Approximately-Strategyproof and Tractable Multi-Unit Auctions

Anshul Kothari * David C. Parkes† Subhash Suri‡

November 18, 2002

Abstract

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. We consider both a reverse auction variation (single buyer, multiple sellers), and a forward auction variation (single seller, multiple buyers). The auction problems lead to a novel and interesting generalization of the classical knapsack problem, for which we develop a fully polynomial-time approximation algorithm. A secondary computational issue addressed in our paper is the computation of the payments to agents in the Vickrey-Clarke-Groves mechanism. Naively, computing these payments for n agents requires n solutions of the original winner determination problem. Instead, we give an scheme that computes these payments in worst-case time $O(T \log n)$, where T is the time complexity to compute the solution to a single allocation problem.

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 auctions 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 (single buyer, multiple sellers), and a forward auction variation (single seller, multiple buyers). The bidding language in our auctions allows marginal-decreasing piecewise constant curves, which are both compact and expressive.

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* on the number of units she demands. The forward variation models, for example, an auction to sell excess inventory in flexible-sized lots.

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

*Computer Science, University of California, Santa Barbara, CA 93106. Email: kothari@cs.ucsb.edu
†Engineering & Applied Sciences, Harvard University, Cambridge, MA 02138. Email: parkes@eecs.harvard.edu
‡Computer Science, University of California, Santa Barbara, CA 93106. Email: suri@cs.ucsb.edu

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.

Even with marginal-decreasing bid curves, the resulting computational problem turns out to (weakly) intractable—for instance, the classical 0/1 knapsack is a special case of this problem.[1] Our auction problems with piecewise-constant bids and volume constraints can be modeled by a novel and interesting generalization of the classical knapsack problem, for which we develop a fully polynomial-time approximation scheme. We implement an approximation scheme for the Vickrey-Clarke-Groves [28, 3, 9] mechanism for the multi-unit auctions. The Vickrey-Clarke-Groves (VCG) mechanism has a number of interesting economics properties in this setting, including strategyproofness, such that truthful bidding is a dominant strategy for buyers, and *allocative efficiency*, such that the outcome maximizes the total surplus in the system. The economic properties are discussed in more detail in Section 2.

A straightforward computation of the VCG payments requires time $O(nT)$, where T is the time complexity to compute the solution to a single allocation problem. Using our approximation scheme, we show that the VCG payments to all n agents can be computed in worst-case time $O(\alpha T \log n)$, where T is the time complexity to compute the solution to a single allocation problem, and 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.

Given an approximation to within $(1 + \varepsilon)$ of the optimal allocation, our approximate VCG mechanism is $(\frac{\varepsilon}{1+\varepsilon})$-strategyproof, which means that a bidder can gain at most $(\frac{\varepsilon}{1+\varepsilon})V$ 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, but this is not an example of what Feigenbaum & Shenker refer to as a *tolerably-manipulable* mechanism [6] because we have not bounded the effect of such a manipulation on the efficiency of the outcome. Nevertheless, 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.

[1] However, if we remove all capacity constraints from the seller and all minimum-lot size constraints from the buyers, then the problem can be solved easily by a greedy scheme.

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 is both NP-complete and inapproximable [15]. Thus, the main contribution of this paper is to identify a non-trivial but approximable allocation problem, in particular one in which bidders are provided with an expressive "exclusive-or" bidding language. The bid taker is allowed to accept at most one point on the bid curve, but no more. In comparison, all non-trivial polynomial time special cases of the general combinatorial allocation require "additive-or" bidding languages in which the bid taker can accept any number of bids from each bidder [5, 24].[2]

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. Finally, Section **??** describes related work in algorithmic mechanism design, and Section 5 concludes.

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, and then introduce the VCG mechanisms for the problems and the ε-dominant results for approximations to VCG outcomes. We also discuss the economic properties of VCG mechanisms in these forward and reverse auction settings.

2.1 Marginal-Decreasing Piecewise Bid Curves

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

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

[2] One exception is the simple assignment problem, in which agents submit exclusive-or bids across individual items.

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 $(0, u_i^1)$ or $(u_i^{m_i}, \infty)$. See Figure 1. 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 $j = 1, 2, \ldots, m_i - 1$. In order to resolve the boundary condition, we assume that the bid price for the upper bound quantity $u_i^{m_i}$ is $p_{\text{bid},i}(u_i^{m_i}) = u_i^{m_i} p_i^{m_i - 1}$.

Similarly, a bid from seller i in the reverse auction, can be written as a list of (quantity-range, unit price) tuples, $((u_i^1, p_i^1), (u_i^2, p_i^2), \ldots, (u_i^{m_i-1}, p_i^{m_i-1}))$, with an upper bound $u_i^{m_i}$ on the quantity. The interpretation is that the bidder's 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. (Again, see Figure 1.) We interpret the bid list as defining a price function, $p_{\text{ask},i}(q) = qp_i^j$, if $u_i^j \leq q < u_i^{j+1}$.

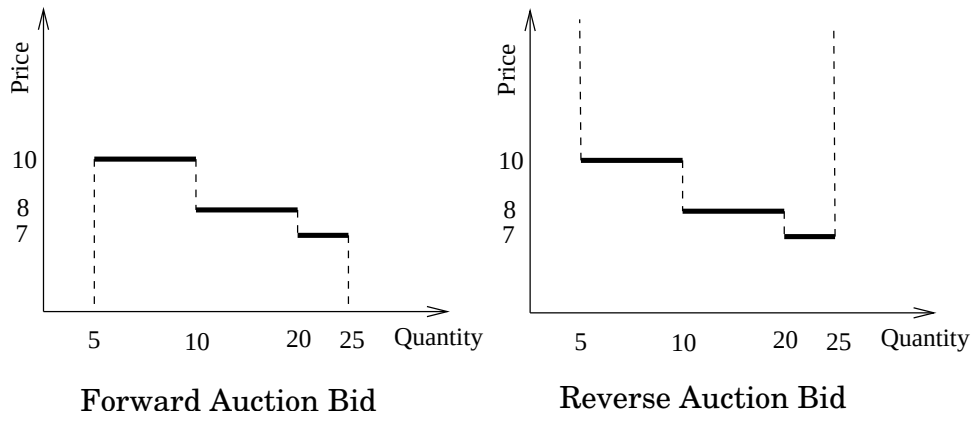


Figure 1: Marginal-decreasing, piecewise constant bids. In the forward auction bid, the bidder offers \$10 per unit for quantity in the range $[5, 10)$, \$8 per unit in the range $[10, 20)$, and \$7 in the range $[20, 25]$. Her valuation is zero for quantities outside the range $[10, 25]$. In the reverse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

2.2 Forward VCG Auction

In the forward auction, there is a seller with M units to sell. The winner-determination problem is to determine the allocation x^* maximizing the revenue to the seller. We can write this as a mixed

integer program:

$$\max_{x,y} \sum_i \sum_{j<m_i} x_i^j p_i^j \qquad \text{[FAP]}$$

$$\text{s.t.} \sum_i \sum_{j<m_i} x_i^j \leq M$$

$$x_i^j \leq L y_{ij}, \quad \forall i, \forall j < m_i \qquad (1)$$

$$\sum_{j<m_i} y_{ij} \leq 1, \quad \forall i \qquad (2)$$

$$y_{ij} u_i^j \leq x_i^j, \quad \forall i, \forall j < m_i \qquad (3)$$

$$x_i^j < u_i^{j+1}, \quad \forall i, \forall j < m_i \qquad (4)$$

$$y_{ij} \in \{0,1\}, x_i^j \text{ integer}$$

Variable x_i^j is the quantity of items sold to buyer i, where $x_i^j > 0 \Rightarrow (u_i^j \leq x_i^j < u_i^{j+1})$ by (3) and (4). By (1) and (2), the auction is constrained to choose items from at most one bid interval for each bidder. Variable $y_i^j = 1$ if and only if $x_i^j > 0$, to indicate which interval is selected. Let $V(\mathcal{I})$ denote the value of the solution to FAP, where $\mathcal{I}$ denotes the set of all bidders. Similarly, $V(\mathcal{I} \setminus i)$ denotes the solution to the same problem without agent i. We are interested in the VCG mechanism, which is defined as follows:

1. Receive bids from all the buyers.
2. Solve FAP with all bidders, compute value $V(\mathcal{I})$, and allocation, x^*.
3. For every successful bidder, i, solve FAP without that bidder, and compute value $V(\mathcal{I} \setminus i)$. Compute the VCG payment by bidder i as $p_{\text{vcg},i} = p_{\text{bid},i}(q_i^*) - [V(\mathcal{I}) - V(\mathcal{I} \setminus i)]$, where $q_i^* = \sum_j (x_i^j)^*$.

The approximate VCG mechanism, as defined by the approximation scheme presented in Section 4, implements an $(1+\varepsilon)$ approximation to the optimal solution x^*, in worst-case time $T = O(n^3/\varepsilon)$, where n is the number of bidders, and 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 cn for each occurrence of n.) In addition, if $V(\mathcal{I})/V(\mathcal{I} \setminus i) \leq \alpha$, then our approximation scheme computes an $(1+\varepsilon)$ approximation to the second-best solutions, without each buyer i in turn, in *total* time $O(\alpha T \log(\alpha n/\varepsilon))$. A constant upper bound, α, can be justified as a "no monopoly" condition, because it is a bound on the marginal value that a single buyer brings to the auction.

2.3 Reverse VCG Auction

In a 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. The winner-determination problem in the reverse auction is to determine the allocation, x^*, that minimizes the cost to the seller:

$$\min_{x,y} \sum_i \sum_{j<m_i} x_i^j p_i^j \qquad \text{[RAP]}$$
$$\text{s.t.} \sum_i \sum_{j<m_i} x_i^j \geq M$$
$$(1),(2),(3),(4)$$
$$y_i^j \in \{0,1\}, \;\; x_i^j \text{ integer}$$

If $x_i^j > 0$, then x_i^j units of the item are purchased from seller i at per-item price p_i^j. Let $C(\mathcal{I})$ denote the value of the solution to RAP, and let $C(\mathcal{I} \setminus i)$ denote the value of the solution to RAP without seller i. We are interested in the VCG mechanism. It is convenient to assume that whenever there is a surplus-increasing trade, such that $C(\mathcal{I}) < V$, then we also have $C(\mathcal{I} \setminus i) < V$ for all $i \in \mathcal{I}$, where V is the value of the buyer. As discussed below, this is a necessary, although not sufficient, condition for budget-balance. The VCG mechanism is defined as follows:

1. Receive bids from all the sellers.
2. Solve RAP with all sellers, compute $C(\mathcal{I})$ and allocation x^*.
3. For every successful seller, i, solve RAP without that seller, and compute cost $C(\mathcal{I}\setminus i)$. Compute the VCG payment to seller i as $p_{\text{vcg},i} = p_{\text{ask},i}(q_i^*) + [C(\mathcal{I} \setminus i) - C(\mathcal{I})]$, where $q_i^* = \sum_j (x_i^j)^*$.

Just like in the forward auction, we can compute an $(1+\varepsilon)$ approximation of the optimal solution x^*, in worst-case time $T = O(n^3/\varepsilon)$, for n bids, each bid with a constant number of pieces. The approximate-VCG payments to all the buyers can be determined 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.

2.4 Economic Analysis

In this section, we discuss the economic properties of the VCG mechanisms, and prove the fact that our approximate VCG mechanism is ε-strategyproof. 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.

Let us first discuss the exact VCG mechanisms. We assume in the reverse auction case that the value, V, of the buyer, is known to the mechanism.

Forward auction. The VCG auction is strategyproof and efficient. Moreover, it is unique amongst the efficient auctions, in that it maximizes the expected revenue to the seller [14].[3] In addition, the payments made by the buyers in the forward VCG auction are always non-negative, and the seller receives an expected positive payoff from participation in the auction.

Reverse auction. The VCG auction is strategyproof for sellers; it is efficient; and it is unique amongst the efficient auctions, in that it minimizes the expected payment by the buyer. However, as discussed below, the reverse auction setting there is often a budget-balance problem, which limits the applicability of the mechanism.

The total payments collected by the sellers can easily exceed the value of the buyer in the reverse VCG auction. The following condition is required for budget-balance in the reverse VCG auction:

$$V - C(\mathcal{I}) \;\geq\; \sum_i \left(C(\mathcal{I} \setminus i) - C(\mathcal{I})\right)$$

In words, the surplus of the efficient allocation must be greater than the total marginal product of each of the sellers. Considering an example with 3 agents $\{1,2,3\}$, the stated condition holds for $V = 100$, $C(123) = 50$, $C(12) = 80$, $C(23) = 80$, $C(13) = 100$, but not for $V = 100$, $C(123) = 50$, $C(12) = 70$, $C(23) = 70$, $C(13) = 100$. The problem occurs because the valuation function of the buyer is *marginal-increasing*: the buyer has zero value for less than M items, and then value V. The problem would not occur if the buyer's valuation were linear, however, in many settings of interest, a linear valuation function is too restrictive. [4]

The payment minimizing property of the VCG mechanism implies a general impossibility result whenever there is a budget balance problem in the VCG: in such a case there can be *no* efficient and balanced mechanism. Moreover, we are aware of no mechanism in the economic literature that is payoff-maximizing for the buyer in settings in which budget-balance is a problem and the buyer needs a fixed quantity, M, of units of an item.[5]

[3] Although not discussed here, a related result shows that with the possibility of resale amnesty buyers, the VCG with reserve prices for the seller also maximizes the revenue across *all* mechanisms [1].

[4] A similar budget-balance problem occurs in the forward auction, at least when the seller has a marginal-decreasing cost function, but not when the seller has a linear cost function subject to a capacity constraint, as in our model.

[5] The analysis of Ausubel & Cramton [2] suggests that VCG mechanisms with reserve prices are payoff maximizing

2.5 ε-Strategyproofness

We now state an ε-strategyproofness property for a VCG mechanism in which there is a PTAS for the allocation problem. It is convenient to abstract away from our problem, and describe the result in quite general terms. Let $\mathcal{K}$ denote a finite set of choices, and let $\theta_i \in \Theta_i$ denote the *type* of agent i, such that agent i has value $v_i(k, \theta_i) \geq 0$ for choice k. In the standard VCG mechanism, the choice k is computed by a social choice function, $f : \Theta \rightarrow \mathcal{K}$, where Θ is the joint space of agent types. In other words, the function $f(\theta)$ solves

$$\max_{k \in \mathcal{K}} \sum_i v_i(k, \theta_i)$$

Let $V(\theta)$ denote the value of this solution. In a VCG mechanism, with worst-case approximation error $(1+\varepsilon)$, the choice k is computed by a social choice function, $\hat{f} : \Theta \rightarrow \mathcal{K}$ with value $\hat{V}(\theta)$, such that

$$(1+\varepsilon)\hat{V}(\theta) \;\geq\; V(\theta), \quad \forall \theta$$

Let $\hat{f}(\theta) = \hat{k}$, and let $\hat{k}_{-i}$ denote the choice selected by $\hat{f}$ without agent i. The utility to agent i in the approximate VCG mechanism is

$$v_i(\hat{k}, \theta_i) + \sum_{j \neq i} v_j(\hat{k}, \theta_j) - \sum_{j \neq i} v_j(\hat{k}_{-i}, \theta_j) \tag{5}$$

The final term is independent of agent i's reported type, but agent i can increase its utility by announcing a reported type, $\hat{\theta}_i$, that corrects for the approximation $\hat{f}$, and sets

$$\hat{f}(\hat{\theta}_i, \theta_{-i}) = f(\theta_i, \theta_{-i})$$

The maximal gain occurs when the initial approximation, $\hat{k}$, has value $V(\theta)/(1+\varepsilon)$, and the maximal gain from manipulation to agent i is

$$V(\theta) - \frac{V(\theta)}{1+\varepsilon} \;=\; \frac{\varepsilon}{1+\varepsilon} V(\theta)$$

Thus, given a $(1+\varepsilon)$ approximation scheme to the allocation problem in the auction schemes, no agent can manipulate her gain by more than $\varepsilon/(1+\varepsilon)$ factor. Thus, we have our ε-strategyproofness result.

Notice that we did not need to use the approximation results for the "second-best allocation problems". The strategyproofness results are derived from the basic Groves mechanism, and do not

when there is a perfect resale market, but it is not clear how to integrate reserve prices into our setting in which the buyer requires a fixed quantity, M, of units.

rely on the particular form of the final term in (5). However, the approximation to the second-best allocation problems is important to make claims about the expected revenue properties of the mechanism given that agents follow truthful strategies.[6]

3 The Generalized Knapsack Problem

In this section, we formulate our auction problems as a *generalized knapsack* problem, and design a fully polynomial approximation scheme for solving the generalized knapsack. Since our formulations for the forward and reverse auctions are completely symmetric, we will describe all our results in the framework of a *reverse auction*—a buyer who wants to procure M units of some good at minimum cost from a set of n sellers.

Before we begin, let us recall the classical 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 item with *minimum value* (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 ith list has the form

$$B_i \;=\; \langle (u_i^1, p_i^1), (u_i^2, p_i^2), \ldots, (u_i^{m_i-1}, p_i^{m_i-1}), (u_i^{m_i}(i), \infty) \rangle,$$

where $u_i^1 < u_i^2 < \cdots < u_i^{m_i}$, $p_i^1 > p_i^2 > \cdots > p_i^{m_i-1}$, 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.

[6] We also suggest that a practical implication of the approximate VCG mechanisms would, for each agent i, adjust the estimate of $V(\theta)$ to be the *maximum* of $\hat{V}(\theta)$ and $\hat{V}(\theta_{-i})$, to ensure individual-rationality without affecting ε-strategyproofness.

Generalized knapsack is a clear generalization of the classical 0/1 knapsack—in the latter, each list consists of a single *point* (s_i, v_i). In fact, because of the "one per list" constraint, the generalized problem is closer in spirit to the *multiple choice knapsack* problem [7], where the underling set of items is partitioned into disjoint subsets $U_1, U_2, \ldots, U_k$, and one can choose at most one item from each subset. 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. In fact, one of the major appeals of our piecewise bidding language is its *compact representation* of the bidder's valuation functions, and we want to preserve that. In our approximation schemes, the problem size will depend only on the number of bidders, not the maximum quantity, which can be unboundedly large, especially in procurement settings.

The connection between the generalized knapsack and our auction problems is transparent: each list encodes a bid, representing multiple *mutually exclusive* quantity intervals; 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.

In a given interval, we are free to choose any quantity, and so this problem has some flavor of the *continuous* knapsack. But there are two major differences that make the 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 classical knapsack, we cannot sort the items (bids) by value/size, since different intervals in one list have different unit costs.

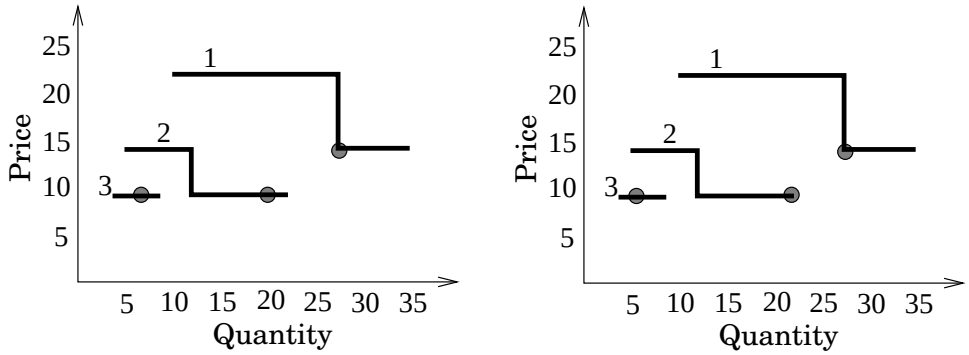
4 Generalized Knapsack Polynomial Approximation Scheme

We begin with a simple property of an optimal knapsack solution; then use this property to develop an $O(n^2)$ time 2-approximation for the generalized knapsack; and then finally use this basic approximation to develop our fully polynomial 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, 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, $\text{cost}(t_i^j) = p_i^j u_i^j$ if $j < m_i$, and $\text{cost}(t_i^{m_i}) = p_i^{m_i-1} u_i^{m_i}$.

In a feasible solution $\{x_1, x_2, \ldots, x_n\}$ of the generalized knapsack, we say that an element $x_i \neq 0$ is an anchor if $x_i = u_k^j$, for some anchor u_k^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. This follows simply because if there are two midrange elements x and x', then we can increment one (with the lower price) and decrement the other (with the higher price) until one of them becomes an anchor. We state this observation for future reference. 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.*



(i) Optimal solution with 2 midrange bids

(ii) Optimal soltution with 1 midrange bid

Figure 2: The left figure shows an optimal solution, where two of the three elements are midrange. The right figure shows an alternative optimal solution, where only one element (corresponding to bidder 3) is midrange; the other two are anchors.

4.1 A 2-Approximation Scheme

We use the anchor property to first obtain a polynomial-time 2-approximation scheme. We do this by solving several instances of a restricted knapsack, which we call *iKnapsack*, where one element is forced to be midrange. Specifically, suppose element x_ℓ is forced to lie in its jth range, $[u_\ell^j, u_\ell^{j+1})$, while all other (non-zero) elements are required to be anchors. We create $n-1$ groups $U_1, \ldots, U_{\ell-1}, U_{\ell+1}, \ldots, U_n$, where group U_i contains all the anchors of the list i in the generalized knapsack. The group U_ℓ is *special*; it contains the *interval* $[0, u_\ell^{j+1} - u_\ell^j)$, and the associated price p_ℓ^j; (Note that since element x_l is midrange it has to contribute at least u_ℓ^j units, the interval represents the excess that can be taken from it.) In any other group U_i, we can choose at most one anchor.

The goal is to obtain at least $M - u_\ell^j$ units with minimum cost. (Note that element x_ℓ has already contributed u_ℓ^j units) The following pseudo-code describes our algorithm for this special version of the generalized knapsack problem.

Algorithm *iKnapsack* (ℓ, j)

1. Let U be the union of all the tuples in U_i's.
2. Sort all the tuples of U in the ascending order of unit price; in case of ties, sort in ascending order of unit quantities.
3. Set $mark(i) = 0$, for all lists $i = 1, 2, \ldots, n$.
   Initialize $R = M - u_\ell^j$, $S = Best = Skip = \emptyset$. ($R$ is the target quantity that remains to be acquired; S is the set of tuples accepted in the current tentative solution; $Best$ maintains the best solution seen so far; $Skip$ is the current set of rejected tuples).
4. Scan the tuples in the sorted order. Suppose the next tuple is t_i^k.
   If $mark(i) = 1$, ignore this tuple; otherwise, set $mark(i) = 1$ and do the following steps:

   - if $size(t_i^k) > R$ and $i = \ell$
     return min $\{\text{cost}(S) + Rp_i^k,\ \text{cost}(Best)\}$;
   - if $size(t_i^k) > R$ and $\text{cost}(t_i^k) \leq \text{cost}(S)$
     return min $\{\text{cost}(S) + \text{cost}(t_i^k),\ \text{cost}(Best)\}$;
   - if $size(t_i^k) \leq R$ then
     add t_i^k to S; subtract $size(t_i^k)$ from R.
   - else set $Best$ to $S \cup \{t_i^k\}$ if current $Best$ has a higher cost. Add t_i^k to $Skip$. ($*$ *The variable Skip is only used in the proof of correctness.* $*$)

In the algorithm described above, S is the set of tuples accepted in current tentative solution, and R is number of units that remain to be acquired. 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 size of t_i^j is much larger than R. We, therefore, use $Best$ to maintain the best solution found so far. 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^j. Since this tuple can be taken fractionally, we update

Best if the sum of S's cost and fractional cost of t_ℓ^j is smaller than the current best solution. The algorithm terminates whenever we find a t_i^j such that $size(t_i^k)$ is greater than R but $cost(t_i^k)$ is less than $cost(S)$.

Due to lack of space, we have moved the proof of the following lemma to the appendix.

Lemma 2 *Suppose O^* is an optimal solution of the generalized knapsack, where element $x_\ell \in [u_\ell^j, u_\ell^{j+1})$ is the midrange element; all others are anchors. If the cost returned by the algorithm iKnapsack (ℓ, j) is $V(\ell, j)$, then*

$$V(\ell, j) + \text{cost}(t_\ell^j) \;\leq\; 2O^*$$

Proof: Please see the appendix. □

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

Theorem 1 *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).*

4.2 A FPTAS for Generalized Knapsack

We now use the 2-approximation algorithm presented in the preceding section to develop a fully polynomial approximation for the generalized knapsack problem. The high level idea is fairly standard, but the details require technical care. We use the 2-approximation algorithm to get an upper bound on the value of the solution; then use scaling on the cost dimension to reduce the size of the dynamic programming (DP) table.

Once again, our dynamic program will actually solve $O(n)$ instances of the *iKnapsack* problem. Suppose the midrange element is x_ℓ, which falls in the range $[u_\ell^j, u_\ell^{j+1})$. We will construct a dynamic programming table to compute the minimum cost at which $M - u_\ell^{j+1}$ units can be obtained using the remaining $n-1$ lists in the generalized knapsack. Suppose $A[i, c]$ denotes the *maximum* number of units that can be obtained at cost at most c using only the first $i-1$ lists in the generalized knapsack. Then, the following recurrence relation describes how to construct the dynamic programming table:

$$A[i,c] = \max \left\{ \begin{array}{l} A[i-1, c] \\ \max_{1 \leq j \leq m_i} \{ A[i-1,\ c - cost(t_i^j)] + u_i^j \qquad \text{if } cost(t_i^j) \leq c \ \} \end{array} \right\}$$

The dynamic programming algorithm described above is only pseudo-polynomial, since it depends on the total cost. We convert it into a fully-polynomial approximation scheme by *scaling* the cost dimension, as follows. Suppose V is the upper bound on the total cost given by our 2-approximation algorithm, and ε is the approximation factor we are aiming for. Then, the scaled cost of a tuple t_i^j, denoted $scost(t_i^j)$, is given as

$$scost(t_i^j) \;=\; \lceil \frac{n\ cost(t_i^j)}{V\varepsilon} \rceil$$

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 εV.

Lemma 3 *Let x and y be two distinct feasible solutions of iKnapsack (ℓ, j), excluding their midrange elements. If x and y have equal scaled costs, then their unscaled costs cannot differ by more than $V\varepsilon$.*

Proof: See the appendix.

Given the dynamic programming table for *iKnapsack* (ℓ, j), we consider all the entries in the last row of this table that have a value greater than $M - u_\ell^{j+1}$, which is the minimum size that the lists other than ℓ must contribute. Among all these entries $A[n-1,\ c]$, we choose the one that *minimizes* the total cost, defined as follows:

$$cost(A[n-1,c]) \;+\; \max\ \{u_\ell^j,\ M - A[n-1,c]\} \times p_\ell^j,$$

where $cost()$ is the original, unscaled cost associated with the DP entry $A[n-1,c]$. That is, we obtain $A[n-1,c]$ units from the lists $i \neq \ell$, and the remaining units from the midrange tuple t_ℓ^j. The following theorem shows that we achieve a $(1+\varepsilon)$-approximation.

Lemma 4 *Suppose O^* is an optimal solution of the generalized knapsack; where element $x_\ell \in [u_\ell^j, u_\ell^{j+1})$ is the midrange element; all others are anchor. If O is the solution from running FP-TAS on iKnapsack (ℓ, j) then*

$$O \leq (1+2\varepsilon)O^*$$

Proof: See the appendix.

Our approximation scheme for the generalized knapsack problem will iterate the scheme described above for each choice of the midrange element, and choose the best solution from among these $O(nc)$ solutions. Also for a fixed 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 2 *We can compute an* $(1+\varepsilon)$ *approximation to the solution of a generalized knapsack problem in worst-case time* $O(n^3/\varepsilon)$.

4.3 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 time roughly $O(T \log n)$, where T is the complexity of solving the allocation problem once. This result depends on a natural "no monopoly" assumption, which says that removing one agent does not increase the allocation cost by more than a factor α; that is, $C(\mathcal{I} \setminus i)/C(\mathcal{I})$ is at most α, for any agent i. We summarize the main result in the following theorem.

Theorem 3 *There is an* $O(\frac{\alpha}{\varepsilon} n^3 \log \frac{n\alpha}{\varepsilon})$ *time fully polynomial approximation scheme for computing the VCG payments for all* n *agents in a forward or reverse auction using our marginal-decreasing, piecewise constant bidding language.*

Proof: See the appendix.

5 Related Work

The idea of using approximations within mechanisms, while retaining either full-strategyproofness or ε-dominance has received some previous attention. Lehmann et al. [15] propose a greedy and strategyproof approximation to a single-minded combinatorial auction problem. Nisan & Ronen [17] 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. Feigenbaum & Shenker have recently revisited this topic of *strategically-faithful* approximations [6]. Schummer [26] and Parkes et al [20]

have previously considered ε-dominance, in the context of economic impossibility results, for example in combinatorial exchanges.

The works most similar to ours are [11, 4, 8]. The focus and contributions of these papers, however, are very different from ours. For instance, the procurement problem studied in [11] is similar to ours, but for a different volume discount model. Moreover, this paper contains no *algorithmic* results: it simply formulates the problem as a general mixed integer linear program, and give some empirical results on synthetic data. Kalagnanam & Davenport [4] addresses 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 cost minimization for the buyer (reverse auctions) or revenue maximization for the seller (forward auctions). The work [8] addresses double auctions with a more general discount model than us but uses *heuristics* to solve the optimization problem. The heuristics are analyzed using experiments without any theoretical bounds.

6 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 $\varepsilon > 0$. 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 in which bidders can use the expressive "exclusive-or" bidding language. A secondary computational issue addressed in our paper is the computation of the payments to agents in the Vickrey-Clarke-Groves mechanism. Naively, computing these payments for n agents requires n solutions of the original winner determination problem. Instead, we give an scheme that computes these payments in worst-case time $O(T \log n)$, where T is the time complexity to compute the solution to a single allocation problem.

The techniques developed in our paper can be extended to also incorporate another constraint that is commonly used in markets: bounding the number of different winning bidders. That is, in a reverse auction, a buyer can put a minimum and a maximum bound on the number of winning sellers.

References

[1] L. M. Ausubel and P. Cramton. The optimality of being efficient. Technical report, University of Maryland, 1999.

[2] L. M. Ausubel and P. Cramton. Vickrey auctions with reserve pricing. Technical report, University of Maryland, 1999.

[3] E. H. Clarke. Multipart pricing of public goods. *Public Choice*, 11:17–33, 1971.

[4] A. Davenport, J. Kalagnanam and H.S. Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. In *Electronic Commerce Journal*, volume 1(3), July 2001.

[5] S. de Vries and R. V. Vohra. Combinatorial auctions: A survey. *Informs Journal on Computing*, 2002. forthcoming.

[6] J. Feigenbaum and S. Shenker. Distributed Algorithmic Mechanism Design: ecent 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.

[7] M. R. Garey and D. S. Johnson *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, 1979.

[8] V. Gottemukkala, A. Dailianas, J. Sairamesh and A. Jhingran. Profit-driven matching in e-marketplaces: Trading composable commodities. In *Agent Mediated Electronic Commerce (IJCAI Workshop)*, pages 153 – 179, 1999.

[9] T. Groves. Incentives in teams. *Econometrica*, 41:617–631, 1973.

[10] J. Hstad. Clique is hard to approximate within $n^{1-\epsilon}$. In *Acta Mathematica*, page 1999, 105 – 142.

[11] J. Kalagnanam, M. Eso, S. Ghosh and L. Ladanyi. Bid evaluation in procurement auctions with piece-wise linear supply curves. Technical Report RC 22219, IBM Research, 2001.

[12] R. M. Karp. Reducibility among combinatorial problems. In R. E. MIller and J. W. Thatcher, editors, *Complexity of Computer Computations*, volume 43, pages 85 – 103. Plenum Press, 1972.

[13] F. Kelly and R. Steinberg. A combinatorial auction with multiple winners for universal services. In *Management Science*, volume 46, pages 586 – 596, 2000.

[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. O'Callaghan and Y. Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. In *Proc. 1st ACM Conf. on Electronic Commerce (EC-99)*, pages 96–102, 1999.

[16] S. Martello and P. Toth. *Knapsack Problems - Algorithms and Computer Implementations*. John Wiley and Sons, New York, 1991.

[17] N. Nisan and A. Ronen. Computationally feasible VCG mechanisms. In *Proc. 2nd ACM Conf. on Electronic Commerce (EC-00)*, pages 242–252, 2000.

[18] N. Nisan and A. Ronen. Algorithmic mechanism design. In *Proc. of the 31st ACM Symp. on Theory of Computing*, pages 129–140, 1999.

[19] O. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the Association for Computing Machinery*, 22:463 – 468, 1975.

[20] D. C. Parkes, J. R. Kalagnanam and M. Eso. Vickrey-based surplus distribution in combinatorial exchanges. In *Proc. 17th International Joint Conference on Artificial Intelligence (IJCAI-01)*, 2001.

[21] D. C. Parkes and L. H. Ungar. Iterative combinatorial auctions: Theory and practice. In *Proc. 17th National Conference on Artificial Intelligence (AAAI-00)*, pages 74–81, July 2000.

[22] U. Pferschy, A. Caprara, H. Kellerer and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333 – 345, 2000.

[23] R. Rivest, T. Cormen, C. Leiserson. *Introduction to Algorithms*. MIT Press, Cambridge, 1990.

[24] M. H. Rothkopf, A. Pekeč and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44(8):1131–1147, 1998.

[25] S. Sahni, E. Horowitz and S. Rajasekaran. *Computer Algorithms.* W.H.Freeman Press, 1998.

[26] J. Schummer. Almost dominant strategy implementation. Technical report, MEDS Department, Kellogg Graduate School of Management, 2001.

[27] V. L. Smith, S. J. Rassenti and R. L. Bulfin. A combinatorial auction mechanism for airport time slot allocation. *Bell J. Econ.*, 13:402–417, 1982.

[28] W. Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *Journal of Finance*, 16:8–37, 1961.

[29] M. P. Wellman, W. E. Walsh and F. Ygge. Combinatorial auctions for supply chain formation. In *Proc. of the 2nd ACM Conference on Electronic Commerce (EC 00)*, pages 260 – 269, 2000.

A Proof of Lemma 2

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

$$V(\ell, j) \;\leq\; \text{cost}(Best) \;\leq\; cost(S_t) + cost(t).$$

This implies that $V(\ell, j) < 2\text{cost}(t)$, because $t \in Skip$ and $\text{cost}(t) \geq \text{cost}(S_t)$. On the other hand, since t is included in O, we have $\text{cost}(O) \geq \text{cost}(t)$. These two inequalities imply the desired bound:

$$\text{cost}(O) \;\leq\; V(\ell, j) \;<\; 2\text{cost}(O).$$

In the second case, imagine a modified instance of *iKnapsack*, which *excludes* all the tuples of the set *Skip*. Since none of these tuples were included in O, 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 on the modified instance, and let $S_{t'}$ be the corresponding tentative solution set. 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 $\text{cost}(S_{t'}) < \text{cost}(O)$.

Since t' was the last tuple considered, by our early termination condition, we have $\text{cost}(t') \leq cost(S_{t'})$. Therefore, we have the following inequalities:

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

The preceding argument has shown that the value returned by our approximation algorithm is within a factor 2 of the optimal for *iKnapsack* (ℓ, j). We now show that the value $V(\ell, j)$ plus $\text{cost}(t_\ell^j)$ is a 2-approximation of the original generalized knapsack problem. Let x be an optimal solution of the generalized knapsack, in which element x_ℓ^j is midrange. Let $x_{-\ell}$ to be set of the remaining elements (i.e. anchors) in this solution. Furthermore, define $x'_\ell = x_\ell^j - u_\ell^j$. Thus,

$$\text{cost}(x) = \text{cost}(x'_l) + \text{cost}(t_l^j) + \text{cost}(x_{-l})$$

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

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

This completes the proof of Lemma 2. □

B Proof of Lemma 3

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,

$$\sum_{i\neq\ell}\sum_{k} scost(t_i^k) I_x[i,k] \;=\; \sum_{i\neq\ell}\sum_{k} scost(t_i^k) I_y[i,k] \tag{6}$$

However, the scaled costs satisfy the following inequalities:

$$\frac{(scost(t_i^k)-1)\varepsilon V}{n} \;\leq\; cost(t_i^k) \;\leq\; \frac{scost(t_i^k)\varepsilon V}{n} \tag{7}$$

These two sets of inequalities imply that

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

where the last inequalities 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. □

C Proof of Lemma 4

Suppose the optimal solution has cost O^*; assume that the only midrange element in this solution is $x_\ell^j \in [u_\ell^j,\; u_\ell^{j+1})$. Let $x_{-\ell}$ denote the vector of the remaining (anchor) elements in this solution. Then, by definition, $O^* \;=\; cost(x_{-\ell}) + p_\ell^j x_\ell^j$.

Let $c = scost(x_{-\ell})$ be the scaled cost associated with the vector $x_{-\ell}$. Now consider the dynamic programming table constructed for the *iKnapsack* instance (ℓ, j), and consider its entry $A[n-1, c]$. 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 εV of each other; that is, $cost(y_{-\ell}) - cost(x_{-\ell}) \;\leq\; \varepsilon V$.

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 O, then we must have

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

where the last inequality uses the fact that $V \leq 2O^*$. □

D A FPTAS for VCG Payments

Our overall strategy will be to build the dynamic programming tables once, and then determine the VCG payments by combining answers from these tables. By Lemma 1, we can limit ourselves to those allocations in which at most one agent is cleared in midrange. We will fix one particular tuple, say, t_ℓ^j, and compute all VCG payments *under the constraint* that only the element x_ℓ^j is midrange. As in our approximation scheme for the generalized knapsack, we will iterate over all $O(n)$ choices of t_ℓ^j. So, in the following, let us concentrate on computing the allocation values with x_ℓ^j being midrange and each agent removed in turn.

We begin with a simple scheme, which is computationally *inefficient* but it helps illustrate the main idea of our final scheme.

D.1 A Simple Approximation Scheme

We begin by solving two instances of RAP($\mathcal{I}$), one with sellers ordered $1, 2, \ldots, \ell-1, \ell+1, \ldots, n$, and the other with sellers ordered $n, n-1, \ldots, \ell+1, \ell-1, \ldots, 1$. (The agent ℓ is ignored while building these tables because she is already forced to be cleared in midrange.) We use our fully polynomial approximation scheme of Section 4.2 and build a dynamic programming table for each of these instances. We call the first table, with sellers listed in order $1, 2, \ldots, n$, the *forward* table, and denote it F_ℓ; the other table is called the *backward* table, and denoted B_ℓ; the subscript ℓ reminds us that the agent ℓ is midrange; To be technically precise, we should index the tables with both ℓ and j, since the jth tuple of the list ℓ is forced to be midrange. But to avoid additional clutter in our notation, we have omitted the reference to j.

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

$$scost(t_i^j) \;=\; \lceil \frac{n\ cost(t_i^j)}{V\varepsilon} \rceil$$

where V is the upper bound given by our 2-approximation scheme. However, because $C(\mathcal{I} \setminus i)$ can be α times $C(\mathcal{I})$, the cost dimension of our dynamic program's table will be $n\alpha/\varepsilon$.

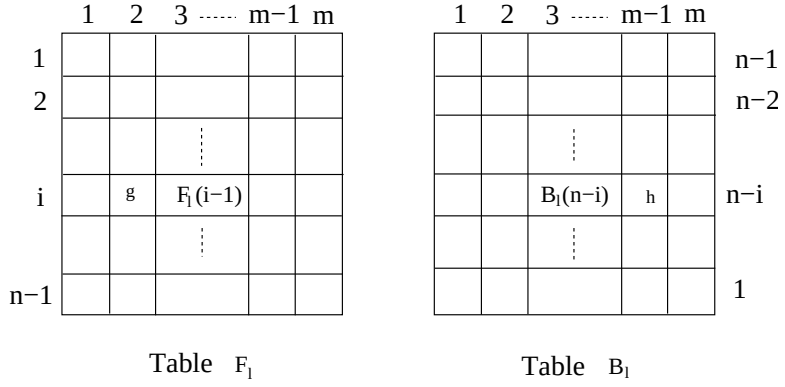


Figure 3: Computing VCG payments.

Recall that the ith row of our DP table stores the best solution possible using only the first i agents (all of them cleared at anchors); the *first* here means according to the order of sellers.

Now, suppose we want to compute a $(1+\varepsilon)$-approximation of $C(\mathcal{I}\setminus i)$. Consider the $(i-1)$th row of F_ℓ, denoted $F_\ell(i-1)$, and the $(n-i)$th row of B_ℓ, denoted $B_\ell(n-i)$. These rows, respectively, contain the quantities that can be acquired from agents $\{1, 2, \ldots, i-1\}$, and agents $\{i+1, i+2, \ldots, n\}$; the (scaled) costs for acquiring these units are the column indices for these entries. In order to compute $C(\mathcal{I} \setminus i)$, we need to choose one entry from $F_\ell(i-1)$ and one from $B_\ell(n-i)$ such that their quantity sum exceeds $M - u_\ell^{j+1}$ and their combined cost is minimum over all such combinations; recall that M is the original target and u_ℓ^{j+1} is the maximum amount the agent ℓ can provide in the range $[u_\ell^j, u_\ell^{j+1})$. Given an entry $g \in F_\ell(i-1)$, let $size(g)$ and $cost(g)$, resp., denote the number of units and the *unscaled* cost associated with this (partial) solution. Similarly, let $size(h)$ and $cost(h)$ be the size and costs for an entry $h \in B_\ell(n-i)$. Then, we minimize the following *subject to the condition* that $size(g) + size(h) > M - u_\ell^{j+1}$:

$$\min_{g \in F_\ell(i-1),\ h \in B_\ell(n-i)} \Big\{ cost(g) + cost(h) + p_\ell^j \max\{u_\ell^j,\ M - size(g) - size(h)\} \Big\} \tag{8}$$

Lemma 5 *The expression in Eq. 8 computes the $(1+\varepsilon)$-approximation of $C(\mathcal{I} \setminus i)$, for any agent i, under the constraint that agent ℓ is cleared midrange.*

Proof: Let x be an optimal solution for $C(\mathcal{I} \setminus i)$ with x_l^j being the midrange element. We split x into three disjoint parts: x_l corresponds to the midrange seller, x_i corresponds to first $i-1$ sellers and x_{-i} corresponds to last $n-i$ sellers.

$$C(\mathcal{I} \setminus i) = \text{cost}(x_i) + \text{cost}(x_{-i}) + p_\ell^j x_\ell^j \tag{9}$$

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

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

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

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

where we use shorthand $size(x)$ to denote total number of units in all tuples in x. Now, define $y_l^j = max(u_l^j, M - size(y_i) - size(y_{-i}))$. From the preceding inequalities, we have $y_l^j \leq x_l^j$. Since (y_l^j, y_i, y_{-i}) is also a feasible solution, the value returned by equation 8 is at most

$$\begin{aligned} cost(y_i) + cost(y_{-i}) + p_l^j y_l^j &\leq C(\mathcal{I} \setminus i) + V\varepsilon \\ &\leq C(\mathcal{I} \setminus i) + 2C(\mathcal{I})\varepsilon \\ &\leq C(\mathcal{I} \setminus i) + 2C(\mathcal{I} \setminus i)\varepsilon \end{aligned}$$

This completes the proof. □.

A naive implementation of this scheme will be inefficient because it might check $(n\alpha/\varepsilon)^2$ pairs for each $C(\mathcal{I} \setminus i)$. In the next section, we present an efficient way of computing Eq. 8.

D.1.1 Improved Approximation Scheme

We use the key property that elements in $F_\ell(i-1)$ and $B_\ell(n-i)$ are sorted; specifically, both, unscaled cost(*cost*) and quantity(*size*), increases from left to right. Recall our goal is to compute

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

For rest of the discussion, we use g as a generic entry in $F_\ell(i-1)$ and h as a generic entry in $B_\ell(n-i)$. To efficiently compute Eq. 8, we break the problem into two parts: one where $u_l^j \geq M - size(g) - size(h)$ and one where it is not.

In the first case, assuming $size(g) + size(h) \geq M - u_l^j$, the problem reduces to

$$\min_{g \in F_\ell(i-1),\ h \in B_\ell(n-i)} \Big\{ cost(g) + cost(h) + p_l^j u_\ell^j \Big\} \tag{10}$$

We can compute it by doing a forward and backward walk on $F_\ell(i-1)$ and $B_\ell(n-i)$ respectively. Call a pair (g,h) *feasible* if $size(g)+size(h) \geq M-u_l^j$. Let (g,h) be the current pair being considered. If (g,h) is feasible, we decrement B's pointer(that is, move backward) otherwise we increment F's pointer. We compute Eq. 10 using the feasible pairs found during the walk. The complexity of this step is linear in size of $F_\ell(i-1)$, which is $O(n\alpha/\varepsilon)$.

In the second case, assuming $M - u_l^{j+1} \leq size(g) + size(h) \leq M - u_l^j$, the problem reduces to

$$\min_{g \in F_\ell(i-1),\ h \in B_\ell(n-i)} \Big\{ cost(g) + cost(h) + p_l^j (M - size(g) - size(h)) \Big\} \tag{11}$$

We consider a new problem with modified cost which is defined as

$$mcost(g) = cost(g) - p_l^j size(g) \ \ \& \ \ mcost(h) = cost(h) - p_l^j size(h)$$

The new problem will compute

$$\min_{g \in F_\ell(i-1),\ h \in B_\ell(n-i)} \Big\{ mcost(g) + mcost(h) + p_l^j M) \Big\} \tag{12}$$

Though using modified cost simplifies the problem but unfortunately the elements are no longer sorted, with respect to *mcost*, in $F_\ell(i-1)$ and $B_\ell(n-i)$. Still the elements are sorted in quantity and we use this property to compute Eq. 12 efficiently.

Call a pair (g,h) *feasible* if $M - u_l^{j+1} \leq size(g) + size(h) \leq M - u_l^j$. Define *feasible set* of g to be all the elements in $B_\ell(n-i)$ which are feasible with it. 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. Therefore, we can compute Eq. 12 by doing a forward and backward walk on $F_\ell(i-1)$ and $B_\ell(n-i)$ respectively. We walk on $B_\ell(n-i)$ 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*

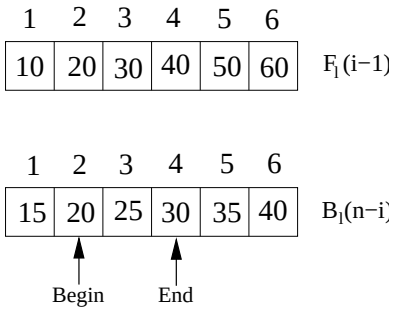


Figure 4: *Feasible Set* for $g = 3$ is $\{2, 3, 4\}$ when $M - u_l^{j+1} = 50$ and $M - u_l^j = 60$. *Begin* and *End* are used as start and end pointers of *feasible set*.

to remove elements from feasible set which are no longer feasible and then using *Begin* to add new feasible elements. To compute Eq. 12, for a given g, the only element which we need to consider in g's feasible set is one with minimum modified cost which can be computed in constant time. So, the main complexity of the computation lies in heap updates. Since, any element is added or deleted at most once, there are $O(\frac{n\alpha}{\varepsilon})$ heap updates and the time complexity of this step is $O(\frac{n\alpha}{\varepsilon} \log \frac{n\alpha}{\varepsilon})$.

For a seller, i, the time complexity to compute Eq. 8 for a fixed midrange, t_l^j, is $O(\frac{n\alpha}{\varepsilon} \log \frac{n\alpha}{\varepsilon})$. To compute $C(\mathcal{I} \setminus i)$, we will iterate over all $O(n)$ choices of t_l^j. Therefore, the time complexity of computing $C(\mathcal{I} \setminus i)$ is $O(\frac{n^2\alpha}{\varepsilon} \log \frac{n\alpha}{\varepsilon})$. In 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 $O(\frac{n^3\alpha}{\varepsilon} \log \frac{n\alpha}{\varepsilon})$.