

Approximately-Strategyproof and Tractable Multi-Unit Auctions

Anshul Kothari*

David C. Parkes†

Subhash Suri*

ABSTRACT

We present an approximately-efficient and approximately-strategyproof auction mechanism for a single-good multi-unit allocation problem. The bidding language in our auctions allows marginal-decreasing piecewise constant curves. First, we develop a fully polynomial-time approximation scheme for the multi-unit allocation problem, which computes a $(1 + \epsilon)$ -approximation in worst-case time $T = O(n^3/\epsilon)$, given n bids each with a constant number of pieces. Second, we embed this approximation scheme within a Vickrey-Clarke-Groves (VCG) mechanism and compute payments to n agents for an asymptotic cost of $O(T \log n)$. The maximal possible gain from manipulation to a bidder in the combined scheme is bounded by $\epsilon/(1 + \epsilon)V$, where V is the total surplus in the efficient outcome.

Categories and Subject Descriptors

F.2 [Theory of Computation]: Analysis of Algorithms and Problem Complexity; J.4 [Computer Applications]: Social and Behavioral Sciences—*Economics*.

General Terms

Algorithms, Economics.

Keywords

Approximation Algorithm, Multi-unit Auctions, Strategyproof.

*Department of Computer Science, University of California at Santa Barbara, CA 93106. Email: {kothari, suri}@cs.ucsb.edu. Supported in part by NSF grant IIS-0121562.

†Division of Engineering and Applied Sciences, 33 Oxford Street, Harvard University, Cambridge, MA 02138. Email: parkes@eecs.harvard.edu. Supported in part by NSF grant IIS-0238147.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EC'03, June 9–12, 2003, San Diego, California, USA.

Copyright 2003 ACM 1-58113-679-X/03/0006 ...\$5.00.

1. INTRODUCTION

In this paper we present a fully polynomial-time approximation scheme for the single-good multi-unit auction problem. Our scheme is both approximately efficient and approximately strategyproof. The auction settings considered in our paper are motivated by recent trends in electronic commerce; for instance, corporations are increasingly using auctions for their strategic sourcing. We consider both a reverse auction variation and a forward auction variation, and propose a compact and expressive bidding language that allows marginal-decreasing piecewise constant curves.

In the **reverse auction**, we consider a single buyer with a demand for M units of a good and n suppliers, each with a marginal-decreasing piecewise-constant cost function. In addition, each supplier can also express an *upper bound*, or capacity constraint on the number of units she can supply. The reverse variation models, for example, a *procurement auction* to obtain raw materials or other services (e.g. circuit boards, power suppliers, toner cartridges), with flexible-sized lots.

In the **forward auction**, we consider a single seller with M units of a good and n buyers, each with a marginal-decreasing piecewise-constant valuation function. A buyer can also express a *lower bound*, or minimum lot size, on the number of units she demands. The forward variation models, for example, an auction to sell excess inventory in flexible-sized lots.

We consider the computational complexity of implementing the Vickrey-Clarke-Groves [22, 5, 11] mechanism for the multi-unit auction problem. The Vickrey-Clarke-Groves (VCG) mechanism has a number of interesting economic properties in this setting, including *strategyproofness*, such that truthful bidding is a dominant strategy for buyers in the forward auction and sellers in the reverse auction, and *allocative efficiency*, such that the outcome maximizes the total surplus in the system. However, as we discuss in Section 2, the application of the VCG-based approach is limited in the reverse direction to instances in which the total payments to the sellers are less than the value of the outcome to the buyer. Otherwise, either the auction must run at a loss in these instances, or the buyer cannot be expected to voluntarily choose to participate. This is an example of the budget-deficit problem that often occurs in efficient mechanism design [17].

The computational problem is interesting, because even with marginal-decreasing bid curves, the underlying allocation problem turns out to (weakly) intractable. For instance, the classic 0/1 knapsack is a special case of this problem.¹ We model the

¹However, the problem can be solved easily by a greedy scheme if we remove all capacity constraints from the seller and all

allocation problem as a novel and interesting generalization of the classic knapsack problem, and develop a fully polynomial-time approximation scheme, computing a $(1 + \epsilon)$ -approximation in worst-case time $T = O(n^3/\epsilon)$, where each bid has a fixed number of piecewise constant pieces.

Given this scheme, a straightforward computation of the VCG payments to all n agents requires time $O(nT)$. We compute approximate VCG payments in worst-case time $O(\alpha T \log(\alpha n/\epsilon))$, where α is a constant that quantifies a reasonable “no-monopoly” assumption. Specifically, in the reverse auction, suppose that $C(\mathcal{I})$ is the minimal cost for procuring M units with all sellers \mathcal{I} , and $C(\mathcal{I} \setminus i)$ is the minimal cost without seller i . Then, the constant α is defined as an upper bound for the ratio $C(\mathcal{I} \setminus i)/C(\mathcal{I})$, over all sellers i . This upper-bound tends to 1 as the number of sellers increases.

The approximate VCG mechanism is $(\frac{\epsilon}{1+\epsilon})$ -strategyproof for an approximation to within $(1 + \epsilon)$ of the optimal allocation. This means that a bidder can gain at most $(\frac{\epsilon}{1+\epsilon})V$ from a non-truthful bid, where V is the total surplus from the efficient allocation. As such, this is an example of a computationally-tractable ϵ -dominance result.² In practice, we can have good confidence that bidders without good information about the bidding strategies of other participants will have little to gain from attempts at manipulation.

Section 2 formally defines the forward and reverse auctions, and defines the VCG mechanisms. We also prove our claims about ϵ -strategyproofness. Section 3 provides the generalized knapsack formulation for the multi-unit allocation problems and introduces the fully polynomial time approximation scheme. Section 4 defines the approximation scheme for the payments in the VCG mechanism. Section 5 concludes.

1.1 Related Work

There has been considerable interest in recent years in characterizing polynomial-time or approximable special cases of the general combinatorial allocation problem, in which there are multiple different items. The combinatorial allocation problem (CAP) is both NP-complete and inapproximable (e.g. [6]). Although some polynomial-time cases have been identified for the CAP [6, 20], introducing an expressive *exclusive-or* bidding language quickly breaks these special cases. We identify a non-trivial but approximable allocation problem with an expressive *exclusive-or* bidding language—the bid taker in our setting is allowed to accept at most one point on the bid curve.

The idea of using approximations within mechanisms, while retaining either full-strategyproofness or ϵ -dominance has received some previous attention. For instance, Lehmann et al. [15] propose a greedy and strategyproof approximation to a single-minded combinatorial auction problem. Nisan & Ronen [18] discussed approximate VCG-based mechanisms, but either appealed to particular *maximal-in-range* approximations to retain *full* strategyproofness, or to resource-bounded agents with information or computational limitations on the ability to compute strategies. Feigen-

minimum-lot size constraints from the buyers.

²However, this may not be an example of what Feigenbaum & Shenker refer to as a *tolerably-manipulable* mechanism [8] because we have not tried to bound the effect of such a manipulation on the efficiency of the outcome. VCG mechanisms do have a natural “self-correcting” property, though, because a useful manipulation to an agent is a reported value that *improves* the total value of the allocation based on the reports of other agents and the agent’s own value.

baum & Shenker [8] have defined the concept of *strategically faithful* approximations, and proposed the study of approximations as an important direction for algorithmic mechanism design. Schummer [21] and Parkes et al [19] have previously considered ϵ -dominance, in the context of economic impossibility results, for example in combinatorial exchanges.

Eso et al. [7] have studied a similar procurement problem, but for a different volume discount model. This earlier work formulates the problem as a general mixed integer linear program, and gives some empirical results on simulated data. Kalagnanam et al. [12] address double auctions, where multiple buyers and sellers trade a *divisible good*. The focus of this paper is also different: it investigates the *equilibrium* prices using the demand and supply curves, whereas our focus is on efficient mechanism design. Ausubel [1] has proposed an *ascending-price* multi-unit auction for buyers with marginal-decreasing values [1], with an interpretation as a primal-dual algorithm [2].

2. APPROXIMATELY-STRATEGYPROOF VCG AUCTIONS

In this section, we first describe the marginal-decreasing piecewise bidding language that is used in our forward and reverse auctions. Continuing, we introduce the VCG mechanism for the problem and the ϵ -dominance results for approximations to VCG outcomes. We also discuss the economic properties of VCG mechanisms in these forward and reverse auction multi-unit settings.

2.1 Marginal-Decreasing Piecewise Bids

We provide a piecewise-constant and marginal-decreasing bidding language. This bidding language is expressive for a natural class of valuation and cost functions: fixed unit prices over intervals of quantities. See Figure 1 for an example. In addition, we slightly relax the marginal-decreasing requirement to allow: a bidder in the forward auction to state a *minimal purchase amount*, such that she has *zero* value for quantities smaller than that amount; a seller in the reverse auction to state a *capacity constraint*, such that she has an effectively infinite cost to supply quantities in excess of a particular amount.

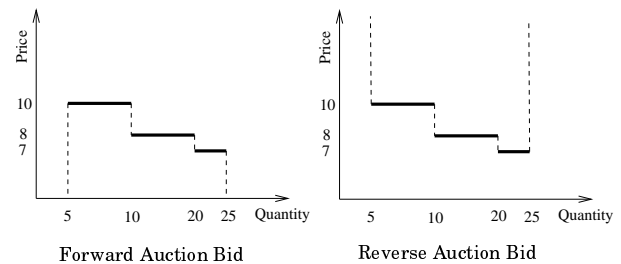


Figure 1: Marginal-decreasing, piecewise constant bids. In the forward auction bid, the bidder offers \$10 per unit for quantity in the range $[5, 10)$, \$8 per unit in the range $[10, 20)$, and \$7 in the range $[20, 25]$. Her valuation is zero for quantities outside the range $[10, 25]$. In the reverse auction bid, the cost of the seller is ∞ outside the range $[10, 25]$.

In detail, in a forward auction, a bid from buyer i can be written as a list of (quantity-range, unit-price) tuples, $((u_i^1, p_i^1), (u_i^2, p_i^2), \dots, (u_i^{m_i-1}, p_i^{m_i-1}))$, with an upper bound $u_i^{m_i}$ on the quantity. The interpretation is that the bidder’s valuation in the

(semi-open) quantity range $[u_i^j, u_i^{j+1})$ is p_i^j for each unit. Additionally, it is assumed that the valuation is 0 for quantities less than u_i^1 as well as for quantities more than u_i^m . This is implemented by adding two dummy bid tuples, with zero prices in the range $[0, u_i^1)$ and (u_i^m, ∞) . We interpret the bid list as defining a price function, $p_{\text{bid},i}(q) = qp_i^j$, if $u_i^j \leq q < u_i^{j+1}$, where $j = 1, 2, \dots, m_i - 1$. In order to resolve the boundary condition, we assume that the bid price for the upper bound quantity $u_i^{m_i}$ is $p_{\text{bid},i}(u_i^{m_i}) = u_i^{m_i} p_i^{m_i - 1}$.

A seller's bid is similarly defined in the reverse auction. The interpretation is that the bidder's cost in the (semi-open) quantity range $[u_i^j, u_i^{j+1})$ is p_i^j for each unit. Additionally, it is assumed that the cost is ∞ for quantities less than u_i^1 as well as for quantities more than u_i^m . Equivalently, the unit prices in the ranges $[0, u_i^1)$ and (u_i^m, ∞) are infinity. We interpret the bid list as defining a price function, $p_{\text{ask},i}(q) = qp_i^j$, if $u_i^j \leq q < u_i^{j+1}$.

2.2 VCG-Based Multi-Unit Auctions

We construct the tractable and approximately-strategyproof multi-unit auctions around a VCG mechanism. We assume that all agents have quasilinear utility functions; that is, $u_i(q, p) = v_i(q) - p$, for a buyer i with valuation $v_i(q)$ for q units at price p , and $u_i(q, p) = p - c_i(q)$ for a seller i with cost $c_i(q)$ at price p . This is a standard assumption in the auction literature, equivalent to assuming risk-neutral agents [13]. We will use the term *payoff* interchangeably for *utility*.

In the **forward auction**, there is a seller with M units to sell. We assume that this seller has no intrinsic value for the items. Given a set of bids from \mathcal{I} agents, let $V(\mathcal{I})$ denote the maximal revenue to the seller, given that at most one point on the bid curve can be selected from each agent and no more than M units of the item can be sold. Let $x^* = (x_1^*, \dots, x_N^*)$ denote the solution to this winner-determination problem, where x_i^* is the number of units sold to agent i . Similarly, let $V(\mathcal{I} \setminus i)$ denote the maximal revenue to the seller without bids from agent i . The VCG mechanism is defined as follows:

1. Receive piecewise-constant bid curves and capacity constraints from all the buyers.
2. Implement the outcome x^* that solves the winner-determination problem with all buyers.
3. Collect payment $p_{\text{vcg},i} = p_{\text{bid},i}(x_i^*) - [V(\mathcal{I}) - V(\mathcal{I} \setminus i)]$ from each buyer, and pass the payments to the seller.

In this forward auction, the VCG mechanism is *strategyproof* for buyers, which means that truthful bidding is a dominant strategy, i.e. utility maximizing whatever the bids of other buyers. In addition, the VCG mechanism is *allocatively-efficient*, and the payments from each buyer are always positive.³ Moreover, each buyer pays less than its value, and receives payoff $V(\mathcal{I}) - V(\mathcal{I} \setminus i)$ in equilibrium; this is precisely the marginal-value that buyer i contributes to the economic efficiency of the system.

In the **reverse auction**, there is a buyer with M units to buy, and n suppliers. We assume that the buyer has value $V > 0$ to purchase all M units, but zero value otherwise. To simplify the mechanism design problem we assume that the buyer will truthfully announce this value to the mechanism.⁴ The winner-

³In fact, the VCG mechanism maximizes the expected payoff to the seller across all efficient mechanisms, even allowing for Bayesian-Nash implementations [14].

⁴Without this assumption, the Myerson-Satterthwaite [17] impossibility result would already imply that we should not expect an efficient trading mechanism in this setting.

determination problem in the reverse auction is to determine the allocation, x^* , that *minimizes* the cost to the buyer, or forfeits trade if the minimal cost is greater than value, V .

Let $C(\mathcal{I})$ denote the minimal cost given bids from all sellers, and let $C(\mathcal{I} \setminus i)$ denote the minimal cost without bids from seller i . We can assume, without loss of generality, that there is an efficient trade and $V \geq C(\mathcal{I})$. Otherwise, then the efficient outcome is *no trade*, and the outcome of the VCG mechanism is no trade and no payments.

The VCG mechanism implements the outcome x^* that minimizes cost based on bids from all sellers, and then provides payment $p_{\text{vcg},i} = p_{\text{ask},i}(x_i^*) + [V - C(\mathcal{I}) - \max(0, V - C(\mathcal{I} \setminus i))]$ to each seller. The total payment is collected from the buyer. Again, in equilibrium each seller's payoff is exactly the marginal-value that the seller contributes to the economic efficiency of the system; in the simple case that $V \geq C(\mathcal{I} \setminus i)$ for all sellers i , this is precisely $C(\mathcal{I} \setminus i) - C(\mathcal{I})$.

Although the VCG mechanism remains strategyproof for sellers in the reverse direction, its applicability is limited to cases in which the total payments to the sellers are less than the buyer's value. Otherwise, there will be instances in which the buyer will not choose to voluntarily participate in the mechanism, based on its own value and its beliefs about the costs of sellers. This leads to a loss in efficiency when the buyer chooses not to participate, because efficient trades are missed. This problem with the size of the payments, does not occur in simple single-item reverse auctions, or even in multi-unit reverse auctions with a buyer that has a *constant* marginal-valuation for each additional item that she procures.⁵

Intuitively, the problem occurs in the reverse multi-unit setting because the buyer demands a fixed number of items, and has zero value without them. This leads to the possibility of the trade being contingent on the presence of particular, so-called "pivotal" sellers. Define a seller i as pivotal, if $C(\mathcal{I}) \leq V$ but $C(\mathcal{I} \setminus i) > V$. In words, there would be no efficient trade without the seller. Any time there is a pivotal seller, the VCG payments to that seller allow her to extract *all* of the surplus, and the payments are too large to sustain with the buyer's value unless this is the only winning seller.

Concretely, we have this participation problem in the reverse auction when the total payoff to the sellers, in equilibrium, exceeds the total payoff from the efficient allocation:

$$V - C(\mathcal{I}) \geq \sum_i [V - C(\mathcal{I}) - \max(0, V - C(\mathcal{I} \setminus i))]$$

As stated above, first notice that we require $V > C(\mathcal{I} \setminus i)$ for all sellers i . In other words, there must be no pivotal sellers. Given this, it is then necessary and sufficient that:

$$V - C(\mathcal{I}) \geq \sum_i (C(\mathcal{I} \setminus i) - C(\mathcal{I})) \quad (1)$$

⁵To make the reverse auction symmetric with the forward direction, we would need a buyer with a *constant* marginal-value to buy the first M units, and zero value for additional units. The payments to the sellers would never exceed the buyer's value in this case. Conversely, to make the forward auction symmetric with the reverse auction, we would need a seller with a constant (and high) marginal-cost to sell anything less than the first M units, and then a low (or zero) marginal cost. The total payments received by the seller can be less than the seller's cost for the outcome in this case.

In words, the surplus of the efficient allocation must be greater than the total marginal-surplus provided by each seller.⁶

Consider an example with 3 agents $\{1, 2, 3\}$, and $V = 150$ and $C(123) = 50$. Condition (1) holds when $C(12) = C(23) = 70$ and $C(13) = 100$, but not when $C(12) = C(23) = 80$ and $C(13) = 100$. In the first case, the agent payoffs $\pi = (\pi_0, \pi_1, \pi_2, \pi_3)$, where 0 is the seller, is $(10, 20, 50, 20)$. In the second case, the payoffs are $\pi = (-10, 30, 50, 30)$.

One thing we do know, because the VCG mechanism will maximize the payoff to the buyer across all efficient mechanisms [14], is that whenever Eq. 1 is not satisfied there can be *no* efficient auction mechanism.⁷

2.3 ϵ -Strategyproofness

We now consider the same VCG mechanism, but with an approximation scheme for the underlying allocation problem. We derive an ϵ -strategyproofness result, that bounds the maximal gain in payoff that an agent can expect to achieve through a unilateral deviation from following a simple truth-revealing strategy. We describe the result for the forward auction direction, but it is quite a general observation.

As before, let $V(\mathcal{I})$ denote the value of the optimal solution to the allocation problem with truthful bids from all agents, and $V(\mathcal{I} \setminus i)$ denote the value of the optimal solution computed without bids from agent i . Let $\hat{V}(\mathcal{I})$ and $\hat{V}(\mathcal{I} \setminus i)$ denote the value of the allocation computed with an approximation scheme, and assume that the approximation satisfies:

$$(1 + \epsilon)\hat{V}(\mathcal{I}) \geq V(\mathcal{I})$$

for some $\epsilon > 0$. We provide such an approximation scheme for our setting later in the paper. Let \hat{x} denote the allocation implemented by the approximation scheme.

The payoff to agent i , for announcing valuation \hat{v}_i , is:

$$v_i(\hat{x}_i) + \sum_{j \neq i} \hat{v}_j(\hat{x}_j) - \hat{V}(\mathcal{I} \setminus i)$$

The final term is independent of the agent's announced value, and can be ignored in an incentive-analysis. However, agent i can try to improve its payoff through the effect of its announced value on the allocation \hat{x} implemented by the mechanism. In particular, agent i wants the mechanism to select \hat{x} to maximize the sum of its *true* value, $v_i(\hat{x}_i)$, and the reported value of the other agents, $\sum_{j \neq i} \hat{v}_j(\hat{x}_j)$. If the mechanism's allocation algorithm is optimal, then all the agent needs to do is truthfully state its value and the mechanism will do the rest. However, faced with an approximate allocation algorithm, the agent can try to improve its payoff by announcing a value that *corrects* for the approximation, and causes the approximation algorithm to implement the allocation that exactly maximizes the total reported value of the other agents together with its own actual value [18].

⁶This condition is implied by the *agents are substitutes* requirement [3], that has received some attention in the combinatorial auction literature because it characterizes the case in which VCG payments can be supported in a competitive equilibrium. Useful characterizations of conditions that satisfy agents are substitutes, in terms of the underlying valuations of agents have proved quite elusive.

⁷Moreover, although there is a small literature on *maximally-efficient* mechanisms subject to requirements of voluntary-participation and budget-balance (i.e. with the mechanism neither introducing or removing money), analytic results are only known for simple problems (e.g. [16, 4]).

We can now analyze the best possible gain from manipulation to an agent in our setting. We first assume that the other agents are truthful, and then relax this. In both cases, the maximal benefit to agent i occurs when the initial approximation is worst-case. With truthful reports from other agents, this occurs when the value of choice \hat{x} is $V(\mathcal{I})/(1 + \epsilon)$. Then, an agent could hope to receive an improved payoff of:

$$V(\mathcal{I}) - \frac{V(\mathcal{I})}{1 + \epsilon} = \frac{\epsilon}{1 + \epsilon} V(\mathcal{I})$$

This is possible if the agent is able to select a reported type to correct the approximation algorithm, and make the algorithm implement the allocation with value $V(\mathcal{I})$. Thus, if other agents are truthful, and with a $(1 + \epsilon)$ -approximation scheme to the allocation problem, then no agent can improve its payoff by more than a factor $\epsilon/(1 + \epsilon)$ of the value of the optimal solution.

The analysis is very similar when the other agents are not truthful. In this case, an individual agent can improve its payoff by no more than a factor $\epsilon/(1 + \epsilon)$ of the *value of the optimal solution given the values reported by the other agents*.

Let V in the following theorem define the total value of the efficient allocation, given the *reported* values of agents $j \neq i$, and the true value of agent i .

THEOREM 1. *A VCG-based mechanism with a $(1 + \epsilon)$ -allocation algorithm is $(\frac{\epsilon}{1 + \epsilon} - V)$ strategyproof for agent i , and agent i can gain at most this payoff through some non-truthful strategy.*

Notice that we did not need to bound the error on the allocation problems without each agent, because the ϵ -strategyproofness result follows from the accuracy of the first-term in the VCG payment and is independent of the accuracy of the second-term. However, the accuracy of the solution to the problem without each agent is important to implement a good approximation to the *revenue* properties of the VCG mechanism.

3. THE GENERALIZED KNAPSACK PROBLEM

In this section, we design a fully polynomial approximation scheme for the generalized knapsack, which models the winner-determination problem for the VCG-based multi-unit auctions. We describe our results for the reverse auction variation, but the formulation is completely symmetric for the forward-auction.

In describing our approximation scheme, we begin with a simple property (the *Anchor property*) of an optimal knapsack solution. We use this property to develop an $O(n^2)$ time 2-approximation for the generalized knapsack. In turn, we use this basic approximation to develop our fully polynomial-time approximation scheme (FPTAS).

One of the major appeals of our piecewise bidding language is its *compact representation* of the bidder's valuation functions. We strive to preserve this, and present an approximation scheme that will depend only on the number of bidders, and not the maximum quantity, M , which can be very large in realistic procurement settings.

The FPTAS implements an $(1 + \epsilon)$ approximation to the optimal solution x^* , in worst-case time $T = O(n^3/\epsilon)$, where n is the number of bidders, and where we assume that the piecewise bid for each bidder has $O(1)$ pieces. The dependence on the number of pieces is also polynomial: if each bid has a maximum

of c pieces, then the running time can be derived by substituting nc for each occurrence of n .

3.1 Preliminaries

Before we begin, let us recall the classic 0/1 knapsack problem: we are given a set of n items, where the item i has *value* v_i and *size* s_i , and a knapsack of capacity M ; all sizes are integers. The goal is to determine a subset of items of maximum value with total size at most M . Since we want to focus on a reverse auction, the equivalent knapsack problem will be to choose a set of items with *minimum value* (i.e. cost) whose size *exceeds* M . The *generalized knapsack problem* of interest to us can be defined as follows:

Generalized Knapsack:

Instance: A target M , and a set of n lists, where the i th list has the form

$$B_i = \langle (u_i^1, p_i^1), \dots, (u_i^{m_i-1}, p_i^{m_i-1}), (u_i^{m_i}(i), \infty) \rangle,$$

where u_i^j are increasing with j and p_i^j are decreasing with j , and u_i^j, p_i^j, M are positive integers.

Problem: Determine a set of integers x_i^j such that

1. (One per list) At most one x_i^j is non-zero for any i ,
2. (Membership) $x_i^j \neq 0$ implies $x_i^j \in [u_i^j, u_i^{j+1})$,
3. (Target) $\sum_i \sum_j x_i^j \geq M$, and
4. (Objective) $\sum_i \sum_j p_i^j x_i^j$ is minimized.

This generalized knapsack formulation is a clear generalization of the classic 0/1 knapsack. In the latter, each list consists of a single *point* (s_i, v_i) .⁸

The connection between the generalized knapsack and our auction problem is transparent. Each list encodes a bid, representing multiple *mutually exclusive* quantity intervals, and one can choose any quantity in an interval, but at most one interval can be selected. Choosing interval $[u_i^j, u_i^{j+1})$ has cost p_i^j per unit. The goal is to procure at least M units of the good at minimum possible cost. The problem has some flavor of the *continuous knapsack problem*. However, there are two major differences that make our problem significantly more difficult: (1) intervals have boundaries, and so to choose interval $[u_i^j, u_i^{j+1})$ requires that at least u_i^j and at most u_i^{j+1} units must be taken; (2) unlike the classic knapsack, we cannot sort the items (bids) by value/size, since different intervals in one list have different unit costs.

3.2 A 2-Approximation Scheme

We begin with a definition. Given an instance of the generalized knapsack, we call each tuple $t_i^j = (u_i^j, p_i^j)$ an *anchor*. Recall that these tuples represent the breakpoints in the piecewise constant curve bids. We say that the *size* of an anchor t_i^j is u_i^j ,

⁸In fact, because of the ‘‘one per list’’ constraint, the generalized problem is closer in spirit to the *multiple choice knapsack problem* [9], where the underlying set of items is partitioned into disjoint subsets U_1, U_2, \dots, U_k , and one can choose at most one item from each subset. PTAS do exist for this problem [10], and indeed, one can convert our problem into a *huge* instance of the multiple choice knapsack problem, by creating one group for each list; put a (quantity, price) point tuple (x, p) for each possible quantity for a bidder into his group (subset). However, this conversion explodes the problem size, making it infeasible for all but the most trivial instances.

the minimum number of units available at this anchor’s price p_i^j . The *cost* of the anchor t_i^j is defined to be the minimum total price associated with this tuple, namely, $cost(t_i^j) = p_i^j u_i^j$ if $j < m_i$, and $cost(t_i^{m_i}) = p_i^{m_i-1} u_i^{m_i}$.

In a feasible solution $\{x_1, x_2, \dots, x_n\}$ of the generalized knapsack, we say that an element $x_i \neq 0$ is an *anchor* if $x_i = u_i^j$, for some anchor u_i^j . Otherwise, we say that x_i is *midrange*. We observe that an optimal knapsack solution can always be constructed so that at most one solution element is midrange. If there are two midrange elements x and x' , for bids from two different agents, with $x \leq x'$, then we can increment x' and decrement x , until one of them becomes an anchor. See Figure 2 for an example.

LEMMA 1. [Anchor Property] *There exists an optimal solution of the generalized knapsack problem with at most one midrange element. All other elements are anchors.*

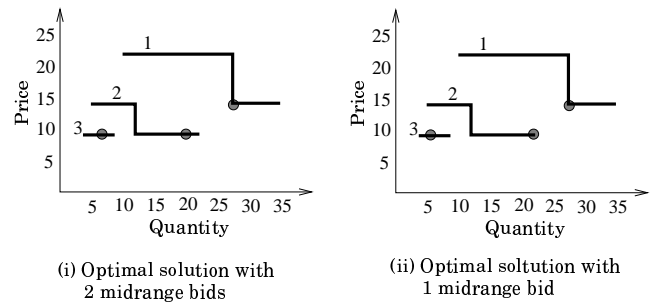


Figure 2: (i) An optimal solution with more than one bid not anchored (2,3); (ii) an optimal solution with only one bid (3) not anchored.

We use the anchor property to first obtain a polynomial-time 2-approximation scheme. We do this by solving several instances of a restricted generalized-knapsack problem, which we call *iKnapsack*, where one element is forced to be midrange for a particular interval.

Specifically, suppose element x_ℓ for agent l is forced to lie in its j th range, $[u_\ell^j, u_\ell^{j+1})$, while all other elements, $x_1, \dots, x_{l-1}, x_{l+1}, x_n$, are required to be anchors, or zero. This corresponds to the restricted problem *iKnapsack*(ℓ, j), in which the goal is to obtain at least $M - u_\ell^j$ units with minimum cost. Element x_ℓ is assumed to have already contributed u_ℓ^j units. The value of a solution to *iKnapsack*(ℓ, j) represents the minimal *additional* cost to purchase the rest of the units.

We create $n - 1$ groups of potential anchors, where i th group contains all the anchors of the list i in the generalized knapsack. The group for agent l contains a single element that represents the interval $[0, u_\ell^{j+1} - u_\ell^j)$, and the associated unit-price p_ℓ^j . This interval represents the excess number of units that can be taken from agent l in *iKnapsack*(ℓ, j), in addition to u_ℓ^j , which has already been committed. In any other group, we can choose at most one anchor.

The following pseudo-code describes our algorithm for this restriction of the generalized knapsack problem. U is the union of all the tuples in n groups, including a tuple t_ℓ for agent l . The *size* of this special tuple is defined as $u_\ell^{j+1} - u_\ell^j$, and the *cost* is defined as $p_\ell^j (u_\ell^{j+1} - u_\ell^j)$. R is the number of units that remain to be acquired. S is the set of tuples accepted in the current tentative

solution. $Best$ is the best solution found so far. Variable $Skip$ is only used in the proof of correctness.

Algorithm Greedy(ℓ, j)

1. Sort all tuples of U in the ascending order of unit price; in case of ties, sort in ascending order of unit quantities.
2. Set $mark(i) = 0$, for all lists $i = 1, 2, \dots, n$.
Initialize $R = M - u_i^j$, $S = Best = Skip = \emptyset$.
3. Scan the tuples in U in the sorted order. Suppose the next tuple is t_i^k , i.e. the k th anchor from agent i .
If $mark(i) = 1$, ignore this tuple;
otherwise do the following steps:
 - if $size(t_i^k) > R$ and $i = \ell$
return $\min \{cost(S) + Rp_\ell^j, cost(Best)\}$;
 - if $size(t_i^k) > R$ and $cost(t_i^k) \leq cost(S)$
return $\min \{cost(S) + cost(t_i^k), cost(Best)\}$;
 - if $size(t_i^k) > R$ and $cost(t_i^k) > cost(S)$
Add t_i^k to $Skip$; Set $Best$ to $S \cup \{t_i^k\}$ if cost improves;
 - if $size(t_i^k) \leq R$ then
add t_i^k to S ; $mark(i) = 1$; subtract $size(t_i^k)$ from R .

The approximation algorithm is very similar to the approximation algorithm for knapsack. Since we wish to minimize the total cost, we consider the tuples in order of increasing per unit cost. If the size of tuple t_i^k is smaller than R , then we add it to S , update R , and delete from U all the tuples that belong to the same group as t_i^k . If $size(t_i^k)$ is greater than R , then S along with t_i^k forms a feasible solution. However, this solution can be far from optimal if the size of t_i^k is much larger than R . If total cost of S and t_i^k is smaller than the current best solution, we update $Best$. One exception to this rule is the tuple t_ℓ . Since this tuple can be taken fractionally, we update $Best$ if the sum of S 's cost and fractional cost of t_ℓ is an improvement.

The algorithm terminates in either of the first two cases, or when all tuples are scanned. In particular, it terminates whenever we find a t_i^k such that $size(t_i^k)$ is greater than R but $cost(t_i^k)$ is less than $cost(S)$, or when we reach the tuple representing agent l and it gives a feasible solution.

LEMMA 2. *Suppose A^* is an optimal solution of the generalized knapsack, and suppose that element (l, j) is midrange in the optimal solution. Then, the cost $V(l, j)$, returned by Greedy(ℓ, j), satisfies:*

$$V(\ell, j) + cost(t_\ell^j) \leq 2cost(A^*)$$

PROOF. Let $V(\ell, j)$ be the value returned by Greedy(ℓ, j) and let $V^*(\ell, j)$ be an optimal solution for $iKnapsack(\ell, j)$. Consider the set $Skip$ at the termination of Greedy(ℓ, j). There are two cases to consider: either some tuple $t \in Skip$ is also in $V^*(\ell, j)$, or no tuple in $Skip$ is in $V^*(\ell, j)$. In the first case, let S_t be the tentative solution S at the time t was added to $Skip$. Because $t \in Skip$ then $size(t) > R$, and S_t together with t forms a feasible solution, and we have:

$$V(\ell, j) \leq cost(Best) \leq cost(S_t) + cost(t).$$

Again, because $t \in Skip$ then $cost(t) > cost(S_t)$, and we have $V(\ell, j) < 2cost(t)$. On the other hand, since t is included in

$V^*(\ell, j)$, we have $V^*(\ell, j) \geq cost(t)$. These two inequalities imply the desired bound:

$$V^*(\ell, j) \leq V(\ell, j) < 2V^*(\ell, j).$$

In the second case, imagine a modified instance of $iKnapsack(\ell, j)$, which *excludes* all the tuples of the set $Skip$. Since none of these tuples were included in $V^*(\ell, j)$, the optimal solution for the modified problem should be the same as the one for the original. Suppose our approximation algorithm returns the value $V'(\ell, j)$ for this modified instance. Let t' be the last tuple considered by the approximation algorithm before termination on the modified instance, and let $S_{t'}$ be the corresponding tentative solution set in that step. Since we consider tuples in order of increasing per unit price, and none of the tuples are going to be placed in the set $Skip$, we must have $cost(S_{t'}) < V^*(\ell, j)$ because $S_{t'}$ is the optimal way to obtain $size(S_{t'})$.

We also have $cost(t') \leq cost(S_{t'})$, and the following inequalities:

$$\begin{aligned} V(\ell, j) \leq V'(\ell, j) &\leq cost(S_{t'}) + cost(t') \\ &< 2V^*(\ell, j) \end{aligned}$$

The inequality $V(\ell, j) \leq V'(\ell, j)$ follows from the fact that a tuple in the $Skip$ list can only affect the $Best$ but not the tentative solutions. Therefore, dropping the tuples in the set $Skip$ can only make the solution worse.

The above argument has shown that the value returned by Greedy(ℓ, j) is within a factor 2 of the optimal solution for $iKnapsack(\ell, j)$. We now show that the value $V(\ell, j)$ plus $cost(t_\ell^j)$ is a 2-approximation of the original generalized knapsack problem.

Let A^* be an optimal solution of the generalized knapsack, and suppose that element x_ℓ^j is midrange. Let $x_{-\ell}$ be set of the remaining elements, either zero or anchors, in this solution. Furthermore, define $x'_\ell = x_\ell^j - u_\ell^j$. Thus,

$$cost(A^*) = cost(x'_\ell) + cost(t_\ell^j) + cost(x_{-\ell})$$

It is easy to see that $(x_{-\ell}, x'_\ell)$ is an optimal solution for $iKnapsack(\ell, j)$. Since $V(\ell, j)$ is a 2-approximation for this optimal solution, we have the following inequalities:

$$\begin{aligned} V(\ell, j) + cost(t_\ell^j) &\leq cost(t_\ell^j) + 2(cost(x'_\ell) + cost(x_{-\ell})) \\ &\leq 2(cost(x'_\ell) + cost(t_\ell^j) + cost(x_{-\ell})) \\ &\leq 2cost(A^*) \end{aligned}$$

This completes the proof of Lemma 2. \square

It is easy to see that, after an initial sorting of the tuples in U , the algorithm Greedy(ℓ, j) takes $O(n)$ time. We have our first polynomial approximation algorithm.

THEOREM 2. *A 2-approximation of the generalized knapsack problem can be found in time $O(n^2)$, where n is number of item lists (each of constant length).*

PROOF. We run the algorithm Greedy(ℓ, j) once for each tuple (l, j) as a candidate for midrange. There are $O(n)$ tuples, and it suffices to sort them once, the total cost of the algorithm is $O(n^2)$. By Lemma 1, there is an optimal solution with at most one midrange element, so our algorithm will find a 2-approximation, as claimed. \square

The dependence on the number of pieces is also polynomial: if each bid has a maximum of c pieces, then the running time is $O((nc)^2)$.

3.3 An Approximation Scheme

We now use the 2-approximation algorithm presented in the preceding section to develop a fully polynomial approximation (FPTAS) for the generalized knapsack problem. The high level idea is fairly standard, but the details require technical care. We use a dynamic programming algorithm to solve $iKnapsack(\ell, j)$ for each possible midrange element, with the 2-approximation algorithm providing an upper bound on the value of the solution and enabling the use of scaling on the cost dimension of the dynamic programming (DP) table.

Consider, for example, the case that the midrange element is x_ℓ , which falls in the range $[u_\ell^j, u_\ell^{j+1})$. In our FPTAS, rather than using a greedy approximation algorithm to solve $iKnapsack(\ell, j)$, we construct a dynamic programming table to compute the minimum cost at which at least $M - u_\ell^{j+1}$ units can be obtained using the remaining $n - 1$ lists in the generalized knapsack.

Suppose $G[i, r]$ denotes the *maximum* number of units that can be obtained at cost at most r using only the first i lists in the generalized knapsack. Then, the following recurrence relation describes how to construct the dynamic programming table:

$$G[0, r] = 0$$

$$G[i, r] = \max \left\{ \begin{array}{l} G[i-1, r] \\ \max_{j \in \beta(i, r)} \{G[i-1, r - \text{cost}(t_i^j)] + u_i^j\} \end{array} \right\}$$

where $\beta(i, r) = \{j : 1 \leq j \leq m_i, \text{cost}(t_i^j) \leq r\}$, is the set of anchors for agent i . As convention, agent i will index the *row*, and cost r will index the *column*.

This dynamic programming algorithm is only pseudo-polynomial, since the number of column in the dynamic programming table depends upon the total cost. However, we can convert it into a FPTAS by *scaling* the cost dimension.

Let A denote the 2-approximation to the generalized knapsack problem, with total cost, $\text{cost}(A)$. Let ε denote the desired approximation factor. We compute the *scaled cost* of a tuple t_i^j , denoted $\text{scost}(t_i^j)$, as

$$\text{scost}(t_i^j) = \left\lceil \frac{n \text{cost}(t_i^j)}{\varepsilon \text{cost}(A)} \right\rceil \quad (2)$$

This scaling improves the running time of the algorithm because the number of columns in the modified table is at most $\lceil \frac{n}{\varepsilon} \rceil$, and independent of the total cost. However, the computed solution might not be an optimal solution for the original problem. We show that the error introduced is within a factor of ε of the optimal solution.

As a prelude to our approximation guarantee, we first show that if two different solutions to the $iKnapsack$ problem have equal scaled cost, then their *original* (unscaled) costs cannot differ by more than $\varepsilon \text{cost}(A)$.

LEMMA 3. *Let x and y be two distinct feasible solutions of $iKnapsack(\ell, j)$, excluding their midrange elements. If x and y have equal scaled costs, then their unscaled costs cannot differ by more than $\varepsilon \text{cost}(A)$.*

PROOF. Let I_x and I_y , respectively, denote the indicator functions associated with the anchor vectors x and y —there is 1 in position $I_x[i, k]$ if the $x_i^k > 0$. Since x and y has equal scaled cost,

$$\sum_{i \neq \ell} \sum_k \text{scost}(t_i^k) I_x[i, k] = \sum_{i \neq \ell} \sum_k \text{scost}(t_i^k) I_y[i, k] \quad (3)$$

However, by (2), the scaled costs satisfy the following inequalities:

$$\frac{(\text{scost}(t_i^k) - 1)\varepsilon \text{cost}(A)}{n} \leq \text{cost}(t_i^k) \leq \frac{\text{scost}(t_i^k)\varepsilon \text{cost}(A)}{n} \quad (4)$$

Substituting the upper-bound on scaled cost from (4) for $\text{cost}(x)$, the lower-bound on scaled cost from (4) for $\text{cost}(y)$, and using equality (3) to simplify, we have:

$$\text{cost}(x) - \text{cost}(y) \leq \frac{\varepsilon \text{cost}(A)}{n} \sum_{i \neq \ell} \sum_k I_y[i, k] \leq \varepsilon \text{cost}(A),$$

The last inequality uses the fact that at most n components of an indicator vector are non-zero; that is, any feasible solution contains at most n tuples. \square

Finally, given the dynamic programming table for $iKnapsack(\ell, j)$, we consider all the entries in the last row of this table, $G[n-1, r]$. These entries correspond to optimal solutions with all agents except l , for different levels of cost. In particular, we consider the entries that provide at least $M - u_\ell^{j+1}$ units. Together with a contribution from agent l , we choose the entry in this set that *minimizes* the total cost, defined as follows:

$$\text{cost}(G[n-1, r]) + \max \{u_\ell^j, M - G[n-1, r]\} p_\ell^j,$$

where $\text{cost}()$ is the original, unscaled cost associated with entry $G[n-1, r]$. It is worth noting, that unlike the 2-approximation scheme for $iKnapsack(\ell, j)$, the value computed with this FPTAS includes the cost to acquire u_ℓ^j units from l .

The following lemma shows that we achieve a $(1+\varepsilon)$ -approximation.

LEMMA 4. *Suppose A^* is an optimal solution of the generalized knapsack problem, and suppose that element (l, j) is midrange in the optimal solution. Then, the solution $A(l, j)$ from running the scaled dynamic-programming algorithm on $iKnapsack(\ell, j)$ satisfies*

$$\text{cost}(A(l, j)) \leq (1 + 2\varepsilon) \text{cost}(A^*)$$

PROOF. Let $x_{-\ell}$ denote the vector of the elements in solution A^* without element l . Then, by definition, $\text{cost}(A^*) = \text{cost}(x_{-\ell}) + p_\ell^j x_\ell^j$. Let $r = \text{scost}(x_{-\ell})$ be the scaled cost associated with the vector $x_{-\ell}$. Now consider the dynamic programming table constructed for $iKnapsack(\ell, j)$, and consider its entry $G[n-1, r]$. Let A denote the 2-approximation to the generalized knapsack problem, and $A(l, j)$ denote the solution from the dynamic-programming algorithm.

Suppose $y_{-\ell}$ is the solution associated with this entry in our dynamic program; the components of the vector $y_{-\ell}$ are the quantities from different lists. Since both $x_{-\ell}$ and $y_{-\ell}$ have equal scaled costs, by Lemma 3, their unscaled costs are within $\varepsilon \text{cost}(A)$ of each other; that is,

$$\text{cost}(y_{-\ell}) - \text{cost}(x_{-\ell}) \leq \varepsilon \text{cost}(A).$$

Now, define $y_\ell^j = \max\{u_\ell^j, M - \sum_{i \neq \ell} \sum_j y_i^j\}$; this is the contribution needed from l to make $(y_{-\ell}, y_\ell^j)$ a feasible solution. Among all the equal cost solutions, our dynamic programming tables chooses the one with maximum units. Therefore,

$$\sum_{i \neq \ell} \sum_j y_i^j \geq \sum_{i \neq \ell} \sum_j x_i^j$$

Therefore, it must be the case that $y_\ell^j \leq x_\ell^j$. Because $(y_\ell^j, y_{-\ell})$ is also a feasible solution, if our algorithm returns a solution with cost $\text{cost}(A(l, j))$, then we must have

$$\begin{aligned} \text{cost}(A(l, j)) &\leq \text{cost}(y_{-\ell}) + p_\ell^j y_\ell^j \\ &\leq \text{cost}(x_{-\ell}) + \varepsilon \text{cost}(A) + p_\ell^j x_\ell^j \\ &\leq (1 + 2\varepsilon) \text{cost}(A^*), \end{aligned}$$

where we use the fact that $\text{cost}(A) \leq 2\text{cost}(A^*)$. \square

Putting this together, our approximation scheme for the generalized knapsack problem will iterate the scheme described above for each choice of the midrange element (l, j) , and choose the best solution from among these $O(n)$ solutions.

For a given midrange, the most expensive step in the algorithm is the construction of dynamic programming table, which can be done in $O(n^2/\varepsilon)$ time assuming constant intervals per list. Thus, we have the following result.

THEOREM 3. *We can compute an $(1 + \varepsilon)$ approximation to the solution of a generalized knapsack problem in worst-case time $O(n^3/\varepsilon)$.*

The dependence on the number of pieces is also polynomial: if each bid has a maximum of c pieces, then the running time can be derived by substituting cn for each occurrence of n .

4. COMPUTING VCG PAYMENTS

We now consider the related problem of computing the VCG payments for all the agents. A naive approach requires solving the allocation problem n times, removing each agent in turn. In this section, we show that our approximation scheme for the generalized knapsack can be extended to determine all n payments in total time $O(\alpha T \log(\alpha n/\varepsilon))$, where $1 \leq C(\mathcal{I} \setminus i)/C(\mathcal{I}) \leq \alpha$, for a constant upper bound, α , and T is the complexity of solving the allocation problem once. This α -bound can be justified as a “no monopoly” condition, because it bounds the marginal value that a single buyer brings to the auction. Similarly, in the reverse variation we can compute the VCG payments to each seller in time $O(\alpha T \log(\alpha n/\varepsilon))$, where α bounds the ratio $C(\mathcal{I} \setminus i)/C(\mathcal{I})$ for all i .

Our overall strategy will be to build two dynamic programming tables, forward and backward, for each midrange element (l, j) once. The forward table is built by considering the agents in the order of their indices, where as the backward table is built by considering them in the reverse order. The optimal solution corresponding to $C(\mathcal{I} \setminus i)$ can be broken into two parts: one corresponding to first $(i - 1)$ agents and the other corresponding to last $(n - i)$ agents. As the $(i - 1)$ th row of the forward table corresponds to the sellers with first $(i - 1)$ indices, an approximation to the first part will be contained in $(i - 1)$ th row of the forward table. Similarly, $(n - i)$ th row of the backward table will contain an approximation for the second part. We first present a simple but an inefficient way of computing the approximate value of $C(\mathcal{I} \setminus i)$, which illustrates the main idea of our algorithm. Then we present an improved scheme, which uses the fact that the elements in the rows are sorted, to compute the approximate value more efficiently.

In the following, we concentrate on computing an allocation with x_ℓ^j being midrange, and some agent $i \neq l$ removed. This will be a component in computing an approximation to $C(\mathcal{I} \setminus i)$, the value of the solution to the generalized knapsack without bids from agent i . We begin with the simple scheme.

4.1 A Simple Approximation Scheme

We implement the scaled dynamic programming algorithm for $i\text{Knapsack}(\ell, j)$ with two alternate orderings over the other sellers, $k \neq l$, one with sellers ordered $1, 2, \dots, n$, and one with sellers ordered $n, n - 1, \dots, 1$. We call the first table the *forward* table, and denote it F_ℓ , and the second table the *backward* table, and denote it B_ℓ . The subscript ℓ reminds us that the agent ℓ is midrange.⁹

In building these tables, we use the same scaling factor as before; namely, the cost of a tuple t_i^j is scaled as follows:

$$\text{scost}(t_i^j) = \left\lceil \frac{n \text{cost}(t_i^j)}{\varepsilon \text{cost}(A)} \right\rceil$$

where $\text{cost}(A)$ is the upper bound on $C(\mathcal{I})$, given by our 2-approximation scheme. In this case, because $C(\mathcal{I} \setminus i)$ can be α times $C(\mathcal{I})$, the scaled value of $C(\mathcal{I} \setminus i)$ can be at most $n\alpha/\varepsilon$. Therefore, the cost dimension of our dynamic program’s table will be $n\alpha/\varepsilon$.

Table F_ℓ
Table B_ℓ

Figure 3: Computing VCG payments. $m = \frac{n\alpha}{\varepsilon}$

Now, suppose we want to compute a $(1 + \varepsilon)$ -approximation to the generalized knapsack problem restricted to element (l, j) midrange, and further restricted to *remove* bids from some seller $i \neq l$. Call this problem $i\text{Knapsack}^{-i}(\ell, j)$.

Recall that the i th row of our DP table stores the best solution possible using only the first i agents excluding agent l , all of them either cleared at zero, or on anchors. These first i agents are a different subset of agents in the forward and the backward tables. By carefully combining one row of F_ℓ with one row of B_ℓ we can compute an approximation to $i\text{Knapsack}^{-i}(\ell, j)$. We consider the row of F_ℓ that corresponds to solutions constructed from agents $\{1, 2, \dots, i - 1\}$, skipping agent l . We consider the row of B_ℓ that corresponds to solutions constructed from agents $\{i + 1, i + 2, \dots, n\}$, again skipping agent l . The rows are labeled $F_\ell(i - 1)$ and $B_\ell(n - i)$ respectively.¹⁰ The scaled costs for acquiring these units are the column indices for these entries. To solve $i\text{Knapsack}^{-i}(\ell, j)$ we choose one entry from row $F_\ell(i - 1)$ and one from row $B_\ell(n - i)$ such that their total quantity exceeds $M - u_\ell^{j+1}$ and their combined cost is minimum over all such combinations. Formally, let $g \in F_\ell(i - 1)$, and $h \in B_\ell(n - i)$ denote entries in each row, with $\text{size}(g)$, $\text{size}(h)$, denoting the number of units and $\text{cost}(g)$ and $\text{cost}(h)$ denoting the *unscaled* cost associated with the entry. We compute the following, *subject*

⁹We could label the tables with both ℓ and j , to indicate the j th tuple is forced to be midrange, but omit j to avoid clutter.

¹⁰To be precise, the index of the rows are $(i - 2)$ and $(n - i)$ for F_ℓ and B_ℓ when $l < i$, and $(i - 1)$ and $(n - i - 1)$, respectively, when $l > i$.

to the condition that g and h satisfy $size(g) + size(h) > M - u_\ell^{j+1}$:

$$\min_{g \in F_\ell(i-1), h \in B_\ell(n-i)} \left\{ cost(g) + cost(h) + p_\ell^j \cdot \max\{u_\ell^j, M - size(g) - size(h)\} \right\} \quad (5)$$

LEMMA 5. Suppose A^{-i} is an optimal solution of the generalized knapsack problem without bids from agent i , and suppose that element (l, j) is the midrange element in the optimal solution. Then, the expression in Eq. 5, for the restricted problem $iKnapsack^{-i}(\ell, j)$, computes a $(1 + \varepsilon)$ -approximation to A^{-i} .

PROOF. From earlier, we define $cost(A^{-i}) = C(\mathcal{I} \setminus i)$. We can split the optimal solution, A^{-i} , into three disjoint parts: x_l corresponds to the midrange seller, x_i corresponds to first $i - 1$ sellers (skipping agent l if $l < i$), and x_{-i} corresponds to last $n - i$ sellers (skipping agent l if $l > i$). We have:

$$cost(A^{-i}) = cost(x_i) + cost(x_{-i}) + p_\ell^j x_\ell^j$$

Let $r_i = scost(x_i)$ and $r_{-i} = scost(x_{-i})$. Let y_i and y_{-i} be the solution vectors corresponding to scaled cost r_i and r_{-i} in $F_\ell(i - 1)$ and $B_\ell(n - i)$, respectively. From Lemma 3 we conclude that,

$$cost(y_i) + cost(y_{-i}) - cost(x_i) - cost(x_{-i}) \leq \varepsilon cost(A)$$

where $cost(A)$ is the upper-bound on $C(\mathcal{I})$ computed with the 2-approximation.

Among all equal scaled cost solutions, our dynamic program chooses the one with maximum units. Therefore we also have,

$$(size(y_i) \geq size(x_i)) \text{ and } (size(y_{-i}) \geq size(x_{-i}))$$

where we use shorthand $size(x)$ to denote total number of units in all tuples in x .

Now, define $y_i^j = \max\{u_\ell^j, M - size(y_i) - size(y_{-i})\}$. From the preceding inequalities, we have $y_i^j \leq x_i^j$. Since (y_i^j, y_i, y_{-i}) is also a feasible solution to the generalized knapsack problem without agent i , the value returned by Eq. 5 is at most

$$\begin{aligned} cost(y_i) + cost(y_{-i}) + p_\ell^j y_i^j &\leq C(\mathcal{I} \setminus i) + \varepsilon cost(A) \\ &\leq C(\mathcal{I} \setminus i) + 2cost(A^*)\varepsilon \\ &\leq C(\mathcal{I} \setminus i) + 2C(\mathcal{I} \setminus i)\varepsilon \end{aligned}$$

This completes the proof. \square

A naive implementation of this scheme will be inefficient because it might check $(n\alpha/\varepsilon)^2$ pairs of elements, for any particular choice of (l, j) and choice of dropped agent i . In the next section, we present an efficient way to compute Eq. 5, and eventually to compute the VCG payments.

4.2 Improved Approximation Scheme

Our improved approximation scheme for the winner-determination problem without agent i uses the fact that elements in $F_\ell(i - 1)$ and $B_\ell(n - i)$ are sorted; specifically, both, unscaled $cost$ and quantity (i.e. $size$), increases from left to right. As before, let g and h denote generic entries in $F_\ell(i - 1)$ and $B_\ell(n - i)$ respectively. To compute Eq. 5, we consider all the tuple pairs, and first divide the tuples that satisfy condition $size(g) + size(h) > M - u_\ell^{j+1}$ into two disjoint sets. For each set we compute the best solution, and then take the best between the two sets.

[case I: $size(g) + size(h) \geq M - u_\ell^j$]

The problem reduces to

$$\min_{g \in F_\ell(i-1), h \in B_\ell(n-i)} \left\{ cost(g) + cost(h) + p_\ell^j u_\ell^j \right\} \quad (6)$$

We define a pair (g, h) to be *feasible* if $size(g) + size(h) \geq M - u_\ell^j$. Now to compute Eq. 6, we do a forward and backward walk on $F_\ell(i - 1)$ and $B_\ell(n - i)$ respectively. We start from the smallest index of $F_\ell(i - 1)$ and move right, and from the highest index of $B_\ell(n - i)$ and move left. Let (g, h) be the current pair. If (g, h) is feasible, we decrement B 's pointer (that is, move backward) otherwise we increment F 's pointer. The feasible pairs found during the walk are used to compute Eq. 6. The complexity of this step is *linear* in size of $F_\ell(i - 1)$, which is $O(n\alpha/\varepsilon)$.

[case II: $M - u_\ell^{j+1} \leq size(g) + size(h) \leq M - u_\ell^j$]

The problem reduces to

$$\min_{g \in F_\ell(i-1), h \in B_\ell(n-i)} \left\{ cost(g) + cost(h) + p_\ell^j (M - size(g) - size(h)) \right\}$$

To compute the above equation, we transform the above problem to another problem using modified cost, which is defined as:

$$\begin{aligned} mcost(g) &= cost(g) - p_\ell^j \cdot size(g) \\ mcost(h) &= cost(h) - p_\ell^j \cdot size(h) \end{aligned}$$

The new problem is to compute

$$\min_{g \in F_\ell(i-1), h \in B_\ell(n-i)} \left\{ mcost(g) + mcost(h) + p_\ell^j M \right\} \quad (7)$$

The modified cost simplifies the problem, but unfortunately the elements in $F_\ell(i - 1)$ and $B_\ell(n - i)$ are no longer sorted with respect to $mcost$. However, the elements are still sorted in quantity and we use this property to compute Eq. 7. Call a pair (g, h) *feasible* if $M - u_\ell^{j+1} \leq size(g) + size(h) \leq M - u_\ell^j$. Define the *feasible set* of g as the elements $h \in B_\ell(n - i)$ that are feasible given g . As the elements are sorted by quantity, the feasible set of g is a contiguous subset of $B_\ell(n - i)$ and shifts left as g increases.

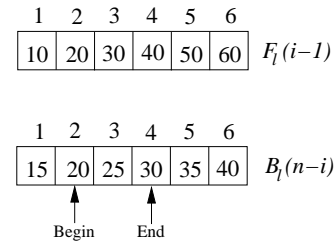


Figure 4: The feasible set of $g = 3$, defined on $B_\ell(n - i)$, is $\{2, 3, 4\}$ when $M - u_\ell^{j+1} = 50$ and $M - u_\ell^j = 60$. *Begin* and *End* represent the start and end pointers to the feasible set.

Therefore, we can compute Eq. 7 by doing a forward and backward walk on $F_\ell(i - 1)$ and $B_\ell(n - i)$ respectively. We walk on $B_\ell(n - i)$, starting from the highest index, using two pointers, *Begin* and *End*, to indicate the start and end of the current feasible set. We maintain the feasible set as a *min heap*, where the key is modified cost. To update the feasible set, when we increment F 's pointer (move forward), we walk left on B , first using *End* to remove elements from feasible set which are no longer

feasible and then using *Begin* to add new feasible elements. For a given g , the only element which we need to consider in g 's feasible set is the one with minimum modified cost which can be computed in constant time with the *min heap*. So, the main complexity of the computation lies in heap updates. Since, any element is added or deleted at most once, there are $O(\frac{n\alpha}{\epsilon})$ heap updates and the time complexity of this step is $O(\frac{n\alpha}{\epsilon} \log \frac{n\alpha}{\epsilon})$.

4.3 Collecting the Pieces

The algorithm works as follows. First, using the 2 approximation algorithm, we compute an upper bound on $C(\mathcal{I})$. We use this bound to scale down the tuple costs. Using the scaled costs, we build the forward and backward tables corresponding to each tuple (l, j) . The forward tables are used to compute $C(\mathcal{I})$. To compute $C(\mathcal{I} \setminus i)$, we iterate over all the possible midrange tuples and use the corresponding forward and backward tables to compute the locally optimal solution using the above scheme. Among all the locally optimal solutions we choose one with the minimum total cost.

The most expensive step in the algorithm is computation of $C(\mathcal{I} \setminus i)$. The time complexity of this step is $O(\frac{n^2\alpha}{\epsilon} \log \frac{n\alpha}{\epsilon})$ as we have to iterate over all $O(n)$ choices of t_l^j , for all $l \neq i$, and each time use the above scheme to compute Eq. 5. In the worst case, we might need to compute $C(\mathcal{I} \setminus i)$ for all n sellers, in which case the final complexity of the algorithm will be $O(\frac{n^3\alpha}{\epsilon} \log \frac{n\alpha}{\epsilon})$.

THEOREM 4. *We can compute an $\epsilon/(1+\epsilon)$ -strategyproof approximation to the VCG mechanism in the forward and reverse multi-unit auctions in worst-case time $O(\frac{n^3\alpha}{\epsilon} \log \frac{n\alpha}{\epsilon})$.*

It is interesting to recall that $T = O(\frac{n^3}{\epsilon})$ is the time complexity of the FPTAS to the generalized knapsack problem with all agents. Our combined scheme computes an approximation to the complete VCG mechanism, including payments to $O(n)$ agents, in time complexity $O(T \log(n/\epsilon))$, taking the no-monopoly parameter, α , as a constant. Thus, our algorithm performs much better than the naive scheme, which computes the VCG payment for each agent by solving a new instance of generalized knapsack problem. The speed up comes from the way we solve $iKnapsack^{-i}(\ell, j)$. Time complexity of computing $iKnapsack^{-i}(\ell, j)$ by creating a new dynamic programming table will be $O(\frac{n^2}{\epsilon})$ but by using the forward and backward tables, the complexity is reduced to $O(\frac{n}{\epsilon} \log \frac{n}{\epsilon})$. We can further improve the time complexity of our algorithm by computing Eq. 5 more efficiently. Currently, the algorithm uses heap, which has logarithmic update time. In worst case, we can have two heap update operations for each element, which makes the time complexity super linear. If we can compute Eq. 5 in linear time then the complexity of computing the VCG payment will be same as the complexity of solving a single generalized knapsack problem.

5. CONCLUSIONS

We presented a fully polynomial-time approximation scheme for the single-good multi-unit auction problem, using marginal decreasing piecewise constant bidding language. Our scheme is both approximately efficient and approximately strategyproof within any specified factor $\epsilon > 0$. As such it is an example of computationally tractable ϵ -dominance result, as well as an example of a non-trivial but approximable allocation problem. It

is particularly interesting that we are able to compute the payments to n agents in a VCG-based mechanism in worst-case time $O(T \log n)$, where T is the time complexity to compute the solution to a single allocation problem.

6. REFERENCES

- [1] L M Ausubel and P R Milgrom. Ascending auctions with package bidding. *Frontiers of Theoretical Economics*, 1:1–42, 2002.
- [2] S Bikchandani, S de Vries, J Schummer, and R V Vohra. Linear programming and Vickrey auctions. Technical report, Anderson Graduate School of Management, U.C.L.A., 2001.
- [3] S Bikchandani and J M Ostroy. The package assignment model. *Journal of Economic Theory*, 2002. Forthcoming.
- [4] K Chatterjee and W Samuelson. Bargaining under incomplete information. *Operations Research*, 31:835–851, 1983.
- [5] E H Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.
- [6] S de Vries and R V Vohra. Combinatorial auctions: A survey. *Inform Journal on Computing*, 2002. Forthcoming.
- [7] M Eso, S Ghosh, J R Kalagnanam, and L Ladanyi. Bid evaluation in procurement auctions with piece-wise linear supply curves. Technical report, IBM TJ Watson Research Center, 2001. in preparation.
- [8] J Feigenbaum and S Shenker. Distributed Algorithmic Mechanism Design: Recent Results and Future Directions. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pages 1–13, 2002.
- [9] M R Garey and D S Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H.Freeman and Company, New York, 1979.
- [10] G V Gens and E V Levner. Computational complexity of approximation algorithms for combinatorial problems. In *Mathematical Foundation of Computer Science*, 292-300, 1979.
- [11] T Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.
- [12] J R Kalagnanam, A J Davenport, and H S Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. *Electronic Commerce Journal*, 1(3):221–238, 2001.
- [13] V Krishna. *Auction Theory*. Academic Press, 2002.
- [14] V Krishna and M Perry. Efficient mechanism design. Technical report, Pennsylvania State University, 1998. Available at: <http://econ.la.psu.edu/~vkrishna/vcg18.ps>.
- [15] D Lehmann, L I O'Callaghan, and Y Shoham. Truth revelation in approximately efficient combinatorial auctions. *JACM*, 49(5):577–602, September 2002.
- [16] R B Myerson. Optimal auction design. *Mathematics of Operation Research*, 6:58–73, 1981.
- [17] R B Myerson and M A Satterthwaite. Efficient mechanisms for bilateral trading. *Journal of Economic Theory*, 28:265–281, 1983.
- [18] N Nisan and A Ronen. Computationally feasible VCG mechanisms. In *ACM-EC*, pages 242–252, 2000.
- [19] D C Parkes, J R Kalagnanam, and M Eso. Achieving budget-balance with Vickrey-based payment schemes in exchanges. In *IJCAI*, 2001.
- [20] M H Rothkopf, A Pekeć, and R M Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.
- [21] J Schummer. Almost dominant strategy implementation. Technical report, MEDS Department, Kellogg Graduate School of Management, 2001.
- [22] W Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.