Fast Algorithms for Knapsack via Convolution and Prediction*†

MohammadHossein Bateni
Google Inc.
New York, NY, USA
bateni@google.com

MohammadTaghi Hajiaghayi
University of Maryland
College Park, MD, USA
hajiagha@cs.umd.edu

Saeed Seddighin
University of Maryland
College Park, MD, USA
saeedreza.seddighin@gmail.com

Cliff Stein
Columbia University
New York, NY, USA
cliff@ieor.columbia.edu

ABSTRACT

The knapsack problem is a fundamental problem in combinatorial optimization. It has been studied extensively from theoretical as well as practical perspectives as it is one of the most well-known NP-hard problems. The goal is to pack a knapsack of size t with the maximum value from a collection of n items with given sizes and values.

Recent evidence suggests that a classic $O(nt)$ dynamic programming solution for the knapsack problem might be the fastest in the worst case. In fact, solving the knapsack problem was shown to be computationally equivalent to the $(\min,+)$ convolution problem, which is thought to be facing a quadratic-time barrier. This hardness is in contrast to the more famous $(+,\cdot)$ convolution (generally known as polynomial multiplication), that has an $O(n \log n)$-time solution via Fast Fourier Transform.

Our main results are algorithms with near-linear running times (in terms of the size of the knapsack and the number of items) for the knapsack problem, if either the values or sizes of items are small integers. More specifically, if item sizes are integers bounded by $s_{\max}$, the running time of our algorithm is $\tilde{O}((n+t)s_{\max})$. If the item values are integers bounded by $v_{\max}$, our algorithm runs in time $\tilde{O}(n + tv_{\max})$. Best previously known running times were $O(nt)$, $O(n^2 s_{\max})$, $O(n^2 v_{\max})$ and $O(ns_{\max}v_{\max})$.

At the core of our algorithms lies the *prediction technique*: Roughly speaking, this new technique enables us to compute the convolution of two vectors in time $\widetilde{O}(ne_{\max})$ when the solution satisfies a *monotonic structure* and an approximation of the solution within an additive error of $e_{\max}$ is available.

Our results also improve the best known solutions for knapsack whose running times do not depend on t. In the limited size setting, when the items have multiplicities, the fastest algorithms for knapsack run in time $O(n^2 {s_{\max}}^2)$ and $O(n^3 {s_{\max}}^2)$ for the cases of infinite and given multiplicities, respectively. Our results improve both running times by a factor of $\widetilde{\Omega}(n \max\{1, n/s_{\max}\})$.

CCS CONCEPTS

• **Theory of computation → Dynamic programming**;

KEYWORDS

knapsack, convolution, prediction

ACM Reference Format:
MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. 2018. Fast Algorithms for Knapsack via Convolution and Prediction. In *Proceedings of 50th Annual ACM SIGACT Symposium on the Theory of Computing (STOC'18).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3188745.3188876

1 INTRODUCTION

The knapsack problem is a fundamental problem in combinatorial optimization. It has been studied extensively from theoretical as well as practical perspectives (*e.g.*, [3, 7, 11, 16, 17]), as it is one of the most well-known NP-hard problems [10]. The goal is to pack a knapsack of size t with the maximum value from a collection of n items with given sizes and values. More formally, item i has size s_i and value v_i, and we want to maximize $\sum_{i\in S} v_i$ such that $S \subseteq [n]$ and $\sum_{i\in S} s_i \leq t$.

Recent evidence suggests that a classic $O(nt)$ DP solution for the knapsack problem [3] may not be improved to $O((nt)^{0.999})$. In fact, solving the knapsack problem was shown to be equivalent to the $(\min,+)$ convolution problem [9], which is thought to be facing a quadratic-time barrier. The two-dimensional extension, called the $(\min,+)$ matrix product problem, appears in several conditional hardness results. These hardness results for $(\min,+)$ matrix product and equivalently $(\max,+)$ matrix product are in contrast to the more famous $(+,\cdot)$ convolution (generally known as polynomial multiplication), that has an $O(n \log n)$-time solution via Fast Fourier Transform (FFT) [8].

Before moving forward, we present the general form of convolution problems. Consider two vectors $a = (a_0, a_1, \ldots, a_{m-1})$ and $b = (b_0, b_1, \ldots, b_{n-1})$. We use the notations $|a| = m$ and $|b| = n$

*Supported in part by NSF CAREER awards CCF-1421161, CCF-1714818, and CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant
†The omitted proofs can be found in the full version of this paper

STOC'18, June 25–29, 2018, Los Angeles, CA, USA

ACM ISBN 978-1-4503-5559-9/18/06…$15.00
https://doi.org/10.1145/3188745.3188876

to denote the size of the vectors. For two associative binary operations $\oplus$ and $\otimes$, the $(\oplus,\otimes)$ convolution of a and b is a vector $c = (c_0, c_1, \ldots, c_{2n-1})$, defined as follows.

$$c_i = \bigoplus_{\substack{j:0\le j<m \\ 0\le i-j<n}} \{a_j \otimes b_{i-j}\}, \qquad \text{for } 0 \le i < m+n-1.$$

The past few years have seen increased attention towards several variants of convolution problems (e.g., [2, 4–6, 9, 14, 15]). Most importantly, many problems, such as tree sparsity and 3-sum, have been shown to have conditional lower bounds on their running times via their intimate connection with (min, +) convolution.

In particular, previous studies have shown that (max,+) convolution, knapsack, and tree sparsity are computationally (almost) equivalent [9]. However, these hardness results are obtained by constructing instances with arbitrarily high item values (in the case of knapsack) or vertex weights (in the case of tree sparsity). A fast algorithm can solve (min, +) convolution in almost linear time when the vector elements are bounded. This raises the question of whether moderate instances of knapsack or tree sparsity can be solved in subquadratic time. The recent breakthrough of Chan and Lewenstein [6] suggests that knapsack and tree sparsity may be solved in barely subquadratic time $O(n^{1.859})$ when the values or weights are small[1].

Our main results are algorithms with near-linear running times (in terms of $n + t$) for the knapsack problem, if either the values or sizes of items are small integers. More specifically, if item sizes are integers bounded by s_{max}, the running time of our algorithm is $\tilde{O}((n + t)s_{max})$. If the item values are integers bounded by v_{max}, our algorithm runs in time $\tilde{O}(n + tv_{max})$. Best previously known running times were $O(nt)$, $O(n^2 s_{max})$, $O(n^2 v_{max})$ [3] and $O(ns_{max}v_{max})$ [17]. As with prior work, we focus on two special cases of 0/1 knapsack (each item may be used at most once) and unbounded knapsack (each item can be used many times), but unlike previous work we present near linear-time exact algorithms for these problems.

Our results are similar in spirit to the work of Zwick [20] (JACM 2002) wherein the author obtains a subcubic time algorithm for the all pairs shortest paths problem (APSP) where the edge weights are small integers. Similar to knapsack and (max, +) convolution, there is a belief that APSP cannot be solved in truly subcubic time. We obtain our results through new sophisticated algorithms for improving the running time of convolution in certain settings whereas Zwick uses the known convolution techniques as black box and develops randomized algorithms to improve the running time of APSP.

We emphasize that our work does not improve the complexity of the general (min, +) convolution problem, for which no truly subquadratic-time algorithm is known to exist. Nevertheless, our techniques provide almost linear running time for the parameterized case of (min, +) convolution when the input numbers are bounded by the parameters.

A summary of the previously known algorithms along with our new results is shown in Table 1. Notice that in 0/1 knapsack, t is always bounded by $n\,s_{max}$ and thus our results improve the previously known algorithms even when t appears in the running time.

[1]The algorithm of Chan and Lewenstein [6] implies a subquadratic time solution for performing (max, +) convolution on two arrays with small distances between neighboring cells. It follows from the reduction of [9] that any subquadratic time algorithm for convolution yields a subquadratic time algorithm for knapsack.

Table 1: n and t denote the number of items and the knapsack size respectively. v_{max} and s_{max} denote the maximum value and size of the items. Notice that when the knapsack problem does not have multiplicity, t is always bounded by ns_{max} and thus our running times are always better than the previously known algorithms. Theorems 6.2, 7.3, and 7.5, as well as Corollary 6.3 are randomized and output a correct solution with probability at least $1 - n^{-10}$.

setting	running time	our improvement
general setting	$O(nt)$ [8]	-
limited size	$O(n^2 s_{max})$ [17]	$\widetilde{O}((n+t)s_{max})$ (Theorem 6.2)
limited size, unlimited multiplicity	$O(n^2 {s_{max}}^2)$ [18]	$\widetilde{O}(ns_{max} + {s_{max}}^2 \min\{n, s_{max}\})$ (Theorem 7.3) $\widetilde{O}((n+t)s_{max})$ (Corollary 6.3)
limited size, given multiplicity	$O(n^3 {s_{max}}^2)$ [18]	$\widetilde{O}(ns_{max}^2 \min\{n, s_{max}\})$ (Theorem 7.5)
limited value	$O(n^2 v_{max})$ [12, 13]	$\widetilde{O}(n + tv_{max})$ (Theorem 4.4)
limited value, unlimited multiplicity	-	$\widetilde{O}(n + tv_{max})$ (Theorem 5.5)
limited value and size	$O(ns_{max}v_{max})$ [17]	$\widetilde{O}((n+t)\min\{v_{max}, s_{max}\})$ (Theorems 4.4 and 6.2)

2 OUR CONTRIBUTION

2.1 Our Technique

Recall that the $(+,\cdot)$ convolution is indeed polynomial multiplication. In this work, we are mostly concerned with (max, +) convolution (which is computationally equivalent to minimum convolution). We may drop all qualifiers and simply call it convolution. We use the notation $a \star b$ for (max, +) convolution and $a \times b$ for polynomial multiplication of two vectors a and b. Also we denote by $a^{\star k}$ the k'th power of a in the (max, +) setting, that is $\underbrace{a \star a \star \ldots \star a}_{k \text{ times}}$.

If there is no size or value constraint, it has been shown that knapsack and (max, +) convolution are computationally equivalent with respect to subquadratic algorithms [9]. In other words, any subquadratic solution for knapsack (in terms of $n + t$) yields a subquadratic solution for (max, +) convolution and vice versa. Following this intuition, our algorithms are closely related to algorithms for computing (max, +) convolution in restricted settings. The main contribution of this work is a technique for computing the (max, +) convolution of two vectors, namely *the prediction technique*. Roughly speaking, the prediction technique enables us to compute the convolution of two vectors in time $\widetilde{O}(ne_{max})$ when an approximation of the solution within an additive error of e_{max} is given and the solution satisfies a monotonic structure. As we show in Sections 4 and 5, this method can be applied to the 0/1 knapsack and unbounded knapsack problems to solve them in $\widetilde{O}(n\,e_{max})$

time (*e.g.*, if $e_{\max} \geq v_{\max}$). In Section 3, we explain the **prediction technique** in three steps:

(1) **Reduction to polynomial multiplication:** We make use of a classic reduction to compute $a \star b$ in time $\widetilde{O}(e_{\max}(|a| + |b|))$ when all values of a and b are integers in the range $[0, e_{\max}]$. This reduction has been used in many previous works (*e.g.*, [2, 5, 6, 19, 20]). In addition to this, we show that when the values are not necessarily integral, an approximation solution with additive error 1 can be found in time $\widetilde{O}(e_{\max}(|a| + |b|))$. In the interest of space, we bring a formal proof for this reduction in the full version of the paper.

(2) **Small distortion case:** Recall that $a \star b$ denotes the (max, +) convolution of vectors a and b. In the second step, we define the "small distortion" case where $a_i + b_j \geq (a \star b)_{i+j} - e_{\max}$ for all i and j. Notice that the case where all input values are in the range $[0, e_{\max}]$ is a special case of the small distortion case. Given such a constraint, we show that $a \star b$ can be computed in time $\tilde{O}(e_{\max} n)$ using the reduction to polynomial multiplication described in the first step. We obtain this result via two observations:
   (a) If we add a constant value C to each component of either a or b, each component of their "product" $a \star b$ increases by the same amount C.
   (b) For a given constant C, adding a quantity iC to every element a_i and b_i of the vectors a and b, for all i, results in an increase of iC in $(a \star b)_i$ for every $0 \leq i < |a \star b|$ (here $|a \star b|$ denotes the size of vector $a \star b$).

   These two operations help us transform the vectors a and b such that all elements fall in the range $[0, O(e_{\max})]$. Next, we approximate the convolution of the transformed vectors via the results of the first step, and eventually compute $a \star b$ in time $\widetilde{O}(e_{\max} n)$. We give more details in Section 3.1.

(3) **Prediction**: We state the *prediction technique* in Section 3.2. Roughly speaking, when an estimate of each component of the convolution is available, with additive error $e_{\max}$, this method lets us compute the convolution in time $\tilde{O}(e_{\max} n)$. More precisely, in the prediction technique, we are given two integer vectors a and b, as well as $|a|$ intervals $[x_i, y_i]$. We are guaranteed that (1) for every $0 \leq i < |a|$ and $x_i \leq j \leq y_i$, the difference between $(a \star b)_{i+j}$ and $a_i + b_j$ is at most $e_{\max}$; (2) for every $0 \leq i < |a \star b|$ we know that for at least one j we have $a_j + b_{i-j} = (a \star b)_i$ and $x_i \leq j \leq y_i$; and (3) if $i < j$, then both $x_i \leq x_j$ and $y_i \leq y_j$ hold. We refer to the intervals as an "uncertain solution" for $a \star b$ within an error of $e_{\max}$.

The reason we call such a data structure an uncertain solution is that given such a structure, one can approximate the solution in almost linear time by iterating over the indices of the resulting vector and for every index i find one j such that $x_j \leq i - j \leq y_j$ and approximate $(a \star b)_i$ by $a_j + b_{i-j}$. Such a j can be found in time $O(\log n)$ via binary search since the boundaries of the intervals are monotone. In the prediction technique, we show that an uncertain solution within an additive error of $e_{\max}$ suffices to compute the convolution of two vectors in time $\widetilde{O}(e_{\max} n)$. We obtain this result by breaking the problem into many subproblems with the small distortion property and applying the result of the second step to compute the solution of each subproblem in time $\widetilde{O}(e_{\max} n)$. We show that all the subproblems can be solved in time $\widetilde{O}(e_{\max} n)$ in total, and based on these solutions, $a \star b$ can be computed in time $\widetilde{O}(e_{\max} n)$. We give more details in Section 3.2.

Theorem 3.5 [restated informally]. *Given two integer vectors a and b and an uncertain solution for $a \star b$ within an error of $e_{\max}$, one can compute $a \star b$ in time $\widetilde{O}(e_{\max} n)$.*

Notice that in Theorem 3.5, there is no assumption on the range of the values in the input vectors and the running time depends linearly on the accuracy of the uncertain solution.

2.2 Main Results

We show in Section 4 that the prediction technique enables us to solve the 0/1 knapsack problem in time $\widetilde{O}(v_{\max} t + n)$. To this end, we define the *knapsack convolution* as follows: given vectors a and b corresponding to the solutions of two knapsack problems k_a and k_b, the goal is to compute $a \star b$. If a vector a is the solution of a knapsack problem, a_i denotes the maximum total value of the items that can be placed in a knapsack of size i. The only difference between knapsack convolution and (max, +) convolution is that in the knapsack convolution both vectors adhere to knapsack structures, whereas in the (max, +) convolution there is no assumption on the values of the vectors. We show that if in the knapsack problems, the values of the items are integers bounded by $v_{\max}$, then an uncertain solution for $a \star b$ within an error of $v_{\max}$ can be computed in almost linear time. The key observation here is that one can approximate the solution of the knapsack problem within an additive error of $v_{\max}$ as follows: sort the items in descending order of v_i/s_i and put the items in the knapsack one by one until either we run out of items or the remaining space of the knapsack is too small for the next item. Based on this algorithm, we compute an uncertain solution for the knapsack convolution in almost linear time and via Theorem 3.5 compute $a \star b$ in time $\widetilde{O}(v_{\max} n)$. Finally, we use the recent technique of [9] to reduce the 0/1 knapsack problem to the knapsack convolution. This yields an $\widetilde{O}(v_{\max} t + n)$ time algorithm for solving the 0/1 knapsack problem when the item values are bounded by $v_{\max}$.

Theorem 4.4 [restated]. *The 0/1 knapsack problem can be solved in time $\widetilde{O}(v_{\max} t + n)$ when the item values are integer numbers in the range $[0, v_{\max}]$.*

As another application of the prediction technique, we present an algorithm that receives a vector a and an integer k as input and computes $a^{\star k}$. We show that if the values of the input vector are integers in the range $[0, e_{\max}]$, the total running time of the algorithm is $\widetilde{O}(e_{\max} |a^{\star k}|)$. This improves upon the trivial $\widetilde{O}(e_{\max}^2 |a^{\star k}|)$. Similar to what we do in Section 4, we again show that the convolution of two powers of a can be approximated within a small additive error. We use this intuition to compute an uncertain solution within an additive error of $O(e_{\max})$ and apply the prediction technique to compute the exact solution in time $\widetilde{O}(e_{\max} |a^{\star k}|)$.

Theorem 5.4 [restated]. *Let a be an integer vector with values in the range $[0, e_{\max}]$. For any integer $k \geq 1$, one can compute $a^{\star k}$ in time*

$\widetilde{O}(e_{\max}|a^{\star k}|)$.

As a consequence of Theorem 5.4, we show that the unbounded knapsack problem can be solved in time $\widetilde{O}(n + v_{\max}t)$.

Theorem 5.5 [restated]. *The unbounded knapsack problem can be solved in time $\widetilde{O}(n + v_{\max}t)$ when the item values are integers in the range $[0, v_{\max}]$.*

To complement our results, we also study the knapsack problem when the item values are unbounded real values, but the sizes are integers in the range $[1, s_{\max}]$[2]. For this case, we present a randomized algorithm that solves the problem w.h.p.[3] in time $\widetilde{O}(s_{\max}(n+t))$. The idea is to first put the items into $t/s_{\max}$ buckets uniformly at random. Next, we solve the problem for each bucket separately, up to a knapsack size $\widetilde{O}(s_{\max})$. We use the Bernstein's inequality to show that w.h.p., only a certain interval of the solution vectors are important and we can neglect the rest of the values, thereby enabling us to merge the solutions of the buckets efficiently. Based on this, we present an algorithm to merge the solutions of the buckets in time $\widetilde{O}(s_{\max}(n+t))$, yielding a randomized algorithm for solving the knapsack problem in time $\widetilde{O}(s_{\max}(n + t))$ w.h.p. when the sizes of the items are bounded by $s_{\max}$. This result is not based on the prediction technique.

Theorem 6.2 [restated]. *There exists a randomized algorithm that correctly computes the solution of the knapsack problem in time $\widetilde{O}(s_{\max}(n + t))$ w.h.p., when the item sizes are integers in the range $[1, s_{\max}]$.*

2.3 Implication to the Limited Settings

When we parameterize the 0/1 knapsack problem by $\max\{s_i\} \leq s_{\max}$, one can set $t' := \min(t, ns_{\max})$ and solve the problem with knapsack size t' in time $\widetilde{O}((t' + n)s_{\max}) = \widetilde{O}(ns_{\max}{}^2)$. This yields a solution for the knapsack problem whose running time is regardless of t. However, this only works when we are allowed to use each item only once. In Section 7, we further extend this solution to the case where each item (s_i, v_i) has a given multiplicity m_i. For this case, our algorithm runs in time $\widetilde{O}(ns_{\max}{}^2 \min\{n, s_{\max}\})$ when m_i's are arbitrary and solves the problem in time $\widetilde{O}(ns_{\max} \min\{n, s_{\max}\})$ when $m_i = \infty$ for all i. Both results improve the algorithms of [18] by a factor of $\widetilde{\Omega}(\max\{n, s_{\max}\})$ in the running time. These results are all implied by Theorem 6.2.

2.4 Further Results

It has been previously shown that tree sparsity, knapsack, and convolution problems are equivalent with respect to their computational complexity. However, these reductions do not hold for the case of small integer inputs. In the full version of the paper, we show some reductions between these problems in the small input setting. In addition to this, we introduce the tree separability problem and explain its connection to the rest of the problems in both general and small integer settings. We also present a linear time algorithm for tree separability when the degrees of the vertices and edge weights are all small integers.

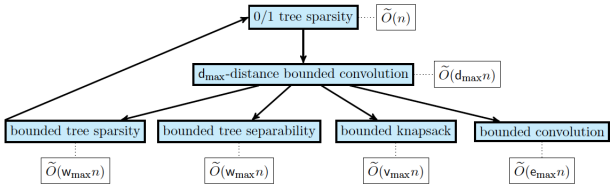

Figure 1: Desired running times are specified in the white boxes. Here $a \rightarrow b$ means that an efficient algorithm for a yields an efficient algorithm for b.

3 THE PREDICTION TECHNIQUE FOR (max, +) CONVOLUTION

In this section, we present several algorithms for computing the (max, +) convolution (computationally equivalent to (min, +) convolution) of two vectors. Recall that in this problem, two vectors a and b are given as input and the goal is to compute a vector c of length $|a| + |b| - 1$ such that

$$c_i = \max_{j=0}^{i}[a_j + b_{i-j}].$$

For this definition only, we assume that each vector a or b is padded on the right with sufficiently many $-\infty$ components: *i.e.*, $a_i = -\infty$ for $i \geq |a|$ and $b_j = -\infty$ for $j \geq |b|$.

Assuming $|a| + |b| = n$, a trivial algorithm to compute c from a and b is to iterate over all pairs of indices and compute c in time $O(n^2)$. Despite the simplicity of this solution, thus far, it has remained one of the most efficient algorithms for computing the (max, +) convolution of two vectors. However, for special cases, more efficient algorithms compute the result in subquadratic time. For instance, if the values of the vectors are integers in the range $[0, e_{\max}]$, one can compute the (max, +) convolution of two vectors in time $\widetilde{O}(e_{\max}n)$.

In this section, we present several novel techniques for multiplying vectors in the (max, +) setting in truly subquadratic time under different assumptions. The main result of this section is *the prediction technique* explained in Section 3.2. Roughly speaking, we define the notion of *uncertain solution* and show that if an uncertain solution of two integer vectors with an error of $e_{\max}$ is given, then it is possible to compute the (max, +) convolution of the vectors in time $\widetilde{O}(e_{\max}n)$. Later in Sections 4 and 5 we use this technique to improve the running time of the knapsack and other problems.

In our algorithm, we subsequently make use of a classic reduction from (max, +) convolution to polynomial multiplication. In the interest of space, we skip this part here and explain it in the full version of the paper. The same reduction has been used as a blackbox in many recent works [2, 5, 6, 19, 20]. Based on this reduction, we show that an $\widetilde{O}(e_{\max}n)$ time algorithm can compute the convolution of two integer vectors whose values are in the range $[0, e_{\max}]$. We further explain that even if the values of the vectors are real but in the range $[0, e_{\max}]$, one can approximate the solution within an additive error less than 1. These results hold even if the input values

[2]Parallel and independent to our work, Axiotis and Tzamos [1] present an $O(n+ts_{\max})$ time algorithm for this case.
[3]With probability at least $1 - n^{-10}$.

can be either in the interval $[0, e_{\max}]$ or in the set $\{-\infty, \infty\}$. We use this technique in Section 3.1 to compute the (max, +) convolution of two integer vectors in time $\widetilde{O}(e_{\max} n)$ when for every i and j we have $|a_i + b_j - (a \star b)_{i+j}| \leq e_{\max}$. Finally, in Section 3.2 we use these results to present the prediction technique for computing the (max, +) convolution of two vectors in time $\widetilde{O}(e_{\max} n)$.

3.1 An $\widetilde{O}(e_{\max} n)$ Time Algorithm for the Case of Small Distortion

In this section we study a variant of the (max, +) convolution problem where every $a_i + b_j$ differs from $(a \star b)_{i+j}$ by at most $e_{\max}$. Indeed this condition is strictly weaker than the case where the input values fall in range $[0, e_{\max}]$. Nonetheless we show that still an $\widetilde{O}(e_{\max} n)$ time algorithm can compute $a \star b$ if the values of the vectors are integers but not necessarily in the range $[0, e_{\max}]$. In the interest of space, we omit the proofs of Lemmas 3.1, 3.2, and 3.3 and include them in the full version of the paper.

We first assume that both vectors a and b have size n. Moreover, since the case of $n = 1$ is trivial, we assume w.l.o.g. that $n > 1$. In order to compute $a \star b$ for two vectors a and b, we transform them into two vectors a' and b' via two operations. In the first operation, we add a constant C to every element of a vector. In the second operation, we fix a constant C and add iC to every element i of **both** vectors. We delicately perform these operations on the vectors to make sure the resulting vectors a' and b' have small values. This enables us to approximate (and not compute since the values of a' and b' are no longer integers) the solution of $a' \star b'$ in time $\widetilde{O}(e_{\max} n)$. Finally, we show how to derive the solution of $a \star b$ from an approximation for $a' \star b'$. We begin by observing a property of the vectors.

Lemma 3.1. *Let a and b be two vectors of size n such that for all $0 \leq i, j < n$ we have $(a \star b)_{i+j} - a_i - b_j \leq e_{\max}$. Then,*

- *for every $0 \leq i, j < n$, we have $|(a_i - b_i) - (a_j - b_j)| \leq e_{\max}$; and*
- *for every $0 \leq i \leq j \leq k < n$ such that $j - i = k - j$, we have $|a_j - (a_i + a_k)/2| \leq e_{\max}$.*

Note that since there is no particular assumption on vector a, the condition of Lemma 3.1 carries over to vector b as well. Next we use Lemma 3.1 to present an $\widetilde{O}(e_{\max} n)$ time algorithm for computing $a \star b$. We obtain this result via two observations:

(1) If we add a constant value C to each component of either a or b, each component of their "product" $a \star b$ increases by the same amount C.
(2) For a given constant C, adding a quantity iC to every element a_i and b_i of the vectors a and b, for all i, results in an increase of iC in $(a \star b)_i$ for every $0 \leq i < |a \star b|$.

These two operations help us transform the vectors a and b such that all elements fall in the range $[0, O(e_{\max})]$. Next, we approximate the convolution of the transformed vectors via the aforementioned algorithm, and finally compute $a \star b$ in time $\widetilde{O}(e_{\max} n)$.

Lemma 3.2. *Let a and b be two integer vectors of size n such that for all $0 \leq i, j < n$ we have $(a \star b)_{i+j} - a_i - b_j \leq e_{\max}$. One can compute $a \star b$ in time $\widetilde{O}(e_{\max} n)$.*

An implicit corollary of Lemma 3.2 is that for any two vectors a and b that meet the conditions of Lemma 3.2, there exist values $\mathcal{A}, \mathcal{A}', \mathcal{B}, \mathcal{B}'$ such that both

$$|a_i - (\mathcal{A} i + \mathcal{A}')| \leq O(e_{\max})|$$

and

$$|b_i - (\mathcal{B} i + \mathcal{B}')| \leq O(e_{\max})|$$

hold.

All that remains is to extend our algorithm to the case where we no longer have $|a| = |b|$. We assume w.l.o.g. that $|b| \geq |a|$ and divide $|b|$ into $\lceil |b|/|a| \rceil$ vectors of length $|a|$ such each b_i appears in at least one of these vectors. Then, in time $O(e_{\max}|a|)$ we compute the (max, +) convolution of a and each of the smaller intervals, and finally use the results to compute $a \star b$ in time $\widetilde{O}(e_{\max}(|a| + |b|))$.

Lemma 3.3. *Let a and b be two integer vectors such that for all $0 \leq i < |a|$ and $0 \leq j < |b|$ we have $(a \star b)_{i+j} - a_i - b_j \leq e_{\max}$. One can compute $a \star b$ in time $\widetilde{O}(e_{\max}(|a| + |b|))$.*

3.2 Prediction

In this section, we explain the prediction technique and show how it can be used to improve the running time of classic problems when the input values are small. Roughly speaking, we show that in some cases an approximation algorithm with an additive error of $e_{\max}$ can be used to compute the exact solution of a (max, +) convolution in time $\widetilde{O}(e_{\max} n)$. In general, an additive approximation of $e_{\max}$ does not suffice to compute the (max, +) convolution in time $\widetilde{O}(e_{\max} n)$. However, we show that under some mild assumptions, an additive approximation yields a faster exact solution. We call this the prediction technique.

Suppose for two integer vectors a and b of size n, we wish to compute $a \star b$. The values of the elements of a and b range over a potentially large (say $O(n)$) interval and thus the previous algorithm does not improve the $O(n^2)$ running time of the trivial solution. However, in some cases we can predict which a_i's and b_j's are far away from $(a \star b)_{i+j}$. For instance, if a and b correspond to the solutions of two knapsack problems whose item weights are bounded by $e_{\max}$, a well-known greedy algorithm can approximate $a \star b$ within an additive error of $e_{\max}$ (a_i and b_i denote the solutions of the knapsack problem for size i). The crux of the argument is that if we sort the items with respect to the ratio of weight over size in descending order and fill the knapsack in this order until we run out of space, we always get a solution of at most $e_{\max}$ away from the optimal. Now, if $a_i + b_j$ is less than the estimated value for $(a \star b)_{i+j}$ for some i and j, then there is no way that the pair (a_i, b_j) contributes to the solution of $a \star b$. With a more involved argument, one could observe that whenever $a_i + b_j$ is at least $e_{\max}$ smaller than the estimated solution for $(a \star b)_{i+j}$, then $a_i + b_k < (a \star b)_{i+k}$ for either all k's in $[j, n-1]$ or all k's in $[0, j]$. We explain this in more details in Section 4.

This observation shows that in many cases, (a_i, b_j) pairs that are far from $(a \star b)_{i+j}$ can be trivially detected and ignored. Therefore, the main challenge is to handle the (a_i, b_j) options that are close to $(a \star b)_{i+j}$. Our prediction technique states that such instances can also be solved in subquadratic time. To this end, suppose that a and b are two integer vectors of size n, and for every $0 \leq i < |a|$ we have an interval $[x_i, y_i]$, and we are guaranteed that $a_i + b_j$ is at most

$e_{\max}$ away from $(a \star b)_{i+j}$ for all $j \in [x_i, y_i]$. Also, we know that for any $0 \le i < |a \star b|$ there exists a j such that $a_j + b_{i-j} = (a \star b)_i$ and $x_j \le i - j \le y_j$. We call such data *an uncertain solution*. We show in Theorem 3.5 that if an uncertain solution is given, then one can compute $a \star b$ in time $\widetilde{O}(e_{\max} n)$. For empty intervals only, y_i is allowed to be smaller than x_i.

Definition 3.4. For two vectors a and b, define an uncertain solution (to $a \star b$) as $|a|$ intervals $[x_i, y_i]$ such that

- $a_i + b_j \ge (a \star b)_{i+j} - e_{\max}$ for all $0 \le i < |a|$ and $j \in [x_i, y_i]$;
- for all $0 \le i < |a \star b|$, there exists an index j such that $a_j + b_{i-j} = (a \star b)_i$ and $x_j \le i - j \le y_j$; and
- $0 \le x_i, y_i < |b|$ for all intervals and $x_i \le x_j$ and $y_i \le y_j$ hold for all $0 \le i < j < |a|$.

Theorem 3.5. *Let a and b be two integer vectors for which an uncertain solution is provided. Then, one can compute $a \star b$ from a, b, and the intervals in time $\widetilde{O}(e_{\max}(|a| + |b|))$.*

Proof. One can set n equal to the smallest power of two greater than $\max\{|a|, |b|\}$ and add extra $-\infty$'s to the end of the vectors to make sure $|a| = |b| = n$. Next, for every newly added element of a we set its corresponding interval $[x_i, y_i]$ to $(n - q, n - q - 1)$ (that is, an empty interval) where q is the number of newly added $-\infty$'s to the end of b. This way, all conditions of the theorem are met and $|a| + |b|$ is multiplied by at most a constant factor. Therefore, from now on, we assume $|a| = |b| = n$ and that n is a power of two. Keep in mind that for every i with property $x_i \le y_i$, none of $\{a_i, b_{x_i}, b_{x_i+1}, \ldots, b_{y_i}\}$ is equal to $-\infty$.

Our algorithm runs in $\log n + 1$ rounds. In every round we split b into several intervals. For an interval $[\alpha, \beta]$ of b we call the projection of $[\alpha, \beta]$ the set of all indices i of a that satisfy both $x_i \le \alpha$ and $y_i \ge \beta$. We denote the projection of an interval $[\alpha, \beta]$ by $\mathcal{P}(\alpha, \beta)$. We first show that for every $0 \le \alpha \le \beta < n$, $\mathcal{P}(\alpha, \beta)$ corresponds to an interval of a. We defer the proof of Observation 3.1 to the full version of the paper.

Observation 3.1. *For every $0 \le \alpha \le \beta < n$, $\mathcal{P}(\alpha, \beta)$ is an interval of a.*

Furthermore, for any pair of disjoint intervals $[\alpha_1, \beta_1]$ and $[\alpha_2, \beta_2]$, we observe that $\mathcal{P}(\alpha_1, \beta_1) \setminus \mathcal{P}(\alpha_2, \beta_2)$ is always an interval. Similar to Observation 3.1, we include the proof of Observation 3.2 in the full version of the paper.

Observation 3.2. *For $0 \le \alpha_1 \le \beta_1 < \alpha_2 \le \beta_2 < n$, both $\mathcal{P}(\alpha_1, \beta_1) \setminus \mathcal{P}(\alpha_2, \beta_2)$ and $\mathcal{P}(\alpha_2, \beta_2) \setminus \mathcal{P}(\alpha_1, \beta_1)$ are intervals of the indices of a.*

The proof for $\mathcal{P}(\alpha_2, \beta_2) \setminus \mathcal{P}(\alpha_1, \beta_1)$ being an interval follows from symmetry.

Before we start the algorithm, we construct a vector c of size $2n-1$ and set all its indices equal to $-\infty$. In Round 1 of our algorithm, we only have a single interval $[\alpha_1, \beta_1] = [0, n-1]$ for b. Therefore, we compute $\mathcal{P}(0, n-1) = [\gamma_1, \delta_1]$ and construct a vector a^1 of size $\delta_1 - \gamma_1 + 1$ and set $a^1_i = a_{i+\gamma}$. Similarly, we construct a vector b^1 of size $\beta_1 - \alpha_1 + 1$ and set $b^1_i = b_{i+\alpha}$. Next, we compute $c^1 = a^1 \star b^1$ using Lemma 3.3, and then based on that we set $c_{i+\alpha+\gamma} \leftarrow \max\{c_{i+\alpha+\gamma}, c'_i\}$ for all $0 \le i < |c^1|$.

The second round is similar to Round 1, except that this time we split b into two intervals $[\alpha_1, \beta_1]$ and $[\alpha_2, \beta_2]$ where $\alpha_1 = 0, \beta_1 = n/2 - 1, \alpha_2 = n/2$, and $\beta_2 = n - 1$. For interval $[\alpha_1, \beta_1]$ of b we compute $[\gamma_1, \delta_1] = \mathcal{P}(\alpha_1, \beta_1) \setminus \mathcal{P}(\alpha_2, \beta_2)$ and similarly for the second interval of b we compute $[\gamma_2, \delta_2] = \mathcal{P}(\alpha_2, \beta_2) \setminus \mathcal{P}(\alpha_1, \beta_1)$. Similar to Round 1, we construct a^1, a^2, b^1, b^2 from a and b with respect to the intervals and compute $c^1 = a^1 \star b^1$ and $c^2 = a^2 \star b^2$. Finally we update the solution based on c^1 and c^2.

More precisely, in Step $s+1$ of the algorithm, we split b into 2^s intervals $[\alpha_i, \beta_i]$ where $\alpha_i = (i-1)2^{(\log n)-s}$ and $\beta_i = i2^{(\log n)-s} - 1$. For odd intervals we compute

$$[\gamma_{2i+1}, \delta_{2i+1}] = \mathcal{P}(\alpha_{2i+1}, \beta_{2i+1}) \setminus \mathcal{P}(\alpha_{2i}, \beta_{2i})$$

and for even intervals we compute

$$[\gamma_{2i}, \delta_{2i}] = \mathcal{P}(\alpha_{2i}, \beta_{2i}) \setminus \mathcal{P}(\alpha_{2i+1}, \beta_{2i+1}).$$

Next, we construct vectors $a^1, a^2, \ldots, a^{2^s}$ and $b^1, b^2, \ldots, b^{2^s}$ from a and b and compute $c^i = a^i \star b^i$ using Lemma 3.3 for every $1 \le i \le 2^s$. Finally, for every $1 \le i \le 2^s$ and $0 \le j < |c^i|$, we set $c_{\alpha_i+\gamma_i+j} = \max\{c_{\alpha_i+\gamma_i+j}, c^i_j\}$.

Algorithm 1: PredictionMethod($a, b, e_{\max}, x_i$'s, y_i's)

Data: Two integer vectors a and b of size n, intervals $[x_i, y_i]$ for $0 \le i < n$ meeting the conditions of Theorem 3.5
Result: $a \star b$

```
1  c ← a vector of size 2n − 1 with indices set to ∞ initially;
2  for s ∈ [0, log n] do
3      for i ∈ [1, 2^s] do
4          α_i ← (i − 1)2^{(log n)−s};
5          β_i ← i2^{(log n)−s} − 1;
6      for i ∈ [1, 2^s] do
7          if s = 0 then
8              [γ_i, δ_i] ← P(α_i, β_i);
9          else
10             if i is odd then
11                 [γ_i, δ_i] ← P(α_i, β_i) \ P(α_{i+1}, β_{i+1});
12             else
13                 [γ_i, δ_i] ← P(α_i, β_i) \ P(α_{i−1}, β_{i−1});
14         a^i ← a vector of size δ_i − γ_i + 1 s.t. a^i_j = a_{γ_i+j};
15         b^i ← a vector of size 2^{(log n)−s} s.t. b^i_j = b_{α_i+j};
16         c^i ← DistortedConvolution(a^i, b^i, e_max);
17         for j ∈ [1, |c^i|] do
18             c_{α_i+γ_i+j} ← max{c_{α_i+γ_i+j}, c^i_j};
19 Return c;
```

We show that (i) Algorithm 1 finds a correct solution for $a \star b$, and (ii) its running time is $\widetilde{O}(e_{\max}(|a| + |b|))$. Observe that Line 1 of Algorithm 1 runs in time $O(n)$ and all basic operations (*e.g.*, Lines 4 and 5) run in time $O(1)$ and thus all these lines in total take time $O(n \log n) = \widetilde{O}(n)$. Moreover, for any $[\alpha, \beta]$, $\mathcal{P}(\alpha, \beta)$ can be found in time $O(\log n)$ by binary searching the indices of a. More precisely,

in order to find $\mathcal{P}(\alpha, \beta)$ we need to find an index γ of a such that $x_\gamma \leq \alpha$ and an index δ such that $y_\delta \geq \beta$. Since both x and y are non-decreasing, we can find such indices in time $O(\log n)$. Therefore, the total running times of Lines 8, 11, and 13 is $O(n \log^2 n) = \widetilde{O}(n)$. The running time of the rest of the operations (Lines 14, 15, 16, and 18) depend on the length of the intervals $[\alpha_i, \beta_i]$ and $[\gamma_i, \delta_i]$. For a Round $s+1$, let $\ell_a = |a^1| + |a^2| + \ldots, |a^{2^s}|$ be the total length of the intervals $[\gamma_i, \delta_i]$. Similarly, define $\ell_b = |b^1| + |b^2| + \ldots + |b^{2^s}|$ and $\ell_c = |c^1| + |c^2| + \ldots + |c^{2^s}|$ as the total length of the intervals $[\alpha_i, \beta_i]$ and vectors c^i. It follow from the algorithm that in Round $s+1$, the running time of Lines 14, 15, and 18 is $\widetilde{O}(\ell_c)$ and the running time of Line 16 is $\widetilde{O}(e_{\max}\ell_c)$. Therefore, it only suffices to show that $\ell_c = O(n)$ to prove Algorithm 1 runs in time $\widetilde{O}(e_{\max} n)$.

Notice that in every Round $s+1$ we have $|b_i| = 2^{\log n - s}$ and thus $\ell_b = 2^s 2^{\log n - s} = n$. Moreover, for every c^i we have $c^i = a^i \star b^i$ and thus $|c^i| \leq |a^i| + |b^i|$. Therefore, $\ell_c \leq \ell_a + \ell_b = \ell_a + n$. Thus, in order to show $\ell_c = O(n)$, we need to prove that $\ell_a = O(n)$. To this end, we argue that for every $0 \leq i < n$, the i'th element of a appears in at most two intervals of $[\gamma_i, \delta_i]$. Suppose for the sake of contradiction that for $0 \leq \alpha_{j_1} < \beta_{j_1} < \alpha_{j_2} < \beta_{j_2} < \alpha_{j_3} < \beta_{j_3}$ we have $i \in [\gamma_{j_1}, \delta_{j_1}] \cap [\gamma_{j_2}, \delta_{j_2}] \cap [\gamma_{j_3}, \delta_{j_3}]$. Recall that depending on the parity of j_2, $[\gamma_{j_2}, \delta_{j_2}]$ is either equal to $\mathcal{P}(\alpha_{j_2}, \beta_{j_2}) \setminus \mathcal{P}(\alpha_{j_2+1}, \beta_{j_2+1})$ or $\mathcal{P}(\alpha_{j_2}, \beta_{j_2}) \setminus \mathcal{P}(\alpha_{j_2-1}, \beta_{j_2-1})$ and since $i \in [\gamma_{j_2}, \delta_{j_2}]$ then either of $i \notin \mathcal{P}(\alpha_{j_2-1}, \beta_{j_2-1})$ or $i \notin \mathcal{P}(\alpha_{j_2+1}, \beta_{j_2+1})$ hold. This implies that either $y_i < \beta_{j_2+1}$ or $x_i > \alpha_{j_2-1}$ which imply either $i \notin \mathcal{P}(\alpha_{j_1}, \beta_{j_1})$ or $i \notin \mathcal{P}(\alpha_{j_3}, \beta_{j_3})$ which is a contradiction. Thus, $\ell_a \leq 2n$ and therefore $\ell_c \leq 3n$. This shows that Algorithm 1 runs in time $\widetilde{O}(e_{\max} n)$.

To prove correctness, we show that (i) every a^i and b^i meet the condition of Lemma 3.3, and (ii) for every a_i and b_j such that $j \in [x_i, y_i]$ in some round of the algorithm and for some k, a^k contains a_i and b^k contains b_j.

We start with the former. Due to our algorithm, in every round for every $[\alpha_i, \beta_i]$ we have $[\gamma_i, \delta_i] \subseteq \mathcal{P}(\alpha_i, \beta_i)$. This implies that for every $i' \in [\gamma_i, \delta_i]$ and every $j' \in [\alpha_i, \beta_i]$ we have

$$\begin{aligned} a^i_{i'-\gamma_i} + b^i_{j'-\alpha_i} - e_{\max} &= a_{i'} + b_{j'} - e_{\max} \\ &\geq (a \star b)_{i'+j'} \\ &\geq (a^i \star b^i)_{i'+j'-\gamma_{i'}-\alpha_{j'}}. \end{aligned}$$

Thus, the condition of Lemma 3.3 holds for every a^i and b^i.

We finally show that for every $0 \leq i < n$ and every $0 \leq j < n$ such that $j \in [x_i, y_i]$, in some round of the algorithm we have $j \in [\alpha_k, \beta_k]$ and $i \in [\gamma_k, \delta_k]$ for some k. To this end, consider the first Round $s+1$ in which $i \in \mathcal{P}(\alpha_{\lceil j/2^{\log n - s} \rceil}, \beta_{\lceil j/2^{\log n - s} \rceil})$. We know that this eventually happens in some round since in Round $\log n + 1$ we have $i \in \mathcal{P}(\alpha_{\lceil j/2^{\log n - \log n} \rceil}, \beta_{\lceil j/2^{\log n - \log n} \rceil}) = \mathcal{P}(j, j)$. Round $s+1$ is the first round that $i \in \mathcal{P}(\alpha_{\lceil j/2^{\log n - s} \rceil}, \beta_{\lceil j/2^{\log n - s} \rceil})$ happens and thus either $s = 0$ or $s > 0$. The former completes the proof since it yields $i \in [\alpha_{\lceil j/2^{\log n - s} \rceil}, \beta_{\lceil j/2^{\log n - s} \rceil}]$. The latter implies that $i \notin [\alpha_{\lceil j/2^{\log n - s + 1} \rceil}, \beta_{\lceil j/2^{\log n - s + 1} \rceil}]$ and thus $i \in [\alpha_{\lceil j/2^{\log n - s} \rceil}, \beta_{\lceil j/2^{\log n - s} \rceil}]$. Thus, in Round $s+1$ we have $i \in [\gamma_k, \delta_k]$ and $j \in [\alpha_k, \beta_k]$ for $k = \lceil j/2^{\log n - s} \rceil$. □

4 THE KNAPSACK PROBLEM

In this section, we consider the knapsack problem and present a fast algorithm that can solve this problem for small values. In particular, when the maximum values of the items are constant, our algorithm runs in linear time. In this problem, we have a knapsack of size t, and n items each associated with size s_i and value v_i. The goal is to place a subset of the items into the knapsack with maximum total value subject to their total size being limited by t. In the 0/1 knapsack problem, we are allowed to use each item at most once, whereas in the unbounded knapsack problem, each item may be used several times. From this point on, the term "knapsack problem" refers to the 0/1 knapsack problem unless stated otherwise.

A classic dynamic programming algorithm yields a running time of $O(nt)$ [8] for the knapsack problem. On the negative side, it was shown recently that both the 0/1 and unbounded knapsack problems are as hard as (max, +) convolution. Thus it is unlikely to solve either problem in time $O((n+t)^{2-\epsilon})$ for any $\epsilon > 0$ [9]. However, there is no assumption on the values of the items in these reductions. Hence the hardness results do not carry over to the case of small values. In particular, a barely subquadratic time $(O(t^{1.859} + n))$ algorithm follows from the work of [6] when the item values are constant integer numbers. In what follows, we show that we can indeed solve the problem in truly subquadratic time when the input values are small. We assume throughout this section that the values of the items are integers in range $[0, v_{\max}]$. Using the prediction technique we present an $\widetilde{O}(v_{\max} t + n)$ time algorithm for the knapsack problem.

We begin by defining a knapsack variant of the (max, +) convolution in Section 4.1, and show that if the corresponding knapsack problems have non-negative integer values bounded by $v_{\max}$, then one can compute the (max, +) convolution of two vectors in time $\widetilde{O}(v_{\max} n)$. It follows from the recent technique of [9] that using this type of (max, +) convolution, one can solve the knapsack problem in time $\widetilde{O}(v_{\max} n)$. However, for the sake of completeness, we include a formal proof in the full version of the paper.

4.1 Knapsack Convolution

Let a and b be two vectors that correspond to the solutions of two knapsack instances k_a and k_b. More precisely, a_i is the maximum value of the items in knapsack problem k_a with a total size of at most i. Similarly, b_i is the maximum value of the items in knapsack problem k_b with a total size of at most i. We show that if the values of the items in k_a and k_b are non-negative integers bounded by $v_{\max}$, then one can compute $a \star b$ in time $\widetilde{O}(v_{\max}(|a| + |b|) + n)$ where n is the total number of items in k_a and k_b.

The sketch of the algorithm is as follows: We first define the fractional variants for the knapsack problem and the knapsack convolution. We show that both problems can be efficiently solved in time $O(n \log n)$ where n is the total number of items in each knapsack problem. Next, we observe that any solution of the fractional knapsack problem can be turned into a solution for the knapsack problem with an additive error of at most $v_{\max}$. Similarly, any solution for the fractional knapsack convolution is always at most $2v_{\max}$ away from the solution of the knapsack convolution. We then show that the solutions of the fractional knapsack problem and fractional knapsack convolution have certain properties. We

explore these properties and show that they enable us to find an uncertain solution for the knapsack convolution in time $\widetilde{O}(t+n)$. This yields an $\widetilde{O}(v_{\max}n)$ time solution for knapsack convolution via Theorem 3.5. In the interest of space, we omit some of the proofs of this section and include them in the full version of the paper.

We define the fractional variant of the knapsack problem as follows. Here we are allowed to divide the items into smaller pieces such that the value of each piece is proportional to the size of that piece. More formally, the fractional knapsack problem is defined as follows:

Definition 4.1. Given a knapsack of size t, and n items with sizes $s_1, s_2, \ldots, s_n$ and values $v_1, v_2, \ldots, v_n$, the fractional knapsack problem is to find non-negative real values $f_1, f_2, \ldots, f_n \leq 1$ such that $\sum_i s_i f_i \leq t$, and $\sum_i f_i v_i$ is maximized.

It is well-known that the fractional variant of the knapsack problem can be solved exactly via a greedy algorithm: sort the items in decreasing order of the ratio of value over size, and place them in the knapsack one by one. If at some point there is not enough space for the next item, we break it into a smaller piece that completely fills the knapsack. We stop when either we run out of items or the knapsack is full. We call this the greedy knapsack algorithm and name this simple observation.

Observation 4.1. *The greedy algorithm solves the fractional knapsack problem in time $O(n \log n)$.*

It is easy to see that in any solution of the greedy algorithm for knapsack, at most one item in the knapsack is broken into a smaller piece. Therefore, one can turn any solution of the fractional knapsack problem into a solution of the knapsack problem by removing the only item with $0 < f_i < 1$ (if any). If all the values are bounded by $v_{\max}$, this hurts the solution by at most an additive error of $v_{\max}$. Moreover, the solution of the fractional knapsack problem cannot be worse than that of the integral knapsack problem. Thus, any solution for the fractional knapsack problem can be turned into a solution for the knapsack problem with an additive error of at most $v_{\max}$.

Based on a similar idea, we define the fractional knapsack convolution of two vectors as follows:

Definition 4.2. Let a and b be two vectors corresponding to two knapsack problems k_a and k_b with knapsack sizes t_a and t_b. For a real value t, we define the fractional knapsack convolution of a and b with respect to t as the solution of the knapsack problem with knapsack size t and the union of items of k_a and k_b subject to the following two additional constraints:

- The total size of the items of k_a in the solution is bounded by t_a.
- The total size of the items of k_b in the solution is bounded by t_b.

One can modify the greedy algorithm to compute the solution of the fractional knapsack convolution as well. The only difference is that once the total size of the items of either knapsack instances in the solution reaches the size of that knapsack, we ignore the rest of the items from that knapsack. A similar argument to what we stated for Observation 4.1 proves the correctness of this algorithm.

Algorithm 2: GreedyAlgorithmForFractionalKnapsackConvolution($a, b, \mathsf{k}_a, \mathsf{k}_b$)

Data: t, a, b, and two knapsack instances k_a and k_b corresponding to a and b.
Result: The solution of fractional knapsack convolution of a and b with respect to t

```
Let items be a sequence of size n containing all items of k_a and k_b;
Sort items according to v_i/s_i in non-increasing order.
Answer ← 0;
for i ∈ [1, n] do
    if (s_i, v_i) belongs to k_a then
        cut ← min{s_i, t, t_a};
        Answer ← Answer + v_i cut/s_i;
        t ← t − cut;
        t_a ← t_a − cut;
    else
        cut ← min{s_i, t, t_b};
        Answer ← Answer + v_i cut/s_i;
        t ← t − cut;
        t_b ← t_b − cut;
Return Answer;
```

Observation 4.2. *Algorithm 2 solves the fractional knapsack convolution in time $O(n \log n)$.*

Again, one can observe that in any solution of the greedy algorithm for fractional knapsack convolution, at most two items may be included in the solution fractionally (at most one for each knapsack instance). Thus, we can get a solution with an additive error of at most $2v_{\max}$ for the knapsack convolution problem from the solution of the fractional knapsack convolution.

We explore several properties of the fractional solutions for the knapsack problems and the knapsack convolution. Based on these, we present an algorithm to compute an uncertain solution for the knapsack convolution within an error of $O(v_{\max})$. Define $a' : [0, t_a] \to \mathbb{R}$ and $b' : [0, t_b] \to \mathbb{R}$ as the solutions of the fractional knapsack problems for k_a and k_b, respectively. Therefore, for any real value x in the domain of the functions, $a'(x)$ and $b'(x)$ denote the solution of each fractional knapsack problem for knapsack size x. Moreover, we define a function $c : [0, t_a + t_b] \to \mathbb{R}$ where $c'(x)$ is the solution of the fractional knapsack convolution of a and b with respect to x. Note that for a', b', and c', the parameter x may be a real value. The following observations follow from the greedy solutions for a', b', and c'.

Observation 4.3. *There exist non-decreasing functions $\mathcal{F}_a : [0, t_a + t_b] \to [0, t_a]$ and $\mathcal{F}_b : [0, t_a + t_b] \to [0, t_b]$ such that $c'(x) = a'(\mathcal{F}_a(x)) + b'(\mathcal{F}_b(x))$.*

Since in Algorithm 2 we put the items greedily in the knapsack, for every $0 \leq x \leq t_a$, there exists a $0 \leq y \leq t_a + t_b$ such that $\mathcal{F}_a(y) = x$. Similarly, for every $0 \leq x \leq t_b$, there exists a $0 \leq y \leq t_a + t_b$ such that $\mathcal{F}_b(y) = x$. We define $\mathcal{F}_a^{-1}(x)$ as the smallest y

such that $\mathcal{F}_a(y) = x$. Moreover, $\mathcal{F}_b^{-1}(x)$ is equal to the smallest y such that $\mathcal{F}_b(y) = x$.

Observation 4.4. *For x, y, y' such that $0 \le x \le t_a$ and $0 \le y < y' \le \mathcal{F}_a^{-1}(x) - x$, we have $c'(x+y) - a'(x) + b'(y) \ge c'(x+y') - a'(x) - b'(y')$.*

Observation 4.5. *For x, y, y' such that $0 \le x \le t_a$ and $\mathcal{F}_a^{-1}(x) - x \le y < y' \le t_b$, we have $c'(x+y) - a'(x) + b'(y) \le c'(x+y') - a'(x) - b'(y')$.*

Let us define $g_x(y) = c'(x+y) - a'(x) - b'(y)$ for any $0 \le x \le t_a$. Then Observations 4.4 and 4.5 show that g_x is non-increasing in the interval $[0, \mathcal{F}_a^{-1}(x) - x]$ and non-decreasing in the interval $[\mathcal{F}_a^{-1}(x) - x, t_b]$. Now, for every integer $i \in [0, t_a]$, define α'_i to be the smallest number in range $[0, \mathcal{F}_a^{-1}(x) - x]$ such that $g_i(\alpha'_i) \le 2v_{\max}$. Similarly, define β'_i to be the largest number in range $[\mathcal{F}_a^{-1}(x) - x, t_b]$ such that $g_i(\beta'_i) \le 2v_{\max}$. It follows from Observations 4.4 and 4.5 that $g_i(x) \le 2v_{\max}$ holds in the interval $[\alpha'_i, \beta'_i]$ and $g_i(x) > 2v_{\max}$ holds for any x outside this range. Moreover, Observations 4.6 and 4.7 imply that $[\alpha'_i, \beta'_i]$'s are monotonic.

Observation 4.6. *Let x, x', and y be three real values such that $0 \le x < x' \le t_a$ and $0 \le y \le \mathcal{F}_a^{-1}(x) - x$. Then $c'(x+y) - a'(x) - b'(y) \le c'(x'+y) - a'(x') - b'(y)$.*

Observation 4.7. *Let x, x', and y be three real values such that $0 \le x < x' \le t_a$ and $0 \le \mathcal{F}_a^{-1}(x') - x' \le y$. Then $c'(x+y) - a'(x) - b'(y) \ge c'(x'+y) - a'(x') - b'(y)$.*

Notice that for every pair of integers i and j such that $\alpha'_i \le j \le \beta'_i$, we have $c'(i+j) - a'(i) - b'(j) \le 2v_{\max}$. Recall that a' and b' are the solutions of the fractional knapsack problems, and thus $a'(i) - a_i$ and $b'(j) - b_j$ are bounded by $v_{\max}$. Moreover, since $c'(i+j)$ is always at least as large as c_{i+j}, we have $c_{i+j} - a_i - b_j \le 4v_{\max}$ for all $\alpha'_i \le j \le \beta'_i$. Furthermore, for every integer $j \in [0, t_b] \setminus [\alpha'_i, \beta'_i]$ we have $c'(i+j) - a'(i) - b'(j) > 2v_{\max}$. Similarly, one can argue that $c'(i+j) \le c_{i+j} + 2v_{\max}$, $a'(i) \ge a_i$, and $b'(j) \ge b_j$, and thus $c_{i+j} - a_i - b_j > 0$. Therefore, intervals $[\alpha'_i, \beta'_i]$ make an uncertain solution for $a \star b$ within an error of $4v_{\max}$. To make the intervals integer, we set $\alpha_i = \lceil \alpha'_i \rceil$ and $\beta_i = \lfloor \beta'_i \rfloor$. Since $[\alpha_i, \beta_i]$ is also an uncertain solution within an error of $4v_{\max}$ we can compute $a \star b$ in time $\widetilde{O}(v_{\max}(|a| + |b|) + n)$.

Theorem 4.3. *Let k_a and k_b be two knapsack problems with knapsack sizes t_a and t_b and n items in total. Moreover, let the item values in k_a and k_b be integer values bounded by $v_{\max}$ and a and b be the solutions of these knapsack problems. There exists an $\widetilde{O}(v_{\max}(t_a + t_b) + n)$ time algorithm for computing $a \star b$ using a, b, k_a, and k_b.*

Proof. Let $t = t_a + t_b$ be the largest index of $a \star b$. As shown earlier, intervals $[\alpha_i, \beta_i]$ formulated above make an uncertain solution for $a \star b$ within an error of $4v_{\max}$. Thus, it only suffices to compute these intervals and then using Theorem 3.5 we can compute $a \star b$ in time $\widetilde{O}(v_{\max}(|a| + |b|))$. In order to determine the intervals, we first compute three arrays a', b', and c' with ranges $[0, t_a]$, $[0, t_b]$, and $[0, t_a + t_b]$, respectively. Then for every i in range $[0, t_a]$ we compute a'_i to be the solution to the fractional knapsack problem of k_a with knapsack size i. This can be done in time $O(n \log n + t)$, since we can use the greedy algorithm to determine these values. Similarly, we compute b'_i equal to the solution to the fractional knapsack problem for k_b and c_i equal to the solution of the fractional knapsack convolution for $a \star b$. This step of the algorithm takes a total time of $O(n \log n + t) = \widetilde{O}(n + t)$.

Along with the construction of array c', we also compute two arrays $\mathcal{F}_a$ and $\mathcal{F}_b$ in time $O(t)$ where $c'_i = a'_{\mathcal{F}_a i} + b'_{\mathcal{F}_b i}$. More precisely, every time we compute c'_i for some integer i we also keep track of the total size of the solution corresponding to each knapsack and store these values in arrays $\mathcal{F}_a$ and $\mathcal{F}_b$. Also one can compute an array $\mathcal{F}_a^{-1}$ from $\mathcal{F}_a$ in time $O(t)$. Next, we iterate over all integers i in range $[0, t_a]$ and for each i compute α_i and β_i in time $O(\log n)$. Recall that α_i (β_i) is the smallest (largest) integer j in range $[0, \mathcal{F}_a^{-1}{}_i - i]$ ($[\mathcal{F}_a^{-1}{}_i - i, t_b]$) such that $c'_{i+j} - a'_i - b'_j \le 2v_{\max}$. Moreover, $c'_{i+j} - a'_i - b'_j$ is monotonic in both ranges $[0, \mathcal{F}_a^{-1}{}_i - i]$ and $[\mathcal{F}_a^{-1}{}_i - i, t_b]$ and hence α_i and β_i can be computed in time $O(\log t_b)$ for every i. This makes a total running time of $O(t_a \log t_b) = \widetilde{O}(t)$. Finally, since intervals $[\alpha_i, \beta_i]$ make an uncertain solution for $a \star b$ within an error of $4v_{\max}$, we can compute $a \star b$ in time $\widetilde{O}(v_{\max}(t_a + t_b) + n)$ (Theorem 3.5). □

In the full version of the paper, we show that Theorem 4.3 yields a solution for the 0/1 knapsack problem in time $\widetilde{O}(v_{\max} t + n)$. The algorithm follows from the reduction of [9] from 0/1 knapsack to knapsack convolution.

Theorem 4.4 (A corollary of Theorem 4.3 and the reduction of [9] from 0/1 knapsack to (max, +) convolution). *The 0/1 knapsack problem can be solved in time $\widetilde{O}(v_{\max} t + n)$ when the item values are integer numbers in range $[0, v_{\max}]$.*

5 COMPUTING $a^{\star k}$ AND APPLICATION TO UNBOUNDED KNAPSACK

Throughout this section, we denote $\overbrace{a \star a \star \ldots \star a}^{k \text{ times}}$ by $a^{\star k}$. In this section, we present another application of the prediction technique for computing the k'th power of a vector in the (max, +) setting. The classic algorithm for this problem runs in time $\widetilde{O}(n^2)$ and thus far, there has not been any substantial improvement for this problem. We consider the case where the input values are integers in range $[0, e_{\max}]$, nonetheless, this result carries over to any range of integer numbers within an interval of size $e_{\max}$.[4] Using known FFT-based techniques, one can compute $a \star a$ in time $\widetilde{O}(e_{\max}|a|)$. However, the values of the elements of $a^{\star 2}$ no longer lie in range $[0, e_{\max}]$ and thus computing $a^{\star 2} \star a^{\star 2}$ requires more computation than $a \star a$. In particular, the values of the elements of $a^{\star k/2}$ are in range $[0, e_{\max} k/2]$ and thus computing $a^{\star k/2} \star a^{\star k/2}$ via the known techniques requires a running time of $\widetilde{O}(e_{\max} k |a^{\star k}|)$. The main result of this section is an algorithm for computing $a^{\star k}$ in time $\widetilde{O}(e_{\max}|a^{\star k}|)$. Moreover, we show that any prefix of size n of $a^{\star k}$ can be similarly computed in time $\widetilde{O}(e_{\max} n)$. We later make a connection between this problem and the unbounded knapsack problem and show this results in an $\widetilde{O}(e_{\max} t + n)$ time algorithm for the unbounded knapsack problem when the item values are

[4] It only suffices to add a constant C to every element of the vector to move the numbers to the interval $[0, e_{\max}]$. After computing the solution, we may move the solution back to the original space.

integers in range $[0, e_{\max}]$. Our algorithm is based on the prediction technique explained in Section 3.

We first explore some observations about the powers of a vector in the $(\max, +)$ setting. We begin by showing that if $b = a^{\star k}$ for some positive integer i, then the elements of b are (weakly) monotone.

Lemma 5.1. *Let a be a vector whose values are in range $[0, e_{\max}]$ and $b = a^{\star k}$ for a positive integer k. Then, for $0 \le i < j < |a^{\star k}|$ we have $b_j \ge b_i - e_{\max}$.*

Proof. If $k = 1$, the lemma follows from the fact that all values of the elements of a are in range $[0, e_{\max}]$. For $k > 1$, we define $c = a^{\star k-1}$ and let l be an index of c with the maximum c_i subject to $l \le j$. Since $b = c \star a$ we have $b_i = c_{i'} + a_{i-i'}$ for some i'. Note that $c_{i'} \le c_l$ and also all values of the indices of c are bounded by $e_{\max}$. Thus we have $b_i \le c_l + e_{\max}$. In addition to this, since $l \le j$ we have $b_j \ge c_l + a_{j-i}$. Notice that all values of the indices of a are non-negative and therefore $b_j \ge c_l$. This along with the fact that $b_i \le c_l + e_{\max}$ implies that $b_j \ge b_i - e_{\max}$. □

Another observation that we make is that if for a k and a k' we have $|k - k'| \le 1$, then $a^{\star k} \star a^{\star k'}$ can be computed by just considering a few (i, j) pairs of the vectors with close values.

Lemma 5.2. *Let k and k' be two positive integer exponents such that $|k - k'| \le 1$. Moreover, let a be an integer vector whose elements' values lie in range $[0, e_{\max}]$. Then, for every $0 \le i \le |a^{\star k}|$, there exist two indices j and $i - j$ such that (i) $a_i^{\star k+k'} = a_j^k + a_{i-j}^{\star k'}$ and (ii) $|a_j^k - a_{i-j}^{\star k'}| \le e_{\max}$.*

Proof. By definition $a^{\star k+k'} = \overbrace{a \star a \star \ldots \star a}^{k+k' \text{ times}}$. Therefore, for every $0 \le i < |a^{\star k+k'}|$, there exist $k + k'$ indices $i_1, i_2, \ldots, i_{k+k'}$ such that $a_i^{\star k+k'} = a_{i_1} + a_{i_2} + \ldots + a_{i_{k+k'}}$ and $i_1 + i_2 + \ldots + i_{k+k'} = i$. We assume w.l.o.g. that $a_{i_1} \le a_{i_2} \le \ldots \le a_{i_{k+k'}}$. We separate the odd and even indices of i to form two sequences $i_1, i_3, \ldots$ and $i_2, i_4, \ldots$. Notice that since $|k - k'| \le 1$, the size of one of such sequences is k and the size of the other one is k'. We assume w.l.o.g. that the size of the odd sequence is k and the size of the even sequence is k'. We now define $j = i_1 + i_3 + \ldots$ and $j' = i_2 + i_4 + \ldots$. Since $i_1 + i_2 + \ldots = i$ then $j' = i - j$ holds. Since $a_i^{\star k+k'} = a_{i_1} + a_{i_2} + \ldots + a_{i_{k+k'}}$ we also have $a_j^{\star k} = a_{i_1} + a_{i_3} + \ldots$, $a_{j'}^{\star k'} = a_{i_2} + a_{i_4} + \ldots$, and also $a_i^{\star k+k'} = a_j^{\star k} + a_{j'}^{\star k'}$. To complete the proof, it only suffices to show that $|a_j^{\star k} - a_{j'}^{\star k'}| \le e_{\max}$. This follows from the fact that the value of all indices of a are in range $[0, e_{\max}]$ and that $a_{i_1} \le a_{i_2} \le a_{i_3} \le \ldots \le a_{i_{k+k'}}$. □

What Lemma 5.2 implies is that when computing $a^{\star k} = a^{\star \lceil k/2 \rceil} \star a^{\star \lfloor k/2 \rfloor}$, it only suffices to take into account (i, j) pairs such that $|a_i^{\star \lceil k/2 \rceil} - a_j^{\star \lfloor k/2 \rfloor}| \le e_{\max}$. This observation enables us to compute $a^{\star k} = a^{\star \lceil k/2 \rceil} \star a^{\star \lfloor k/2 \rfloor}$ in time $\widetilde{O}(e_{\max}|a^{\star k}|)$ via the prediction technique. Suppose a is an integer vector with values in range $[0, e_{\max}]$. In addition to this, assume that $\hat{a} = a^{\star \lceil k/2 \rceil}$ and $\bar{a} = a^{\star \lfloor k/2 \rfloor}$. We propose an algorithm that receives $\hat{a}$ and $\bar{a}$ as input and computes $a^{\star k} = \hat{a} \star \bar{a}$ as output. The running time of our algorithm is $\widetilde{O}(e_{\max}|a^{\star k}|)$.

We define two integer vectors $\hat{b}$ and $\bar{b}$ where $\hat{b}_i = \max_{j \le i} \hat{a}_j$. Similarly, $\bar{b}_i = \max_{j \le i} \bar{a}_j$. By definition, both vectors $\hat{b}$ and $\bar{b}$ are non-decreasing. Now, for every index i of $\hat{b}$ we find an interval $[x_i, y_i]$ of $\bar{b}$ such that $\hat{b}_i - 2e_{\max} \le \bar{b}_j \le \hat{b}_i + 2e_{\max}$ for any j within $[x_i, y_i]$. Since both vectors $\hat{b}$ and $\bar{b}$ are non-decreasing, computing each interval takes time $O(\log n)$ via binary search. Finally, we provide these intervals to the prediction technique and compute $a^{\star k} = \hat{a} \star \bar{a}$ in time $\widetilde{O}(e_{\max}|a^{\star k}|)$. In Lemma 5.3, we prove that the intervals adhere to the conditions of the prediction technique and thus Algorithm 3 correctly computes $a^{\star k}$ from $\hat{a}$ and $\bar{a}$ in time $\widetilde{O}(e_{\max}|a^{\star k}|)$.

Algorithm 3: FastPower($\hat{a}, \bar{a}, e_{\max}$)

Data: Two vectors $\hat{a}$ and $\bar{a}$ s.t. $\hat{a} = a^{\star \lceil k/2 \rceil}$ and $\bar{a} = a^{\star \lfloor k/2 \rfloor}$ for some a and k.
Result: $\hat{a} \star \bar{a}$

```
Let b̂, b̄ be two vectors of size |a^{⋆⌈k/2⌉}| and |a^{⋆⌊k/2⌋}| respectively.;
b̂_0 ← â_1;
b̄_0 ← ā_1;
for i ∈ [1, |â| − 1] do
    b̂_i ← max{b̂_{i−1}, â_i};
for i ∈ [1, |ā| − 1] do
    b̄_i ← max{b̄_{i−1}, ā_i};
for i ∈ [1, |a^{⋆k}| − 1] do
    x_i ← the smallest j such that b̄_j ≥ b̄_i − 2e_max;
    y_i ← the largest j such that b̄_j ≤ b̄_i + 2e_max;
c = PredictionMethod(â, ā, 5e_max, x_i's, y_i's);
Return c;
```

Lemma 5.3. *Let a be an integer vector with values in range $[0, e_{\max}]$. For some integer $k > 0$, let $\hat{a} = a^{\star \lceil k/2 \rceil}$ and $\bar{a} = a^{\star \lfloor k/2 \rfloor}$. Given $\hat{a}$ and $\bar{a}$ as input, Algorithm 3 computes $a^{\star k} = \hat{a} \star \bar{a}$ in time $\widetilde{O}(e_{\max}|a^{\star \lceil k \rceil}|)$.*

Proof. The correctness of Algorithm 3 boils down to whether intervals $[x_i, y_i]$ provided for the prediction technique meet the conditions of Theorem 3.5. Before we prove that the conditions are met, we note that by Lemma 5.1, the values of $\hat{b}$ and $\bar{b}$ are at most $e_{\max}$ more than that of $\hat{a}$ and $\bar{a}$. Moreover, by definition, the vectors $\hat{b}$ and $\bar{b}$ are non-decreasing and lower bounded by the values of $\hat{a}$ and $\bar{a}$.

First condition: The first condition is that for every $0 \le i < |a^{\star \lceil k/2 \rceil}|$ and $x_i \le j \le y_i$ we have $\hat{a}_i + \bar{a}_j \ge (\hat{a} \star \bar{a})_{i+j} - O(e_{\max})$. In what follows, we show that in fact $\hat{a}_i + \bar{a}_j \ge (\hat{a} \star \bar{a})_{i+j} - 5e_{\max}$ holds for such i's and j's. Due to Lemma 5.2, for every such i and j, there exist an i' and a j' such that $\hat{a}_{i'} + \bar{a}_{j'} = (\hat{a} \star \bar{a})_{i'+j'}$, $i' + j' = i + j$, and $|\hat{a}_{i'} - \bar{a}_{j'}| \le e_{\max}$. Therefore,

$$\begin{aligned}(\hat{a} \star \bar{a})_{i+j} &= (\hat{a} \star \bar{a})_{i'+j'} \\ &= \hat{a}_{i'} + \bar{a}_{j'} \\ &\le 2\min\{\hat{a}_{i'} + \bar{a}_{j'}\} + e_{\max} \\ &\le 2\min\{\hat{b}_{i'} + \bar{b}_{j'}\} + e_{\max}.\end{aligned}$$

In addition to this, we know that $i + j = i' + j'$ and thus either $i' \le i$ or $j' \le j$. In any case, since both $\hat{b}$ and $\bar{b}$ are non-decreasing, $\max\{\hat{b}_i, \bar{b}_j\} \ge \min\{\hat{b}_{i'}, \hat{b}_{j'}\}$ and therefore,

$$\begin{aligned}(\hat{a} \star \bar{a})_{i+j} &\le 2\min\{\hat{b}_{i'}, \bar{b}_{j'}\} + e_{\max} \\ &\le 2\max\{\hat{b}_i, \bar{b}_j\} + e_{\max}.\end{aligned}$$

Due to Algorithm 3, $\max\{\hat{b}_i, \hat{b}_j\} - \min\{\hat{b}_i, \bar{b}_j\} \le 2e_{\max}$ and hence

$$\begin{aligned}(\hat{a} \star \bar{a})_{i+j} &\le 2\max\{\hat{b}_i, \bar{b}_j\} + e_{\max} \\ &\le \hat{b}_i + \bar{b}_j + 3e_{\max} \\ &\le \hat{a}_i + \bar{a}_j + 5e_{\max}.\end{aligned}$$

Second condition: The second condition is that for every $0 \le i < |a^{\star\lceil k/2\rceil}|$, there exists a $0 \le j \le i$ such that $\hat{a}_j + \bar{a}_{i-j} = (\hat{a} \star \bar{a})_i$ and that $x_j \le i - j \le y_j$. We prove this condition via Lemma 5.2. Lemma 5.2 states that for every $|a^{\star\lceil k/2\rceil}|$ there exists a $0 \le j \le i$ such that satisfies $\hat{a}_j + \bar{a}_{i-j} = (\hat{a} \star \bar{a})_i$ and also $|\hat{a}_j - \bar{a}_{i-j}| \le e_{\max}$. Since the values of $\bar{b}, \hat{b}$ differ from $\hat{a}, \hat{b}$ by an additive factor of at most $e_{\max}$, the latter inequality implies $|\hat{b}_j - \bar{b}_{i-j}| \le 2e_{\max}$. Due to Algorithm 3, if $|\hat{b}_j - \bar{b}_{i-j}| \le 2e_{\max}$ then $i - j$ lies in the interval $[x_j, y_j]$.

Third condition: The third condition is regarding the monotonicity of x_i's and y_i's. This condition directly follows from the fact that both vectors $\hat{b}$ and $\bar{b}$ are non-decreasing and as such, the computed intervals are also non-decreasing.

Apart from an invocation of Algorithm 1, the rest of the operations in Algorithm 3 run in time $\widetilde{O}(n)$ and therefore the total running time of Algorithm 3 is $\widetilde{O}(e_{\max}|a^{\star\lceil k\rceil}|)$. □

Based on Lemma 5.3, for an integer vector with values in range $[0, e_{\max}]$, we can compute $a^{\star k}$ via $O(\log k)$ $\star$ operations, each of which takes time $\widetilde{O}(e_{\max}|a^{\star\lceil k\rceil}|)$. Moreover, we always need to make at most $O(\log k) = \widetilde{O}(1)$ $\star$ operations in order to compute $a^{\star k}$.

Theorem 5.4. *Let a be an integer vector with values in range $[0, e_{\max}]$. For any integer $k \ge 1$, one can compute $a^{\star k}$ in time $\widetilde{O}(e_{\max}|a^{\star\lceil k\rceil}|)$.*

Proof. The proof follows from the correctness of Algorithm 3 and the fact that it runs in time $\widetilde{O}(e_{\max}|a^{\star\lceil k\rceil}|)$. □

Theorem 5.4 provides a strong tool for solving many combinatorial problems including the unbounded knapsack problem. In order to compute the solution of the unbounded knapsack problem, it only suffices to construct a vector a of size t wherein a_i specifies the value of the heaviest items with size i. a itself specifies the solution of the unbounded knapsack problem if we are only allowed to put one item in the bag. Similarly, $a^{\star 2}$ denotes the solution of the unbounded knapsack problem when we can put up to two items in the knapsack. More generally, for every $1 \le k$, $a^{\star k}$ denotes the solution of the unbounded knapsack problem subject to using at most k items. This way, $a^{\star t}$ formulates the solution of the unbounded knapsack problem. Note that in order to solve the knapsack problem, we only need to compute a prefix of size $t + 1$ of $a^{\star t}$. This makes the running time of every $\star$ operation $\widetilde{O}(e_{\max} t)$ and thus computing the first $t + 1$ elements of $a^{\star t}$ takes time $\widetilde{O}(e_{\max} t)$.

Theorem 5.5 (A corollary of Theorem 5.4). *The unbounded knapsack problem can be solved in time $\widetilde{O}(v_{\max} t + n)$ when the item values are integers in range $[0, v_{\max}]$.*

6 KNAPSACK FOR ITEMS WITH SMALL SIZES

We also consider the case where the size of the items is bounded by $s_{\max}$. Note that in such a scenario, the values of the items can be large real values, however, each item has an integer size in range $[1, s_{\max}]$. We propose a randomized algorithm that solves the knapsack problem w.h.p. in time $\widetilde{O}(s_{\max}(n + t))$ in this case. Although our technique is novel, the idea of putting the items in random buckets has been used before (see *e.g.* [5, 9]). Our algorithm is as follows: we randomly put the items in $t/s_{\max}$ different buckets. Using the classic quadratic time knapsack algorithm we solve the problem for each bucket up to a knapsack size $\widetilde{O}(s_{\max})$. Next, we merge the solutions in $\log(t/s_{\max})$ rounds. In the first round, we merge the solutions for buckets 1 and 2, buckets 3 and 4, and so on. This results in $t/2s_{\max}$ different solutions for every pair of buckets at the end of the first round. In the second round, we do the same except that this time the number of buckets is divided by 2. After $\log(t/s_{\max})$ rounds, we only have a single solution and based on that, we determine the maximum value of the solution with a size bounded by t and report that value.

If we use the classic (max, +)-convolution for merging the solutions of two buckets, it takes time $O(t^2)$ for merging two solutions and yields a slow algorithm. The main idea to improve the running time of the algorithm is to merge the solutions via a faster algorithm. We explain the idea by stating a randomized argument. Throughout this paper, every time we use the term w.h.p. we mean with a probability of at least $1 - n^{-10}$.

Lemma 6.1. *Let $(s_1, v_1), (s_2, v_2), \ldots, (s_n, v_n)$ be n items with sizes in range $[1, s_{\max}]$. Let the total size of the items be S. For some $0 < p < 1/2$, we randomly select each item of this set with probability p and denote their total size by S'. If $s_{\max} \le 2pS$ then for some $C = \widetilde{O}(1)$ $|pS - S'| \le C\sqrt{s_{\max} pS}$ holds w.h.p. (with probability at least $1 - n^{-10}$).*

In the interest of space, we omit the proof of Lemma 6.1 here and include it in the full version.

In our analysis, we fix an arbitrary optimal solution of the problem and state our observations based on this solution. Since the sizes of the items are bounded by $s_{\max}$, then either our solution uses all items and has a total size of $\sum s_i$ (if $\sum s_i$ is not larger than t) or leaves some of the items outside the knapsack and therefore has a size in range $[t - s_{\max} + 1, t]$. One can verify in $O(n)$ if the total size of the items is bounded by t and compute the solution in this case. Therefore, from now on, we assume that the total size of the items is at least t and thus the solution size is in $[t - s_{\max} + 1, t]$.

Now, if we randomly distribute the items into $t/s_{\max}$ buckets then the expected size of the solution in each bucket is $O(s_{\max})$ and thus we expect the size of the solution in each bucket to be in range $[0, \widetilde{O}(s_{\max})]$ w.h.p. due to Lemma 6.1. Therefore, it suffices to compute the solution for each bucket up to a size of $\widetilde{O}(s_{\max})$. Next, we use Lemma 6.1 to merge the solutions in faster than quadratic time. Every time we plan to merge the solutions of two sets of items

S_1 and S_2, we expect the size of the solutions in these two sets to be in ranges $[t|S_1|/n - \widetilde{O}(\sqrt{t\mathrm{s}_{\max}|S_1|/n}), t|S_1|/n + \widetilde{O}(\sqrt{t\mathrm{s}_{\max}|S_1|/n})]$ and $[t|S_2|/n - \widetilde{O}(\sqrt{t\mathrm{s}_{\max}|S_2|/n}), t|S_2|/n + \widetilde{O}(\sqrt{t\mathrm{s}_{\max}|S_2|/n})]$ w.h.p. Therefore, if we only consider the values within these ranges, we can merge the solutions correctly w.h.p. and thus one can compute the solution for $S_1 \cup S_2$ w.h.p. in time $\widetilde{O}(\sqrt{t\mathrm{s}_{\max}(|S_1|+|S_2|)/n}^2) = \widetilde{O}(t\mathrm{s}_{\max}(|S_1|+|S_2|)/n)$. This enables us to compute the solution w.h.p. in time $\widetilde{O}(\mathrm{s}_{\max}(n+t))$.

Algorithm 4: KNPKSmallSizes

Data: Knapsack size t and n items (s_i, v_i) where $1 \le s_i \le \mathrm{s}_{\max}$ for all items.
Result: Solution for knapsack size t

```
Randomly distribute the items into t/s_max buckets;
for j ∈ [t/s_max] do
    x_{1,j} = solution of the problem for bucket i up to size (C + 2)s_max;
for i ∈ [2, ⌈log(t/s_max)⌉] do
    for j ∈ ⌈t/s_max/2^i⌉ do
        Combine the solutions of x_{i-1,2j-1} and x_{i-1,2j} into x_{i,j} (based on Lemma 6.1 );
Return max x_{⌈log(t/s_max)⌉,1};
```

Theorem 6.2. *There exists a randomized algorithm that correctly computes the solution of the knapsack problem in time $\widetilde{O}(\mathrm{s}_{\max}(n+t))$ w.h.p., if the item sizes are integers in range $[1, \mathrm{s}_{\max}]$.*

Proof. We assume w.l.o.g. that the total size of the items is at least t and thus the solution size is in range $[t - \mathrm{s}_{\max} + 1, t]$. As outlined earlier, we randomly put the items into $t/\mathrm{s}_{\max}$ buckets. Based on Lemma 6.1, the expected size of the solution in each bucket is in range $[\mathrm{s}_{\max} - 1, \mathrm{s}_{\max}]$. Therefore, by Lemma 6.1 w.h.p. the size of the solution in every bucket is at most $\mathrm{s}_{\max} + \widetilde{O}(\mathrm{s}_{\max}) = \widetilde{O}(\mathrm{s}_{\max})$. Therefore, for each bucket with n_i items we can compute the solution up to size $\widetilde{O}(\mathrm{s}_{\max})$ in time $\widetilde{O}(\mathrm{s}_{\max} n_i)$. Since $\sum n_i = n$, the total running time of this step is $\widetilde{O}(\mathrm{s}_{\max} n)$.

We merge the solutions in $\log(t/\mathrm{s}_{\max})$ rounds. In every round i, we make $t/\mathrm{s}_{\max}/2^i$ merges each corresponding to the solutions of 2^i buckets. By Lemma 6.1, the range of the solution size in every merge is $[\mathrm{s}_{\max}2^i - \widetilde{O}(\sqrt{{\mathrm{s}_{\max}}^2 2^i}), \mathrm{s}_{\max}2^i + \widetilde{O}(\sqrt{{\mathrm{s}_{\max}}^2 2^i})]$ w.h.p. Thus, every merge takes time ${\mathrm{s}_{\max}}^2 2^i$. Moreover, in every round i the number of merges is $t/\mathrm{s}_{\max}/2^i$. Therefore, the total running time of each phase is $\widetilde{O}(\mathrm{s}_{\max} t)$ and thus the algorithm runs in time $\widetilde{O}(\mathrm{s}_{\max}(n+t))$. In order to show our solution is correct with probability at least $1 - n^{-10}$, we argue that we make at most n merges and therefore the total error of our solution is at most $nn^{-10} = n^{-9}$. Thus, if we run Algorithm 4 twice and output the better of the generated answers, our error is bounded by $2(n^{-9})^2 = n^{-18}/2 \le n^{-10}$ and thus the output is correct with probability at least $1 - n^{-10}$. □

As a corollary of Theorem 6.2, we can also solve the unbounded knapsack problem in time $\widetilde{O}(\mathrm{s}_{\max}(n+t))$ if the sizes of the items are bounded by $\mathrm{s}_{\max}$.

Corollary 6.3 (of Theorem 6.2). *There exists a randomized $\widetilde{O}(\mathrm{s}_{\max}(n+t))$ time algorithm that solves the unbounded knapsack problem w.h.p. when the sizes are bounded by $\mathrm{s}_{\max}$.*

Proof. The crux of the argument is that in an instance of the unbounded knapsack problem if the sizes of two items are equal, we never use the item with the smaller value in our solution. Thus, this leaves us with $\mathrm{s}_{\max}$ different items. We also know that we use each item of size s_i at most $\lfloor t/s_i \rfloor$ times and thus if we copy the most profitable item of each size s_i, $\lfloor t/s_i \rfloor$ times, this gives us an instance of the 0/1 knapsack problem with $O(t \log \mathrm{s}_{\max})$ items. Using the algorithm of Theorem 6.2 we can solve this problem in time $\widetilde{O}(\mathrm{s}_{\max} t)$. Since the reduction takes time $O(n)$ the total running time is $\widetilde{O}(\mathrm{s}_{\max}(n+t))$. □

Using the same idea, one can also solve the problem in time $\widetilde{O}((n+t)\mathrm{s}_{\max})$ when each item has a given multiplicity.

7 ALGORITHMS FOR KNAPSACK WITH MULTIPLICITIES

In this section, we study the knapsack problem where items have multiplicities. We assume throughout this section that the sizes of the items are bounded by $\mathrm{s}_{\max}$. More precisely, for every item (s_i, v_i), m_i denotes the number of copies of this item that can appear in any solution. Using our algorithms, we show that when all the sizes are integers bounded by $\mathrm{s}_{\max}$, one can solve the problem in time $\widetilde{O}(n{\mathrm{s}_{\max}}^2 \min\{n, \mathrm{s}_{\max}\})$. Notice that this running time is independent of t. This result improves upon the $O(n^3{\mathrm{s}_{\max}}^2)$ time algorithm of [18].

We begin, as a warm-up, by considering the case where $m_i = \infty$ for all items. We show that in this case, the $O(n^2{\mathrm{s}_{\max}}^2)$ time algorithm of [18] can be improved to an $\widetilde{O}(n\mathrm{s}_{\max} + {\mathrm{s}_{\max}}^2 \min\{n, \mathrm{s}_{\max}\})$ time algorithm. Before we explain our algorithm, we state a mathematical lemma that will be later used in our proofs.

Lemma 7.1. *Let S be a subset of items with integer sizes. If $|S| \ge k$ then there exists a non-empty subset of S whose total size is divisible by k.*

Proof. Select k items of S and give them an arbitrary ordering. Let s_i be the total size of the first i items in this order. Therefore, $0 = s_0 < s_1 < s_2 < \ldots < s_k$ holds. By pigeonhole principal, from set $\{s_0, s_1, \ldots, s_k\}$ two numbers have the same remainder when divided by k. Therefore, for some $i < j$ we have $s_i \bmod k = s_j \bmod k$. This means that the total size of the items in positions $i+1$ to j is divisible by k. □

When all multiplicities are infinity, our algorithm is as follows: define $\mathsf{H} := \arg\max v_i/s_i$ to be the index of an item with the highest ratio of v_i/s_i or in other words, the most profitable item. We claim that there always exists an optimal solution for the knapsack problem in which the total size of all items except $(s_\mathsf{H}, v_\mathsf{H})$ is bounded by ${\mathrm{s}_{\max}}^2$.

Lemma 7.2. *Let I be an instance of the knapsack problem where the multiplicity of every item is equal to infinity and let $(s_\mathsf{H}, v_\mathsf{H})$ be an item with the highest ratio of v_i/s_i. There exists an optimal solution for I in which the number of items except $(s_\mathsf{H}, v_\mathsf{H})$ is smaller than s_H.*

Proof. We begin with an arbitrary optimal solution and modify the solution until the condition of the lemma is met. Due to Lemma 7.1, every set S with at least s_H items contains a subset whose total size is divisible by s_H. Therefore, until the number of items other than (s_H, v_H) drops below s_H, we can always find a subset of such items whose total size is divisible by s_H. Next, we replace this subset with multiple copies of (s_H, v_H) with the same total size. Since v_H/s_H is the highest ratio over all items, the objective value of the solution doesn't hurt, and thus it remains optimal. □

Since $s_i \le s_{max}$ holds for all items, Lemma 7.2 implies that in such a solution, the total size of all items except (s_H, v_H) is bounded by ${s_{max}}^2$. This implies that at least $\max\{0, \lfloor (t - {s_{max}}^2)/s_H \rfloor\}$ copies of item (s_H, v_H) appear in an optimal solution. Thus, one can put these items into the knapsack and solve the problem for the remaining space of the knapsack. Let the remaining space be t' which is bounded by ${s_{max}}^2 + s_{max}$. Therefore, the classic $O(nt')$ time algorithm for knapsack finds the solution in time $O(n{s_{max}}^2)$. Also, by Theorem 6.2, one can solve the problem in time $\widetilde{O}((n + t')s_{max}) = \widetilde{O}(ns_{max} + {s_{max}}^3)$. Thus, the better of two algorithms runs in time $\widetilde{O}(ns_{max} + {s_{max}}^2 \min\{n, s_{max}\})$. This procedure is shown in Algorithm 5.

Algorithm 5: KNPKInfiniteMultiplicities

Data: A knapsack size t and n items with sizes and values (s_i, v_i). $m_i = \infty$ and $s_i \le s_{max}$ hold for all $1 \le i \le n$
Result: The solution of the knapsack problem for knapsack size t

```
H ← arg max v_i/s_i;
cnt ← max{0, ⌊(t − s_max^2)/s_H⌋};
t' ← t − cnt · s_H;
if n ≤ s_max then
    Report cnt · v_H +
    ClassicKNPK(t', n, {(s_1, t_1), ..., (s_n, t_n)}, {m_1, ..., m_n});
else
    Report cnt · v_H +
    KNPKSmallSizes(t', n, {(s_1, t_1), ..., (s_n, t_n)}, {m_1, ..., m_n});
```

Theorem 7.3. *When $s_i \in [s_{max}]$ and $m_i = \infty$ hold for every item, Algorithm 5 computes the solution of the knapsack problem in time $\widetilde{O}(ns_{max} + {s_{max}}^2 \min\{n, s_{max}\})$.*

Proof. The main ingredient of this proof is Lemma 7.2. According to Lemma 7.2, there exists a solution in which apart from (s_H, v_H) type items, the total size of the remaining items is bounded by ${s_{max}}^2$. Therefore, we are guaranteed that at least cnt copies of item (s_H, v_H) appear in an optimal solution of the problem. Thus, the remaining space of the knapsack (t') is at most ${s_{max}}^2 + s_{max}$ and therefore Algorithm 5 solves the problem in time $\widetilde{O}(ns_{max} + {s_{max}}^2 \min\{n, s_{max}\})$. □

Next, we present our algorithm for the general case where every multiplicity $m_i \ge 1$ is a given integer number. Our solution for this case runs in time $\widetilde{O}(n{s_{max}}^2 \min\{n, s_{max}\})$. We assume w.l.o.g. that $t \ge {s_{max}}^2$, otherwise the better of the classic knapsack algorithm and our limited size knapsack algorithm solves the problem in time $\widetilde{O}(ns_{max} + {s_{max}}^2 \min\{n, s_{max}\})$. In addition to this, we assume that the items are sorted in decreasing order of v_i/s_i, that is

$$v_1/s_1 \ge v_2/s_2 \ge \ldots \ge v_n/s_n.$$

We define $t' = t - {s_{max}}^2$ to be a smaller knapsack size which is less than t by an additive factor of ${s_{max}}^2$. We construct a pseudo solution for the smaller knapsack problem, by putting the items one by one into the smaller knapsack (of size t') greedily. We stop when the next item does not fit into the knapsack. Let b_i be the number of copies of item (s_i, v_i) in our pseudo solution for the smaller knapsack problem. In what follows, we show that there exists an optimal solution for the original knapsack problem such that if $b_i \ge s_{max}$ holds for some item (s_i, v_i), then at least $b_i - s_{max}$ copies of (s_i, v_i) appear in this solution.

Lemma 7.4. *Let b_i denote the number of copies of item (s_i, v_i) in our pseudo solution for the smaller knapsack problem. There exists an optimal solution for the original knapsack problem that contains at least $b_i - s_{max}$ copies of each item (s_i, t_i) such that $b_i \ge s_{max}$.*

Proof. To show this lemma, we start with an optimal solution and modify it step by step to make sure the condition of the lemma is met. We denote the number of copies of item (s_i, v_i) in our solution by a_i. In every step, we find the smallest index i such that $a_i < b_i - s_{max}$. Notice that due to the greedy nature of our algorithm for constructing the pseudo solution and the fact that $b_i > 0$ then $b_j = m_j$ for every $j < i$. Hence, $a_j \le m_j = b_j$ holds for all $j \le i$. Since at least one copy of item (s_i, v_i) is not used in the optimal solution, then the unused space in the optimal solution is smaller than s_i. Recall that the total size of the pseudo solution is bounded by $t' = t - {s_{max}}^2$ and since $a_j \le b_j$ for all $j \le i$, then the first i items contribute to at most $t - {s_{max}}^2 - s_{max}s_i$ space units of the solution. Moreover, as we discussed above, the total size of the solution is at least $t - s_{max}$ and thus the rest of the items have a size of at least ${s_{max}}^2$ in our optimal solution. Therefore we have

$$\sum_{j=i+1}^{n} a_j s_j \ge {s_{max}}^2$$

and since $s_j \le s_{max}$ holds, we have $\sum_{j=i+1}^{n} a_j \ge s_{max} \ge s_i$. Based on Lemma 7.1 there exists a subset of these items whose total size is divisible by s_i and thus we can replace them with enough (and at most s_{max}) copies of item (s_i, v_i) without hurting the solution. At the end of this step a_i increases and all a_j for $j < i$ remain intact. Therefore after at most $\sum b_i$ steps, our solution has the desired property. □

What Lemma 7.4 suggests is that although our pseudo solution may be far from the optimal, it gives us important information about the optimal solution of our problem. If our pseudo solution uses all copies of items, it means that all items fit into the knapsack and therefore the solution is trivial. Otherwise, we know that the total size of the pseudo solution is at least $t' - s_{max} = t - {s_{max}}^2 - s_{max}$. Based on Lemma 7.4, for any item with $b_i \ge s_{max}$ we know that at least $b_i - s_{max}$ copies of this item appear in an optimal solution of our problem. Therefore, we can decrease the multiplicity of such items by $b_i - s_{max}$ and decrease the knapsack size by $(b_i - s_{max})s_i$.

We argue that after such modifications, the remaining size of the knapsack is at most $s_{max} + {s_{max}}^2 + n{s_{max}}^2$. Recall that the total size of the pseudo solution is at least $t - {s_{max}}^2 - s_{max}$ and therefore $\sum b_i s_i \geq t - {s_{max}}^2 - s_{max}$. This implies that

$$\begin{aligned}\sum \max\{0, b_i - s_{max}\}s_i &\geq \sum (b_i - s_{max})s_i \\ &= \sum b_i s_i - \sum s_{max} s_i \\ &\geq [t - {s_{max}}^2 - s_{max}] - \sum s_{max} s_i \\ &\geq [t - {s_{max}}^2 - s_{max}] - \sum {s_{max}}^2 \\ &= [t - {s_{max}}^2 - s_{max}] - n{s_{max}}^2 \\ &= t - s_{max} - (n+1){s_{max}}^2\end{aligned}$$

Therefore, after the above modifications, the remaining size of the knapsack is at most $s_{max} + (n+1){s_{max}}^2$. Thus, we can solve the problem in time $\widetilde{O}(n{s_{max}}^3)$ using Lemma 6.2 and solve the problem in time $\widetilde{O}(n^2{s_{max}}^2)$ using the classic knapsack algorithm. This procedure is explained in details in Algorithm 6.

Algorithm 6: KNPKGivenMultiplicities

Data: A knapsack size t and n items with sizes and values (s_i, v_i). n multiplicities $m_1, m_2, \ldots, m_n$. $s_i \leq s_{max}$ holds for all $1 \leq i \leq n$

Result: The solution of the knapsack problem for knapsack size t

$t' \leftarrow \max\{0, t - {s_{max}}^2\}$;
for $i \in [1, n]$ **do**
  $b_i \leftarrow \min\{m_i, \lfloor t'/s_i \rfloor\}$;
  $t' \leftarrow t' - b_i s_i$;
  if $b_i \neq m_i$ **then**
    break;
$t'' \leftarrow t$;
surplus $\leftarrow 0$;
for $i \in [1, n]$ **do**
  $t'' \leftarrow t'' - \max\{0, b_i - s_{max}\}s_i$;
  $m'_i \leftarrow m_i - \max\{0, b_i - s_{max}\}$;
  surplus $\leftarrow$ surplus $+ \max\{0, b_i - s_{max}\}v_i$;
if $n \leq s_{max}$ **then**
  Report surplus + ClassicKNPK$(t'', n, \{(s_1, t_1), \ldots, (s_n, t_n)\}, \{m'_1, \ldots, m'_n\})$;
else
  Report surplus + KNPKSmallSizes$(t'', n, \{(s_1, t_1), \ldots, (s_n, t_n)\}, \{m'_1, \ldots, m'_n\})$;

Theorem 7.5. *Algorithm 6 solves the knapsack problem in time* $\widetilde{O}(n{s_{max}}^2 \min\{n, s_{max}\})$ *when the sizes of the items are integers in range* $[1, s_{max}]$ *and each item has a given integer multiplicity.*

Proof. The proof is based on Lemma 7.4. After determining the values of vector b', we know that for each item (s_i, t_i) at least $b_i - s_{max}$ copies appear in the solution. Thus, we can remove the space required by these items and reduce the knapsack size. As we discussed before, after all these modifications, the new knapsack size (t'') is bounded by $s_{max} + (n+1){s_{max}}^2$ and thus the better of the classic knapsack algorithm and the algorithm of Section 6 solve the problem in time $\widetilde{O}(n{s_{max}}^2 \min\{n, s_{max}\})$. □

ACKNOWLEDGMENT

We would like to thank the anonymous reviewers for their careful reading of our paper and their many insightful comments and suggestions.

REFERENCES

[1] Kyriakos Axiotis and Christos Tzamos. 2018. Capacitated Dynamic Programming: Faster Knapsack and Graph Algorithms. *arXiv preprint arXiv:1802.06440* (2018).
[2] Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. 2017. Better Approximations for Tree Sparsity in Nearly-linear Time. In *SODA*. 2215–2229.
[3] Richard Bellman. 1957. *Dynamic Programming* (first ed.). Princeton University Press, Princeton, NJ, USA.
[4] David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. 2006. Necklaces, Convolutions, and X+Y. In *ESA*. 160–171.
[5] Karl Bringmann. 2017. A near-linear pseudopolynomial time algorithm for subset sum. In *SODA*. 1073–1084.
[6] Timothy M Chan and Moshe Lewenstein. 2015. Clustered integer 3SUM via additive combinatorics. In *STOC*. 31–40.
[7] V. Chvatal. 1980. Hard knapsack problems. *Operations Research* 28 (1980), 1402–1411.
[8] Thomas H.. Cormen, Charles Eric Leiserson, Ronald L Rivest, and Clifford Stein. 2001. *Introduction to algorithms*. MIT press Cambridge.
[9] Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. 2017. On problems equivalent to (min, +)-convolution. In *ICALP*. 22:1–22:15.
[10] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA.
[11] Ellis Horowitz and Sartaj Sahni. 1974. Computing Partitions with Applications to the Knapsack Problem. *J. ACM* 21, 2 (April 1974), 277–292.
[12] Hans Kellerer and Ulrich Pferschy. 2004. Improved Dynamic Programming in Connection with an FPTAS for the Knapsack Problem. *J. Comb. Optim.* 8, 1 (2004), 5–11. https://doi.org/10.1023/B:JOCO.0000021934.29833.6b
[13] Hans Kellerer, Ulrich Pferschy, and David Pisinger. 2004. *Knapsack problems*. Springer.
[14] Konstantinos Koiliaris and Chao Xu. 2017. A Faster Pseudopolynomial Time Algorithm for Subset Sum. In *SODA*. 1062–1072.
[15] Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. 2017. On the Fine-grained Complexity of One-Dimensional Dynamic Programming. In *ICALP*. 21:1–21:15.
[16] Silvano Martello and Paolo Toth. 1990. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., New York, NY, USA.
[17] David Pisinger. 1999. Linear Time Algorithms for Knapsack Problems with Bounded Weights. *Journal of Algorithms* 33 (1999), 1–14.
[18] Arie Tamir. 2009. New pseudopolynomial complexity bounds for the bounded and other integer Knapsack related problems. *Operations Research Letters* 37, 5 (2009), 303–306.
[19] Uri Zwick. 1998. All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms. In *FOCS*. 310–319.
[20] Uri Zwick. 2002. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *JACM* 49, 3 (2002), 289–317.