relative to the number of bundles than

Figure 5: 100 agents, 200 goods, 600 bundles



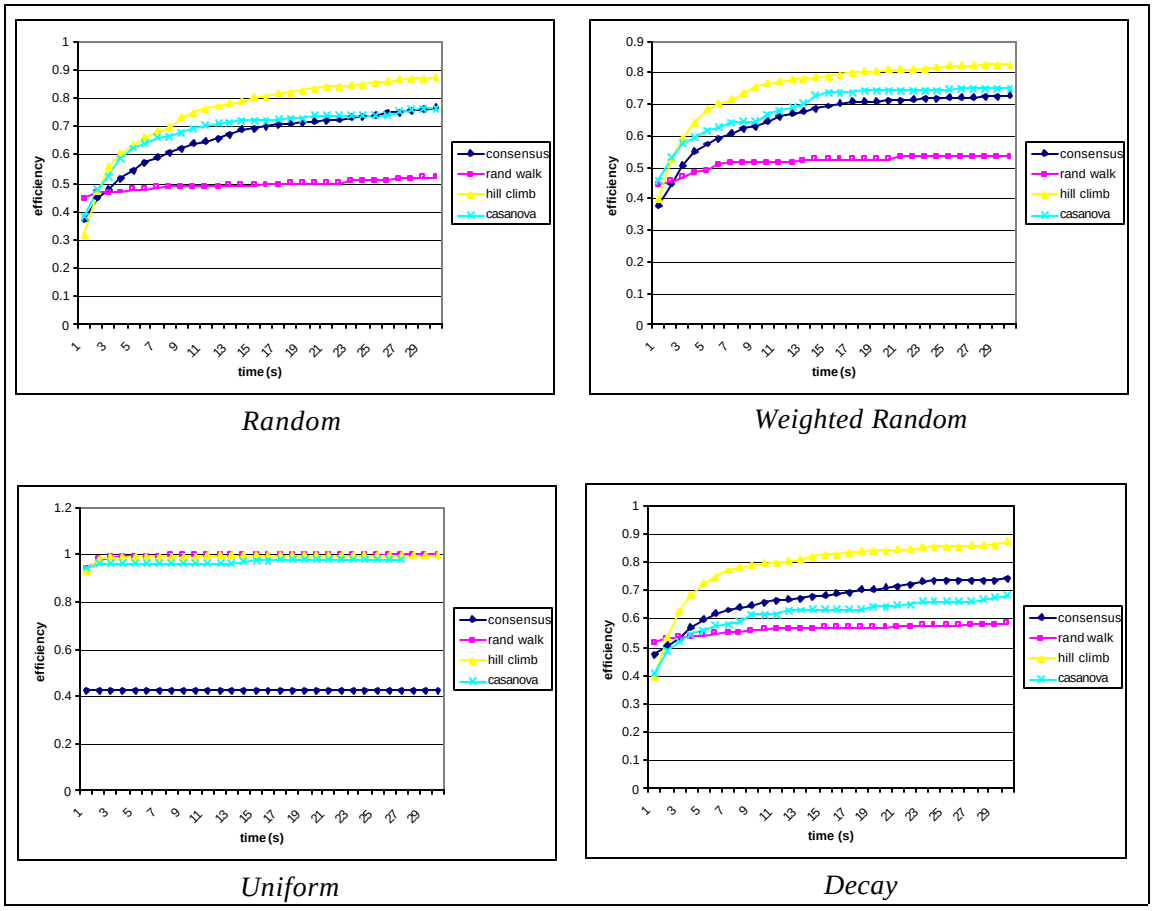
in the previous problems, which means that, compared to the previous problems, a larger percentage of bundles overlap and hence there are a smaller number of partitions relative

to the number of bundles. We notice in this case that all four of the algorithms have relative performance similar to that of the previous problems, but the absolute performance of all four falls, and the speed with which the hills are climbed in the consensus, directed hill climbing, and Casanova algorithms falls; this observation makes sense because although there are fewer partitions relative to the number of bundles, in absolute terms there are more partitions than in the previous problems because there are significantly more bundles, so we have a larger search space.

40 agents, 80 goods, 480 bundles

Figure 6 presents the average results of auctions containing 40 agents, 80 goods, and 480 bundles. Among the four algorithms, we see relative performance similar to that of the first two problems, but significantly better absolute performance for consensus, random walk hill-climbing, and directed hill-climbing. Because the number of goods relative to the number of bundles is significantly smaller in this problem, the number of feasible partitions is also smaller, for a greater percentage of bundles overlap. Also, since the number of agents is at least half of the number in the previous problems, the number of feasible allocations for each partition is also smaller (meaning that the linear program that is solved in each step takes less time, so the number of steps for a given unit of time is larger, so the search process is quicker). Because the search process is quicker and there is a smaller space in which to search, it is not surprising that the algorithms have better absolute performance, for they are able to more quickly find a particular partition.

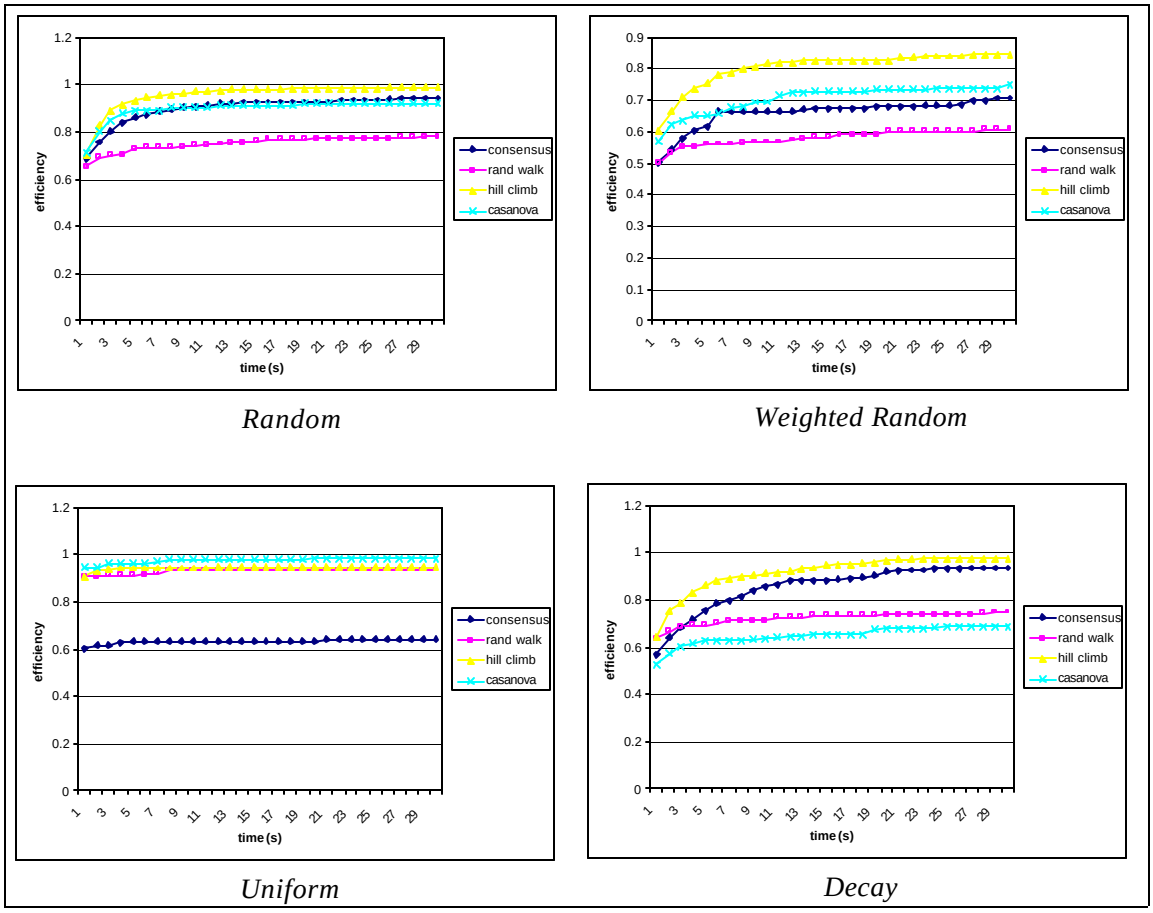
Figure 6: 40 agents, 80 goods, 480 bundles



10 agents, 100 goods, 600 bundles

Figure 7 illustrates the average performance of the four algorithms on auctions with 10 agents, 100 goods, and 600 bundles. The performance of the consensus algorithm on auctions of this type is important because it is the auctions with a small number of agents that can lead to drastic manipulations, for the small number means that each agent has a high degree of power in altering value calculations. As shown, in all distributions except for uniform, the consensus algorithm performs remarkably well. It should be noted that the other three algorithms, because they are susceptible to manipulation, would in

Figure 7: 10 agents, 100 goods, 600 bundles



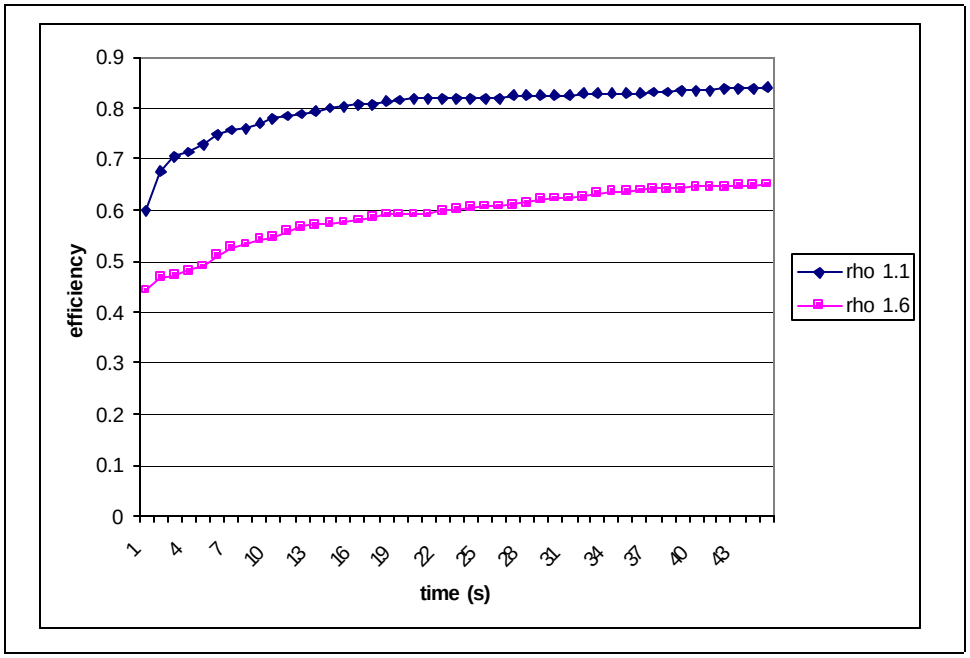
reality perform a lot worse than illustrated if a single agent decides to manipulate, because if there are only ten agents participating, and hence a single agent can have a significant impact on value calculations, a single agent can greatly alter the performance of a manipulable algorithm.

8.3 The effect ρ has on efficiency

In our theoretical analysis chapter, we explained why a smaller value of ρ for a given *q* produces better results, for as ρ rises for a given *q* the consensus algorithm

becomes less accurate, and this loss in accuracy can be thought of as a reduction in the information that is fed to the consensus algorithm to guide its search. To illustrate experimentally what this loss of information means, we ran the consensus algorithm for 45 seconds on a problem consisting of 30 agents, 20 goods, and 600 bundles for $\rho = 1.1$ and $\rho = 1.6$ with near-optimal *q* values. Figure 8 shows the average efficiency results over 40 runs.

Figure 8: Approximation quality for $\rho = 1.1$ and 1.6



It is clear from the graph that an increase in ρ results in a decrease in the performance of the consensus algorithm. We emphasize again that ρ is a domain property and not something that the mechanism can be designed for.

8.4 Varying *q* for a Given ρ

Our theoretical section also discussed the effect that the value of *q* has, for a given ρ , on the performance of the consensus algorithm. We decided experimentally to examine this effect. On a problem with 8 agents, 20 goods, and 40 bundles, we graphed how average efficiency after 3 seconds in 100 runs is affected as q moves from 0 to 1 in increments of .05 for ρ values of 1.05, 1.1, 1.25, 1.6, 2.0, and 3.0. Figure 9 shows the results, with a polynomial trend line of degree 3 added to each graph. Based on the trend lines, we graphed, in figure 10, the efficiency-maximizing value of *q* as ρ varies. We observe that the optimal *q* value appears to decrease as ρ increases. We notice also that varying *q* has a less drastic effect as ρ increases.

We may wonder why in the graphs of Figure 9 the efficiency at $q = 0$ is higher than at $q = 1$, for, according to the theoretical analysis, the consensus algorithm reduces to random walk for both values. However, as also explained by our theoretical analysis, the reason that the consensus algorithm is able to better perform with 0 probability of consensus than with 1 is that when $q = 0$ the algorithm is performing hill-climbing random walk, while when $q = 1$ it is performing a completely random walk. It is no surprise that hill-climbing random walk performs better than non-hill-climbing random walk, since hill-climbing returns the best partition seen across all iterations whereas non-hill-climbing returns a completely random partition; for this reason the performance at $q = 0$ is better than that at $q = 1$, confirming our theoretical analysis of which *q* value would result in a more efficient mechanism.

Figure 9: Varying *q* for a given ρ

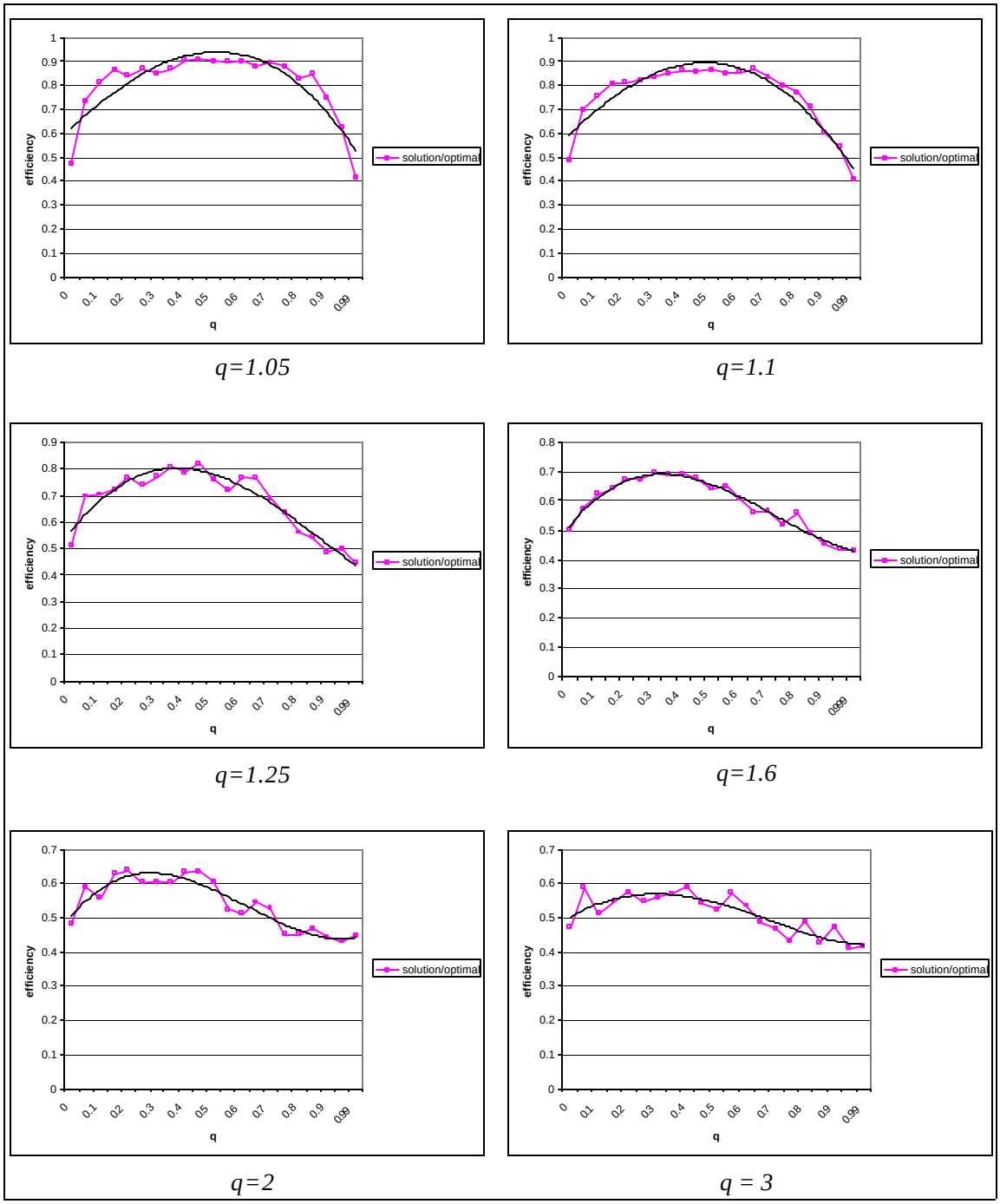
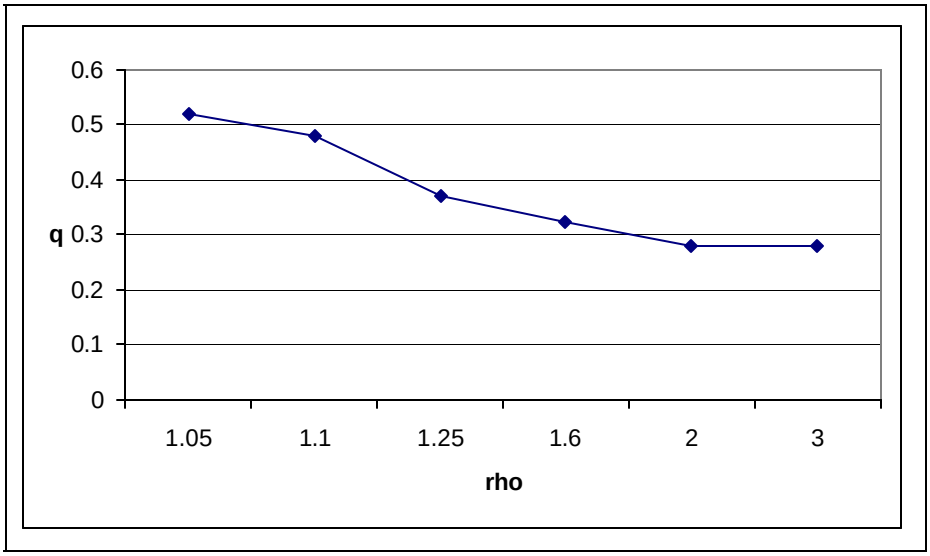


Figure 10: Efficiency-maximizing value of *q* as ρ varies



8.5 The Marginal Effect of an Agent

A Fixed Problem

In attempting to get a sense of appropriate ρ values for specific domains, it helps to examine the marginal effect that a bidder who bids values that are the highest possible in a particular domain can have on the calculated value of a partition. We attempted through experimentation to get a sense of this marginal effect on a problem with 80 agents, 320 goods, and 400 bundles. In this set of experiments, we assigned all agents except for agent 0 valuation functions according to the distribution in which the experiment was run, and gave agent 0 for each bundle the maximum value allowed under that distribution; for example, for a random distribution a value of 1 for every bundle, and for a weighted random distribution a value for each bundle equal to the number of goods in that bundle. Over 100 runs for each distribution, we recorded the ratio of the value of a random

partition with agent 0 in the system to the value of the same partition without agent 0. Table 6 lists the average, median, maximum, and minimum ratios encountered for each of the distributions.

Table 6

	Average	Median	Max	Min
Random	1.016183	1.016748	1.01831	1.009009
Weighted Random	1.015738	1.015197	1.028962	1.00347
Uniform	1.002188	1.001322	1.009056	1
Decay	1.017999	1.015598	1.049795	1.010332

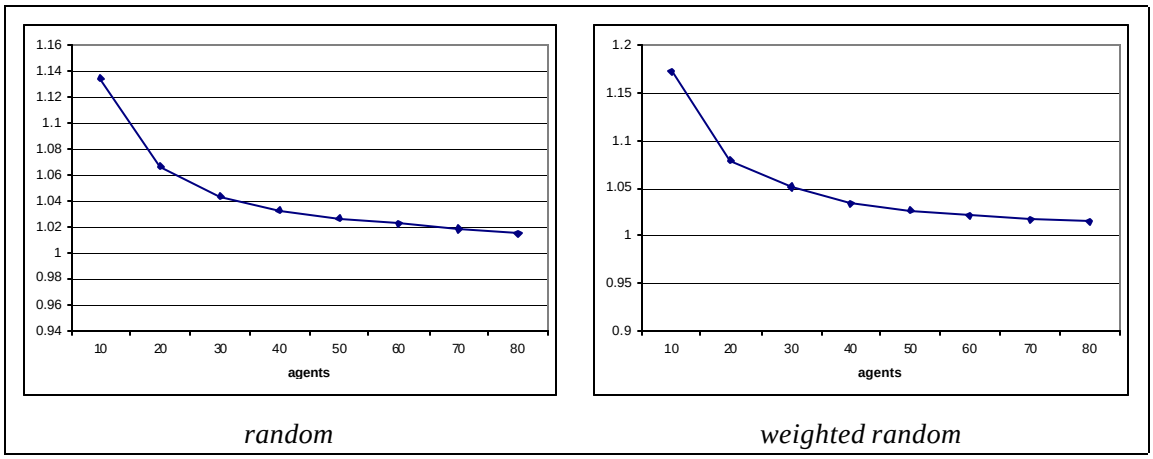
The results suggest that, for example, a value of 1.1 for ρ in this problem is more than realistic, for even the maximum factors by which a single agent can feasibly influence the value of a random partition are nowhere near 1.1.

Scaling the Number of Agents

We also wanted to get a sense of the maximum marginal effect that an agent, bidding within the realm of feasible bids, can have as we scale the number of agents. Thus, for a problem with 200 goods, we scaled the number of agents from 10 to 80 in increments of 10, adding 5 bundles for every agent added. Like before, our experiments assigned all agents except for agent 0 valuation functions according to the distribution in which the experiment was run, and gave agent 0 for each bundle the maximum value allowed under that distribution. Our experiments calculated, over 100 runs per problem, the average ratio of the value of a random partition with agent 0 in the system to the value of the same par-

tition without agent 0. The results for the random and weighted random distributions are shown in Figure 11. The results show that, for example, with 80 agents in the random dis-

Figure 11: The marginal effect of an agent as the number of agents scales



random *weighted random*

tribution, agent 0 on average increases the calculated value of a partition by a factor less than 1.02 of the value without agent 0, suggesting that a ρ value of 1.1 is more than safe for that particular problem. As expected, as the number of agents falls, the marginal effect that agent 0 can have grows, for agent 0's bids make up a greater fraction of the overall value assigned to a partition as the number of agents decreases. Thus, for a given problem, as the number of agents increases, the minimum value of a suitable ρ decreases.

Chapter 9: Conclusion

We have presented a local-search-based approximation mechanism that is strategyproof with high probability.

Prior to this work, two primary classes of approximation algorithms existed. The first, based on local search, used local search's power to provide quality approximations; unfortunately, they were not strategyproof. The second were non-VCG mechanisms that were strategyproof, but placed restrictions on agent preference domains that were quite limiting. The first had a major advantage over the second in that they were anytime. We wanted to construct a mechanism that has the good approximation and anytime properties of the first class, but with the strategyproofness that comes with the second class, but without the limitations of the second class. We proposed such a mechanism, one that does not have the second class's restrictions on agent preferences, builds on the strengths of local-search-based algorithms to provide quality anytime approximations, and is strategyproof.

Our Consensus mechanism is anytime and strategyproof with high probability, and is able to have this strategyproof property while also using agent bids to guide the search direction. Furthermore, our mechanism allows the entity running it to specify the probability with which the mechanism is strategyproof, enabling the appropriate level of trade-off between strategyproofness and efficiency to be specified at runtime.

We built a robust combinatorial auction environment to ensure that the modifications and constraints we added to basic local search through our use of consensus functions were not too limiting from an efficiency viewpoint. Comparing our Consensus

algorithm across several different problem sizes to three benchmark algorithms that ignored incentives, we were quite pleased—almost surprised—by how well it performed in spite of the restrictions we placed upon it. We also wanted to see experimental evidence of how q and ρ fit into the mix, and our results confirmed our theoretical analysis of the trade-off between approximation quality and strategyproof probability, of the different flavors of random walk local search that extreme values of q (near 0 and near 1) produce, and of the importance in making ρ as small as possible. We also discovered experimentally that values of ρ which are realistic for the distributions we considered allow our mechanism to have both high strategyproof probability and high approximation quality.

All-in-all, we are pleased with the theoretical and experimental results of our consensus mechanism. We are excited about what we have seen so far and would love to explore the mechanism further, hoping most of all that the contributions we have made will aid others in discovering more anytime strategyproof approximation algorithms that quickly provide approximations of even higher quality.

Future work

We are quite excited about our Consensus mechanism idea and about the theoretical and experimental results we have presented. We hope to continue exploring the idea in more depth on several fronts. Firstly, we would like to compare our mechanism to more mechanisms based on non-strategyproof approximation algorithms, like tabu search, in order to get an even better sense of how well the mechanism does in relation to other non-

strategyproof ones that have been proposed. Along the same lines, we would like to test the Consensus mechanism experimentally on more problem sizes and distributions in order to make our benchmarking comparisons more robust.

We also would like to compare our mechanism experimentally to the performance of GrowRange to determine on which bid distributions each is better suited.

Appendix

.

Table 7: Inputs to Combinatorial Auction Environment

Input	*Description*
`num_agents`	The number of bidders in the system.
`num_goods`	The number of goods being sold in each auction.
`bundles_per_agent`	The number of bundles in which any one agent is interested in bidding.
`num_neighbors`	The number of neighbors to consider in each round of a local search (used in directed hill climbing and consensus algorithms).
`time_per_auction`	The amount of time (in milliseconds) for which each auction should run. If set to 0, the auction runs for as long as the optimal algorithm took to solve.
`num_auctions`	The number of auction instances that are run.
`lp_type`	The linear program that is used. If set to 1, then an agent can be awarded at most one bundle in any allocation. If set to 0, there is no limit on the number of bundles an agent can be allocated.
`rho_value`	The ρ value used by the consensus algorithm.
`consensus_pr`	The *q* value used by the consensus algorithm.
`wp`	The *wp* value fed to casanova.
`np`	The *np* value fed to casanova.
`distribution_type`	If 1, then a random distribution is used. If 2, a weighted random. If 3, a uniform. If 4, a decay.
`distribution_m`	The value of *m* used in the distributions.
`distribution_alpha`	The value of α used in the decay distribution.

Table 7: Inputs to Combinatorial Auction Environment

`output_increments`	The increments, in seconds, for which the efficiency of the current best allocation determined by a given algorithm should be output to a file. If set to 0, only outputs the data from a algorithm after it has halted.
`time_tracking`	If set to 1, will write to a file the amount of time for which optimal and the four other algorithms ran.
`consensus`	If set to 1, the consensus algorithm will run.
`directed_hill_climbing`	If set to 1, the directed hill climbing algorithm will run.
`random_walk`	If set to 1, random walk hill climbing will run.
`casanova`	If set to 1, casanova will run.

References

[Andersson *et al*.] Arne Andersson, Mattias Tenhunen, Fredri Ygge. Integer Programming for Combinatorial Auction Winner Determination.

[Collins, 2002]. John Collins. Solving Combinatorial Auctions with Temporal Constraints in Economic Agents, 2002.

[Greenberg and Naim, 2004]. Brian Greenberg and Edward W. Naim. Market Based Routing in Sensor Networks, 2004.

[Goldberg and Hartline]. Andrew Goldberg and Jason Hartline. Competitiveness via Consensus.

[Hoos and Boutilier, 2000]. Holger H. Hoos and Craig Boutilier. Solving Combinatorial Auctions using Stochastic Local Search, 2000.

[Naim, 2003]. Edward W. Naim. Solving Combinatorial Auctions Using Tabu Search, 2003.

[Ng *et al*., 2000] Chaki Ng, David C Parkes, and Margo Seltzer. Strategyproof Computing: Systems Infrastructures for Self-Interested Parties, 2003.

[Nisan, 2000]. Noam Nisan. Bidding and Allocation in Combinatorial Auctions, 2000.

[Parkes, 2001]. David C. Parkes. Iterative Combinatorial Auctions: Achieving Economic and Computational Efficiency, 2001.

[Parkes and Schoenebeck, 2004] David Parkes and Grant Schoenebeck. GrowRange: Anytime VCG-Based Mechanisms, 2004.

[Sandholm, 1999]. Tuomas Sandholm. An algorithm for optimal winner determination in combinatorial auctions, 1999.

[Sandholm *et al*., 2001]. CABOB: A Fast Optimal Algorithm for Combinatorial Auctions. Tuomas Sandholm, Subhash Suri, Andrew Gilpin, and David Levine, 2001.

[Vries and Vohra, 2001]. Sven de Vries and Rakesh Vohra. Combinatorial Auctions: A Survey, 2001.