Fast Algorithms for Knapsack via Convolution and Prediction

MohammadHossein Bateni[*] MohammadTaghi HajiAghayi[†‡] Saeed Seddighin[‡]
Clifford Stein[§¶]

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 $\mathsf{s}_{\max}$, the running time of our algorithm is $\tilde{O}((n+t)\mathsf{s}_{\max})$. If the item values are integers bounded by $\mathsf{v}_{\max}$, our algorithm runs in time $\tilde{O}(n+t\mathsf{v}_{\max})$. Best previously known running times were $O(nt)$, $O(n^2\mathsf{s}_{\max})$ and $O(n\mathsf{s}_{\max}\mathsf{v}_{\max})$ (Pisinger, J. of Alg., 1999).

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 $\widetilde{O}(n\mathsf{e}_{\max})$ when an approximation of the solution within an additive error of $\mathsf{e}_{\max}$ is available.

Our results also improve the best known strongly polynomial time solutions for knapsack. In the limited size setting, when the items have multiplicities, the fastest strongly polynomial time algorithms for knapsack run in time $O(n^2{\mathsf{s}_{\max}}^2)$ and $O(n^3{\mathsf{s}_{\max}}^2)$ for the cases of infinite and given multiplicities, respectively. Our results improve both running times by a factor of $\widetilde{\Omega}(n\max\{1,n/\mathsf{s}_{\max}\})$.

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.*, [2, 9, 13, 19, 20]), as it is one of the most well-known NP-hard problems [12]. 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 $\sum_{i\in S} s_i\leq t$.

[*]Google Inc., New York
[†]University of Maryland, College Park
[‡]Supported in part by NSF CAREER award CCF-1053605, NSF BIGDATA grant IIS-1546108, NSF AF:Medium grant CCF-1161365, DARPA GRAPHS/AFOSR grant FA9550-12-1-0423, and another DARPA SIMPLEX grant.
[§]Columbia University, New York, NY 10027, USA
[¶]Research supported in part by NSF grants CCF-1421161 and CCF-1714818. Some research done while visiting Google

Recent evidence suggests that a classic $O(nt)$ dynamic-programming solution for the knapsack problem [2] might be the fastest in the worst case. In fact, solving the knapsack problem was shown to be equivalent to the $(\min,+)$ convolution problem [11], 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) [10].

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 $|b|=n$ 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.*, [1, 3, 4, 8, 11, 16, 17]). Most importantly, many problems, such as tree sparsity, subset sum, and 3-sum, have been shown to have conditional lower bounds on their running time via their intimate connection with $(\min,+)$ convolution.

In particular, previous studies have shown that (max,+) convolution, knapsack, and tree sparsity are computationally (almost) equivalent [11]. 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 [8] implicitly 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 for the knapsack problem, if either the values or sizes of items are small integers. More specifically, if item sizes are integers bounded by $\mathsf{s}_{\max}$, the running time of our algorithm is $\tilde{O}((n+t)\mathsf{s}_{\max})$. If the item values are integers bounded by $\mathsf{v}_{\max}$, our algorithm runs in time $\tilde{O}(n+t\mathsf{v}_{\max})$. Best previously known running times were $O(nt)$, $O(n^2\mathsf{s}_{\max})$ and $O(n\mathsf{s}_{\max}\mathsf{v}_{\max})$ [20]. 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 [26] (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 strongly subquadratic-time algorithm is known to exist. Nevertheless, our

[1]It follows from the reduction of [11] that any subquadratic algorithm for convolution yields a subquadratic algorithm for knapsack.

techniques provide almost linear running time for the parameterized case of $(\min, +)$ convolution when the input numbers are bounded by the parameters.

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 $\widetilde{O}(n\,\mathsf{e}_{\max})$ when an approximation of the solution within an additive error of $\mathsf{e}_{\max}$ is available. Note that such an approximation is not difficult to find for a typical knapsack instance (*e.g.*, for $\mathsf{e}_{\max} \geq \mathsf{v}_{\max}$): simply sort the items in decreasing order of value to cost ratio and greedily pack your knapsack with them. A summary of the previous known results along with our new[2] results is shown in Table 1. Notice that in 0/1 knapsack, t is always bounded by $n\,\mathsf{s}_{\max}$ and thus our results improve the previously known algorithms even when t appears in the running time.

Table 1: n and t denote the number of items and the knapsack size respectively. $\mathsf{v}_{\max}$ and $\mathsf{s}_{\max}$ denote the maximum value and size of the items. Notice that when the knapsack problem does not have multiplicity, t is always bounded by $n\mathsf{s}_{\max}$ and thus our running times are always better than the previously known algorithms. Theorems C.2, D.3, and D.5, as well as Corollary C.3 are randomized and output a correct solution with probability at least $1 - n^{-10}$.

setting	running time	our improvement
general setting	$O(nt)$ [10]	-
limited size knapsack	$O(n^2\mathsf{s}_{\max})$ [20]	$\widetilde{O}((n+t)\mathsf{s}_{\max})$ (Theorem C.2)
limited size knapsack, unlimited multiplicity	$O(n^2{\mathsf{s}_{\max}}^2)$ [22]	$\widetilde{O}(n\mathsf{s}_{\max} + {\mathsf{s}_{\max}}^2 \min\{n, \mathsf{s}_{\max}\})$ (Theorem D.3) $\widetilde{O}((n+t)\mathsf{s}_{\max})$ (Corollary C.3)
limited size knapsack, given multiplicity	$O(n^3{\mathsf{s}_{\max}}^2)$ [22]	$\widetilde{O}(n{\mathsf{s}_{\max}}^2 \min\{n, \mathsf{s}_{\max}\})$ (Theorem D.5)
limited value knapsack	-	$\widetilde{O}(n + t\mathsf{v}_{\max})$ (Theorem A.4)
limited value knapsack, unlimited multiplicity	-	$\widetilde{O}(n + t\mathsf{v}_{\max})$ (Theorem B.5)
limited value and size	$O(n\mathsf{s}_{\max}\mathsf{v}_{\max})$ [20]	$\widetilde{O}((n+t)\min\{\mathsf{v}_{\max}, \mathsf{s}_{\max}\})$ (Theorems A.4 and C.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 $\underbrace{a \star a \star \ldots \star a}_{k \text{ times}}$.

[2]We wish to emphasize that, as far as we know, our results are in no way implied by previous work (e.g., [14, 15], which tend to have in the running time the *sum* of the profit values rather than their *maximum*.).

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 [11]. In other words, any subquadratic solution for knapsack 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 $\widetilde{O}(n\mathsf{e}_{\max})$ when an approximation of the solution within an additive error of $\mathsf{e}_{\max}$ is given. As we show in Sections A and B, this method can be applied to the 0/1 knapsack and unbounded knapsack problems to solve them in $\widetilde{O}(n\,\mathsf{e}_{\max})$ time (*e.g.*, if $\mathsf{e}_{\max} \geq \mathsf{v}_{\max}$). 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 $\widetilde{O}(\mathsf{e}_{\max}(|a| + |b|))$ when all values of a and b are integers in the range $[0, \mathsf{e}_{\max}]$. This reduction has been used in many previous works (*e.g.*, [1, 4, 8, 25, 26]). 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 $\widetilde{O}(\mathsf{e}_{\max}(|a| + |b|))$. We give a detailed explanation of this in Section H.

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 $a_i + b_j \geq (a \star b)_{i+j} - \mathsf{e}_{\max}$ for all i and j. Notice that the case where all input values are in the range $[0, \mathsf{e}_{\max}]$ is a special case of the small distortion case. Given such a constraint, we show that $a \star b$ can be computed in time $\widetilde{O}(\mathsf{e}_{\max} n)$ 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 $[0, O(\mathsf{e}_{\max})]$. Next, we approximate the convolution of the transformed vectors via the results of the first step, and eventually compute $a \star b$ in time $\widetilde{O}(\mathsf{e}_{\max} n)$. 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 $\mathsf{e}_{\max}$, this method lets us compute the convolution in time $\widetilde{O}(\mathsf{e}_{\max} n)$. 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 $\mathsf{e}_{\max}$; (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 $\mathsf{e}_{\max}$.

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 $\mathsf{e}_{\max}$ suffices to compute the convolution of two vectors in time $\widetilde{O}(\mathsf{e}_{\max}n)$. 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 $\widetilde{O}(\mathsf{e}_{\max}n)$. We show that all the subproblems can be solved in time $\widetilde{O}(\mathsf{e}_{\max}n)$ in total, and based on these solutions, $a \star b$ can be computed in time $\widetilde{O}(\mathsf{e}_{\max}n)$. We give more details in Section 3.2.

Theorem 3.4 [restated informally]. *Given two integer vectors a and b and an uncertain solution for $a \star b$ within an error of $\mathsf{e}_{\max}$, one can compute $a \star b$ in time $\widetilde{O}(\mathsf{e}_{\max}n)$.*

Notice that in Theorem 3.4, 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 A that the prediction technique enables us to solve the 0/1 knapsack problem in time $\widetilde{O}(\mathsf{v}_{\max}t + n)$. 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 $\mathsf{v}_{\max}$, then an uncertain solution for $a \star b$ within an error of $\mathsf{v}_{\max}$ 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 $\mathsf{v}_{\max}$ 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 3.4 compute $a \star b$ in time $\widetilde{O}(\mathsf{v}_{\max}n)$. Finally, we use the recent technique of [11] to reduce the 0/1 knapsack problem to the knapsack convolution. This yields an $\widetilde{O}(\mathsf{v}_{\max}t + n)$ time algorithm for solving the 0/1 knapsack problem when the item values are bounded by $\mathsf{v}_{\max}$.

Theorem A.4 [restated]. *The 0/1 knapsack problem can be solved in time $\widetilde{O}(\mathsf{v}_{\max}t + n)$ when the item values are integer numbers in the range $[0, \mathsf{v}_{\max}]$.*

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 $[0, \mathsf{e}_{\max}]$, the total running time of the algorithm is $\widetilde{O}(\mathsf{e}_{\max}|a^{\star k}|)$. This improves upon the trivial $\widetilde{O}({\mathsf{e}_{\max}}^2|a^{\star k}|)$. Similar to what we do in Section A, 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 $O(\mathsf{e}_{\max})$ and apply the prediction technique to compute the exact solution in time $\widetilde{O}(\mathsf{e}_{\max}|a^{\star k}|)$.

Theorem B.4 [restated]. *Let a be an integer vector with values in the range $[0, \mathsf{e}_{\max}]$. For any integer $k \geq 1$, one can compute $a^{\star k}$ in time $\widetilde{O}(\mathsf{e}_{\max}|a^{\star k}|)$.*

As a consequence of Theorem B.4, we show that the unbounded knapsack problem can be solved in time $\widetilde{O}(n + \mathsf{v}_{\max}t)$.

Theorem B.5 [restated]. *The* unbounded knapsack *problem can be solved in time* $\widetilde{O}(n + \mathsf{v}_{\max} t)$ *when the item values are integers in the range* $[0, \mathsf{v}_{\max}]$.

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 $[1, \mathsf{s}_{\max}]$. For this case, we present a randomized algorithm that solves the problem w.h.p.[3] in time $\widetilde{O}(\mathsf{s}_{\max}(n + t))$. The idea is to first put the items into $t/\mathsf{s}_{\max}$ buckets uniformly at random. Next, we solve the problem for each bucket separately, up to a knapsack size $\widetilde{O}(\mathsf{s}_{\max})$. 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 $\widetilde{O}(\mathsf{s}_{\max}(n + t))$, yielding a randomized algorithm for solving the knapsack problem in time $\widetilde{O}(\mathsf{s}_{\max}(n + t))$ w.h.p. when the sizes of the items are bounded by $\mathsf{s}_{\max}$.

Theorem C.2 [restated]. *There exists a randomized algorithm that correctly computes the solution of the* knapsack *problem in time* $\widetilde{O}(\mathsf{s}_{\max}(n+t))$ *w.h.p., when the item sizes are integers in the range* $[1, \mathsf{s}_{\max}]$.

2.3 Implication to Strongly Polynomial Time Algorithms

When we parameterize the 0/1 knapsack problem by $\max\{s_i\} \leq \mathsf{s}_{\max}$, one can set $t' := \min(t, n\mathsf{s}_{\max})$ and solve the problem with knapsack size t' in time $\widetilde{O}((t' + n)\mathsf{s}_{\max}) = \widetilde{O}(n{\mathsf{s}_{\max}}^2)$. This yields a strongly polynomial time solution for the knapsack problem. However, this only works when we are allowed to use each item only once. In Section D, 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 $\widetilde{O}(n{\mathsf{s}_{\max}}^2 \min\{n, \mathsf{s}_{\max}\})$ when m_i's are arbitrary and solves the problem in time $\widetilde{O}(n\mathsf{s}_{\max} \min\{n, \mathsf{s}_{\max}\})$ when $m_i = \infty$ for all i. Both results improve the algorithms of [22] by a factor of $\widetilde{\Omega}(\max\{n, \mathsf{s}_{\max}\})$ in the running time. These results are all implied by Theorem C.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 Sections F and G, 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.

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

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

[3]With probability at least $1 - n^{-10}$.

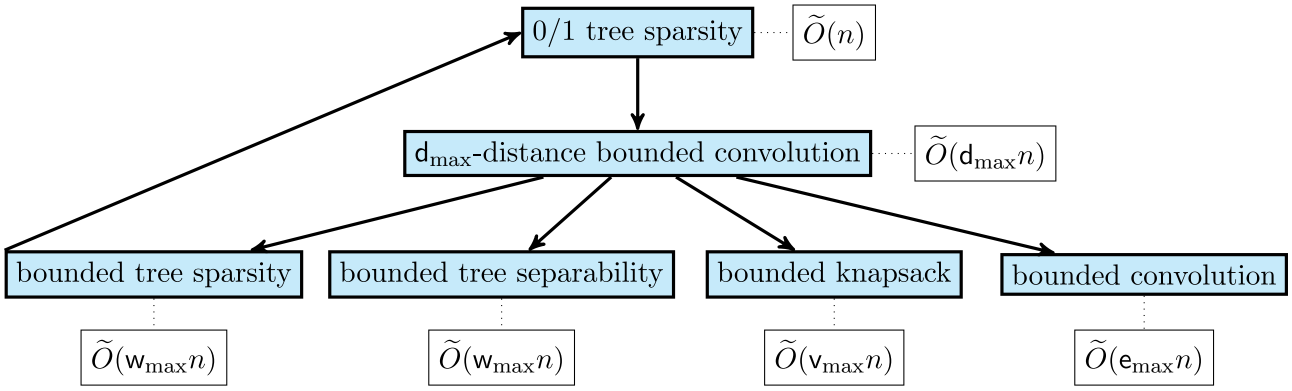
0/1 tree sparsity
$\widetilde{O}(n)$
$\mathsf{d}_{\max}$-distance bounded convolution
$\widetilde{O}(\mathsf{d}_{\max}n)$
bounded tree sparsity
bounded tree separability
bounded knapsack
bounded convolution
$\widetilde{O}(\mathsf{w}_{\max}n)$
$\widetilde{O}(\mathsf{w}_{\max}n)$
$\widetilde{O}(\mathsf{v}_{\max}n)$
$\widetilde{O}(\mathsf{e}_{\max}n)$

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

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, as we show in Section H, if the values of the vectors are integers in the range $[0, \mathsf{e}_{\max}]$, one can compute the $(\max, +)$ convolution of two vectors in time $\widetilde{O}(\mathsf{e}_{\max}n)$.

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 $\mathsf{e}_{\max}$ is given, then it is possible to compute the $(\max, +)$ convolution of the vectors in time $\widetilde{O}(\mathsf{e}_{\max}n)$. Later in Sections A and B 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 Section H. The same reduction has been used as a blackbox in many recent works [1, 4, 8, 25, 26]. Based on this reduction, we show that an $\widetilde{O}(\mathsf{e}_{\max}n)$ time algorithm can compute the convolution of two integer vectors whose values are in the range $[0, \mathsf{e}_{\max}]$. We further explain in Section H that even if the values of the vectors are real but in the range $[0, \mathsf{e}_{\max}]$, one can approximate the solution within an additive error less than 1. These results hold even if the input values can be either in the interval $[0, \mathsf{e}_{\max}]$ 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 $\widetilde{O}(\mathsf{e}_{\max}n)$ when for every i and j we have $|a_i + b_j - (a \star b)_{i+j}| \leq \mathsf{e}_{\max}$. Finally, in Section 3.2 we use these results to present the prediction technique for computing the $(\max, +)$ convolution of two vectors in time $\widetilde{O}(\mathsf{e}_{\max}n)$.

3.1 An $\widetilde{O}(\mathsf{e}_{\max}n)$ 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 $\mathsf{e}_{\max}$. Indeed this condition is strictly weaker than the condition studied in Section H. Nonetheless we show that still an $\widetilde{O}(\mathsf{e}_{\max}n)$ time algorithm can compute $a \star b$ if the values of the vectors are integers but not necessarily in the range $[0, \mathsf{e}_{\max}]$. In the interest of space, we omit the proofs of Lemmas 3.1, 3.2, and 3.3 and include them in Section I.

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 $\widetilde{O}(\mathsf{e}_{\max} n)$. 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 $0 \le i, j < n$ we have $(a \star b)_{i+j} - a_i - b_j \le \mathsf{e}_{\max}$. Then,*

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

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 $\widetilde{O}(\mathsf{e}_{\max} n)$ 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 $0 \le i < |a \star b|$.

These two operations help us transform the vectors a and b such that all elements fall in the range $[0, O(\mathsf{e}_{\max})]$. Next, we approximate the convolution of the transformed vectors via the results of Section H, and eventually compute $a \star b$ in time $\widetilde{O}(\mathsf{e}_{\max} n)$.

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

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 $|b| \ge |a|$ 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 $O(\mathsf{e}_{\max}|a|)$ 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 $\widetilde{O}(\mathsf{e}_{\max}(|a| + |b|))$.

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

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 $\mathsf{e}_{\max}$ can be used to compute the exact solution of a $(\max, +)$ convolution in time $\widetilde{O}(\mathsf{e}_{\max} n)$. In general, an additive approximation of $\mathsf{e}_{\max}$ does not suffice to compute the $(\max, +)$ convolution in time $\widetilde{O}(\mathsf{e}_{\max} n)$. 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 Algorithm 7 doesn't 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 $\mathsf{e}_{\max}$, a well-known greedy algorithm can approximate $a \star b$ within an additive error of $\mathsf{e}_{\max}$ (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 $\mathsf{e}_{\max}$ 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 $\mathsf{e}_{\max}$ 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 A.

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 $\mathsf{e}_{\max}$ away from $(a \star b)_{i+j}$ for all $j \in [x_i, y_i]$. Also, we know that for any $0 \leq i < |a \star b|$ there exists a j such that $a_j + b_{i-j} = (a \star b)_i$ and $x_j \leq i - j \leq y_j$. We call such data *an uncertain solution.* We show in Theorem 3.4 that if an uncertain solution is given, then one can compute $a \star b$ in time $\widetilde{O}(\mathsf{e}_{\max} n)$. For empty intervals only, y_i is allowed to be smaller than x_i.

Theorem 3.4. *Let a and b be two integer vectors and assume we have $|a|$ intervals $[x_i, y_i]$ such that*

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

Then, one can compute $a \star b$ from a, b, and the intervals in time $\widetilde{O}(\mathsf{e}_{\max}(|a| + |b|))$.

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 $x_i \leq y_i$, 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 $x_i \leq \alpha$ and $y_i \geq \beta$. We denote the projection of an interval $[\alpha, \beta]$ by $\mathcal{P}(\alpha, \beta)$. We first show that for every $0 \leq \alpha \leq \beta < n$, $\mathcal{P}(\alpha, \beta)$ corresponds to an interval of a. We defer the proof of Observation 3.1 to Appendix J.

Observation 3.1. *For every $0 \leq \alpha \leq \beta < n$, $\mathcal{P}(\alpha, \beta)$ 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 Appendix J.

Observation 3.2. *For $0 \leq \alpha_1 \leq \beta_1 < \alpha_2 \leq \beta_2 < n$, 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 $0 \leq i < |c^1|$.

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 $1 \leq i \leq 2^s$. Finally, for every $1 \leq i \leq 2^s$ and $0 \leq j < |c^i|$, we set $c_{\alpha_i+\gamma_i+j} = \max\{c_{\alpha_i+\gamma_i+j}, c^i_j\}$.

We show that (i) Algorithm 1 finds a correct solution for $a \star b$, and (ii) its running time is $\widetilde{O}(\mathsf{e}_{\max}(|a| + |b|))$. 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 $\widetilde{O}(\mathsf{e}_{\max}\ell_c)$. Therefore, it only suffices to show that $\ell_c = O(n)$ to prove Algorithm 1 runs in time $\widetilde{O}(\mathsf{e}_{\max} n)$.

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 $x_i > \alpha_{j_2-1}$

Algorithm 1: $\mathsf{ConvolutionViaPredictionMethod}(a, b, \mathsf{e}_{\max}, x_i\text{'s}, y_i\text{'s})$

```
Data: Two integer vectors $a$ and $b$ of size $n$, intervals $[x_i, y_i]$ for $0 \leq i < n$ meeting the
      conditions of Theorem 3.4
Result: $a \star b$
$c \leftarrow$ a vector of size $2n-1$ with indices set to $\infty$ initially;
for $s \in [0, \log n]$ do
    for $i \in [1, 2^s]$ do
        $\alpha_i \leftarrow (i-1)2^{(\log n)-s}$;
        $\beta_i \leftarrow i2^{(\log n)-s} - 1$;
    for $i \in [1, 2^s]$ do
        if $s = 0$ then
            $[\gamma_i, \delta_i] \leftarrow \mathcal{P}(\alpha_i, \beta_i)$;
        else
            if $i$ is odd then
                $[\gamma_i, \delta_i] \leftarrow \mathcal{P}(\alpha_i, \beta_i) \setminus \mathcal{P}(\alpha_{i+1}, \beta_{i+1})$;
            else
                $[\gamma_i, \delta_i] \leftarrow \mathcal{P}(\alpha_i, \beta_i) \setminus \mathcal{P}(\alpha_{i-1}, \beta_{i-1})$;
        $a^i \leftarrow$ a vector of size $\delta_i - \gamma_i + 1$ s.t. $a^i_j = a_{\gamma_i + j}$;
        $b^i \leftarrow$ a vector of size $2^{(\log n)-s}$ s.t. $b^i_j = b_{\alpha_i + j}$;
        $c^i \leftarrow \mathsf{DistortedConvolution}(a^i, b^i, \mathsf{e}_{\max})$;
        for $j \in [1, |c^i|]$ do
            $c_{\alpha_i+\gamma_i+j} \leftarrow \max\{c_{\alpha_i+\gamma_i+j}, c^i_j\}$;
Return $c$;
```

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 $\widetilde{O}(\mathsf{e}_{\max} n)$.

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

$$a^i_{i'-\gamma_i} + b^i_{j'-\alpha_i} - \mathsf{e}_{\max} = a_{i'} + b_{j'} - \mathsf{e}_{\max} \geq (a \star b)_{i'+j'} \geq (a^i \star b^i)_{i'+j'-\gamma_{i'}-\alpha_{j'}}.$$

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 $k = \lceil j/2^{\log n - s}\rceil$. □

References

[1] A. Backurs, P. Indyk, and L. Schmidt. Better approximations for tree sparsity in nearly-linear time. In *SODA*, pages 2215–2229, 2017.

[2] R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, first edition, 1957.

[3] D. Bremner, T. M. Chan, E. D. Demaine, J. Erickson, F. Hurtado, J. Iacono, S. Langerman, M. Patrascu, and P. Taslakian. Necklaces, convolutions, and X+Y. In *ESA*, pages 160–171, 2006.

[4] K. Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *SODA*, pages 1073–1084, 2017.

[5] K. Bringmann, F. Grandoni, B. Saha, and V. V. Williams. Truly sub-cubic algorithms for language edit distance and RNA-folding via fast bounded-difference min-plus product. In *FOCS*, pages 375–384, 2016.

[6] M. Bussieck, H. Hassler, G. J. Woeginger, and U. T. Zimmermann. Fast algorithms for the maximum convolution problem. *Oper. Res. Lett.*, 15(3):133–141, Apr. 1994.

[7] M. L. Carmosino, J. Gao, R. Impagliazzo, I. Mihajlin, R. Paturi, and S. Schneider. Nondeterministic extensions of the strong exponential time hypothesis and consequences for non-reducibility. In *ITCS*, pages 261–270, 2016.

[8] T. M. Chan and M. Lewenstein. Clustered integer 3SUM via additive combinatorics. In *STOC*, pages 31–40, 2015.

[9] V. Chvatal. Hard knapsack problems. *Operations Research*, 28:1402–1411, 1980.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to algorithms.* MIT press Cambridge, 2001.

[11] M. Cygan, M. Mucha, K. Wegrzycki, and M. Wlodarczyk. On problems equivalent to $(\min, +)$-convolution. In *ICALP*, pages 22:1–22:15, 2017.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness.* W. H. Freeman & Co., New York, NY, USA, 1990.

[13] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *J. ACM*, 21(2):277–292, April 1974.

[14] H. Kellerer and U. Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004.

[15] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack problems.* Springer, 2004.

[16] K. Koiliaris and C. Xu. A faster pseudopolynomial time algorithm for subset sum. In *SODA*, pages 1062–1072, 2017.

[17] M. Künnemann, R. Paturi, and S. Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *ICALP*, pages 21:1–21:15, 2017.

[18] P. Maragos. Differential morphology. In S. Mitra and G. Sicuranza, editors, *Nonlinear Image Processing*, chapter 10, pages 289–329. Academic Press, 2000.

[19] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, Inc., New York, NY, USA, 1990.

[20] D. Pisinger. Linear time algorithms for knapsack problems with bounded weights. *Journal of Algorithms*, 33:1–14, 1999.

[21] D. D. Sleator and R. E. Tarjan. A data structure for dynamic trees. In *STOC*, pages 114–122, 1981.

[22] A. Tamir. New pseudopolynomial complexity bounds for the bounded and other integer knapsack related problems. *Operations Research Letters*, 37(5):303–306, 2009.

[23] J. V. Uspensky. *Introduction to mathematical probability.* McGraw-Hill, 1937.

[24] D. B. West. *Introduction to graph theory.* Prentice Hall, second edition, 2001.

[25] U. Zwick. All pairs shortest paths in weighted directed graphs-exact and almost exact algorithms. In *FOCS*, pages 310–319, 1998.

[26] U. Zwick. All pairs shortest paths using bridging sets and rectangular matrix multiplication. *JACM*, 49(3):289–317, 2002.

A 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 value of the items is constant, our algorithm runs in linear time. In this problem, we have a knapsack of size t and n items each associated to a 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, we can use each item several times. From now on, every time we use the term knapsack problem we mean the 0/1 knapsack problem unless stated otherwise.

A classic dynamic programming algorithm yields a running time of $O(nt)$ [10] for the knapsack problem. On the negative side, recently it has been shown that both the 0/1 and unbounded knapsack problems are as hard as $(\max, +)$ convolution and thus it is unlikely to solve any of these problems in time $O((n+t)^{2-\epsilon})$ for any $\epsilon > 0$ [11]. However, there is no assumption on the values of the items in these reductions and thus the hardness results don't 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 [8] 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 $[0, \mathsf{v}_{\max}]$. Using the prediction technique we present an $\widetilde{O}(\mathsf{v}_{\max} t + n)$ time algorithm for the knapsack problem.

We begin by defining a knapsack variant of the $(\max, +)$ convolution in Section A.1 and show that if the corresponding knapsack problems have non-negative integer values bounded by $\mathsf{v}_{\max}$, then one can compute the $(\max, +)$ convolution of two vectors in time $\widetilde{O}(\mathsf{v}_{\max} n)$. It follows from the recent technique of [11] that using this type of $(\max, +)$ convolution, one can solve the knapsack problem in time $\widetilde{O}(\mathsf{v}_{\max} n)$. However, for the sake of completeness, we include a formal proof in Appendix K.

A.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 $\mathsf{v}_{\max}$, then one can compute $a \star b$ in time $\widetilde{O}(\mathsf{v}_{\max}(|a|+|b|)+n)$ 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 of both 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 $\mathsf{v}_{\max}$. Similarly, any solution for the fractional knapsack convolution is always at most $2\mathsf{v}_{\max}$ away from the solution of the knapsack convolution. We then show that both the solution of the fractional knapsack problem and the solution of fractional knapsack convolution have some 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 $\widetilde{O}(\mathsf{v}_{\max} n)$ time solution for knapsack convolution via Theorem 3.4. In the interest of space, we omit some of the proofs of this section and include them in Appendix L.

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

Definition A.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 n non-negative real values $f_1, f_2, \ldots, f_n$ such that $\sum s_i f_i \leq t$, $f_i \leq 1$ for all i, and $\sum f_i v_i$ is maximized.*

One well-known observation is that the fractional variant of the knapsack problem can be solved exactly via a greedy algorithm: sort the items according to the ratio of value over size and put the items into the knapsack accordingly. 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 A.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, there is at most one item in the knapsack which 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 from the knapsack with $f_i < 1$ (if any). If all the values are bounded by $\mathsf{v}_{\max}$, this hurts the solution by at most an additive factor of $\mathsf{v}_{\max}$. Moreover, the solution of the fractional knapsack problem is always no less than the solution 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 $\mathsf{v}_{\max}$.

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

Definition A.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 times 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 A.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 $\mathsf{k}_a$ and $\mathsf{k}_b$ corresponding to $a$ and $b$.
Result: The solution of fractional knapsack convolution of $a$ and $b$ with respect to $t$
1  Let items be a sequence of size $n$ containing all items of $\mathsf{k}_a$ and $\mathsf{k}_b$;
2  Sort the items of items according to $v_i/s_i$ in non-increasing order.
3  $Answer \leftarrow 0$;
4  for $i \in [1, n]$ do
5      if $(s_i, v_i)$ belongs to $\mathsf{k}_a$ then
6          $cut \leftarrow \min\{s_i, t, t_a\}$;
7          $Answer \leftarrow Answer + v_i cut/s_i$;
8          $t \leftarrow t - cut$;
9          $t_a \leftarrow t_a - cut$;
10     else
11         $cut \leftarrow \min\{s_i, t, t_b\}$;
12         $Answer \leftarrow Answer + v_i cut/s_i$;
13         $t \leftarrow t - cut$;
14         $t_b \leftarrow t_b - cut$;
15 Return $Answer$;
```

Observation A.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, there are at most two items that are fractionally included in the solution (at most one for each knapsack instance). Thus, we can get a solution with an additive error of at most $2\mathsf{v}_{\max}$ 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 and based on them, we present an algorithm to compute an uncertain solution for the knapsack convolution within an error of $O(\mathsf{v}_{\max})$. 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 all a', b', and c', parameter x may be a real value. The following observations follow from the greedy solutions for a', b', and c'.

Observation A.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 A.4. *For an $0 \leq x \leq t_a$, y, and y' such that $0 \leq y < y' \leq \mathcal{F}_a^{-1}(x) - x$ we have $c'(x+y) - a'(x) + b'(y) \geq c'(x+y') - a'(x) - b'(y')$.*

Observation A.5. *For an $0 \leq x \leq t_a$, y, and y' such that $\mathcal{F}_a^{-1}(x) - x \leq y < y' \leq t_b$ we have $c'(x+y) - a'(x) + b'(y) \leq c'(x+y') - a'(x) - b'(y')$.*

Observations A.4 and A.5 show that for any $0 \leq x \leq t_a$, if we define $g_x(y) = c'(x+y) - a'(x) - b'(y)$ then g_x is non-decreasing in the interval $[0, \mathcal{F}_a^{-1}(x) - x]$ and non-increasing 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 $g_i(\alpha'_i) \leq 2\mathsf{v}_{\max}$. Similarly, define β'_i to be the largest number in range $[\mathcal{F}_a^{-1}(x) - x, t_b]$ such that $g_i(\beta'_i) \leq 2\mathsf{v}_{\max}$. It follows from Observations A.4 and A.5 that $g_i(x) \leq 2\mathsf{v}_{\max}$ holds in the interval $[\alpha'_i, \beta'_i]$ and $g_i(x) > 2\mathsf{v}_{\max}$ holds for any x outside this range. Moreover, Observations A.6 and A.7 imply that $[\alpha'_i, \beta'_i]$'s are monotonic.

Observation A.6. *Let x, x', and y be three real values such that $0 \leq x < x' \leq t_a$ and $0 \leq y \leq \mathcal{F}_a^{-1}(x) - x$. Then $c'(x+y) - a'(x) - b'(y) \leq c'(x'+y) - a'(x') - b'(y)$.*

Observation A.7. *Let x, x', and y be three real values such that $0 \leq x < x' \leq t_a$ and $0 \leq \mathcal{F}_a^{-1}(x') - x' \leq y$. Then $c'(x+y) - a'(x) - b'(y) \geq c'(x'+y) - a'(x') - b'(y)$.*

Notice that for every pair of integers i and j such that $\alpha'_i \leq j \leq \beta'_i$ we have $c'(i+j) - a'(i) - b'(j) \leq 2\mathsf{v}_{\max}$. 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 $\mathsf{v}_{\max}$. Moreover, since $c'(i+j)$ is always at least as large as c_{i+j}, we have $c_{i+j} - a_i - b_j \leq 4\mathsf{v}_{\max}$ for all $\alpha'_i \leq j \leq \beta'_i$. Furthermore, for every integer $j \in [0, t_b] \setminus [\alpha'_i, \beta'_i]$ we have $c'(i+j) - a'(i) - b'(j) > 2\mathsf{v}_{\max}$. Similarly, one can argue that $c'(i+j) \leq c_{i+j} + 2\mathsf{v}_{\max}$, $a'(i) \geq a_i$, and $b'(j) \geq b_j$ and thus $c_{i+j} - a_i - b_j > 0$ which means that intervals $[\alpha'_i, \beta'_i]$ make an uncertain solution for $a \star b$ within an error of $4\mathsf{v}_{\max}$. 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 $4\mathsf{v}_{\max}$ we can compute $a \star b$ in time $\widetilde{O}(\mathsf{v}_{\max}(|a| + |b|) + n)$.

Theorem A.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 $\mathsf{v}_{\max}$ and a and b be the solutions of these knapsack problems. There exists an $\widetilde{O}(\mathsf{v}_{\max}(t_a + t_b) + n)$ 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 $4\mathsf{v}_{\max}$. Thus, it only suffices to compute these intervals and then using Theorem 3.4 we can compute $a \star b$ in time $\widetilde{O}(\mathsf{v}_{\max}(|a| + |b|))$. 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 $c'_{i+j} - a'_i - b'_j \leq 2\mathsf{v}_{\max}$. 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 $4\mathsf{v}_{\max}$, we can compute $a \star b$ in time $\widetilde{O}(\mathsf{v}_{\max}(t_a + t_b) + n)$ (Theorem 3.4). $\square$

In Appendix K we show that Theorem A.3 yields a solution for the 0/1 knapsack problem in time $\widetilde{O}(\mathsf{v}_{\max} n)$. The algorithm follows from the reduction of [11] from 0/1 knapsack to knapsack convolution.

Theorem A.4 (a corollary of Theorem A.3 and the reduction of [11] from 0/1 knapsack to $(\max, +)$ convolution)**.** *The 0/1 knapsack problem can be solved in time $O(\mathsf{v}_{\max} t + n)$ when the item values are integer numbers in range $[0, \mathsf{v}_{\max}]$.*

B Computing $a^{\star k}$ and Application to Unbounded Knapsack

Throughout this section, any time we mention $a^{\star k}$ we mean $\overbrace{a \star a \star \ldots \star a}^{k \text{ times}}$. 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 $[0, \mathsf{e}_{\max}]$, nonetheless, this result carries over to any range of integer numbers within an interval of size $\mathsf{e}_{\max}$.[4] Using known FFT-based techniques, one can compute $a \star a$ in time $\widetilde{O}(\mathsf{e}_{\max}|a|)$ (see Section H). However, the values of the elements of $a^{\star 2}$ no longer lie in range $[0, \mathsf{e}_{\max}]$ 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 $[0, \mathsf{e}_{\max} k/2]$ and thus computing $a^{\star k/2} \star a^{\star k/2}$ via the known techniques requires a running time of $\widetilde{O}(\mathsf{e}_{\max} k |a^{\star k}|)$. The main result of this section is an algorithm for computing $a^{\star k}$ in time $\widetilde{O}(\mathsf{e}_{\max}|a^{\star k}|)$. Moreover, we show that any prefix of size n of $a^{\star k}$ can be similarly computed in time $\widetilde{O}(\mathsf{e}_{\max} n)$. We later make a connection between this problem and the unbounded knapsack problem and show this results in an $\widetilde{O}(\mathsf{e}_{\max} t + n)$ time algorithm for the unbounded knapsack problem when the item values are integers in range $[0, \mathsf{e}_{\max}]$. 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 B.1. *Let a be a vector whose values are in range $[0, \mathsf{e}_{\max}]$ and $b = a^{\star k}$ for a positive integer k. Then, for $0 \leq i < j < |a^{\star k}|$ we have $b_j \geq b_i - \mathsf{e}_{\max}$.*

Proof. If $k = 1$, the lemma follows from the fact that all values of the elements of a are in range $[0, \mathsf{e}_{\max}]$. 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 $l \leq j$. Since $b = c \star a$ we have $b_i = c_{i'} + a_{i-i'}$ for some i'. Note that $c_{i'} \leq c_l$ and also all values of the indices of c are bounded by $\mathsf{e}_{\max}$. Thus we have $b_i \leq c_l + \mathsf{e}_{\max}$. In addition to this, since $l \leq j$ we have $b_j \geq c_l + a_{j-i}$. Notice that all values of the indices of a are non-negative and therefore $b_j \geq c_l$. This along with the fact that $b_i \leq c_l + \mathsf{e}_{\max}$ implies that $b_j \geq b_i - \mathsf{e}_{\max}$. □

Another observation that we make is that if for a k and a k' we have $|k - k'| \leq 1$, 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 B.2. *Let k and k' be two positive integer exponents such that $|k - k'| \leq 1$. Moreover, let a be an integer vector whose elements' values lie in range $[0, \mathsf{e}_{\max}]$. Then, for every $0 \leq i \leq |a^{\star k}|$, 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) $|a_j^k - a_{i-j}^{\star k'}| \leq \mathsf{e}_{\max}$.*

Proof. By definition $a^{\star k+k'} = \overbrace{a \star a \star \ldots \star a}^{k+k' \text{ times}}$. Therefore, for every $0 \leq i < |a^{\star k+k'}|$, 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 $a_{i_1} \leq a_{i_2} \leq \ldots \leq a_{i_{k+k'}}$. 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 $|k - k'| \leq 1$, 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,

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

it only suffices to show that $|a^{\star k}_{j} - a^{\star k'}_{j'}| \le \mathsf{e}_{\max}$. This follows from the fact that the value of all indices of a are in range $[0, \mathsf{e}_{\max}]$ and that $a_{i_1} \le a_{i_2} \le a_{i_3} \le \ldots \le a_{i_{k+k'}}$. $\square$

What Lemma B.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 $|a^{\star \lceil k/2 \rceil}_i - a^{\star \lfloor k/2 \rfloor}_j| \le \mathsf{e}_{\max}$. This observation enables us to compute $a^{\star k} = a^{\star \lceil k/2 \rceil} \star a^{\star \lfloor k/2 \rfloor}$ in time $\widetilde{O}(\mathsf{e}_{\max}|a^{\star k}|)$ via the prediction technique. Suppose a is an integer vector with values in range $[0, \mathsf{e}_{\max}]$. 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 $\widetilde{O}(\mathsf{e}_{\max}|a^{\star k}|)$.

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 $\hat{b}_i - 2\mathsf{e}_{\max} \le \bar{b}_j \le \hat{b}_i + 2\mathsf{e}_{\max}$ 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 $\widetilde{O}(\mathsf{e}_{\max}|a^{\star k}|)$. In Lemma B.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 $\widetilde{O}(\mathsf{e}_{\max}|a^{\star k}|)$.

```
Algorithm 3: FastPower($\hat{a}, \bar{a}, \mathsf{e}_{\max}$)
    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: $\hat{a} \star \bar{a}$
 1  Let $\hat{b}, \bar{b}$ be two vectors of size $|a^{\star \lceil k/2 \rceil}|$ and $|a^{\star \lfloor k/2 \rfloor}|$ respectively.;
 2  $\hat{b}_0 \leftarrow \hat{a}_1$;
 3  $\bar{b}_0 \leftarrow \bar{a}_1$;
 4  for $i \in [1, |\hat{a}| - 1]$ do
 5      $\hat{b}_i \leftarrow \max\{\hat{b}_{i-1}, \hat{a}_i\}$;
 6  for $i \in [1, |\bar{a}| - 1]$ do
 7      $\bar{b}_i \leftarrow \max\{\bar{b}_{i-1}, \bar{a}_i\}$;
 8  for $i \in [1, |a^{\star k}| - 1]$ do
 9      $x_i \leftarrow$ the smallest $j$ such that $\bar{b}_j \ge \bar{b}_i - 2\mathsf{e}_{\max}$;
10      $y_i \leftarrow$ the largest $j$ such that $\bar{b}_j \le \bar{b}_i + 2\mathsf{e}_{\max}$;
11  $c = $ PolynomialMultiplicationViaPredictionMethod($\hat{a}, \bar{a}, 5\mathsf{e}_{\max}, x_i$'s, $y_i's$);
12  Return $c$;
```

Lemma B.3. *Let a be an integer vector with values in range $[0, \mathsf{e}_{\max}]$. 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 $\widetilde{O}(\mathsf{e}_{\max}|a^{\star \lceil k \rceil}|)$.*

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 3.4. Before we prove that the conditions are met, we note that by Lemma B.1, the values of $\hat{b}$ and $\bar{b}$ are at most $\mathsf{e}_{\max}$ 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 $\hat{a}_i + \bar{a}_j \ge (\hat{a} \star \bar{a})_{i+j} - O(\mathsf{e}_{\max})$. In what follows, we show that in fact $\hat{a}_i + \bar{a}_j \ge (\hat{a} \star \bar{a})_{i+j} - 5\mathsf{e}_{\max}$ holds for such i's and j's. Due to Lemma B.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 $|\hat{a}_{i'} - \bar{a}_{j'}| \leq \mathsf{e}_{\max}$. Therefore,

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

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

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

Due to Algorithm 3, $\max\{\hat{b}_i, \bar{b}_j\} - \min\{\hat{b}_i, \bar{b}_j\} \leq 2\mathsf{e}_{\max}$ and hence

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

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

Based on Lemma B.3, for an integer vector with values in range $[0, \mathsf{e}_{\max}]$, we can compute $a^{\star k}$ via $O(\log k)$ $\star$ operations, each of which takes time $\widetilde{O}(\mathsf{e}_{\max}|a^{\star\lceil k\rceil}|)$. Moreover, we always need to make at most $O(\log k) = \widetilde{O}(1)$ $\star$ operations in order to compute $a^{\star k}$.

Theorem B.4. *Let a be an integer vector with values in range $[0, \mathsf{e}_{\max}]$. For any integer $k \geq 1$, one can compute $a^{\star k}$ in time $\widetilde{O}(\mathsf{e}_{\max}|a^{\star\lceil k\rceil}|)$.*

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

Theorem B.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 $1 \leq k$, $a^{\star k}$ 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 $\widetilde{O}(\mathsf{e}_{\max} t)$ and thus computing the first $t+1$ elements of $a^{\star t}$ takes time $\widetilde{O}(\mathsf{e}_{\max} t)$.

Theorem B.5 (a corollary of Theorem B.4)**.** *The unbounded knapsack problem can be solved in time* $\widetilde{O}(\mathsf{v}_{\max} t + n)$ *when the item values are integers in range* $[0, \mathsf{v}_{\max}]$.

C Knapsack for Items with Small Sizes

We also consider the case where the size of the items is bounded by $\mathsf{s}_{\max}$. 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 $[1, \mathsf{s}_{\max}]$. We propose a randomized algorithm that solves the knapsack problem w.h.p. in time $\widetilde{O}(\mathsf{s}_{\max}(n+t))$ in this case. Our algorithm is as follows: we randomly put the items in $t/\mathsf{s}_{\max}$ different buckets. Using the classic quadratic time knapsack algorithm we solve the problem for each bucket up to a knapsack size $\widetilde{O}(\mathsf{s}_{\max})$. Next, we merge the solutions in $\log(t/\mathsf{s}_{\max})$ rounds. In the first round, we merge the solutions for buckets 1 and 2, buckets 3 and 4, and so on. This results in $t/2\mathsf{s}_{\max}$ 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 $\log(t/\mathsf{s}_{\max})$ 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 C.1. *Let $(s_1,v_1),(s_2,v_2),\dots,(s_n,v_n)$ be n items with sizes in range $[1,\mathsf{s}_{\max}]$. 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 $\mathsf{s}_{\max} \le 2pS$ then for some $\mathsf{C}=\widetilde{O}(1)$ $|pS-S'| \le \mathsf{C}\sqrt{\mathsf{s}_{\max}pS}$ holds w.h.p. (with probability at least $1-n^{-10}$).*

Proof. This lemma follows from the Bernstein's inequality [23]. Bernstein's inequality states that if $x_1,x_2,\dots,x_n$ are n independent random variables strictly bounded by the intervals $[a_i,b_i]$ and $\bar{x}=\sum x_i$ then we have:

$$\mathsf{Pr}[|\bar{x}-\mathbb{E}[\bar{x}]|>y] \le 2\exp\Big(-\frac{y^2/2}{V+Zy/3}\Big)$$

where $V=\sum \mathbb{E}[(x_i-\mathbb{E}[x_i])^2]$ and $Z=\max\{b_i-a_i\}$.

To prove the lemma, we use Bernstein's inequality in the following way: for every item we put a variable x_i which identifies whether item (s_i,v_i) is selected in our set. If so, we set $x_i=s_i$, otherwise we set $x_i=0$. As such, the value of every variable x_i is in range $[0,s_i]$ and thus $a_i=0$ and $b_i=s_i$ for all $1\le i\le n$. This way we have

$$\mathbb{E}[\bar{x}]=\sum \mathbb{E}[x_i]=\sum ps_i = p(\sum s_i)=pS.$$

Moreover, $s_i \le \mathsf{s}_{\max}$ holds for all i and for each x_i we have $\mathbb{E}[(x_i-\mathbb{E}[x_i])^2]=p(1-p)s_i^2 \le p(1-p)\mathsf{s}_{\max}s_i$. Thus,

$$V=\sum \mathbb{E}[(x_i-\mathbb{E}[x_i])^2] \le \sum p(1-p)\mathsf{s}_{\max}s_i = (\sum s_i)p(1-p)\mathsf{s}_{\max} = p(1-p)\mathsf{s}_{\max}S.$$

By replacing $\mathbb{E}[\bar{x}]$ by pS, Z by $\mathsf{s}_{\max}$, and V by $p(1-p)\mathsf{s}_{\max}S$ we get

$$\mathsf{Pr}[|\bar{x}-pS]|>y] \le 2\exp\Big(-\frac{y^2/2}{p(1-p)\mathsf{s}_{\max}S+\mathsf{s}_{\max}y/3}\Big).$$

We set $\mathsf{C}=40\log n$ and $y=\mathsf{C}\sqrt{\mathsf{s}_{\max}pS}$ to bound the probability that $|\sum x_i - pS| > \mathsf{C}\sqrt{\mathsf{s}_{\max}pS}$ happens. Thus we obtain

$$\mathsf{Pr}[|\bar{x}-pS]|>\mathsf{C}\sqrt{\mathsf{s}_{\max}pS}] \le 2\exp\Big(-\frac{\mathsf{C}^2\mathsf{s}_{\max}pS/2}{p(1-p)\mathsf{s}_{\max}S+\mathsf{C}\mathsf{s}_{\max}\sqrt{\mathsf{s}_{\max}pS}/3}\Big).$$

Since $1-p \leq 1$ we have

$$\frac{\mathsf{C}^2\mathsf{s}_{\max}pS/2}{p(1-p)\mathsf{s}_{\max}S} = \frac{\mathsf{C}^2/2}{(1-p)} \geq \mathsf{C}^2/2. \tag{1}$$

Moreover, by the assumption of the lemma $p \leq 1/2$ holds. In addition to this, $\mathsf{s}_{\max} \leq 2pS$ and therefore

$$\frac{\mathsf{C}^2\mathsf{s}_{\max}pS/2}{\mathsf{C}\mathsf{s}_{\max}\sqrt{\mathsf{s}_{\max}pS}/3} = \frac{\mathsf{C}pS/2}{\sqrt{\mathsf{s}_{\max}pS}/3} = \frac{\mathsf{C}\sqrt{pS}/2}{\sqrt{\mathsf{s}_{\max}}/3} = 3\frac{\mathsf{C}\sqrt{pS}/2}{\sqrt{\mathsf{s}_{\max}}} \geq 3\frac{\mathsf{C}\sqrt{pS}/2}{\sqrt{2pS}} = 3\frac{\mathsf{C}/2}{\sqrt{2}} \geq 3\mathsf{C}/(2\sqrt{2}). \tag{2}$$

It follows from Inequalities (1) and (2) that

$$\begin{aligned}\frac{\mathsf{C}^2\mathsf{s}_{\max}pS/2}{p(1-p)\mathsf{s}_{\max}S + \mathsf{C}\mathsf{s}_{\max}\sqrt{\mathsf{s}_{\max}pS}/3} &= \frac{\mathsf{C}^2\mathsf{s}_{\max}pS/2}{\left[p(1-p)\mathsf{s}_{\max}S\right] + \left[\mathsf{C}\mathsf{s}_{\max}\sqrt{\mathsf{s}_{\max}pS}/3\right]}\\ &\geq \min\{\frac{\mathsf{C}^2\mathsf{s}_{\max}pS/2}{p(1-p)\mathsf{s}_{\max}S}, \frac{\mathsf{C}^2\mathsf{s}_{\max}pS/2}{\mathsf{C}\mathsf{s}_{\max}\sqrt{\mathsf{s}_{\max}pS}/3}\}/2\\ &\geq \min\{\mathsf{C}^2/2, \frac{3\mathsf{C}}{2\sqrt{2}}\} \geq \frac{3\mathsf{C}}{4\sqrt{2}}\\ &\geq \mathsf{C}/2\\ &= 20\log n.\end{aligned}$$

This implies that $\exp(-\frac{\mathsf{C}^2\mathsf{s}_{\max}pS/2}{p(1-p)\mathsf{s}_{\max}S+\mathsf{C}\mathsf{s}_{\max}\sqrt{\mathsf{s}_{\max}pS}/3}) \leq \exp(-20\log n) \leq n^{-10}$ and thus $|pS - S'| \leq \mathsf{C}\sqrt{\mathsf{s}_{\max}pS}$ holds w.h.p. □

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 $\mathsf{s}_{\max}$, 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 $[t-\mathsf{s}_{\max}+1, t]$. 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 $[t-\mathsf{s}_{\max}+1, t]$.

Now, if we randomly distribute the items into $t/\mathsf{s}_{\max}$ buckets then the expected size of the solution in each bucket is $O(\mathsf{s}_{\max})$ and thus we expect the size of the solution in each bucket to be in range $[0, \widetilde{O}(\mathsf{s}_{\max})]$ w.h.p. due to Lemma C.1. Therefore, it suffices to compute the solution for each bucket up to a size of $\widetilde{O}(\mathsf{s}_{\max})$. Next, we use Lemma C.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 $[t|S_1|/n - \widetilde{O}(\sqrt{t\mathsf{s}_{\max}|S_1|/n}), t|S_1|/n + \widetilde{O}(\sqrt{t\mathsf{s}_{\max}|S_1|/n})]$ and $[t|S_2|/n - \widetilde{O}(\sqrt{t\mathsf{s}_{\max}|S_2|/n}), t|S_2|/n + \widetilde{O}(\sqrt{t\mathsf{s}_{\max}|S_2|/n})]$ 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 $\widetilde{O}(\sqrt{t\mathsf{s}_{\max}(|S_1|+|S_2|)/n}^2) =$

$\widetilde{O}(t\mathsf{s}_{\max}(|S_1|+|S_2|)/n)$. This enables us to compute the solution w.h.p. in time $\widetilde{O}(\mathsf{s}_{\max}(n+t))$.

Algorithm 4: KnapsackForSmallSizes

```
  Data: Knapsack size $t$ and $n$ items $(s_i, v_i)$ where $1 \le s_i \le \mathsf{s}_{\max}$ for all items.
  Result: Solution for knapsack size $t$
1 Randomly distribute the items into $t/\mathsf{s}_{\max}$ buckets;
2 for $j \in [t/\mathsf{s}_{\max}]$ do
3     $x_{1,j}$ = solution of the problem for bucket $i$ up to size $(\mathsf{C}+2)\mathsf{s}_{\max}$;
4 for $i \in [2, \lceil \log(t/\mathsf{s}_{\max}) \rceil]$ do
5     for $j \in \lceil t/\mathsf{s}_{\max}/2^i \rceil$ do
6         Combine the solutions of $x_{i-1,2j-1}$ and $x_{i-1,2j}$ into $x_{i,j}$ (based on Lemma C.1 );
7 Return  $\max x_{\lceil \log(t/\mathsf{s}_{\max}) \rceil, 1}$;
```

Theorem C.2. *There exists a randomized algorithm that correctly computes the solution of the knapsack problem in time $\widetilde{O}(\mathsf{s}_{\max}(n+t))$ w.h.p., if the item sizes are integers in range $[1, \mathsf{s}_{\max}]$.*

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

We merge the solutions in $\log(t/\mathsf{s}_{\max})$ rounds. In every round i, we make $t/\mathsf{s}_{\max}/2^i$ merges each corresponding to the solutions of 2^i buckets. By Lemma C.1, the range of the solution size in every merge is $[\mathsf{s}_{\max}2^i - \widetilde{O}(\sqrt{\mathsf{s}_{\max}{}^2 2^i}), \mathsf{s}_{\max}2^i + \widetilde{O}(\sqrt{\mathsf{s}_{\max}{}^2 2^i})]$ w.h.p. Thus, every merge takes time $\mathsf{s}_{\max}{}^2 2^i$. Moreover, in every round i the number of merges is $t/\mathsf{s}_{\max}/2^i$. Therefore, the total running time of each phase is $\widetilde{O}(\mathsf{s}_{\max} t)$ and thus the algorithm runs in time $\widetilde{O}(\mathsf{s}_{\max}(n+t))$. 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 C.2, we can also solve the unbounded knapsack problem in time $\widetilde{O}(\mathsf{s}_{\max}(n+t))$ if the sizes of the items are bounded by $\mathsf{s}_{\max}$.

Corollary C.3 (of Theorem C.2)**.** *There exists a randomized $\widetilde{O}(\mathsf{s}_{\max}(n+t))$ time algorithm that solves the unbounded knapsack problem w.h.p. when the sizes are bounded by $\mathsf{s}_{\max}$.*

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 $\mathsf{s}_{\max}$ 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 $O(t \log \mathsf{s}_{\max})$ items. Using the algorithm of Theorem C.2 we can solve this problem in time $\widetilde{O}(\mathsf{s}_{\max} t)$. Since the reduction takes time $O(n)$ the total running time is $\widetilde{O}(\mathsf{s}_{\max}(n+t))$. ☐

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

D Strongly Polynomial Time 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 $\mathsf{s}_{\max}$. 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. We show that when all the sizes are integers bounded by $\mathsf{s}_{\max}$, one can solve the problem in time $\widetilde{O}(n\mathsf{s}_{\max}{}^2 \min\{n, \mathsf{s}_{\max}\})$. Notice that this running time is independent of t and thus our algorithm runs in strongly polynomial time. This result improves upon the $O(n^3\mathsf{s}_{\max}{}^2)$ time algorithm of [22].

We begin, as a warm-up, by considering the case where $m_i = \infty$ for all items. We show that in this case, the $O(n^2\mathsf{s}_{\max}{}^2)$ time algorithm of [22] can be improved to an $\widetilde{O}(n\mathsf{s}_{\max}+\mathsf{s}_{\max}{}^2 \min\{n, \mathsf{s}_{\max}\})$ time algorithm. Before we explain our algorithm, we state a mathematical lemma that will be later used in our proofs.

Lemma D.1. *Let S be a subset of items with integer sizes. If $|S| \geq k$ 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 $s_i \mathsf{~Mod~} k = s_j \mathsf{~Mod~} k$. This means that the total size of the items in positions $i + 1$ to j is divisible by k. $\square$

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 $\mathsf{s}_{\max}{}^2$.

Lemma D.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 D.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 $(s_\mathsf{H}, v_\mathsf{H})$ 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 $(s_\mathsf{H}, v_\mathsf{H})$ with the same total size. Since $v_\mathsf{H}/s_\mathsf{H}$ is the highest ratio over all items, the objective value of the solution doesn't hurt, and thus it remains optimal. $\square$

Since $s_i \leq \mathsf{s}_{\max}$ holds for all items, Lemma D.2 implies that in such a solution, the total size of all items except $(s_\mathsf{H}, v_\mathsf{H})$ is bounded by $\mathsf{s}_{\max}{}^2$. This implies that at least $\max\{0, \lfloor(t - \mathsf{s}_{\max}{}^2)/s_\mathsf{H}\rfloor\}$ copies of item $(s_\mathsf{H}, v_\mathsf{H})$ 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 $\mathsf{s}_{\max}{}^2 + \mathsf{s}_{\max}$. Therefore, the classic $O(nt')$ time algorithm for knapsack finds the solution in time $O(n\mathsf{s}_{\max}{}^2)$. Also, by Theorem C.2, one can solve the problem in time $\widetilde{O}((n + t')\mathsf{s}_{\max}) = \widetilde{O}(n\mathsf{s}_{\max} + \mathsf{s}_{\max}{}^3)$. Thus, the better of two algorithms runs in time $\widetilde{O}(n\mathsf{s}_{\max} + \mathsf{s}_{\max}{}^2 \min\{n, \mathsf{s}_{\max}\})$. This procedure is shown in Algorithm 5.

Theorem D.3. *When $s_i \in [\mathsf{s}_{\max}]$ and $m_i = \infty$ hold for every item, Algorithm 5 computes the solution of the knapsack problem in time $\widetilde{O}(n\mathsf{s}_{\max} + \mathsf{s}_{\max}{}^2 \min\{n, \mathsf{s}_{\max}\})$.*

Algorithm 5: KnapsackWithInfiniteMultiplicities

```
Data: A knapsack size $t$ and $n$ items with sizes and values $(s_i, v_i)$. $m_i = \infty$ and $s_i \leq \mathsf{s}_{\max}$
      hold for all $1 \leq i \leq n$
Result: The solution of the knapsack problem for knapsack size $t$
1 $\mathsf{H} \leftarrow \arg\max v_i/s_i$;
2 $\mathsf{cnt} \leftarrow \max\{0, \lfloor (t - {\mathsf{s}_{\max}}^2)/s_{\mathsf{H}}\} \rfloor$;
3 $t' \leftarrow t - \mathsf{cnt} \cdot s_{\mathsf{H}}$;
4 if $n \leq \mathsf{s}_{\max}$ then
5     Report $\mathsf{cnt} \cdot v_{\mathsf{H}} + \mathsf{ClassicKnapsack}(t', n, \{(s_1,t_1),(s_2,t_2),\ldots,(s_n,t_n)\}, \{m_1,m_2,\ldots,m_n\})$;
6 else
7     Report
      $\mathsf{cnt} \cdot v_{\mathsf{H}} + \mathsf{KnapsackForSmallSizes}(t', n, \{(s_1,t_1),(s_2,t_2),\ldots,(s_n,t_n)\}, \{m_1,m_2,\ldots,m_n\})$;
```

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

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

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

We define $t' = t - {\mathsf{s}_{\max}}^2$ to be a smaller knapsack size which is less than t by an additive factor of ${\mathsf{s}_{\max}}^2$. 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 $b_i \geq \mathsf{s}_{\max}$ holds for some item (s_i, v_i), then at least $b_i - \mathsf{s}_{\max}$ copies of (s_i, v_i) appear in this solution.

Lemma D.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 $b_i - \mathsf{s}_{\max}$ copies of each item (s_i, t_i) such that $b_i \geq \mathsf{s}_{\max}$.*

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 $a_i < b_i - \mathsf{s}_{\max}$. 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, $a_j \leq m_j = b_j$ holds for all $j \leq i$. 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 $t' = t - {\mathsf{s}_{\max}}^2$ and since $a_j \leq b_j$ for all $j \leq i$, then the first i items contribute to at most $t - {\mathsf{s}_{\max}}^2 - \mathsf{s}_{\max} s_i$ space units of the solution. Moreover, as we discussed above, the total size of the solution is at least $t - \mathsf{s}_{\max}$

and thus the rest of the items have a size of at least ${\mathsf{s}_{\max}}^2$ in our optimal solution. Therefore we have

$$\sum_{j=i+1}^{n} a_j s_j \geq {\mathsf{s}_{\max}}^2$$

and since $s_j \leq \mathsf{s}_{\max}$ holds, we have $\sum_{j=i+1}^{n} a_j \geq \mathsf{s}_{\max} \geq s_i$. Based on Lemma D.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 $\mathsf{s}_{\max}$) 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 D.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 $t' - \mathsf{s}_{\max} = t - {\mathsf{s}_{\max}}^2 - \mathsf{s}_{\max}$. Based on Lemma D.4, for any item with $b_i \geq \mathsf{s}_{\max}$ we know that at least $b_i - \mathsf{s}_{\max}$ copies of this item appear in an optimal solution of our problem. Therefore, we can decrease the multiplicity of such items by $b_i - \mathsf{s}_{\max}$ and decrease the knapsack size by $(b_i - \mathsf{s}_{\max})s_i$. We argue that after such modifications, the remaining size of the knapsack is at most $\mathsf{s}_{\max} + {\mathsf{s}_{\max}}^2 + n{\mathsf{s}_{\max}}^2$. Recall that the total size of the pseudo solution is at least $t - {\mathsf{s}_{\max}}^2 - \mathsf{s}_{\max}$ and therefore $\sum b_i s_i \geq t - {\mathsf{s}_{\max}}^2 - \mathsf{s}_{\max}$. This implies that

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

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

Theorem D.5. *Algorithm 6 solves the knapsack problem in time $\widetilde{O}(n{\mathsf{s}_{\max}}^2 \min\{n, \mathsf{s}_{\max}\})$ when the sizes of the items are integers in range $[1, \mathsf{s}_{\max}]$ and each item has a given integer multiplicity.*

Proof. The proof is based on Lemma D.4. After determining the values of vector b', we know that for each item (s_i, t_i) at least $b_i - \mathsf{s}_{\max}$ 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 $\mathsf{s}_{\max} + (n+1){\mathsf{s}_{\max}}^2$ and thus the better of the classic knapsack algorithm and the algorithm of Section C solve the problem in time $\widetilde{O}(n{\mathsf{s}_{\max}}^2 \min\{n, \mathsf{s}_{\max}\})$. □

Algorithm 6: KnapsackWithGivenMultiplicities

```
Data: A knapsack size $t$ and $n$ items with sizes and values $(s_i, v_i)$. $n$ multiplicities
      $m_1, m_2, \ldots, m_n$. $s_i \leq \mathsf{s}_{\max}$ holds for all $1 \leq i \leq n$
Result: The solution of the knapsack problem for knapsack size $t$
1  $t' \leftarrow \max\{0, t - \mathsf{s}_{\max}{}^2\}$;
2  for $i \in [1, n]$ do
3      $b_i \leftarrow \min\{m_i, \lfloor t'/s_i \rfloor\}$;
4      $t' \leftarrow t' - b_i s_i$;
5      if $b_i \neq m_i$ then
6          break;
7  $t'' \leftarrow t$;
8  surplus $\leftarrow 0$;
9  for $i \in [1, n]$ do
10     $t'' \leftarrow t'' - \max\{0, b_i - \mathsf{s}_{\max}\} s_i$;
11     $m'_i \leftarrow m_i - \max\{0, b_i - \mathsf{s}_{\max}\}$ ;
12     surplus $\leftarrow$ surplus $+ \max\{0, b_i - \mathsf{s}_{\max}\} v_i$;
13 if $n \leq \mathsf{s}_{\max}$ then
14     Report surplus + ClassicKnapsack($t'', n, \{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}, \{m'_1, m'_2, \ldots, m'_n\}$);
15 else
16     Report
       surplus + KnapsackForSmallSizes($t'', n, \{(s_1, t_1), (s_2, t_2), \ldots, (s_n, t_n)\}, \{m'_1, m'_2, \ldots, m'_n\}$);
```

E Related Convolution Problems

In Sections G and F we discuss other convolution type problems that are similar to knapsack. We mentioned in the introduction that several problems seem to be closely related to the knapsack and convolution problems. We define and mention some previous results for some of those problems here. The tree sparsity problem asks for a maximum-value subtree of size k from a given node-valued tree. The best-known algorithm for the problem runs in $O(kn)$ time, which is quadratic for $k = \Theta(n)$. Backurs *et al.* [1] show that it is unlikely to obtain a strongly subquadratic-time algorithm for this problem, since it implies the same for the $(\min, +)$ convolution problem. They provide the first single-criterion $(1+\epsilon)$-approximation for tree sparsity that runs in near-linear time, and works on arbitrary trees. Given a set of integers and a target, the subset sum problem looks for a subset whose sum matches the target. To find all the *realizable* integers up to u takes time $\tilde{O}(\min\{\sqrt{n}u, u^{4/3}, \sigma\})$, where σ is the sum of the given input integers [16], improving upon the simple $O(nu)$ dynamic-programming solution [2]. Finding out whether a specific t is realizable may be done in $\tilde{O}(n+t)$ randomized time, matching certain conditional lower bounds [4]. The least-value sequence problem is studied in Künnemann *et al.* [17]: given a sequence of n items and a (perhaps succinctly represented) not necessarily positive value function for every pair, find a subsequence that minimizes the sum of values of adjacent pairs. Several problems such as longest chain of nested boxes, vector domination, and a coin change problem fit in this category and are considered. For each of these, the authors identify "core" problems, which help to either demonstrate hardness or design fast algorithms. The fastest algorithms for language edit distance is based on computing the $(\min, +)$ product of two $n \times n$ matrices, which also solves all pairs shortest paths. Bringmann *et al.* [5] show that the matrix product can be computed in subcubic time if one matrix has bounded differences in either rows or columns.

While minimum convolution[5] admits a near linear-time $(1+\epsilon)$-approximation [1, 7], we do not know of a strongly subquadratic-time exact algorithm for it. The best-known algorithm runs in time $O(n^2(\log\log n)^3/\log^2 n)$ [3]. Some special cases have faster algorithms, though: $O(n)$ time for convex sequences, and $O(n\log n)$ time for randomly permuted sequences [6, 18]. Moreover additive combinatorics allows us to solve the convolution problem for increasing integers bounded by $O(n)$ in randomized time $O(n^{1.859})$ and deterministic $O(n^{1.864})$ [8].

Cygan *et al.* [11] study minimum convolution as a hardness assumption, and identify several problems that are as hard. First of all, minimum convolution is known to reduce to either three-sum or all pairs shortest paths problem, though no reduction in the other direction is known, and the relation of the latter two is not known. (The three-sum problem asks whether three elements of a given set of n numbers sum to zero.) Despite the recent progress on the subset sum problem, which is a special case of the 0/1 knapsack problem, the latter is shown to be equivalent to minimum convolution. (The former reduces to $(\vee,\wedge)$ convolution that can be solved via FFT.) A similar reduction exists for the unbounded knapsack problem.

[5]It is also called $(\min,+)$ convolution, min-sum convolution, inf-convolution, infimal convolution or the epigraphical sum in the literature.

F Tree Separability

0/1 knapsack, unbounded knapsack, and tree sparsity along with a few other combinatorial optimization problems have been shown to be computationally equivalent with respect to subquadratic algorithms [11]. In other words, a subquadratic algorithm for any problem in this list yields a subquadratic algorithm for the rest of the problems. In this section, we introduce the tree separability problem and show that this problem is indeed computationally equivalent to the rest of the problems of the list. Next, in Section F.2, we show that in some cases, a bounded weight tree separability problem can be solved in better than subquadratic time. This result in nature is similar to the algorithms we provide for knapsack problems.

In the tree separability problem, we are given a tree T with n nodes and $n-1$ edges. Every edge $e=(i,j)$ is associated with a weight w_e. The goal of this problem is to partition the vertices of T into two (not necessarily connected) partitions of size m and $n-m$ in a way that the total weight of the crossing edges is minimized. A special case of the problem where $|m-(n-m)| \leq 1$ is known as tree bisection.

F.1 Equivalence with $(\max,+)$ Convolution

To show a subquadratic equivalence, we first present an indirect reduction from $(\max,+)$ convolution to tree separability. We use the *MaxCov-UpperBound* as an intermediary problem in our reduction. Cygan *et al.* [11] show that any subquadratic algorithm for MaxCov-UpperBound yields a subquadratic solution for $(\max,+)$ convolution.

Definition F.1. *In the MaxCov-UpperBound problem, we are given two vectors a and b of size n and a vector c of size $2n-1$. The goal is to find out whether there exists an i such that $(a \star b)_i > c_i$.*

Lemma F.2 (proven in [11])**.** *Any subquadratic solution for MaxCov-UpperBound yields a subquadratic solution for the $(\max,+)$ convolution.*

The main idea of our reduction is as follows: Given three vectors a, b, and c with sizes n, n, and $2n-1$ one can construct a tree consisted of three paths joining at a vertex r. We show that based on the solution of the tree separability on this tree, one can determine if $(a \star b)_i \geq c_i$ for some $0 \leq i < 2n-1$.

Lemma F.3. *Any subquadratic algorithm for tree separability results in a subquadratic algorithm for the $(\max,+)$ convolution.*

Proof. As we mentioned earlier, we prove this reduction through MaxCov-UpperBound. Suppose we are given two vectors a and b of size n and a vector c of size $2n-1$ and are asked if $(a \star b)_i > c_i$ for some $0 \leq i < |c|$. We answer this question by constructing a tree of size $8n$ as follows: Let $M = 10\max\{1, |a_0|, |a_1|, \ldots, |a_{n-1}|, |b_0|, |b_1|, \ldots, |b_{n-1}|\}$ be a large enough number. The root of the tree is a vertex r and three paths are connected to vertex r. The vertices of each path correspond to the elements of one vector. Thus, we denote the vertices of the paths by a'_i, b'_i, and c'_i respectively. For every c'_i we set the weight of the edge between c'_i and c'_{i-1} (or r in case of $i=0$) equal to $M + c_{2n-2-i}$. Similar to this, for every $0 \leq i < n$, we set the weight of the edge between a'_{i+n} and a'_{i+n-1} equal to $M - a_i$ and the weight of the edge between b'_{i+n} and b'_{i+n-1} equal to $M - b_i$. The rest of the edges have weight ∞. This construction is illustrated in Figure 2. Our claim is that for $m = 4n-1$ the solution of the tree separability problem is at least $3M$ if and only if $a \star b$ is bounded by c.

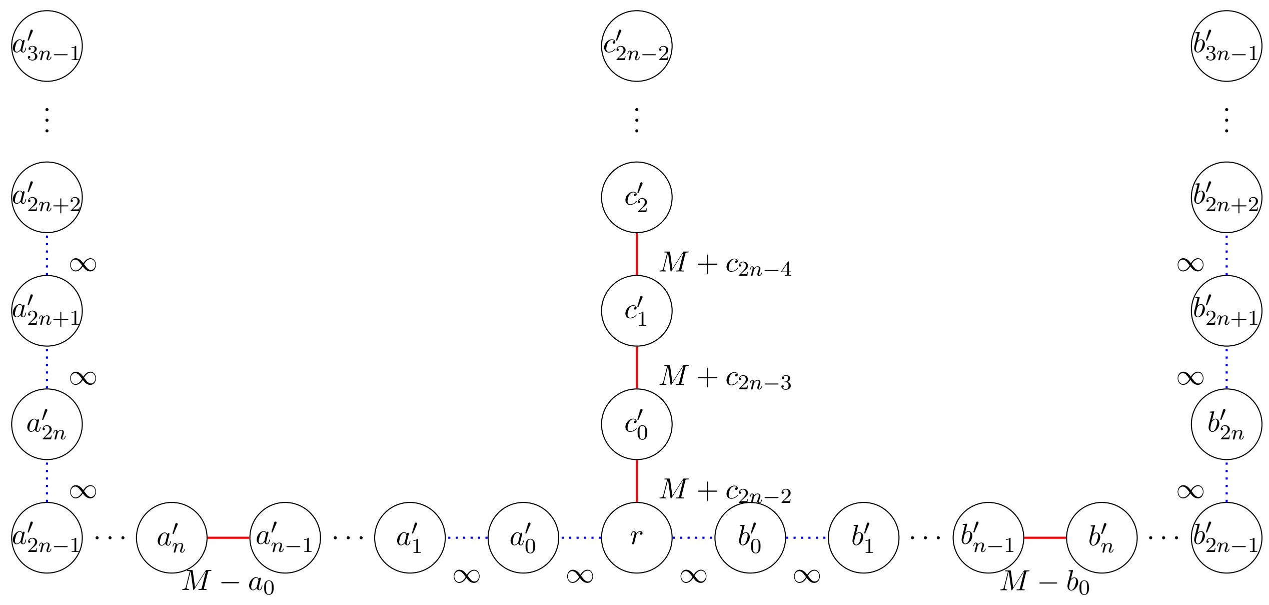

Figure 2: Dashed edges have weight ∞ while the weight of the solid edges is based on the value of the vectors a, b, and c.

We first prove that if for some a_i and b_j we have $a_i + b_j = (a \star b)_{i+j} > c_{i+j}$ then the solution of the tree separability problem for $m = 4n - 1$ is smaller than $3M$. To this end, we put the following vertices in one partition and the rest of the vertices in the second partition:

$$\{r, a'_0, a_1, \ldots, a'_{n+i-1}, b'_0, b_1, \ldots, b'_{n+j-1}, c'_0, c'_1, \ldots, c'_{2n-i-j-3}\}.$$

Notice that the above list contains exactly $4n - 1$ vertices. Moreover, the only crossing edges of this solution are the ones connected to a'_{n+i}, b'_{n+j}, and $c'_{2n-i-j-2}$ with weights $M - a_i$, $M - b_j$, and $M + c_{i+j}$. Since $a_i + b_j > c_{i+j}$ we have $M - a_i + M - b_j + M + c_{i+j} < 3M$ and thus the solution of the tree separability problem is smaller than $3M$.

Finally, we show that if $a_i + b_j \geq c_{i+j}$ holds for all i and j, then the solution of the tree separability problem is at least $3M$. Notice that since M is large enough, in order for a solution to have a weight smaller than $3M$ it has to meet the following constraints:

- The solution should not contain an edge with weight ∞.
- The number of crossing edges between the partitions should be at most 3.

In any solution that meets the above constraints, the partition with size $4n - 1$ contains vertex r. Moreover, none of the crossing edges in parts a' and b' have weight ∞ and thus the crossing edges correspond to two vertices a'_{n+i} and b'_{n+j} with $0 \leq i, j < n$ and therefore their weights are a_i and b_j. Since the size of the partition is $4n - 1$, the third crossing edge has a weight of c_{i+j}. Recall that we assume $c_{i+j} \geq a_i + b_j$ and therefore $M - a_i + M - b_j + M + c_{i+j} \geq 3M$.

□

We also show in Appendix M that a $T(n)$ time algorithm for computing the $(\max, +)$ convolution of two vectors yields an $\widetilde{O}(T(n))$ time algorithm for solving tree separability. The proof is very similar to the works of Cygan *et al.* [11] and Backurs *et al.* [1]. If the height of the tree is small, the classic dynamic program yields a running time of $T(n)$. We use the spine decomposition of [21] to deal with cases where the height of the tree is large.

Lemma F.4. *Any $T(n)$ time algorithm for solving $(\max,+)$ convolution yields an $\widetilde{O}(T(n))$ time algorithm for tree separability.*

Lemmas F.3 and F.4 imply that $(\max,+)$ convolution and tree separability are computationally equivalent.

Theorem F.5 (A corollary of Lemmas F.3 and F.4)**.** *$(\max,+)$ convolution and tree separability are computationally equivalent with respect to subquadratic algorithms.*

F.2 Fast Algorithm for Special Cases

We show that when the maximum degree of the tree and the weights of the edges are bounded by $\mathsf{d}_{\max}$ and $\mathsf{w}_{\max}$, one can solve the problem in time $\widetilde{O}(\mathsf{d}_{\max}\mathsf{w}_{\max}n)$. Note that our algorithm only works when the edge weights are integers. In particular, when both $\mathsf{w}_{\max}$ and $\mathsf{d}_{\max}$ are $O(1)$ our algorithm runs in (almost) linear time.

Our main observation is the following: for any $1 \le m < n$, there exists a partitioning of a given tree T into two partitions of sizes m and $n-m$ such that the number of crossing edges is bounded by $2\mathsf{d}_{\max}\log n$.

Lemma F.6. *Let T be a tree of size n and $1 \le m < n$ be an integer number. There exists a partitioning of T into two partitions of sizes m and $n-m$ with at most $2\mathsf{d}_{\max}\log n$ crossing edges where $\mathsf{d}_{\max}$ is the maximum degree of a vertex in T.*

Proof. It is a well-known fact that every tree of size n has a vertex v such that if we remove v from the tree, the size of each connected component of the tree is bounded by $(2/3)n$ [24]. Based on this observation, we inductively construct a solution with no more than $2\mathsf{d}_{\max}\log n$ crossing edges for any tree with n vertices and a given partition size m. The base case is $n \le 2$ for which the lemma holds trivially. Now, for a given tree T of size n, we find its center v with the above property. Next, we remove v from the tree to obtain d_v connected components $T_1, T_2, \ldots, T_{d_v}$. To construct a solution, we first put vertex v in the partition that is to be of size m. We then continue growing this partition by adding the subtrees to it one by one. We stop when adding any subtree to the solution increases the size of the partition to more than m. Let m' be the size of the partition and T_i be the first subtree that cannot be entirely added to the solution. If $m' = m$ our solution is valid, otherwise we recursively partition T_i into two partitions of sizes $m - m'$ and $|T_i| - (m - m')$ and update the solution by adding the $m - m'$ part to it. Note that we have at most $\mathsf{d}_{\max}$ crossing edges for vertex v and also based on the induction hypothesis, the number of crossing edges in partitions of T_i is at most $2\mathsf{d}_{\max}\log|T_i|$. In addition to this, since v is a center of the tree, we have $|T_i| \le n2/3$ and therefore $2\mathsf{d}_{\max}\log|T_i| \le (2\log n - 1)\mathsf{d}_{\max}$. Thus, the total number of crossing edges in our solution is bounded by $2\mathsf{d}_{\max}\log n$. $\square$

It follows from Lemma F.6 that when the maximum degree of a tree is bounded by $\mathsf{d}_{\max}$ and the maximum weight of the edges is bounded by $\mathsf{w}_{\max}$ then the solution of the tree separability problem is bounded by $2\mathsf{d}_{\max}\mathsf{w}_{\max}\log n$. Therefore, the values of the solutions for every subproblem of the tree separability problem are bounded by $2\mathsf{d}_{\max}\mathsf{w}_{\max}\log n$. Thus, every time we wish to compute the convolution of two vectors a and b corresponding to the solutions of the subproblems, the $(\max,+)$ convolution can be computed in time $\widetilde{O}(\mathsf{d}_{\max}\mathsf{w}_{\max}(|a|+|b|))$ (Lemma H.1). Thus, based on Lemma F.4, we can compute the solution of the tree separability problem in time $\widetilde{O}(\mathsf{d}_{\max}\mathsf{w}_{\max}n)$.

Theorem F.7 (A corollary of Lemmas F.6 and F.4)**.** *Given a tree T with n nodes whose maximum degree is bounded by $\mathsf{d}_{\max}$. If the weights of the edges are integers bounded by $\mathsf{w}_{\max}$, one can compute the solution of the tree separability problem for T in time $\widetilde{O}(\mathsf{d}_{\max}\mathsf{w}_{\max}n)$.*

G 0/1 Tree Sparsity

We show in Sections 3, A, and F that convolution, knapsack, and tree separability problems can be solved in almost linear time in special cases. One of the important problems that lies in the same computational category with these problems is the tree sparsity problem. Therefore, an important question that remains open is whether an almost linear time algorithm can solve the tree sparsity problem when the weights of the vertices are small integers. In particular, is it possible to solve the 0/1 tree sparsity problem (in which the weight of every vertex is either 0 or 1) in $\widetilde{O}(n)$ time? We show in this section that tree sparsity is the hardest problem of this category when it comes to small weights. More precisely, we show that an $\widetilde{O}(n)$ time algorithm for 0/1 tree sparsity immediately implies linear time solutions for the rest of the problems when the input values are small. We assume that the goal of the tree sparsity problem is to find for every i what is the weight of the heaviest connected component of the tree of size i.

To this end, we define the $\mathsf{d}_{\max}$-distance bounded $(\max,+)$ convolution problem as follows: given two vectors a and b with the condition that $\max|a_i - a_{i-1}| \leq \mathsf{d}_{\max}$ and $\max|b_i - b_{i-1}| \leq \mathsf{d}_{\max}$ hold for every i. The goal is to compute $a \star b$. We show that a $T(n)$ time algorithm for 0/1 tree sparsity yields an $\widetilde{O}(T(\mathsf{d}_{\max}n))$ time algorithm for $\mathsf{d}_{\max}$-distance bounded $(\max,+)$ convolution of two integer vectors a and b where $n = |a|+|b|$. This yields fast algorithms for convolution, knapsack, tree separability, and tree separability when the input values are small integers. Indeed we already know that convolution and knapsack problems admit almost linear time algorithms for such special cases, nonetheless such a reduction sheds light on the connection between these problems.

We begin by showing that a $T(n)$ time algorithm for 0/1 tree sparsity yields an $\widetilde{O}(T(\mathsf{d}_{\max}n))$ time algorithm for $\mathsf{d}_{\max}$-distance bounded convolution.

Lemma G.1. *Given a $T(n)$ time algorithm for 0/1 tree sparsity, one can solve the $\mathsf{d}_{\max}$-distance bounded convolution for integer vectors in time $O(T(\mathsf{d}_{\max}n))$.*

Proof. Suppose we are given two $\mathsf{d}_{\max}$-distance bounded vectors a and b and wish to compute $a \star b$. We assume w.l.o.g. that both a and b are of size n. Also, we can assume w.l.o.g. that both vectors are increasing because of the following fact: If we add $i(\mathsf{d}_{\max}+1)$ to every element i of both vectors a and b, they both become increasing since $|a_i - a_{i-1}| \leq \mathsf{d}_{\max}$ and $|b_i - b_{i-1}| \leq \mathsf{d}_{\max}$ hold for the original vectors. Moreover, if we compute $c = a \star b$ for the new vectors, one can compute the solution for the convolution of the original vectors by just subtracting $i(\mathsf{d}_{\max}+1)$ from every element i of vector c. In addition to this, since the original vectors are $\mathsf{d}_{\max}$-distance bounded, after adding $i(\mathsf{d}_{\max}+1)$ to every element i of the vectors, the resulting vectors are $(2\mathsf{d}_{\max}+1)$-distance bounded.

For the rest of the proof, we assume both vectors a and b are increasing and of size n. Moreover, both vectors are $\mathsf{d}_{\max}$-distance bounded and thus both $|a_i - a_{i-1}| \leq \mathsf{d}_{\max}$ and $|b_i - b_{i-1}| \leq \mathsf{d}_{\max}$ hold. We also assume $a_0 = b_0 = 0$ since one can ensure that constraint by shifting the values. To compute $a \star b$, we construct an instance of the 0/1 tree sparsity problem with $n+1+a_{n-1}+b_{n-1}$ vertices. The underlying tree has a root r connected to three different paths a', b', c'. Path c' contains n consecutive vertices, each with weight 1. Paths a' and b' correspond to vectors a and b. Vertices of path a' are denoted by $a'_1, a'_2, \ldots, a'_{a_{n-1}}$. The weight of each a'_i is 1 if and only if there exists a j such that $a_j = i$. Similar to this, the vertices of path b' are denoted by $b'_1, b_2, \ldots, b'_{b_{n-1}}$ and the weight of a vertex b'_i is equal to 1 if and only if $b_j = i$ for some j. An example of such construction is shown in Figure 3.

Let s be the solution vector to the tree sparsity problem explained above. In other words, s is a vector of size $n+2+a_{n-1}+b_{n-1}$ where s_i is the heaviest connected subtree of size i. We argue that for every $0 \leq i < |(a \star b)|$, $(a \star b)_i$ is equal to $j-n-1$ for the smallest j such that $s_j = n+i+1$. If

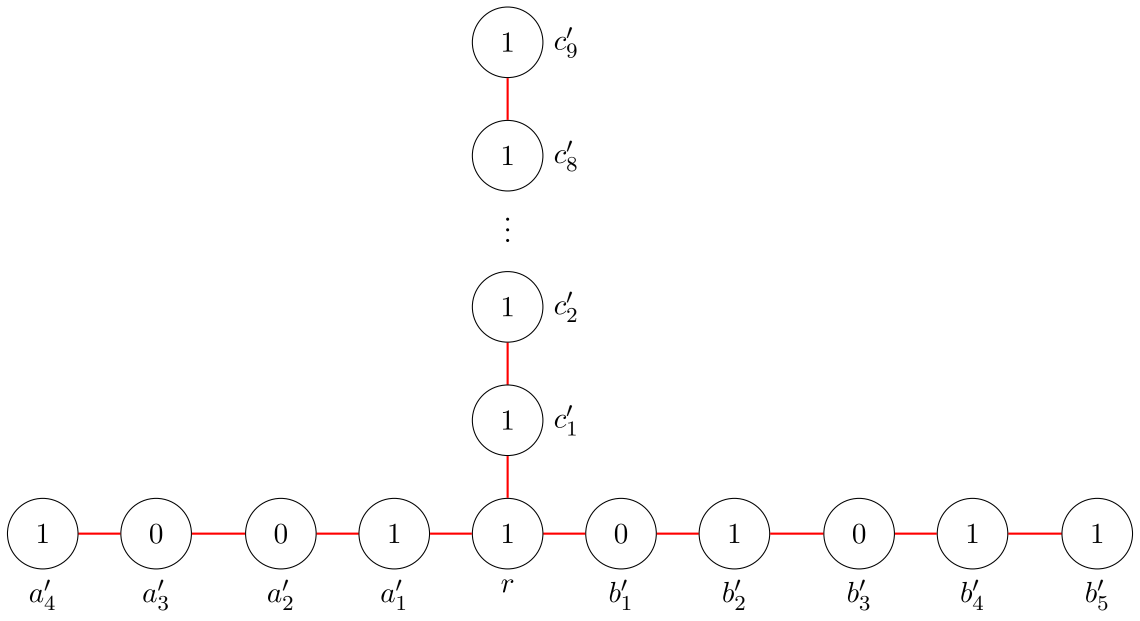

Figure 3: The above tree corresponds to vectors $a = \langle 0, 1, 3, 4\rangle$ and $b = \langle 0, 2, 4, 5\rangle$.

this claim is correct then computing $(a \star b)$ can be trivially done provided that the solution vector s is given.

In order to prove the above statement, we first start with an observation.

Observation G.1. *For any $1 \le i \le n+1+a_{n-1}+b_{n-1}$ there exists an optimal solution for size i that contains vertex r.*

Proof. For $i \le n+1$, one can simply start with vertex r and go along path c' to collect i vertices with weight 1. Obviously, this is the best we can achieve since the weight of every vertex is bounded by 1. For $i > n+1$ we argue that the weight of the solution is at least $n+1$ since path c' along with vertex r and some subpath of a' and b' suffice to have a solution weight at least $n+1$. In addition to this, if we remove vertex r from the tree, any connected subtree contains at most n vertices with weight 1 and thus any solution not containing vertex r has a weight of at most n. Thus, one can obtain an optimal solution containing path c' and vertex r from any optimal solution. □

Observation G.2. *For any $i \ge n+1$, there always exists an optimal solution of size i that contains both the entire path c' and vertex r.*

Proof. By Observation G.1 we already know that there always exists an optimal solution of size i that contains vertex r. Notice that the weight of all vertices in path c' is equal to 1. Thus, if the solution doesn't contain all vertices of path c', one can iteratively remove a leaf from the solution and instead add the next vertex in path c‘ to the solution. This does not hurt the solution since the weight of all vertices in path c' is equal to 1. □

Now, for an $i \ge n+1$ consider such a solution of the tree sparsity problem for size i. Apart from $n+1$ vertices of path c' and r, such a solution contains a prefix of path a' and a prefix of path b'. The number of vertices with weight one in each of these paths indicates how many indices in the corresponding vectors have a value equal to or smaller than the number of vertices of the solution in that path. This implies that if vector s is the solution of the tree sparsity problem then $(a \star b)_i$ is equal to $j - n - 1$ for the smallest j such that $s_j = n + 1 + i$.

Notice that since a and b are $\mathsf{d}_{\max}$-distance bounded, both a_{n-1} and b_{n-1} are bounded by $\mathsf{d}_{\max}n$ and thus one can solve the corresponding tree sparsity problem in time $T((\mathsf{d}_{\max} + 1)n + 1)$. Since T is at most quadratic, the running time is $O(T(\mathsf{d}_{\max}n))$. $\square$

In Section 3, we discussed that when input values for convolution are small integers bounded by $\mathsf{e}_{\max}$, an $\widetilde{O}(\mathsf{e}_{\max}n)$ time algorithm can solve the problem exactly. Notice that when the input values are bounded by $\mathsf{e}_{\max}$, the input vectors are also $\mathsf{e}_{\max}$-distance bounded and thus an $\widetilde{O}(T(\mathsf{e}_{\max}n))$ is also possible via a reduction to 0/1 tree sparsity. We showed in Section A that the knapsack problem reduces to knapsack convolution and used the prediction technique to solve the knapsack convolution in time $\widetilde{O}(\mathsf{v}_{\max}n)$ when the item values are bounded by $\mathsf{v}_{\max}$. However, it follows from the definition that if the item values are integers bounded by $\mathsf{v}_{\max}$ then knapsack convolution is a special case of $\mathsf{v}_{\max}$-distance bounded convolution. Thus, a $T(n)$ time solution for 0/1 tree sparsity implies a $\widetilde{O}(T(\mathsf{v}_{\max}n))$ time solution for bounded value knapsack. It is interesting to observe that the solutions of both tree sparsity and tree separability are $\mathsf{w}_{\max}$-distance bounded if the weights are integers bounded by $\mathsf{w}_{\max}$. Therefore, a $T(n)$ time algorithm for 0/1 tree sparsity also implies a $\widetilde{O}(T(\mathsf{w}_{\max}n))$ time algorithm for tree sparsity and tree separability when the weights are integers bounded by $\mathsf{w}_{\max}$. In particular, an $\widetilde{O}(n)$ time algorithm for 0/1 tree sparsity yields $\widetilde{O}(\mathsf{w}_{\max}n)$ time algorithms for both tree sparsity and tree separability when the weights are integers bounded by $\mathsf{w}_{\max}$.

Theorem G.2. *A $T(n)$ time solution for 0/1 tree sparsity implies the following:*

- *A $\widetilde{O}(T(\mathsf{e}_{\max}n))$ time algorithm for bounded knapsack convolution when the values are integers in range $[0, \mathsf{e}_{\max}]$.*
- *A $\widetilde{O}(T(\mathsf{v}_{\max}n))$ time algorithm for 0/1 knapsack when the item values are integers in range $[0, \mathsf{v}_{\max}]$.*
- *A $\widetilde{O}(T(\mathsf{w}_{\max}n))$ time algorithm for tree sparsity when the vertex weights are integers in range $[0, \mathsf{w}_{\max}]$.*
- *A $\widetilde{O}(T(\mathsf{w}_{\max}n))$ time algorithm for tree separability when the vertex weights are integers in range $[0, \mathsf{w}_{\max}]$.*

H Reduction to Polynomial Convolution

Given two integer vectors a and b with the condition that all a_i's and b_i's are in range $[0, \mathsf{e}_{\max}]$ we wish to find the convolution of the two vectors in the $(\max, +)$ setting. We denote this by $a \star b$. Let $n = |a| + |b|$ and define a' as a vector with the same size as a as follows:

$$\forall 0 \le i < |a| \qquad\qquad a'_i = (n+1)^{a_i}.$$

Similarly, we assume b' is a vector with the same size as b such that

$$\forall 0 \le i < |b| \qquad\qquad b'_i = (n+1)^{b_i}.$$

This way, for all i and j we have $a'_i b'_j = (n+1)^{a_i}(n+1)^{b_i} = (n+1)^{a_i+b_j}$. Let $c = a \star b$ be the solution of the problem and $c' = a' \times b'$ be the polynomial multiplication of a' and b', thus, for every $0 \le i < |c'|$ we have

$$c'_i = \sum_{j=0}^{i} a'_i b'_{i-j} = \sum_{j=0}^{i} (n+1)^{a_j+b_{i-j}} \le i \max_{j=0}^{i} (n+1)^{a_j+b_{i-j}} \le i(n+1)^{\max_{j=0}^{i} a_j+b_{i-j}} \le i(n+1)^{c_i} \le n(n+1)^{c_i}.$$

Similarly one can show that

$$c'_i = \sum_{j=0}^{i} a'_i b'_{i-j} = \sum_{j=0}^{i} (n+1)^{a_j+b_{i-j}} \ge \max_{j=0}^{i} (n+1)^{a_j+b_{i-j}} \ge (n+1)^{\max_{j=0}^{i} a_j+b_{i-j}} \ge (n+1)^{c_i},$$

and thus $(n+1)^{c_i} \le c'_i \le n(n+1)^{c_i}$. Therefore, $c_i = \lfloor \log_{n+1} c'_i \rfloor$ and thus one can determine vector c from c'. This reduction shows that any algorithm for computing c' in time $\widetilde{O}(\mathsf{e}_{\max} n)$ yields a similar running time for computing c. Polynomial multiplication can be computed in time $\widetilde{O}(n)$ via FFT when the running time of each arithmetic operation is $O(1)$ [10]. However, here, every element of a' and b' can be as large as $(n+1)^{\mathsf{e}_{\max}}$ and thus every arithmetic operation for such values takes time $\widetilde{O}(\mathsf{e}_{\max})$. Thus, the total running time of the FFT method for computing c' is $\widetilde{O}(\mathsf{e}_{\max} n)$. This yields an $\widetilde{O}(\mathsf{e}_{\max} n)$ time algorithm for computing $a \star b$.

Algorithm 7: BoundedRangeConvolution(a, b)

```
  Data: Two vectors $a$ and $b$
  Result: $a \star b$
1 $a' \leftarrow$ a vector of size $|a|$ s.t $a'_i = (n+1)^{a_i}$;
2 $b' \leftarrow$ a vector of size $|b|$ s.t $b'_i = (n+1)^{b_i}$;
3 $c' = $ FFTPolynomialConvolution$(a', b')$;
4 $c \leftarrow$ a vector of size $|c'|$ s.t $c_i := \lfloor \log_{n+1} c'_i \rfloor$;
5 Return $c$;
```

Lemma H.1 (also used in [1, 4, 8, 25, 26])**.** *Given two vectors a and b whose values are integers in the range $[0, \mathsf{e}_{\max}]$, there exists an algorithm to compute $a \star b$ in time $\widetilde{O}(\mathsf{e}_{\max} n)$ where $n = |a| + |b|$.*

Of course Lemma H.1 holds whenever the range of the values of the vectors is an interval of length $\mathsf{e}_{\max}$. It has been also shown that Lemma H.1 holds even when the input values are in set $\{0, 1, \ldots, \mathsf{e}_{\max}, -\infty\}$. The reason behind this is that since the numbers are small, one can replace $-\infty$ by $-2\mathsf{e}_{\max}$ and solve the problem in time $\widetilde{O}(n\mathsf{e}_{\max})$ using the same procedure. Then, we replace every negative value of the solution by $-\infty$. The same idea can solve the problem when input values are allowed to be ∞ as well as $-\infty$. An immediate consequence of Lemma H.1 is that even if the vectors do not have integer values, still one can use this method to approximate the solution within a small additive error. We show this in Lemma H.2.

Lemma H.2. *Let a and b be two given vectors whose values are real numbers in range $[0, \mathsf{e}_{\max}]$. One can compute in time $\widetilde{O}(\mathsf{e}_{\max} n)$ a vector c with the same size as $|a \star b|$ such that $(a \star b)_i - 1 < c_i \le (a \star b)_i$ holds for all $0 \le i < |c|$ where $n = |a| + |b|$.*

Proof. For a vector x, let $\lfloor x \rfloor$ be an integer vector of the same size where $\lfloor x \rfloor_i = \lfloor x_i \rfloor$. Moreover, for a vector x we define αx as a vector of the same size where $(\alpha x)_i = \alpha x_i$. Note that for a given vector x, both $\lfloor x \rfloor$ and αx can be computed in time $O(n)$ from x. We argue that $c = 1/2(\lfloor 2a \rfloor \star \lfloor 2b \rfloor)$ meets the conditions of the lemma. This observation proves the lemma since all values of $\lfloor 2a \rfloor$ and $\lfloor 2b \rfloor$ are integers in range $[0, 2\mathsf{e}_{\max}]$ and thus one can compute $\lfloor 2a \rfloor \star \lfloor 2b \rfloor$ in time $\widetilde{O}(\mathsf{e}_{\max} n)$. With additional $O(n)$ operations we compute c from $\lfloor 2a \rfloor \star \lfloor 2b \rfloor$.

Notice that for every i we have $2a_i - 1 < \lfloor 2a_i \rfloor \le 2a_i$ and similarly $2b_i - 1 < \lfloor 2b_i \rfloor \le 2b_i$. Therefore, for every $0 \le i < |a \star b|$ we have $(2a \star 2b)_i - 2 < (\lfloor 2a \rfloor \star \lfloor 2b \rfloor)_i \le (2a \star 2b)_i$. This implies that $(a \star b)_i - 1 < (1/2(\lfloor 2a \rfloor \star \lfloor 2b \rfloor))_i \le (a \star b)_i$ holds for all $0 \le i < |a \star b|$ and thus the proof is complete. ☐

Similar to Lemma H.1, Lemma H.2 also holds when $-\infty$ and ∞ are allowed in the input.

Algorithm 8: ApproximateConvolution(a, b)

```
Data: Two vectors $a$ and $b$
Result: An approximate solution to $a \star b$
1 $a' \leftarrow \lfloor 2a \rfloor$;
2 $b' \leftarrow \lfloor 2b \rfloor$;
3 $c' = $ BoundedRangeConvolution$(a', b')$;
4 $c = c'/2$;
5 Return $c$;
```

I Omitted Proofs of Section 3.1

Proof. [of Lemma 3.1]

first condition: Suppose for the sake of contradiction that the condition of the lemma doesn't hold for some i and j. We assume w.l.o.g. that $a_i - b_i \geq a_j - b_j$ and thus $a_i - b_i > a_j - b_j + \mathsf{e}_{\max}$. This implies that $a_i + b_j > a_j + b_i + \mathsf{e}_{\max}$ and hence $(a \star b)_{i+j} \geq a_i + b_j > a_j + b_i + \mathsf{e}_{\max}$ which contradicts the first assumption of the lemma.

second condition: Based on the assumption of the lemma we have $(a \star b)_{i+k} - \mathsf{e}_{\max} \leq a_i + b_k \leq (a \star b)_{i+k}$. Similarly, $(a \star b)_{2j} - \mathsf{e}_{\max} \leq a_j + b_j \leq (a \star b)_{2j}$. Since $j - i = k - j$ then $2j = i + k$ and thus both $a_j + b_j$ and $a_i + b_k$ are lower bounded by $(a \star b)_{i+k} - \mathsf{e}_{\max}$ and upper bounded by $(a \star b)_{i+k}$. Hence we have $|(a_i + b_k) - (a_j + b_j)| \leq \mathsf{e}_{\max}$. If we add the term $[(b_k - a_k) - (b_j - a_j)]$ to the expression $(a_i + a_k) - 2a_j$ we obtain

$$\begin{aligned} |(a_i + a_k) - 2a_j + [(b_k - a_k) - (b_j - a_j)]| &= |(a_i + (b_k - a_k) + a_k) - (2a_j + (b_j - a_j))| \\ &= |(a_i + b_k) - (a_j + b_j)| \\ &\leq \mathsf{e}_{\max}, \end{aligned}$$

which implies $|(a_i + a_k) - 2a_j| \leq \mathsf{e}_{\max} + |(b_k - a_k) - (b_j - a_j)|$. Recall that we proved $|(b_k - a_k) - (b_j - a_j)| \leq \mathsf{e}_{\max}$ and thus $|(a_i + a_k) - 2a_j| \leq 2\mathsf{e}_{\max}$. □

Proof. [of Lemma 3.2] We present a simple algorithm and show that (i) it provides a correct solution for the problem and (ii) its running time is $\widetilde{O}(\mathsf{e}_{\max} n)$. In this algorithm, we construct two vectors a' and b' from a and b such that all values of a' and b' are in range $[0, 6\mathsf{e}_{\max}]$ and $a \star b$ can be computed from $a' \star b'$. The key idea here is that if we add a constant C to all components of either a or b, this value is added to all elements of $(a \star b)$. Moreover, for a fixed C, if we add a value of iC to every element i of both a and b, then every $(a \star b)_i$ is increased by exactly iC. Based on these observations, we construct two vectors a' and b' of size n from a and b as follows:

$$\begin{aligned} a'_i &:= a_i + [3\mathsf{e}_{\max} - a_0] && + i[(a_0 - a_{n-1})/(n-1)] \\ b'_i &:= b_i + [3\mathsf{e}_{\max} + a_{n-1} - a_0 - b_{n-1}] && + i[(a_0 - a_{n-1})/(n-1)]. \end{aligned}$$

The transformation formulas are basically the application of the above operations to a and b which are delicately chosen to make sure the values of a' and b' fall in range $[0, 6\mathsf{e}_{\max}]$. Notice that vectors a' and b' might have fractional values. However, we show that all the values of these vectors are in range $[0, 6\mathsf{e}_{\max}]$. Thus, we can use the algorithm of Lemma H.2 to compute in time $\widetilde{O}(\mathsf{e}_{\max} n)$ an approximate solution c' to $a' \star b'$ within an error less than 1. Next, based on vector c' we construct a solution c as follows:

$$c_i := \lceil c'_i - [6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] - i[(a_0 - a_{n-1})/(n-1)] \rceil.$$

Finally, we report c as the solution to $a \star b$. This procedure is shown in Algorithm 9.

In what follows, we show that c is indeed equal to $a \star b$ and that our algorithm runs in time $\widetilde{O}(\mathsf{e}_{\max} n)$. We first point out a few observations regarding a' and b':

Observation I.1. $a'_0 = a'_{n-1} = b'_{n-1} = 3\mathsf{e}_{\max}$.

Proof. According to the formula,

$$\begin{aligned} a'_{n-1} &= a_{n-1} + [3\mathsf{e}_{\max} - a_0] + (n-1)[(a_0 - a_{n-1})/(n-1)] \\ &= a_{n-1} + [3\mathsf{e}_{\max} - a_0] + (a_0 - a_{n-1}) \\ &= 3\mathsf{e}_{\max} + [a_0 - a_0] + (a_{n-1} - a_{n-1}) \\ &= 3\mathsf{e}_{\max}. \end{aligned}$$

Algorithm 9: DistortedNTimesNConvolution(a, b)

```
Data: Two integer vectors $a$ and $b$ of size $n$ meeting the condition of Lemma 3.2
Result: $a \star b$
1 $a' \leftarrow$ a vector of size $n$ s.t $a'_i = a_i + [3\mathsf{e}_{\max} - a_0] + i[(a_0 - a_{n-1})/(n-1)]$;
2 $b' \leftarrow$ a vector of size $n$ s.t $b'_i = b_i + [3\mathsf{e}_{\max} + a_{n-1} - a_0 - b_{n-1}] + i[(a_0 - a_{n-1})/(n-1)]$;
3 $c' \leftarrow \mathsf{ApproximateConvolution}(a', b')$;
4 $c \leftarrow$ a vector of size $2n-1$ s.t $c_i := \lceil c'_i - [6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] - i[(a_0 - a_{n-1})/(n-1)]\rceil$;
5 Return $c$;
```

Moreover,

$$\begin{aligned}
a'_0 &= a_0 + [3\mathsf{e}_{\max} - a_0] + 0[(a_0 - a_{n-1})/(n-1)] \\
&= a_0 + [3\mathsf{e}_{\max} - a_0] \\
&= 3\mathsf{e}_{\max} + [a_0 - a_0] \\
&= 3\mathsf{e}_{\max}.
\end{aligned}$$

Finally, for b'_{n-1} we have

$$\begin{aligned}
b'_{n-1} &= b_{n-1} + [3\mathsf{e}_{\max} + a_{n-1} - a_0 - b_{n-1}] + (n-1)[(a_0 - a_{n-1})/(n-1)] \\
&= b_{n-1} + [3\mathsf{e}_{\max} + a_{n-1} - a_0 - b_{n-1}] + (a_0 - a_{n-1}) \\
&= 3\mathsf{e}_{\max} + [b_{n-1} + a_0 - a_0 - b_{n-1}] + (a_{n-1} - a_{n-1}) \\
&= 3\mathsf{e}_{\max}.
\end{aligned}$$

□

Observation I.2. *For every $0 \le i, j < n$ we have $a'_i + b'_j \ge (a' \star b')_{i+j} - \mathsf{e}_{\max}$ and*

$$(a' \star b')_{i+j} = (a \star b)_{i+j} + [6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] + (i+j)[(a_0 - a_{n-1})/(n-1)].$$

Proof. Based on the construction of a' and b' we have

$$\begin{aligned}
a'_i + b'_j &= a_i + [3\mathsf{e}_{\max} - a_0] + i[(a_0 - a_{n-1})/(n-1)] \\
&\quad + b_j + [3\mathsf{e}_{\max} + a_{n-1} - a_0 - b_{n-1}] + j[(a_0 - a_{n-1})/(n-1)] \\
&= a_i + b_j + [6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] + i[(a_0 - a_{n-1})/(n-1)] + j[(a_0 - a_{n-1})/(n-1)] \\
&= a_i + b_j + [6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] + (i+j)[(a_0 - a_{n-1})/(n-1)].
\end{aligned}$$

Notice that $[6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] + (i+j)[(a_0 - a_{n-1})/(n-1)]$ is the same for all pairs of indices that sum up to $i+j$ and thus

$$(a' \star b')_{i+j} = (a \star b)_{i+j} + [6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] + (i+j)[(a_0 - a_{n-1})/(n-1)].$$

Since we have $a_i + b_j \ge (a \star b)_{i+j} - \mathsf{e}_{\max}$, by adding the $[6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] + (i+j)[(a_0 - a_{n-1})/(n-1)]$ part to both sides of the inequality we get $a'_i + a'_j \ge (a' \star b')_{i+j} - \mathsf{e}_{\max}$. □

Next, we use Observations I.1 and I.2 to show that the values of both vectors a' and b' are in the interval $[0, 6\mathsf{e}_{\max}]$. Observation I.2 states that both a' and b' meet the conditions of Lemma 3.1 and thus for every $0 \le i < j < k < n$ subject to $j - i = k - j$ we have $|a'_i + a'_k - 2a_j| \le 2\mathsf{e}_{\max}$. Moreover, Observations I.1 implies that both a'_0 and a'_{n-1} are equal to $3\mathsf{e}_{\max}$. Now, let $j = \arg\max a'_i$ and suppose for the sake of contradiction that $a'_j > 5\mathsf{e}_{\max}$. If $2j < n$, then by setting $i = 0$ and $k = 2j$ Lemma 3.1 implies that $a'_0 + a'_{2j} \ge 2a'_j - 2\mathsf{e}_{\max}$. Therefore, since $a'_0 = 3\mathsf{e}_{\max}$ this implies

$a'_{2j} - a'_j \geq a'_j - 5\mathsf{e}_{\max}$ and thus if $a'_j > 5\mathsf{e}_{\max}$ then $a'_{2j} - a'_j > 0$ contradicts the maximality of a'_j. If $2j \geq n$, then by setting $i = 2j - n - 1$ and $k = n - 1$ one could show that if $a'_j > 5\mathsf{e}_{\max}$, then $a'_{2j-n-1} > a'_j$ holds which again contradicts the maximality of a'_j. One can make a similar argument and show that if $j = \arg\min a'_i$ and $a'_j < \mathsf{e}_{\max}$, then either of a'_{2j} or a'_{2j-n-1} should be less than a'_j which contradicts the minimality of a'_j. Therefore, all the values of vector a' lie in the interval $[\mathsf{e}_{\max}, 5\mathsf{e}_{\max}]$.

Recall that by Observation I.2, a' and b' meet the condition of Lemma 3.1 and thus for all $0 \leq i < n$ we have

$$\begin{aligned}|(a'_i - b'_i) - (a'_{n-1} - b'_{n-1})| &= |(a'_i - b'_i) - (3\mathsf{e}_{\max} - 3\mathsf{e}_{\max})| \\ &= |(a'_i - b'_i)| \\ &\leq \mathsf{e}_{\max}.\end{aligned}$$

Since for all $0 \leq i < n$, $\mathsf{e}_{\max} \leq a'_i \leq 5\mathsf{e}_{\max}$ holds, we have $0 \leq b'_i \leq 6\mathsf{e}_{\max}$ which shows that the values of both a' and b' lie in the interval $[0, 6\mathsf{e}_{\max}]$. However, since the values are not integer, still we cannot compute $a' \star b'$ in time $\widetilde{O}(\mathsf{e}_{\max}n)$. Instead, we can compute in time $\widetilde{O}(\mathsf{e}_{\max}n)$ a vector c' such that $(a' \star b')_i - 1 < c'_i \leq (a' \star b')_i$ holds for all $0 \leq i < |c'|$. Observation I.2 implies that for all $0 \leq i < |c'|$ we have

$$(a \star b)_i - 1 < c'_i - [6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] - i[(a_0 - a_{n-1})/(n-1)] \leq (a \star b)_i.$$

Notice that since both a and b are integer vectors, $a \star b$ is also an integer vector and thus all its elements have integer values. Therefore, $\lceil c'_i - [6\mathsf{e}_{\max} + a_{n-1} - 2a_0 - b_{n-1}] - i[(a_0 - a_{n-1})/(n-1)]\rceil = (a \star b)_i$ for all $0 \leq i < |c'|$ and hence our algorithm computes $a \star b$ correctly.

With regard to the running time, all steps of the algorithm run in time $O(n)$, except where we approximate c' from a' and b' which takes time $\widetilde{O}(\mathsf{e}_{\max}n)$ since $0 \leq a'_i, b'_i \leq 6\mathsf{e}_{\max}$ (Lemma H.2). Thus, the total running time of our algorithm is $\widetilde{O}(\mathsf{e}_{\max}n)$.

Proof. [of Lemma 3.3] As mentioned earlier, we show this lemma by a direct reduction to Lemma 3.2. We assume w.l.o.g. that $|b| \geq |a|$. Let $l = \lceil |b|/|a| \rceil$ and construct l intervals (x_i, y_i) of length $|a|$ (i.e. $y_i = x_i + |a| - 1$ for all i) as follows: for $1 \leq i < l$ set $x_i = (i-1)|a|$ and $y_i = i|a| - 1$. Also, set $x_l = |b| - |a|$ and $y_l = |b| - 1$. This way, every $0 \leq i < |b|$ appears in at least one interval.

Next, we construct l vectors b^1, b^2, $\ldots$, b^l from b where every b^i is a vector of length $|a|$ and $b^i_j = b_{x_i+j}$. We next compute $c^i = a \star b^i$ for all $1 \leq i \leq l$, each in time $\widetilde{O}(\mathsf{e}_{\max}|a|)$ using Lemma 3.2. Thus, the total running time of this step is $\widetilde{O}(\mathsf{e}_{\max}|a|l) = \widetilde{O}(\mathsf{e}_{\max}(|a| + |b|))$.

Finally, we construct a solution of size $|a| + |b| - 1$ initially containing $-\infty$ for all elements and for every vector $0 \leq j < 2|a| - 1$ and c^i, we set $c_{x_i+j} := \max\{c_{x_i+j}, c^i_j\}$. This takes a total time of $O(|a|l) = O(|a| + |b|)$ and therefore the total running time of the algorithm is $\widetilde{O}(\mathsf{e}_{\max}(|a| + |b|))$.

We argue that for all $0 \leq i < |a| + |b| - 1$, $c_i \leq (a \star b)_i$ holds. To this end, suppose for the sake of contradiction that $c_i > (a \star b)_i$ for an $0 \leq i < |c|$. Due to our algorithm, $c_i = c^k_j$ for some $1 \leq k \leq l$ and $0 \leq j < 2|a| - 1$ such that $x_k + j = i$. Hence, $c_i = (a \star b^k)_j \leq (a \star b)_{x_k+j}$ and since $x_k + j = i$ we have $c_i \leq (a \star b)_i$ which contradicts $c_i > (a \star b)_i$. Also, if $c_i < (a \star b)_i$ for some i, then we argue that by definition $(a \star b)_i = b_j + a_{i-j}$ for some j. Since every element of b appears in at least one interval, there exists a k such that $x_k \leq j \leq y_k$. Since $b^k_{j-x_k} = b_j$ we have $c^k_{i-x_k} = c^k_{j-x_k+(i-j)} \geq b_j + a_{i-j} = (a \star b)_i$. Note that $c_i \geq c^k_{i-x_k}$ and thus $c_i \geq (a \star b)_i$ which contradicts $c_i < (a \star b)_i$. □

Algorithm 10: DistortedConvolution(a, b)

```
Data: Two integer vectors $a$ and $b$ meeting the condition of Lemma 3.3
Result: $a \star b$
1  $l \leftarrow \lceil |b|/|a| \rceil$;
2  for $i \in [1, l-1]$ do
3      $x_i \leftarrow (i-1)|a|$;
4      $y_i \leftarrow i|a| - 1$;
5  $x_l \leftarrow |b| - |a|$;
6  $y_l \leftarrow |b| - 1$;
7  for $i \in [1, i]$ do
8      $b^i \leftarrow$ a vector of size $|a|$ s.t. $b^i_j = b_{x_i+j}$;
9      $c^i \leftarrow$ DistortedNTimesNConvolution$(a, b^i)$;
10 $c \leftarrow$ a vector of size $|a| + |b| - 1$ with values set to $-\infty$ initially;
11 for $i \in [1, l]$ do
12     for $j \in [0, 2|a| - 2]$ do
13         $c_{x_i+j} \leftarrow \max\{c_{x_i+j}, c^i_j\}$;
14 Return $c$;
```

J Omitted Proofs of Section 3.2

Proof. [of Observation 3.1] Suppose for the sake of contradiction that $\mathcal{P}(\alpha, \beta)$ is not an interval of a. This means that there are three integers $i < j < k$ such that $i, k \in \mathcal{P}(\alpha, \beta)$ but $j \notin \mathcal{P}(\alpha, \beta)$. Therefore, we have $x_i \leq x_k \leq \alpha$ and $y_k \geq y_i \geq \beta$ but either $x_j > \alpha$ or $y_j < \beta$. However, since the intervals are monotone, $x_j \leq x_k$ and also $y_j \geq y_i$ and thus $x_j \leq \alpha$ and $y_j \geq \beta$ which is a contradiction. $\square$

Proof. [of Observation 3.2] Similar to Observation 3.1, suppose for the sake of contradiction that $\mathcal{P}(\alpha_1, \beta_1) \setminus \mathcal{P}(\alpha_2, \beta_2)$ is not an interval. Since both of $\mathcal{P}(\alpha_1, \beta_1)$ and $\mathcal{P}(\alpha_2, \beta_2)$ are intervals (see Observation 3.1), this implies that there exist $i < j < k$ such that $i, j, k \in \mathcal{P}(\alpha_1, \beta_1)$, $i, k \notin \mathcal{P}(\alpha_2, \beta_2)$, and $j \in \setminus \mathcal{P}(\alpha_2, \beta_2)$. In other words, $[\alpha_1, \beta_1]$ is a subset of $[x_i, y_i]$, $[x_j, y_j]$, and $[x_k, y_k]$. Moreover, none of $[x_i, y_i]$ and $[x_k, y_k]$ entirely contain $[\alpha_2, \beta_2]$ but $[x_j, y_j]$ contains $[\alpha_2, \beta_2]$. Notice that if $[x_i, y_i]$ contains β_2 the monotonicity of the intervals implies that $[x_i, y_i]$ contains $[\alpha_2, \beta_2]$. Similarly, we can imply that $\alpha_2 \notin [x_k, y_k]$ and thus $[x_i, y_i] \cap [x_k, y_k] \subseteq [\alpha_2, \beta_2]$. Since $[\alpha_1, \beta_1]$ and $[\alpha_2, \beta_2]$ are disjoint, this implies $[\alpha_1, \beta_1]$ is empty and thus the solution is empty and contradicts our assumption. $\square$

K An $\widetilde{O}(\mathsf{v}_{\max}t+n)$ Time Algorithm for Knapsack

The result of this section follows from the reduction of [11] from knapsack to $(\max,+)$ convolution. However, for the sake of completeness we restate the reduction of [11] and use Theorem A.3 to solve knapsack in time $\widetilde{O}(\mathsf{v}_{\max}t+n)$, in case the item values are integers in range $[0,\mathsf{v}_{\max}]$.

Proof. [of Theorem A.4] For simplicity, we assume that the goal of the knapsack problem is to find the solution for all knapsack sizes $0 \le i \le t$. The blueprint of the reduction is as follows: We first divide the items into $\lceil \log t \rceil$ buckets in a way that the sizes of the items in every bucket differ by at most a multiplicative factor of two. Next, for each of the buckets, we solve the problem with respect to the items of that bucket. More precisely, for every bucket i we compute a vector c^i of size $t+1$ where $(c^i)_j$ is the solution to the knapsack problem for bucket i and knapsack size j. Once we have these solutions, it only suffices to compute $c^1 \star c^2 \star \ldots \star c^{\lceil \log t \rceil}$ and report the first $t+1$ elements as the solution. Therefore, the problem boils down to finding the solution for each of the buckets.

In every bucket i, the size of the items is in range $[2^{i-1}, 2^i-1]$. Now, if we fix a range $[r_1, r_2]$ for the item sizes, the maximum number of items used in any solution is t/r_1. Therefore, if we randomly put the items in t/r_1 categories, any fixed solution will have no more than $\mathsf{polylog}(t)$ items in every category. Based on this, we propose the following algorithm to solve the problem for item sizes in range $[r_1, r_2]$: randomly put the items into t/r_1 categories. For every category, solve the problem up to a knapsack size $r_2\mathsf{polylog}(t)$ and merge the solutions. We show that merging the solutions can be done via some convolution invocations of total size $\widetilde{O}(t)$. Thus, the only non-trivial part is to solve the problem up to a knapsack size $r_2\mathsf{polylog}(t)$ for every category of items. Since $r_2/r_1 \le 2$, each of these solutions consists of at most $\mathsf{polylog}(t)$ items. Cygan *et al.* [11] show that if we again put these items in $\mathsf{polylog}(t)$ random groups, then using $(\max,+)$ convolution one can solve the problem in almost linear time. We bring the pseudocode of the algorithms below. For correctness, we refer the reader to [11]. Here, we just show that the algorithm runs in time $\widetilde{O}(\mathsf{v}_{\max}t+n)$ if we use the knapsack convolution. Since the algorithm is probabilistic, in order to bring the success probability close to 1, we use a factor $\mathsf{C} = \mathsf{polylog}(t)$ in our algorithm and run the procedures C times and take the best solution found in these runs. We do not specify the exact value of C, however, since C is logarithmically small, it does not have an impact on the running time of our algorithms since we use the $\widetilde{O}$ notation.

Algorithm 11 solves the problem when the solution consists of at most C items.

Algorithm 11: $\mathsf{BoundedSolutionKnapsackAlgorithm}(t, \{(s_1,v_1),(s_2,v_2),\ldots\})$

```
Data: Knapsack size $t$ and items $(s_1,v_1),(s_2,v_2),\ldots,(s_n,v_n)$
Result: A solution vector $c$
1 $c \leftarrow$A vector of size $t+1$ with all 0's initially;
2 for $cnt \in [1,\mathsf{C}]$ do
3     Randomly put the items in $\mathsf{C}^2$ lists $l_1, l_2, \ldots, l_{\mathsf{C}}$;
4     for $i \in [1,\mathsf{C}^2]$ do
5         $c^i \leftarrow$ A vector of size $t+1$ where $c^i_j$ is the highest value of an item in $l_i$ with size at most $j$;
6     $c' \leftarrow c^1 \star c^2 \star \ldots c^{\mathsf{C}^2}$;
7     for $i \in [0,t]$ do
8         $c_i = \max\{c_i, c'_i\}$;
9 return c;
```

Notice that C is logarithmically small in size of the original knapsack. The only time consuming operation of the algorithm is Line 6 which takes time $\widetilde{O}(\mathsf{v}_{\max}t)$ due to Lemma H.1 since the item values are bounded by $\mathsf{v}_{\max}$. Moreover, Algorithm 11 iterates over all items at least once. Thus, the total running time of Algorithm 11 is $\widetilde{O}(\mathsf{v}_{\max}t + n)$ for a given knapsack size t and n items. Algorithm 12 uses Algorithm 11 to solve the knapsack problem when all the item sizes are in range $[r_1, r_2]$ and $r_2 \le 2r_1$.

```
Algorithm 12: BoundedRangeKnapsackAlgorithm($t, \{(s_1, v_1), (s_2, v_2), \ldots\}, [r_1, r_2]$)
  Data: Knapsack size $t$, items $(s_1, v_1), (s_2, v_2), \ldots, (s_n, v_n)$, and range $[r_1, r_2]$
  Result: A solution vector $c$
1 $c \leftarrow$A vector of size $t+1$ with all 0's initially;
2 for $cnt \in [1, \mathsf{C}]$ do
3     Randomly put the items in $\lceil t/r_1 \rceil$ lists $l_1, l_2, \ldots, l_{\lceil t/r_1 \rceil}$;
4     for $i \in [1, \lceil t/r_1 \rceil]$ do
5         $c^i \leftarrow$ BoundedSolutionKnapsackAlgorithm($\mathsf{C}r_2, l_i$);
6     $c' \leftarrow$ Merge($\{c^1, c^2, \ldots, c^{\lceil t/r_1 \rceil}\}$);
7     for $i \in [0, t]$ do
8         $c_i = \max\{c_i, c'_i\}$;
9 return c;
```

Algorithm 12 puts the items into $\lceil t/r_1 \rceil$ different categories and solves each category using Algorithm 11. Since the running time of Algorithm 11 is $\widetilde{O}(\mathsf{v}_{\max}t + n)$, except the part where we merge the solutions. In the following, we describe the algorithm for merging the solutions and show that its running time is $\widetilde{O}(\mathsf{v}_{\max}t)$ where t is the original knapsack size.

```
Algorithm 13: Merge($\{c^1, c^2, \ldots, c^k\}$)
  Data: $k$ vectors $c^1, c^2, \ldots, c^k$ with total size $t$
  Result: $c^1 \star c^2 \star c^3 \ldots c^k$
1 if $k = 1$ then
2     return $c^1$
3 else
4     $a \leftarrow$ Merge($c^1, c^2, \ldots, c^{\lfloor k/2 \rfloor}$);
5     $b \leftarrow$ Merge($c^{\lfloor k/2 \rfloor+1}, c^{\lfloor k/2 \rfloor+2}, \ldots, c^k$);
6     return KnapsackConvolution($a, b$);
7 return c;
```

Notice that Algorithm 13 uses the knapsack convolution to merge the vectors. Every merge for vectors with total size n takes time $\widetilde{O}(\mathsf{v}_{\max}n)$. Moreover, the total size of the vectors is $\widetilde{O}(t)$ and due to Algorithm 13, the total length of the vectors in all convolutions is $\widetilde{O}(t)$. Thus, Algorithm 13 runs in time $\widetilde{O}(\mathsf{v}_{\max}t)$.

Finally, in Algorithm 14 we merge the solutions of different buckets and report the result.

```
Algorithm 14: KnapsackViaConvolution($t, \{(s_1, v_1), (s_2, v_2), \ldots\}$)
1 $l_1, l_2, \ldots, l_{\lceil \log t \rceil+1} \leftarrow \lceil \log t \rceil + 1$ lists of items initially empty;
2 for $i \in [1, \lceil \log t \rceil + 1]$ do
3     Put all items with size in range $[2^{i-1}, 2^i - 1]$ in $l_i$;
4     $c^i \leftarrow$ BoundedRangeKnapsackAlgorithm($t, l_i, [2^{i-1}, 2^i - 1]$);
5 $c \leftarrow c^1 \star c^2 \star \ldots \star c^{\lceil \log t \rceil+1}$;
6 return the first $t+1$ elements of $c$;
```

Since we use the knapsack convolution for merging the solutions of different buckets, the running time of Algorithm 14 is also $\widetilde{O}(\mathsf{v}_{\max}t + n)$; $\square$

L Omitted Proofs of Section A.1

Proof. [of Observation A.1] We argue that in any optimal solution, if for two items i and j we have $w_i/s_i > w_j/s_j$ then either $f_i = 1$ or $f_j = 0$. If not, one can increase f_i by ϵ and decrease f_j by $s_j\epsilon/s_i$ and obtain a better solution. Notice that for items with the same ratio of w_i/s_i it doesn't matter which items are put in the knapsack so long as the total size of these items in the knapsack is fixed. Thus, the greedy algorithm provides an optimal solution. The running time of the algorithm is $O(n \log n)$ since after sorting the items we only make an iteration over the items in time $O(n)$. □

Proof. [of Observation A.2] Similar to Observation A.1, in order to maximize the weight we always add the item with the highest ratio of w_i/s_i to the knapsack. Therefore, this yields the maximum total weight for any knapsack size t. The running time of the algorithm is $O(n \log n)$ since it sorts the items and puts them in the knapsack one by one. □

Proof. [of Observation A.3] This observation follows from the greedy algorithm for knapsack. Notice that we add the items to the knapsack greedily and the solution consists of two types of items: items of knapsack k_a and items of knapsack k_b. Since the algorithm greedily adds the items to the solution, the order of items is based on w_i/s_i and thus the order of items added to the solution for each type is also based on w_i/s_i. Thus, if for some knapsack size x we define $\mathcal{F}_a(x)$ to be the total size of the items in the solution of x that belong to k_a and set $\mathcal{F}_b(x)$ equal to the size of the solution for items of knapsack k_b, then $c'(x) = a'(\mathcal{F}_a(x)) + b'(\mathcal{F}_b(x))$. The monotonicity of $\mathcal{F}_a$ and $\mathcal{F}_b$ follow from the fact that in order to update the solution we only add items and we never remove any item from the solution. □

In the proofs of Observations A.4 and A.5 we refer to the solution of $a'(x)$ and $b'(x)$ as the solution that the greedy algorithm for fractional knapsack provides for knapsack size x and knapsack problems k_a and k_b, respectively. Similarly, we denote by the solution of $c'(x)$ the solution that Algorithm 2 provides for the fractional convolution of $a \star b$ with respect to knapsack size x. We say two solutions differ in at most one item, if they are the same except for one item.

Proof. [of Observation A.4] We assume w.l.o.g. that y and y' are close enough to make sure the solution of $c'(x+y)$ differs from the solution of $c'(x+y')$ by at most one item. Similarly, we assume w.l.o.g. that the solutions of $b'(y)$ and $b'(y')$ differ by at most one item. If the statement of Observation A.4 is correct for such y and y' then it extends to all $y < y'$ in range $[0, \mathcal{F}_a^{-1}(x) - x]$ since for every $y < y'$ one can write $y < y_1 < y_2 < \ldots < y'$ such that every two consecutive elements are close enough. Therefore, the statement holds for any pairs of consecutive elements and thus holds for y and y'. In order to compare $c'(x+y) - a'(x) - b'(y)$ with $c(x+y') - a'(x) - b'(y')$ it only suffices to compare $c'(x+y') - c'(x+y)$ with $b'(y') - b'(y)$.

It follows from the monotonicity of $\mathcal{F}_a$ that since $y < y' < \mathcal{F}_a^{-1}(x) - x$, then $\mathcal{F}_a(x+y) \leq \mathcal{F}_a(x+y') \leq x$ and therefore $\mathcal{F}_b(x+y') \geq y'$. Let (s_i, w_i) be the last item in the solution of knapsack problem k_b for knapsack size y'. Hence, due to Algorithm 2, any item not included in the solution of $c'(x+y)$ has a ratio of weight over size which is upper bounded by w_i/s_i. Therefore, $c'(x+y') - c'(x+y) \leq (y'-y)(w_i/s_i)$. Since $b'(y)$ and $b'(y')$ differ in at most one item we have $b'(y) - b'(y) = (y'-y)(w_i/s_i)$. Thus, $c'(x+y') - c'(x+y) \leq b'(y) - b'(y)$ and therefore $c'(x+y') - a'(x) - b'(y') \leq c'(x+y) - a'(x) - b'(y)$. □

Proof. [of Observation A.5] The proof is similar to that of Observation A.4. We assume w.l.o.g. that the solutions of $c'(x+y)$ and $c'(x+y')$ differ in at most one item and also the solutions of $b'(y)$ and $b'(y')$ differ in at most one item. By monotonicity of $\mathcal{F}_a$ we have $\mathcal{F}_a(x+y') \geq x$ and thus $\mathcal{F}_b(x+y') \leq y'$. This means that if (s_i, w_i) is the last item of $c'(x+y')$ then any item not included in $b'(y)$ has a ratio of weight over size of at most w_i/s_i. This implies that $b'(y') - b'(y) \leq (y' - y)w_i/s_i = c'(x+y') - c'(x+y)$ which implies $c'(x+y) - a'(x) + b'(y) \leq c'(x+y') - a'(x) - b'(y')$. $\square$

Proof. [of Observation A.6] Since $x < x'$ and $y \leq \mathcal{F}_a^{-1}(x) - x$ then we have $\mathcal{F}_b^{-1}(y) - y \leq x < x'$. Since a' and b' are symmetric, this observation follows from Observation A.5. $\square$

Proof. [of Observation A.7] Similar to Observation A.6, we have $x < x' \leq \mathcal{F}_b^{-1}(y) - y$ and the observation reduces to Observation A.5 by switching a' and b'. $\square$

M Omitted Proof of Section F.1

Proof. [of Lemma F.4] As aforementioned, the proof follows from the ideas of Cygan *et al.* [11] and Backurs *et al.* [1]. We assume that the tree is rooted at some arbitrary vertex and for a vertex v we refer to the subtree rooted at v by $T(v)$. We say the solution of a subtree $T(v)$ is a vector a of size $|T(v)|+1$, where every a_i denotes the minimum cost for putting the vertices of $T(v)$ into two disjoint components of sizes i and $|T(v)|-i$ where vertex v itself in the part with size i.

Let u and v be two disjoint subtrees of the graph and denote by a and b the solutions of these subtrees. If we add an edge from u to v and wish to compute the solution for the combined tree, one can derive the solution vector from a and b. To this end, there are two possibilities to consider: either u and v are in the same component in which case the solution is equal to $a \star b$. Otherwise, we construct a vector b' where $b'_i = 1 + b_{|T(v)|-i}$ and compute $a \star b'$ to find the answer Therefore, merging the solutions of two subtrees reduces to computing the convolution of two solution vectors.

If the tree is balanced, and therefore the height of the tree is $O(\log n)$, the standard dynamic program yields a running time of $T(n)$ since the total lengths of the convolutions we make is $O(n \log n)$. However, if the tree is not balanced, in the worst case, the height of the tree also appears in the running time. A classic tool to overcome this challenge is the spine decomposition of [21] to break the tree into a number of spines. Every spine is a path starting from a vertex and ending at some leaf. We say a spine x is above a spine y, if there exists a vertex in y such at least one parent of that vertex appears in x. We denote this relation with $x \prec y$. Such a spine decomposition satisfies the property that every sequence of spines $x_1 \prec x_2 \prec \ldots \prec x_k$ has a length of at most $O(\log n)$. This enables us to solve the dynamic program in time $\widetilde{O}(T(n))$. The overall idea is that instead of updating every vertex at each stage, we update the solution of a spine. Since every spine is a path, we can again reduce the problem of combining the solutions of a spine to convolution. This way, the height of the updates reduces to $O(\log n)$ and thus our algorithm runs in time $\widetilde{O}(T(n))$. We refer the reader to [11] and [1] for a formal proof. □