Interval Subset Sum and Uniform-Price Auction Clearing*

Anshul Kothari[1], Subhash Suri[1], and Yunhong Zhou[2]

[1] Computer Science Depart, University of California, Santa Barbara, CA 93106
{kothari, sur}@cs.ucsb.edu
[2] HP Labs, 1501 Page Mill Rd, Palo Alto, CA 94304 yunhong.zhou@hp.com

Abstract. We study the *interval subset sum problem* (ISSP), a generalization of the classic *subset-sum* problem, where given a set of intervals, the goal is to choose a set of integers, *at most one from each interval*, whose sum best approximates a target integer T. For the *cardinality constrained* interval subset-sum problem (kISSP), at least $k_{\min}$ and at most $k_{\max}$ integers must be selected. Our main result is a fully polynomial time approximation scheme for ISSP, with time and space both $O(n \cdot 1/\varepsilon)$. For kISSP, we present a 2-approximation with time $O(n)$, and a FPTAS with time $O(n \cdot k_{\max} \cdot 1/\varepsilon)$.

Our work is motivated by auction clearing for uniform-price multi-unit auctions, which are increasingly used by security firms to allocate IPO shares, by governments to sell treasury bills, and by corporations to procure a large quantity of goods. These auctions use the *uniform price* rule—the bids are used to determine who wins, but all winning bidders receive the same price. For procurement auctions, a firm may even limit the number of winning suppliers to the range $[k_{\min},\ k_{\max}]$. We reduce the auction clearing problem to ISSP, and use approximation schemes for ISSP to solve the original problem. The cardinality constrained auction clearing problem is reduced to kISSPand solved accordingly.

1 Introduction

We introduce the *interval subset sum problem* (ISSP), where given a set of intervals $[a_i, b_i]$, for nonnegative integers a_i, b_i, $i = 1, 2, \ldots, n$, and a target integer T, the goal is to choose a set of integers, at most one from each interval, that best approximates T. ISSP is very similar to the classical *subset sum problem*, with each integer item replaced by an interval of integer values. By limiting $a_i = b_i$ for all i, ISSP degenerates into the subset sum problem. Because the subset sum problem is NP-complete [7], it is easy to show that ISSP is also NP-complete. In this paper we study the optimization version of ISSP, and present approximation schemes for it.

ISSP is motivated by uniform-price multi-unit combinatorial auctions. In many practical scenarios, additional requirements are imposed on the structure of the optimal solution. A quite frequent and natural restriction concerns the

* Kothari and Suri were partially supported by NSF grants CCR-9901958 and IIS-0121562. Most of the work was done while Kothari was an intern at HP Labs.

number of items included in an optimal solution. Therefore, we also consider the *cardinality constrained* version of ISSP, where a feasible solution should contain at most $k_{\max}$ integers and at least $k_{\min}$ integers. We call this the *cardinality constrained interval subset sum problem* (kISSP) and will also present approximation algorithms for it.

1.1 Motivations

The problem we consider in this paper is motivated by single-item multi-unit auctions, where the auctioneer wants to allocate multiple indistinguishable units among a set of bidders, optimizing certain metrics. The metric can be revenue maximization as in the case of forward auctions like IPO auctions, treasury auctions etc or cost minimization as in the case of reverse auctions like procurement auctions.

One of the most important aspects of any auction is bidding format or how does a bidder express its bid. Traditionally, single item auctions have used *point bid* format, where a bid is a pair (p, u), where p is the per unit price and u is the number of units the bidder is interested in. The point bidding language is very restrictive as the auctioneer can allocate exactly u units to the bidder or nothing at all. In practice, a bidder might be satisfied with some thing close to u. A more expressive bidding format is *interval bids*, where the bidder specifies a range $[u_1, u_2]$ such that he can be allocated any number of units in this range. This type of bidding language was implicitly used by Google for its IPO auction. In its prospectus [17] Google mentioned that a winning bid with asking quantity u will be guaranteed to receive at least 80% of that number, i.e., any quantity in the interval $[0.8u, u]$.

Another important application of interval bids is procurement auctions used by firms to procure raw materials. In these auctions, the *suppliers* wish to express *volume discounts* in their bids: instead of offering a fixed price per unit, they prefer a more expressive bidding language. The language most widely used has the form of a *piecewise constant* curve. (See Figure 1.) Such a bid can be written as a list of tuples $\langle (u_1, p_1), (u_2, p_2), \ldots, (u_{m-1}, p_{m-1}), (u_m, \infty) \rangle$, where $u_1 < u_2 < \ldots < u_m$ and $p_1 > p_2 > \ldots > p_{m-1}$. The procurement interpretation is that price per unit is p_i for any number of units in the range $[u_i, u_{i+1})$, and u_m is the maximum number of units offered by this bid. As the quantity increases, the unit price decreases (volume discount).

Procurement auctions usually have one additional side constraint. The buyer wishes to control the number of different suppliers that win in the auction: managing many different suppliers is costly, but too few suppliers expose the firm to vulnerability in case of unforeseen events. Thus, the firm puts a constraint that the number of *winning suppliers* be in the range $[k_{\min},\ k_{\max}]$, where $k_{\min}, k_{\max} > 0$ are integers.

Another important aspect of a single-item multi-unit auction is clearing price. There are two kinds of pricing rules: *uniform* and *non-uniform*. In this paper, we consider the *uniform-price* rule: all winners receive the same per unit price. The bids are used to determine who all win the auction and how many units

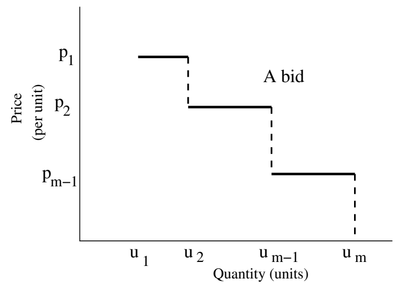

Fig. 1. A piecewise constant bid.

are allocated to them but all the winners receive a common per unit price. This pricing rule has the advantage of being *ex ante* fair, and promotes goodwill among auction participants. Auction sites like *MobShop* use such pricing rules for group buying. For the non-uniform pricing rule, where different winners are offered different prices, the auction problem translates into a generalized form of the knapsack problem, which we have studied in a companion paper [11].

1.2 Related Work

The subset-sum problem is a special case of the 0-1 Knapsack problem where item profits and item weights coincide. A large number of theoretical and practical papers have appeared on variants of knapsack problems. See Kellerer, Pferschy and Pisinger [9] for an extensive treatment of knapsack problems. The subset-sum problem is well-known to be NP-complete [7]. The classical dynamic programming (DP) approach gives a pseudo-polynomial time algorithm with running time $O(nT)$. An optimal algorithm with improved complexity is due to Pisinger [15]. The first fully polynomial time approximation scheme for the subset-sum problem was suggested by Ibarra and Kim [6]. After a series of improvement, the best currently known result is by Kellerer et al [8]. The algorithm finds an approximate solution with relative error less than ε in time $O(\min\{n \cdot 1/\varepsilon, n + 1/\varepsilon^2 \log(1/\varepsilon)\})$ and space $O(n + 1/\varepsilon)$.

Cardinality constraints arise from column generations of cutting stock problems, and were originally studied by Gilmore and Gomory [5]. For the knapsack problem with cardinality constraints, it is formally introduced by Caprara et al. [1], where an upper bound $k_{\max}$ is given on the number of packed items. Caprara et al. also study the cardinality constrained subset-sum problem, and it runs in $O(n \cdot 1/\varepsilon \cdot \ell)$ time and $O(n + \ell^2/\varepsilon)$ space, where $\ell = \min\{1/\varepsilon, k_{\max}\}$. Our work generalizes items into intervals, and considers both an upper bound ($k_{\max}$) and a lower bound ($k_{\min}$) on the number of items selected, thus it demands extra techniques to handle.

There has been a flurry of research activities for large scale procurement in recent years, inspired by the emergence of electronic commerce. In particular, *combinatorial auctions* have been proposed as expressive, economically efficient, and truthful mechanisms for resource distributions [10, 12, 14, 16]. The winner determination problem in combinatorial auctions, unfortunately, is NP-complete and inapproximable [13] in general. Consequently, there is enormous interest in finding the right level of generality at which to address the problem. The setting studied in our paper is eminently practical; for various reasons, firms tend to conduct separate auctions for different goods, thus eliminating the main source of complexity in combinatorial auctions. The other two sources (bounding the numbers of winners, and the expressiveness of bidding language) remain, and our work incorporates them both.

In procurement auctions, the work most similar to ours is [3, 2, 4]. The focus and contributions of these papers, however, are very different from ours. For instance, the procurement problem studied by Eso et al. [4] is similar to ours, but they use a different volume discount model: the supplier asks for unit price p_1 up to some quantity q_1; the discount is offered *only for the additional* units beyond q_1. We believe our model is more natural and commonly used. But the main difference is that 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.* Davenport et al. [3] addresses double auctions, where multiple buyers and sellers trade a *divisible good*, and its focus is also different: it investigates the *equilibrium* prices using the demand and supply curves, whereas our focus is on cost minimization for the buyer. Dailianas et al. [2] addresses double auctions which has a more general discount model than ours but uses *heuristics* to solve the optimization problem. These heuristics have no provable theoretical performance guarantees.

The rest of the paper is organized as follows. We define ISSP and kISSP formally and obtain a canonical solution structure for them in Section 2. We present a FPTAS for ISSP in Section 3 and a FPTAS for kISSP in Section 4. We transform the uniform-price multi-unit auction clearing problem into ISSP (or kISSP) and use FPTAS for ISSP (or kISSP) to solve the original problem in Section 5, and conclude in Section 6.

2 Definitions and Preliminaries

The decision problem of ISSP is the following: given a set of intervals $[a_i, b_i]$ where a_i, b_i are positive integers for $i = 1, \ldots, n$, and a target integer T, decide whether there exists a subset of integers, at most one from each interval, such that their total sum equals to exactly T. ISSP is NP-complete as it contains the classical subset sum problem as a special case ($a_i = b_i$ for all i), which is one of the earliest examples proved to be NP-complete. So we focus on the optimization version of ISSP, and it has two variations: maximization and minimization. The maximization problem is to choose a subset S' whose elements sum up to *at most* T while *maximizing* the sum of the subset. The minimization problem is to choose a subset S' whose elements sum up to *at least* T while *minimizing* the subset sum. Formally these optimization versions are defined as following:

Instance: A set of intervals $[a_i, b_i]$, for integers $b_i \geq a_i > 0$, $i = 1, 2, \ldots, n$, and a target integer T.

ISSP (max) : Determine a set of integers $x_1, x_2, \ldots, x_n$ with $\sum_{i=1}^{n} x_i \leq T$ such that $\sum_{i=1}^{n} x_i$ is maximized, where $x_i > 0$ implies that $x_i \in [a_i, b_i]$.

ISSP (min) : Determine a set of integers $x_1, x_2, \ldots, x_n$ with $\sum_{i=1}^{n} x_i \geq T$ such that $\sum_{i=1}^{n} x_i$ is minimized, where $x_i > 0$ implies that $x_i \in [a_i, b_i]$.

For the cardinality constrained interval subset sum problem (kISSP), there are also two optimization versions: maximization and minimization. The definitions are identical to the definitions for the optimization models of ISSP, with an extra cardinality constraint:

$$k_{\min} \leq |\{x_i \mid x_i > 0\}| \leq k_{\max},$$

where $|X|$ denotes the cardinality of set X.

Intuitively, the minimization version of ISSP (kISSP) corresponds to cost minimization for reverse auctions, while the maximization version corresponds to revenue maximization of forward auctions. The minimization version of ISSP (kISSP) is highly similar to its corresponding maximization version in both computational complexity and algorithmic details, thus we will focus only on the maximization version during this paper. From now on, by default ISSP (kISSP) denotes the maximization version of the corresponding optimization problem.

Our main results are fully polynomial time approximation schemes for both ISSP and kISSP. The algorithm for ISSP produces a solution that is within $(1+\varepsilon)$ of the optimal in $O(n \cdot 1/\varepsilon)$ time. [3] The cardinality constraints add a multiplicative factor of $k_{\max}$ to the complexity, and kISSP can be solved in $O(n \cdot k_{\max} \cdot 1/\varepsilon)$ time. Thus the time complexity of these algorithms for ISSP and kISSP *closely matches* the time complexity for the best known algorithms for their corresponding classical versions of the subset sum problem.

We begin our discussion by describing an important structural property of the optimal solution, which is used by our algorithms.

2.1 A Canonical Optimal Solution

Consider a solution of ISSP (kISSP) $x_1, x_2, \ldots, x_n$. We call a non-zero element x_i an *anchored element* if $x_i = a_i$ or $x_i = b_i$; that is, the element is the leftmost or the rightmost point of its interval. Naturally, if $x_i = a_i$, we call it a *left anchor* and if $x_i = b_i$, we call it a *right anchor*. A non-zero element x_i that is not anchored will be called *midrange*. We identify intervals with the same label as the label of the corresponding element; that is, if x_i is left-anchored, then we say that $[a_i, b_i]$ is left-anchored etc. The following fact is straightforward.

Fact 1 *There exists an optimal solution of ISSP (kISSP) with at most one midrange element.*

[3] For a maximization problem, let A be output value of the algorithm and O be the optimum value, a $(1+\varepsilon)$-approximation guarantees that $O \leq A(1+\varepsilon)$, i.e., $A \geq O/(1+\varepsilon)$. In this paper, if $A \geq (1-\varepsilon)O$, we also consider it as a $(1+\varepsilon)$-approximation. These two definitions are asymptotically equivalent.

If an optimal solution contains only anchored elements, we can pick one of the anchored elements as a proxy midrange element. Therefore, for the rest of the discussion, we assume that the optimal solution has exactly a midrange element. Our next lemma characterizes a property that will be a key to our algorithm. It states that if we consider the input intervals in some arbitrary order, then there exists an optimal solution such that the midrange element separates the right-anchored intervals from the left-anchored intervals. Figure 2 shows an example with 5 intervals, and target $T = 385$. A canonical solution is shown where $[50, 65]$ is the midrange interval, while $[30, 38]$ is left-anchored, and $[200, 300]$ is right-anchored.

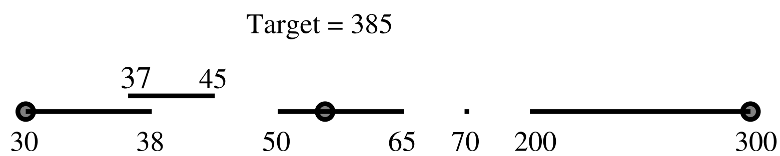

Fig. 2. A canonical optimal solution. Canonical solutions are not unique.

Lemma 1 (Canonical Solution). *Consider the intervals* $\{[a_i, b_i] \mid 1 \le i \le n\}$ *in some arbitrary order. There exists an optimal solution of ISSP (kISSP) with the following property: if* x_r *is the midrange element in the solution, then all the left-anchored intervals precede* $[a_r, b_r]$*, and all the right-anchored intervals follow* $[a_r, b_r]$*.*

3 Approximation Schemes for ISSP

In this sections we consider approximation algorithms for ISSP. We first give a relatively simple algorithm, which gives a fully polynomial time approximation scheme with running time $O(n \cdot 1/\varepsilon^2)$. Then we use more involved analysis to give another fully polynomial time approximation scheme with running time $O(n \cdot 1/\varepsilon)$.

3.1 A Simpler FPTAS

We begin with some simple observations. If $a_i \le T \le b_i$ then there is a trivial solution using just this interval. If $a_i > T$, then x_i must be 0 and we can ignore the interval $[a_i, b_i]$. Therefore, we assume that for all intervals, $b_i < T$. Furthermore, the problem is non-trivial only if $T < \sum_{i=1}^n b_i$. This follows because if $T \ge \sum_{i=1}^n b_i$, then the optimal solution is to choose $x_i = b_i$, for $i = 1, \ldots, n$. If $\sum_{i=1}^n a_i \le T \le \sum_{i=1}^n b_i$, then a solution with sum exactly T can be found easily in $O(n)$ time. Start by setting $x_i = a_i$, $\forall i$. For $i = 1, 2, \ldots, n$, if we can increase x_i from a_i to b_i without violating the T bound, we set $x_i = b_i$. The first time we reach an index j when increasing x_j to b_j makes $\sum_{i=1}^n x_i$ larger than T, we stop and set $x_j = T - \sum_{i \ne j} x_i$. Thus, we assume that $T < \sum_{i=1}^n a_i$. In summary, from now on we only consider the non-trivial case $\max_i b_i < T < \sum_i a_i$.

Next we divide items into two types: *small* items and *large* items. Small items are those such that $a_i \le \varepsilon T$, large items are all others. The crucial observation here is that a relatively large a_i makes the approximation problem difficult, and it really doesn't matter what b_i is. The next lemma guarantees that we only need to consider large items for the approximation algorithm:

Lemma 2. *If we can compute an $(1+\varepsilon)$-approximation for all the large items, we can compute an $(1+\varepsilon)$-approximation for all the items with $O(n)$ extra time.*

According to Lemma 2, only large items need to be considered. Once we obtain a good solution for large items, small items can be added to it afterwards. So from now on we assume that all items are large. We are now ready to describe a $(1+\varepsilon)$-approximation algorithm with running time $O(n/\varepsilon^2)$. For simplicity of exposition, we assume that $1/\varepsilon$ is an integer for the rest of the paper. We exploit the structure of the canonical optimal solution and describe the algorithm in three steps.

First, we scale all interval endpoints by a factor $\varepsilon^2 T$. Specifically, let $a'_i = \lfloor a_i/(\varepsilon^2 T) \rfloor$, $b'_i = \lfloor b_i/(\varepsilon^2 T) \rfloor$, for each i. Thus $a'_i \le b'_i \le 1/\varepsilon^2$. Now we only consider subset sums for the scaled down values, with the upper bound $1/\varepsilon^2$.

Second, we use dynamic programming to build two sets of lists $L_a(i)$, $L_b(i)$ for $i = 1, \ldots, n$. List $L_a(i)$ consists of all possible subset sums ($\le 1/\varepsilon^2$) for values $a'_1, \ldots, a'_i$, sorted in strictly increasing order. List $L_b(i)$ consists of all possible subset sums ($\le 1/\varepsilon^2$) for values $b'_i, \ldots, b'_n$. We build list $L_a(i)$ for $i = 1, \ldots, n$ starting from $i = 1$. List $L_a(1)$ consists of only one single element a'_1. Once we get list $L_a(i-1)$, we build $L_a(i)$ as follows: Let L_{temp} be the list by adding a'_i into each element of $L_a(i-1)$, and discarding all values exceeding $1/\varepsilon^2$. L_{temp} is also a sorted list. Let $L_a(i)$ be the list by merging $L_a(i-1)$ and L_{temp}. Since each list has length bounded by $1/\varepsilon^2$, it takes $O(1/\varepsilon^2)$ time to build $L_a(i)$ from $L_a(i-1)$. $L_b(i)$ is constructed similarly; we start with $i = n$ and build $L_b(i)$ by using $L_b(i+1)$ and b'_i. There are totally $2n$ lists, which takes $O(n \cdot 1/\varepsilon^2)$ time and space to build.

Third, for each possible midrange index $r \in \{1, \ldots, n\}$, let $V(r)$ denote

$$\max\{v_1 + x'_r + v_2 \mid v_1 \in L_a(r-1),\ v_2 \in L_b(r+1),\ x'_r \in [a'_r, b'_r],\ v_1 + x'_r + v_2 \le 1/\varepsilon^2\}.$$

According to Lemma 1, we only need to consider canonical solutions of the form given by $V(r)$, for some midrange index r. In the following we claim that $V(r)$ can be computed in linear time:

Claim. We can compute $V(r)$ in $O(1/\varepsilon^2)$ time for a fixed r.

If the claim is true, then we can compute all the $V(r)$'s for $r = 1, \ldots, n$ in time $O(n \cdot 1/\varepsilon^2)$, and the output will be $\max_{1 \le r \le n} V(r)$. In the following we show how to prove the above claim.

Let v_1 and v_2 be generic entries in the lists $L_a(r-1)$ and $L_b(r+1)$ resp. We call a pair (v_1, v_2) feasible iff $v_1 + v_2 \le 1/\varepsilon^2 - a'_r$. It is easy to see that to compute $V(r)$ we only need to consider feasible pairs.

Next, recall that both of the lists, $L_a(r-1)$ and $L_b(r+1)$, are sorted in the ascending order. Now to compute $V(r)$ we are going to do a linear walk on these two lists: starting from the head of $L_a(r-1)$ and the tail of $L_b(r+1)$. Let $v_1 \in L_a(r-1)$ and $v_2 \in L_b(r+1)$ be the current elements we are considering. If (v_1, v_2) is not a feasible pair then we know that for any $v_2' \geq v_2$, (v_1, v_2') is not going to be feasible either. Therefore, we fix v_1 and reduce v_2 by moving one step left. If (v_1, v_2) is feasible, then we do some extra processing: If $v_1 + v_2 \leq 1/\varepsilon^2 - b_r'$, there exists $x_r \in [a_r', b_r']$ such that $v_1 + v_2 + x_r = 1/\varepsilon^2$. In this case, we have already obtained the optimal solution and stop. Otherwise, we keep track of $v_1 + v_2 + b_r'$ as a candidate for $V(r)$. Now we increase v_1 by moving one step right. Once we reach the end of any of the list, we exit and choose the candidate solution with the maximum value. Given that we walk through these two lists monotonically, the total time spent to compute $V(r)$ will be bounded by a constant factor of the maximum list length, which is $O(1/\varepsilon^2)$. Now we're ready to state our first theorem:

Theorem 2. *The above algorithm computes a* $(1+\varepsilon)$*-approximation of ISSP in time and space* $O(n \cdot 1/\varepsilon^2)$.

3.2 An Improved FPTAS

In this subsection we give an algorithm with improved time and space complexity. Recall that we scale down all the intervals in the previous subsection. Scaling will cause an error for each item value, thus the total aggregated error is large afterwards. To get a better performance, we avoid scaling and use a technique called *relaxed dynamic programming* (relaxed-DP). Relaxed-DP avoids scaling and keeps the total error bounded by εT during the DP process. The basic idea of relaxed-DP is based on Kellerer et al. [8].

Partition the space $(0, T]$ into $1/\varepsilon$ intervals $B_j = ((j-1) \cdot \varepsilon T, j \cdot \varepsilon T]$ for $j = 1, \ldots, 1/\varepsilon$. The relaxed-DP will build two DP tables $L_a(i,j)$ and $L_b(i,j)$ for $i = 0, \ldots, n$ and $j = 1, \ldots, 1/\varepsilon$. Let's describe $L_a(i,j)$ in detail and $L_b(i,j)$ is built similarly.

For values $a_1, \ldots, a_i$, there may be a lot of feasible subset sums falling into the range B_j. To reduce time and space complexity, the algorithm will only keep at most two possible values and store them in $L_a(i,j)$. $L_a(i,j)$ will either be empty, or contain two of the possible subset sums v_{ij}^-, v_{ij}^+ with $v_{ij}^- \leq v_{ij}^+$. Intuitively, v_{ij}^- denotes one of the smallest subset sums in the interval B_j and v_{ij}^+ denotes one of the largest subset sums in B_j. Initially $L_a(0,j)$ is empty for all j. For the dynamic programming step from $i-1$ to i, let's consider item a_i.

Let $\tilde{\Delta}_i$ be all the possible values of $(a_i + v_{i-1,j}^-)$, $(a_i + v_{i-1,j}^+)$ in $(0, T]$ for $j = 1, \ldots, 1/\varepsilon$. The cardinality of $\tilde{\Delta}_i$ is $\leq 2/\varepsilon$. For each possible value $v \in \tilde{\Delta}_i$, find the corresponding interval B_j that it belongs to. If $L_a(i,j)$ is empty, then set $v_{ij}^- = v_{ij}^+ = v$. Otherwise, update $L_a(i,j)$ as the following:

$$v_{ij}^- := \min\{v_{ij}^-, v\}, \quad v_{ij}^+ := \max\{v_{ij}^+, v\}.$$

It is easy to check that the step from $i-1$ to i takes time $O(1/\varepsilon)$ and the whole process to build $L_a(i,j)$ for all i, j can be done in time and space $O(n \cdot 1/\varepsilon)$.

For a fixed i, consider all the values v_{ij}^{-}, $v_{i,j}^{+}$ for $j = 1, \ldots, 1/\varepsilon$. There are totally at most $2/\varepsilon$ of them and we relabel them as an ordered list $v_i^a(1) < \cdots < v_i^a(\lambda_i)$, where $\lambda_i \leq 2/\varepsilon$ is the number of distinct values in $L_a(i, .)$. For convenience, we also use $v_i^a(.)$ to denote the set containing all these λ_i elements. Let Δ_i denote all the possible subset sums for elements $a_1, \cdots, a_i$. The following lemma shows that the sequence $v_i^a(.)$ is an (εT)-approximation of Δ_i.

Lemma 3. *Let $v_i^a(0) = v_i^a(1) - \varepsilon T$, $v_i^a(\lambda_i + 1) = v_i^a(\lambda_i) + \varepsilon T$. For each possible subset sum $\delta \in \Delta_i$, there exists an index λ with $v_i^a(\lambda) \leq \delta \leq v_i^a(\lambda + 1)$ and $v_i^a(\lambda + 1) - v_i^a(\lambda) \leq \varepsilon T$.*

The proof can be found in Kellerer et al. [8]. Similarly, $L_b(i, j)$ is either empty or contains at most two feasible subset sums over $\{b_i, \ldots, b_n\}$ in the range B_j. As before, we can compute all the values $L_b(i, j)$ in time $O(n \cdot 1/\varepsilon)$. And $v_i^b(.)$ is an (εT)-approximation for the subset sums over $b_i, \ldots, b_n$. We are now ready to compute the final solution based on the canonical structure of the optimal solution. Similar as in Section 3.1, for each possible midrange index r, let $V(r)$ denote

$$\max\{v_1 + x_r + v_2 \mid v_1 \in v_{r-1}^a(.),\ x_r \in [a_r, b_r],\ v_2 \in v_{r+1}^b(.),\ v_1 + x_r + v_2 \leq T\}.$$

We claim that $V(r)$ can be computed in $O(1/\varepsilon)$ time for a fixed r. The claim can be proved identically as Claim 3.1, with the only superficial difference that here $v_{r-1}^a(.), v_{r+1}^b(.)$ have their cardinalities bounded by $2/\varepsilon$, while in Claim 3.1 $L_a(r-1), L_b(r+1)$ have their cardinalities bounded by $1/\varepsilon^2$. Because the claim is true, we can compute all the $V(r)$'s for $r = 1, \ldots, n$ in time $O(n \cdot 1/\varepsilon)$. The output of the algorithm will be $\max_{1 \leq r \leq n} V(r)$. Now we're ready to state our main result in this section:

Theorem 3. *We can compute a $(1 + \varepsilon)$-approximation of ISSP in time and space $O(n \cdot 1/\varepsilon)$.*

4 Approximation Schemes for kISSP

In the cardinality constrained interval subset-sum problem (kISSP), two additional integers $k_{\min}$ and $k_{\max}$ are specified. The goal is to reach (approximate) the target sum T using at least $k_{\min}$ and at most $k_{\max}$ (non-zero) elements. With this additional constraint, even finding a simple 2-approximation is not trivial. The following lemma describes such an algorithm.

Lemma 4. *A 2-approximation of kISSP can be computed in $O(n)$ time and space.*

The fully polynomial time approximation scheme for kISSP is similar to the improved FPTAS described in Section 3.2 for ISSP, as it also uses relaxed-DP to improve its performance. Partition the space $(0, T]$ into $1/\varepsilon$ intervals $B_j = ((j-1) \cdot \varepsilon T, j \cdot \varepsilon T]$ for $j = 1, \ldots, 1/\varepsilon$. The relaxed-DP technique will build two DP tables $L_a(i, j, k)$ and $L_b(i, j, k)$ for indexes $i = 0, \ldots, n$, $j = 1, \ldots, 1/\varepsilon$, and $k = 1, \ldots, k_{\max}$. Let's describe $L_a(i,j,k)$ in detail and $L_b(i,j,k)$ is built similarly.

Fix i,j,k as in their specified domains, $L_a(i,j,k)$ will either be empty or contains two possible subset sums over $\{a_1, \ldots, a_i\}$ in the range B_j. These subset sums should be the summation of exactly k (non-zero) values over $a_1, \ldots, a_i$. If $L_a(i,j,k) \neq \emptyset$, then it contains v^-_{ijk}, v^+_{ijk} with $v^-_{ijk} \leq v^+_{ijk}$. Intuitively, v^-_{ijk} denotes one of the smallest subset sums in the interval B_j with cardinality k, and v^+_{ijk} denotes one of the largest subset sums in B_j with cardinality k. By definition, $L_a(i,j,k)$ is empty if $i < k$ as the total number of available values is less than k. And $L_a(0,j,k)$ is empty for any j,k. For the dynamic programming step from $i-1$ to i, let's consider item a_i.

For fixed i,k, let $\tilde{\Delta}_{ik}$ be all the possible values of $(a_i + v^-_{i-1,j,k-1})$, $(a_i + v^+_{i-1,j,k-1})$ for j from 1 to $1/\varepsilon$. There are at most $2/\varepsilon$ of them. For each possible value $v \in \tilde{\Delta}_{ik}$, find the corresponding interval B_j that it belongs to. If $L_a(i,j,k)$ is empty, then set $v^-_{ijk} = v^+_{ijk} = v$. Otherwise, update $L_a(i,j,k)$ as the following:

$$v^-_{ijk} \;:=\; \min\{v^-_{ijk}, v\}, \quad v^+_{ijk} \;:=\; \max\{v^+_{ijk}, v\}.$$

It is easy to check that the step from $i-1$ to i takes time $O(k_{\max} \cdot 1/\varepsilon)$ and the whole process to build $L_a(i,j,k)$ for all i,j,k can be done in time and space $O(nk_{\max} \cdot 1/\varepsilon)$.

For fixed i,k, we consider all the values v^-_{ijk}, v^+_{ijk} for $j = 1, \ldots, 1/\varepsilon$. There are totally $\lambda_{ik} \leq 2/\varepsilon$ distinct elements and we relabel them as $v^a_{ik}(1) < \cdots < v^a_{ik}(\lambda_{ik})$. For convenience, we will also use $v^a_{ik}(.)$ to denote the set containing all these elements. Let Δ_{ik} denote all the possible subset sums (with fixed cardinality k) of $a_1, \cdots, a_i$. The following lemma shows that $v^a_{ik}(.)$ is an (εT)-approximation for Δ_{ik}:

Lemma 5. *Let $v^a_{ik}(0) = v^a_{ik}(1) - \varepsilon T$, $v^a_{ik}(\lambda_{ik}+1) = v^a_{ik}(\lambda_{ik}) + \varepsilon T$. For each possible subset sum $\delta \in \Delta_{ik}$, there exists an index j with $v^a_{ik}(j) \;\leq\; \delta \;\leq\; v^a_{ik}(j+1)$ and $v^a_{ik}(j+1) - v^a_{ik}(j) \leq \varepsilon T$.*

Similarly, we define and compute $L_b(i,j,k)$ for all i,j,k and $v^b_{ik}(.)$ is an (εT)-approximation for all subset sums over $\{b_i, \ldots, b_n\}$ with cardinality k. We are now ready to compute the final solution based on the canonical structure of the optimal solution. As in Section 3.2, we can define $V(r, k_1, k_2)$ as the maximum solution value with the form that $v_1 \in v^a_{r-1,k_1}(.)$, and $v_2 \in v^b_{r+1,k_2}(.)$. This straightforward implementation results in a multiplicative factor of $k^2_{\max}$ for the time complexity. A relatively involved technique can reduce the multiplicative factor from $k^2_{\max}$ to $k_{\max}$, thus *matches* the time complexity of the best algorithm for cardinality constrained subset sum problem. For details about how to improve the time factor from $O(k^2_{\max})$ to $O(k_{\max})$, see the proof of the following theorem, which is also our main result in this section.

Theorem 4. *We can compute a $(1+\varepsilon)$-approximation of kISSP in time and space $O(n \cdot k_{\max} \cdot 1/\varepsilon)$.*

5 Application to Uniform-Price Auction Clearing

In this section we reduce the auction clearing problem for uniform-price multi-unit auctions to ISSP (or kISSP), and use algorithms for ISSP (or kISSP) to solve the auction clearing problem.

For forward auctions, assume that there are T units of a single type of goods and the seller wants to maximize his revenue. There are n bidders, and the i-th bidder submits a bid $(p_i, [a_i, b_i])$ where p_i denotes the unit price and $[a_i, b_i]$ is the desired quantity range. The clearing algorithm works as follows: For each index i, find all the intervals $S_i = \{[a_{i'}, b_{i'}] \mid p_i \leq p_{i'}\}$. S_i makes an instance of ISSP (or kISSP) with target T, and we apply FPTAS for ISSP (or kISSP) to compute the maximum value V_i. Find index i such that $p_i V_i$ is maximized over all the indexes, and p_i will the clearing price. In summary, we have:

Corollary 1. *For uniform-price multi-unit auctions with n bidders and each bid has an interval quantity range, we can compute a $(1+\varepsilon)$-approximation in time $O(n^2 \cdot 1/\varepsilon)$. If cardinality constraints are present, a $(1+\varepsilon)$-approximation can be obtained in time $O(n^2 \cdot k_{\max} \cdot 1/\varepsilon)$.*

For procurement auctions with the piecewise constant supply curve and uniform-price model, similar results can be obtained. Observe that the uniform price offered to all the winning suppliers is going to be one of the price levels in the input functions. For each price level, we solve an instance of ISSP as follows: Consider a unit price p. Let $a_i(p)$ (resp. $b_i(p)$) denote the minimum (resp. maximum) number of units offered by i at price p or lower. These n intervals $[a_i(p), b_i(p)]$ form the input to ISSP, with target T. Since all winning suppliers are paid the common price p, the total cost becomes $pT(p)$, where $T(p)$ is the solution of the corresponding ISSP. Thus we should choose a price level p^* such that $p^* T(p^*)$ is minimized. Therefore the procurement problem can be solved by solving m instances of ISSP, where m is the total number of different price levels in the input. When the number of suppliers must be bounded in the range $[k_{\min}, k_{\max}]$, we use kISSP. In summary, we have

Corollary 2. *For procurement auctions with uniform-price rule and piecewise constant supply curves for suppliers, we can compute a $(1+\varepsilon)$-approximation in time $O(m^2 \cdot 1/\varepsilon)$. Here m is the total number of different price levels. If cardinality constraints are present, a $(1+\varepsilon)$-approximation can be obtained in time $O(m^2 \cdot k_{\max} \cdot 1/\varepsilon)$.*

6 Concluding Remarks

We gave a fully polynomial time approximation scheme for the interval subset sum problem, a generalization of the classical subset sum problem. We also presented a FPTAS to solve the cardinality constrained version of the problem. Our algorithms for both ISSP and kISSP have the same asymptotic complexity as the corresponding classical versions of the subset sum problem. The interval subset-sum problem extends the subset-sum problem to intervals, and it finds a natural application in uniform-price multi-unit auctions. With the advance of auction design mechanism and practice, we foresee that bidder's single quantity bidding format will possibly be replaced by an interval quantity format. Under such circumstance, it is likely for the interval subset sum algorithm to become a useful subroutine for future auction clearing algorithms.

Due to space limit, we didn't consider how to improve space complexity. It is possible to improve the space bounds for our algorithms using standard techniques such as balanced dynamic programming. Currently the auction clearing algorithm invokes the ISSP (or kISSP) procedure $O(n)$ times, so that its time complexity is quadratic of n. Given that these ISSP (or kISSP) instances are almost identical, is it possible to reduce the time complexity from $O(n^2)$ to sub-quadratic, such as $O(n \log n)$? We leave this as an open problem.

References

1. A. Caprara, H. Kellerer, U. Pferschy and D. Pisinger. Approximation algorithms for knapsack problems with cardinality constraints. *European Journal of Operational Research*, 123:333–345, 2000.
2. A. Dailianas, J. Sairamesh, V. Gottemukkala and A. Jhingran. Profit-driven matching in e-marketplaces: Trading composable commodities. *Agent Mediated Electronic Commerce Workshop*, 153–179, 1999.
3. A. Davenport, J. Kalagnanam and H.S. Lee. Computational aspects of clearing continuous call double auctions with assignment constraints and indivisible demand. *Electronic Commerce Research*, 1(3):221–238, 2001.
4. M. Eso et al. Bid evaluation in procurement auctions with piece-wise linear supply curves. Technical Report RC 22219, IBM Research, 2001.
5. P.C. Gilmore and R.E. Gomory. A linear programming approach to the cutting stock problem. *Operations Research*, 9:849-859, 1961.
6. O. Ibarra and C. E. Kim. Fast approximation algorithms for the knapsack and sum of subset problems. *JACM*, 22:463–468, 1975.
7. R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, 43:85–103, 1972.
8. H. Kellerer, R. Mansini, U. Pferschy and M.G. Speranza. An efficient fully polynomial approximation scheme for the Subset-Sum Problem. *Journal of Computer and System Science*, 66:349–370, 2003.
9. H. Kellerer, U. Pferschy and D. Pisinger. *Knapsack Problems*. Springer, 2004.
10. F. Kelly and R. Steinberg. A combinatorial auction with multiple winners for universal services. *Management Science*, 46:586–596, 2000.
11. A. Kothari, D. Parkes and S. Suri. Approximately-strategyproof and tractable multi-unit auctions. *Decision Support Systems*, 39(1):105-121, 2005.
12. N. Nisan and A. Ronen. Algorithmic mechanism design. *STOC*, 129–140, 1999.
13. L. I. O'Callaghan, D. Lehmann and Y. Shoham. Truth revelation in rapid, approximately efficient combinatorial auctions. *Proc. of ACM EC*, 96–102, 1999.
14. A. Pekec, M. H. Rothkopf and R. M. Harstad. Computationally manageable combinatorial auctions. *Management Science*, 44:1131–1147, 1998.
15. D. Pisinger. Linear time algorithms for knapsack problems with bounded weight. *Journal of Algorithms*, 33:1–14, 1999.
16. M. P. Wellman, W. E. Walsh and F. Ygge. Combinatorial auctions for supply chain formation. *Proc. of ACM EC*, 260–269, 2000.
17. Google IPO Prospectus. https://www.ipo.google.com/data/prospectus.html.