Faster Knapsack Algorithms via Bounded Monotone Min-Plus-Convolution

Karl Bringmann
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Alejandro Cassis
Universität des Saarlandes, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We present new exact and approximation algorithms for 0-1-Knapsack and Unbounded Knapsack:

- *Exact Algorithm for 0-1-Knapsack:* 0-1-Knapsack has known algorithms running in time $\widetilde{O}(n + \min\{n \cdot \mathrm{OPT}, n \cdot W, \mathrm{OPT}^2, W^2\})$ [Bellman '57], where n is the number of items, W is the weight budget, and OPT is the optimal profit. We present an algorithm running in time $\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$. This improves the running time in case n, W, OPT are roughly equal.
- *Exact Algorithm for Unbounded Knapsack:* Unbounded Knapsack has known algorithms running in time $\widetilde{O}(n + \min\{n \cdot p_{\max}, n \cdot w_{\max}, p_{\max}^2, w_{\max}^2\})$ [Axiotis, Tzamos '19, Jansen, Rohwedder '19, Chan, He '22], where n is the number of items, $w_{\max}$ is the largest weight of any item, and $p_{\max}$ is the largest profit of any item. We present an algorithm running in time $\widetilde{O}(n + (p_{\max} + w_{\max})^{1.5})$, giving a similar improvement as for 0-1-Knapsack.
- *Approximating Unbounded Knapsack with Resource Augmentation:* Unbounded Knapsack has a known FPTAS with running time $\widetilde{O}(\min\{n/\varepsilon, n + 1/\varepsilon^2\})$ [Jansen, Kraft '18]. We study *weak* approximation algorithms, which approximate the optimal profit but are allowed to overshoot the weight constraint (i.e. resource augmentation). We present the first approximation scheme for Unbounded Knapsack in this setting, achieving running time $\widetilde{O}(n + 1/\varepsilon^{1.5})$. Along the way, we also give a simpler FPTAS with lower order improvement in the standard setting.

For all of these problem settings the previously known results had matching conditional lower bounds. We avoid these lower bounds in the approximation setting by allowing resource augmentation, and in the exact setting by analyzing the time complexity in terms of weight and profit parameters (instead of only weight or only profit parameters).

Our algorithms can be seen as reductions to Min-Plus-Convolution on monotone sequences with bounded entries. These structured instances of Min-Plus-Convolution can be solved in time $\widetilde{O}(n^{1.5})$ [Chi, Duan, Xie, Zhang '22] (in contrast to the conjectured $n^{2-o(1)}$ lower bound for the general case). We complement our results by showing reductions in the opposite direction, that is, we show that achieving our results with the constant 1.5 replaced by any constant < 2 implies subquadratic algorithms for Min-Plus-Convolution on monotone sequences with bounded entries.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Knapsack, Approximation Schemes, Fine-Grained Complexity, Min-Plus Convolution

Digital Object Identifier 10.4230/LIPIcs.ICALP.2022.31

Category Track A: Algorithms, Complexity and Games

Related Version *Full Version*: `https://arxiv.org/abs/2205.08493`

Funding This work is part of the project TIPEA that has received funding from the European Research Council (ERC) under the European Unions Horizon 2020 research and innovation programme (grant agreement No. 850979).

1 Introduction

In this paper we present new exact and approximation algorithms for Knapsack problems.

Exact Pseudopolynomial Algorithms for Unbounded Knapsack. In the UNBOUNDED KNAPSACK problem we are given a set of n items, where each item has a weight w_i and a profit p_i, along with a knapsack capacity W. The goal is to find a *multiset* of items which maximizes the total profit and has total weight at most W. The textbook dynamic programming algorithm for UNBOUNDED KNAPSACK due to Bellman [6] runs in time $O(nW)$ or in time $O(n \cdot \mathrm{OPT})$, where OPT is the value of the optimal solution. Recent literature on UNBOUNDED KNAPSACK studies alternative parameters: Axiotis and Tzamos [3] and Jansen and Rohwedder [25] independently presented algorithms running in time $\widetilde{O}(w_{\max}^2)$ and $\widetilde{O}(p_{\max}^2)$,[1] where $w_{\max}$ is the largest weight of any item and $p_{\max}$ the largest profit of any item – in general $w_{\max}$ could be much smaller than W and $p_{\max}$ much smaller than OPT, so these algorithms improve upon Bellman's algorithm for some parameter settings. Chan and He [15] presented further improvements[2] achieving time $\widetilde{O}(n\, w_{\max})$ and $\widetilde{O}(n\, p_{\max})$ Note that when $w_{\max} \approx p_{\max} \approx n$ all mentioned algorithms require at least quadratic time $\Omega(n^2)$. Can we overcome this quadratic barrier? In this paper we answer this positively by considering the combined parameter $w_{\max} + p_{\max}$.

▶ **Theorem 1.** *UNBOUNDED KNAPSACK can be solved in expected time* $\widetilde{O}(n + (p_{\max} + w_{\max})^{1.5})$.

This result is particularly interesting in light of recent fine-grained lower bounds for UNBOUNDED KNAPSACK. Indeed, for each previous result that we have mentioned above, a matching conditional lower bound is known [19, 28]. For example, UNBOUNDED KNAPSACK cannot be solved in time $O((n + W)^{2-\delta})$ for any constant $\delta > 0$ under a plausible hypothesis. Inspecting these conditional lower bounds, we observe that they construct hard instances where only the profit parameters or only the weight parameters are under control; one of the two must be very large to obtain a hardness reduction. We thus avoid these conditional lower bounds by considering the combined profit and weight parameter $w_{\max} + p_{\max}$.

Exact Pseudopolynomial Algorithms for 0-1 Knapsack. The 0-1 KNAPSACK problem is the variant of UNBOUNDED KNAPSACK where every input item can appear at most once in any solution. Bellman's algorithm also solves 0-1 KNAPSACK in time $O(nW)$ or $O(n \cdot \mathrm{OPT})$. However, the landscape is more diverse when considering other parameters. In particular, it is open whether 0-1 KNAPSACK can be solved in time $\widetilde{O}(n + w_{\max}^2)$ or $\widetilde{O}(n + p_{\max}^2)$.[3] Table 1 shows a non-exhaustive list of pseudopolynomial-time algorithms for 0-1 KNAPSACK using different combinations of the parameters $n, w_{\max}, p_{\max}, W, \mathrm{OPT}$. Note that when all

[1] We use the notation $\widetilde{O}(T) = \cup_{c>0} O(T \log^c T)$ to supress polylogarithmic factors

[2] Note that we can assume that $n \leq w_{\max}$ without loss of generality since if there are multiple items with the same weight, we can keep only the one with the largest profit. Similarly, $n \leq p_{\max}$.

[3] This gap is analogous to the case of SUBSET SUM where we are given a set of numbers X and a target number t. For the unbounded case, where the goal is to find whether a multiset of items in X sums to t, Jansen and Rohwedder [25] gave an algorithm in time $\widetilde{O}(n + u)$ where u is the largest number in the input. For the more standard "0-1" case where we ask for a subset of X summing to t, the best known running times are $\widetilde{O}(n + t)$ [8, 26], $O(nu)$ [33], $\widetilde{O}(n + u^2/n)$ by combining [21] and [8, 26], and $\widetilde{O}(n + u^{3/2})$ by combining [21] and [8, 26] and [33]; see also [11, 34] for generalizations to X being a multiset and related results.

Table 1 Non-exhaustive list of pseudopolynomial-time algorithms for 0-1 Knapsack.

Running Time	Reference
$O(n \cdot \min\{W, \mathrm{OPT}\})$	[6]
$O(n \cdot p_{\max} \cdot w_{\max})$	[33]
$\widetilde{O}(n + w_{\max} \cdot W)$	[27, 5, 3]
$\widetilde{O}(n + p_{\max} \cdot W)$	[5]
$\widetilde{O}(n \cdot \min\{w_{\max}^2, p_{\max}^2\})$	[3]
$\widetilde{O}((n + W) \cdot \min\{w_{\max}, p_{\max}\})$	[5]
$O(n + \min\{w_{\max}^3, p_{\max}^3\})$	[34]
$\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$	**Theorem 2**

these parameters are bounded by $O(n)$, all existing algorithms require at least quadratic time $\Omega(n^2)$. In this paper we show that by considering the combined weight and profit parameter $W + \mathrm{OPT}$ we can overcome this quadratic barrier.

▶ **Theorem 2.** *There is a randomized algorithm for 0-1 Knapsack that runs in time $\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$ and succeeds with high probability.*

Similar to the unbounded case, matching conditional lower bounds ruling out time $O((n + W)^{2-\delta})$ and $O((n + \mathrm{OPT})^{2-\delta})$ for any $\delta > 0$ are known [19, 28]. These lower bounds construct hard instances where only one of W, OPT is under control, the other needs to be very large. We thus avoid these lower bounds by considering the combined weight and profit parameter $W + \mathrm{OPT}$.

Approximation Schemes for Unbounded Knapsack. Since Unbounded Knapsack is well known to be NP-hard, it is natural to study approximation algorithms. In particular, a *fully polynomial-time approximation scheme* (FPTAS) given $0 < \varepsilon < 1$ computes a solution x with total weight $w(x) \le W$ and total profit $p(x) \ge (1 - \varepsilon)\mathrm{OPT}$ in time $\mathrm{poly}(n, 1/\varepsilon)$. The first FPTAS for Unbounded Knapsack was designed by Ibarra and Kim in 1975 [23] and runs in time $\widetilde{O}(n + (1/\varepsilon)^4)$. In 1979 Lawler [30] improved the running time to $\widetilde{O}(n + (1/\varepsilon)^3)$. This was the best known until Jansen and Kraft in 2018 [24] presented an FPTAS running in time $\widetilde{O}(n + (1/\varepsilon)^2)$. This algorithm has a matching conditional lower ruling out time $O((n + 1/\varepsilon)^{2-\delta})$ for any $\delta > 0$ [19, 28, 32].

We present a new FPTAS for Unbounded Knapsack which is (as we believe) simpler than Jansen and Kraft's, and has a lower order improvement in the running time:

▶ **Theorem 3.** *Unbounded Knapsack has an FPTAS with running time $\widetilde{O}\left(n + \frac{(1/\varepsilon)^2}{2^{\Omega(\sqrt{\log(1/\varepsilon)})}}\right)$.*

Bringmann and Nakos [10] recently gave an FPTAS for the related Subset Sum problem which achieves the same running time.

Weak Approximation for Unbounded Knapsack. Motivated by the matching upper and conditional lower bounds for FPTASs for Unbounded Knapsack, we study the relaxed notion of *weak approximation* as coined in [32]: we relax the weight constraint and seek a solution x with total weight $w(x) \le (1 + \varepsilon) \cdot W$ and total profit $p(x) \ge (1 - \varepsilon)\mathrm{OPT}$. Note that OPT is still the optimal value of any solution with weight at most W. This can be interpreted as bicriteria approximation (approximating both the weight and profit constraint) or as resource augmentation (the optimal algorithm is allowed weight W while our algorithm is allowed a slightly larger weight of $(1+\varepsilon)\cdot W$). All of these are well-established relaxations of the standard (=strong[4]) notion of approximation. Such weaker notions of approximation are typically studied when a PTAS for the strong notion of approximation is not known. More generally, studying these weaker notions is justified whenever there are certain limits for strong approximations, to see whether these limits can be overcome by relaxing the notion of approximation. In particular, we want to understand whether this relaxation can overcome the conditional lower bound ruling out time $O((n+1/\varepsilon)^{2-\delta})$. For the related Subset Sum problem this question has been resolved positively: Bringmann and Nakos [10] conditionally ruled out strong approximation algorithms in time $O((n+1/\varepsilon)^{2-\delta})$ for any $\delta > 0$, but Mucha, Węgrzycki and Włodarczyk [32] designed a weak FPTAS in time $\widetilde{O}(n+(1/\varepsilon)^{5/3})$.

In this paper, we give a positive answer for Unbounded Knapsack:

▶ **Theorem 4.** *Unbounded Knapsack has a weak approximation scheme running in expected time* $\widetilde{O}(n+(\frac{1}{\varepsilon})^{1.5})$.

Our theorem gives reason to believe that resource augmentation indeed makes the problem easier. Specifically, obtaining a strong approximation scheme with the same running time as our weak one would refute the existing conditional lower bound for strong approximation.

Min-Plus-Convolution. All conditional lower bounds mentioned above are based on a hypothesis about the MinConv problem: Given sequences $A, B \in \mathbb{Z}^n$ compute their $(\min,+)$-convolution, which is the sequence $C \in \mathbb{Z}^{2n}$ with $C[k] = \min_{i+j=k} A[i]+B[j]$.[5] The MinConv problem can be trivially solved in time $O(n^2)$. This can be improved to time $n^2/2^{\Omega(\sqrt{\log n})}$ via a reduction to $(\min,+)$-matrix product due to Bremner et al. [7], and using Williams' algorithm for the latter [37] (which was derandomized later by Chan and Williams [17]). The lack of truly subquadratic algorithms despite considerable effort has led researchers to postulate the MinConv hypothesis, namely that MinConv cannot be solved in time $O(n^{2-\delta})$ for any constant $\delta > 0$ [19, 28]. Many problems are known to have conditional lower bounds from the MinConv hypothesis, see, e.g., [4, 14, 19, 25, 28, 29].

Central to our work is a reduction from MinConv to Unbounded Knapsack shown independently by Cygan et al. [19] and Künnemann et al. [28]. In particular, they showed that if Unbounded Knapsack on n items and $W = O(n)$ can be solved in subquadratic time, then MinConv can be solved in subquadratic time. This reduction immediately implies matching conditional lower bounds for the previously known exact algorithms with running times $O(n \cdot W)$, $\widetilde{O}(w_{\max}^2)$ and $\widetilde{O}(n \cdot w_{\max})$, as mentioned earlier.

A small modification of this reduction extends to the dual case, i.e., an exact subquadratic-time algorithm for Unbounded Knapsack with $\mathrm{OPT} = O(n)$ would result in a subquadratic-time algorithm for MinConv. This establishes matching conditional lower bounds for the algorithms in time $O(n\cdot\mathrm{OPT})$, $\widetilde{O}(p_{\max}^2)$ and $\widetilde{O}(n\cdot p_{\max})$. Moreover, by setting $\varepsilon = \Theta(1/\mathrm{OPT})$, an FPTAS for Unbounded Knapsack would yield an exact algorithm for MinConv, establishing that the $\widetilde{O}(n+(1/\varepsilon)^2)$-time FPTAS is conditionally optimal. This last observation was pointed out in [32].

[4] From now on, by "strong" approximation we mean the standard (non-weak) notion of approximation.
[5] If we replace the min by a max, we obtain the MaxConv problem, which is equivalent to MinConv after negating the sequences. Therefore, we will sometimes use these two names interchangeably.

In fact, the reduction due to Cygan et al. [19] is even an equivalence, showing that Unbounded Knapsack is intimately connected to MinConv. The same reduction is also known for 0-1 Knapsack [19, 28], so a similar discussion applies to 0-1 Knapsack.

Bounded Monotone Min-Plus-Convolution. Despite its postulated hardness, there are restricted families of instances of MinConv for which subquadratic algorithms are known, see e.g. [9, 12, 16, 18]. Central to this paper is the case when the input sequences $A, B \in \mathbb{Z}^n$ are monotone non-decreasing and have entries bounded by $O(n)$; we call this task BMMinConv.

In a celebrated result, Chan and Lewenstein gave an algorithm for BMMinConv that runs in expected time $O(n^{1.859})$ [16]. As a big hammer, their algorithm uses the famous Balog-Szemerédi-Gowers theorem from additive combinatorics. Very recently, Chi, Duan, Xie and Zhang showed how to avoid this big hammer and improved the running time to expected $\widetilde{O}(n^{1.5})$ [18].

All of our results mentioned so far use this BMMinConv algorithm as a subroutine. That is, our algorithms are reductions from various Knapsack problems to BMMinConv.

Equivalence with Bounded Monotone Min-Plus-Convolution. We complement our results by showing reductions in the opposite direction: Following the same chain of reductions as in [19, 28] but starting from bounded monotone instances of $(\min,+)$-convolution, we reduce BMMinConv to $O(n)$ instances of Unbounded Knapsack with $O(\sqrt{n})$ items each, where it holds that $w_{\max}, p_{\max}, W$ and OPT are all bounded by $O(\sqrt{n})$. Instantiating this reduction for the exact and approximate setting, we show the following theorem.

▶ **Theorem 5** (Equivalence). *For any problems A and B from the following list, if A can be solved in time $\widetilde{O}(n^{2-\delta})$ for some $\delta > 0$, then B can be solved in randomized time $\widetilde{O}(n^{2-\delta/2})$:*

1. *BMMaxConv on sequences of length n*
2. *Unbounded Knapsack on n items and $W, \mathrm{OPT} = O(n)$*
3. *0-1 Knapsack on n items and $W, \mathrm{OPT} = O(n)$*
4. *Weak $(1+\varepsilon)$-approximation for Unbounded Knapsack on n items and $\varepsilon = \Theta(1/n)$*

On the one hand, Theorem 1 solves Unbounded Knapsack in time $\widetilde{O}(n + (p_{\max} + w_{\max})^{1.5})$ by using Chi, Duan, Xie and Zhang's subquadratic BMMinConv algorithm [18]. On the other hand, Theorem 5 shows that any algorithm solving Unbounded Knapsack in time $\widetilde{O}(n + (p_{\max} + w_{\max})^{2-\delta})$ can be transformed into a subquadratic BMMinConv algorithm. This shows that both our exact and approximation algorithms take essentially *the only possible route* to obtain subquadratic algorithms, by invoking a BMMinConv algorithm.

Is randomness necessary? The algorithms given by Theorems 1, 2 and 4 are all randomized. If we insist on deterministic algorithms, we note that by applying Chan and Lewenstein's deterministic $\widetilde{O}(n^{1.864})$-time algorithm for BMMaxConv [16], we can obtain deterministic versions of Theorem 1 and Theorem 4 with exponent 1.864 instead of 1.5 (i.e. the only part where we use randomness is in applying Chi, Duan, Xie and Zhang's algorithm [18]). On the other hand, we do not know[6] how to derandomize Theorem 2.

[6] Our algorithm closely follows Bringmann's algorithm for Subset Sum [8] whose derandomization is an open problem.

Organization of the extended abstract. Due to space constraints, we only include the details of our exact algorithm for Unbounded Knapsack and 0-1 Knapsack in the main part of this manuscript. We defer our approximation schemes and the equivalence between BMMinConv and knapsack problems to the full version.

1.1 Technical Overview

Exact algorithm for Unbounded Knapsack. In Section 3 we present our exact algorithm for Unbounded Knapsack. Let $(\mathcal{I}, W)$ be an instance of Unbounded Knapsack. We denote by $\mathcal{P}_i[0,\dots,W]$ the array where $\mathcal{P}_i[j]$ is the maximum profit of a solution of weight at most j using at most 2^i items. Since each item has weight at least 1, any feasible solution consists of at most W items. Thus, our goal is to compute the value $\mathcal{P}_{\lceil \log W \rceil}[W] = \mathrm{OPT}$.

The natural approach is to use dynamic programming: since $\mathcal{P}_0$ consists of solution of at most one item, it can be computed in time $O(n)$. For $i > 0$ we can compute $\mathcal{P}_i[0,\dots,W]$ by taking the $(\max,+)$-convolution of $\mathcal{P}_{i-1}[0,\dots,W]$ with itself. This gives an algorithm in time $O(W^2 \log W)$.

Jansen and Rohwedder [25] and Axiotis and Tzamos [3] showed that instead of convolving sequences of length W, it suffices to convolve only $O(w_{\max})$ entries of $\mathcal{P}_{i-1}$ in each iteration. This improves the running time to $O(w_{\max}^2 \log W)$ by using the naive algorithm for $(\max,+)$-convolution. The approach of Jansen and Rohwedder [25] is as follows. Suppose x is the optimal solution for a target value $\mathcal{P}_i[j]$. They showed that x can be split into two solutions x_1, x_2 such that (i) the number of items in each part is at most 2^{i-1} and (ii) the difference between the weights of both parts is at most $O(w_{\max})$. Thus, (i) guarantees that both x_1 and x_2 are optimal solutions for two entries of $\mathcal{P}_{i-1}$, and (ii) implies that these entries lie in an interval in $\mathcal{P}_{i-1}$ of length $O(w_{\max})$. In this way, they can afford to perform the $(\max,+)$-convolution of only $O(w_{\max})$ entries in $\mathcal{P}_{i-1}$.

To show the existence of such a partitioning of x they make use of Steinitz' Lemma [36], which shows that any collection of m vectors in $\mathbb{R}^d$ with infinity norm at most 1, whose sum is 0, can be permuted such that every prefix sum has norm at most $O(d)$ (see Lemma 11 for the precise statement). The partitioning of x follows from Steinitz' Lemma by taking the weights of the items picked by x as 1-dimensional vectors. The usage of Steinitz' Lemma to reduce the number of states in dynamic programs was pioneered by Eisenbrand and Weismantel [20], and further refined by Jansen and Rohwedder [25].

In our algorithm, we use Steinitz' Lemma in a similar way to split the number of items and the weight of x, but additionally we use it to ensure that the profits of the solutions x_1, x_2 differ by at most $O(p_{\max})$ (see Lemma 12). In this way, by carefully handling the subproblems $\mathcal{P}_{i-1}$ we can enforce that the values of the $O(w_{\max})$ entries that need to be convolved have values in a range of size $O(p_{\max})$. Since the arrays $\mathcal{P}_i$ are monotone non-decreasing, we can then apply the algorithm for BMMaxConv, and thus handle each subproblem in time $\widetilde{O}((p_{\max} + w_{\max})^{1.5})$.

Exact algorithm for 0-1 Knapsack. Cygan et al. [19] showed that there is a reduction from 0-1 Knapsack to MaxConv. More precisely, they showed that if MaxConv can be solved in time $T(n)$, then 0-1 Knapsack can be solved in randomized time $\widetilde{O}(T(W))$. Their reduction is a generalization of Bringmann's Subset Sum algorithm [8], which can be seen as a reduction from Subset Sum to Boolean convolution. Cygan et al. showed that the reduction for 0-1 Knapsack can be obtained by essentially replacing the Boolean convolutions by $(\max,+)$-convolutions in Bringmann's algorithm.

In Section 4 we observe that this reduction produces instances of MaxConv which are monotone non-decreasing and have entries bounded by OPT. That is, we obtain BMMaxConv instances of size $O(W + \text{OPT})$, and following the analysis of [19] we obtain an algorithm for 0-1 Knapsack in time $\widetilde{O}(n + (W + \text{OPT})^{1.5})$, which yields Theorem 2.

Approximating Unbounded Knapsack. Let $(\mathcal{I}, W)$ be an instance of Unbounded Knapsack. Consider the array $\mathcal{P}[0, \dots, W]$ where $\mathcal{P}[j]$ is the maximum profit of a solution with weight at most j. In particular, $\mathcal{P}[W] = \text{OPT}$ is the optimal value of the instance. We will use that the sequence $\mathcal{P}[0, \dots, W]$ is monotone.

We present the following simple algorithm to compute $\mathcal{P}[0, \dots, W]$, which is based on an algorithm for (unbounded) Subset Sum from Bringmann [8]. Fix an optimal solution x for an entry $\mathcal{P}[j]$. We can split x into three smaller solutions x_1, x_2, x_3 such that the total weight of both x_1 and x_2 is at most $j/2$, and x_3 is consists of at most one item. It follows that if we know $\mathcal{P}[0, \dots, j/2]$, then we can compute $\mathcal{P}[0, \dots, j]$ by taking the $(\max, +)$-convolution of $\mathcal{P}[0, \dots, j/2]$ with itself and possibly adding one extra item from $\mathcal{I}$. In particular, if we compute all entries $\mathcal{P}[0, \dots, W/n]$ as a base case using dynamic programming in time $O(W)$, then we can compute the entire sequence $\mathcal{P}[0, \dots, W]$ by applying $O(\log n)$ $(\max, +)$-convolutions on sequences of length at most W. The overall running time is $O(W^2 \log n)$.

Although this exact algorithm is not particularly exciting or new, it can be nicely extended to the approximate setting. We show that by replacing the exact $(\max, +)$-convolutions with approximate ones, we obtain an FPTAS for Unbounded Knapsack in time $\widetilde{O}(n + 1/\varepsilon^2)$. To this end, we preprocess the item set to get rid of *light* items with weight smaller than $\varepsilon \cdot W$, and *cheap* items with profit smaller than $\varepsilon \cdot \text{OPT}$, while decreasing the optimal value by only $O(\varepsilon \cdot \text{OPT})$. We now proceed as in the exact case, starting with the base case $\mathcal{P}[0, \dots, \varepsilon \cdot W]$, which is all-zeroes since there are no more light items. Then we can build up $\mathcal{P}[0, \dots, 2^j \varepsilon W]$ for increasing values of j by performing approximate $(\max, +)$-convolutions, until we have computed $\mathcal{P}[0, \dots, W]$. For approximating $(\max, +)$-convolutions we use an algorithm due to Chan [13], which in our setting without cheap items runs in time $\widetilde{O}(1/\varepsilon^2)$. Thus, after applying the preprocessing in time $O(n)$, we compute a $(1 + \varepsilon)$-approximation of $\mathcal{P}[0, \dots, W]$ by applying $O(\log 1/\varepsilon)$ approximate $(\max, +)$-convolutions in overall time $\widetilde{O}(n + (1/\varepsilon)^2)$.

Then, we treat the case of weak approximation. The main steps of the algorithm are virtually the same as before. The crucial difference is that now we can afford to round weights. In this way, we can adapt Chan's algorithm and construct MaxConv instances which are monotone non-decreasing and have bounded entries. This yields BMMaxConv instances, and by using Chi, Duan, Xie and Zhang's algorithm for this special case [18], we can compute a weak approximation of $(\max, +)$-convolution in time $\widetilde{O}((1/\varepsilon)^{1.5})$. By similar arguments as for the strong approximation, this yields a weak approximation scheme running in time $\widetilde{O}(n + (1/\varepsilon)^{1.5})$.

Equivalence between BMMinConv and Knapsack problems. As mentioned earlier in the introduction, Cygan et al. [19] and Künnemann et al. [28] independently showed a reduction from MinConv to Unbounded Knapsack. In the full version of the paper, we show that following the same chain of reductions from MaxConv to Unbounded Knapsack but instead starting from BMMaxConv, with minor adaptations we can produce instances of Unbounded Knapsack with $W, \text{OPT} = O(n)$. Together with our exact algorithm for Unbounded Knapsack, which we can phrase as a reduction to BMMaxConv, we obtain an equivalence of BMMaxConv and Unbounded Knapsack with $W, \text{OPT} = O(n)$ – if one of these problems can be solved in subquadratic time, then both can.

Note that for Unbounded Knapsack with $W, \mathrm{OPT} = O(n)$ a weak $(1+\varepsilon)$-approximation for $\varepsilon = \Theta(1/n)$ already computes an exact optimal solution. This yields the reduction from BMMaxConv to the approximate version of Unbounded Knapsack. We similarly obtain a reduction to 0-1 Knapsack with $W, \mathrm{OPT} = O(n)$. This yields our equivalences from Theorem 5.

2 Preliminaries

We write $\mathbb{N} = \{0, 1, 2, \dots\}$. For $t \in \mathbb{N}$ we let $[t] = \{0, 1, \dots, t\}$. For reals $a \le b$ we write $[a, b]$ for the interval from a to b, and for reals a, b with $b \ge 0$ we write $[a \pm b]$ for the interval $[a - b, a + b]$.

We formally define the Unbounded Knapsack problem. We are given a set of items $\mathcal{I} = \{(p_1, w_1), \dots, (p_n, w_n)\}$, where each item i has a profit $p_i \in \mathbb{N}$ and a weight $w_i \in \mathbb{N}$, and a knapsack capacity $W \in \mathbb{N}$. The task is to maximize $\sum_{i=1}^n p_i \cdot x_i$ subject to the constraints $\sum_{i=1}^n w_i \cdot x_i \le W$ and $x \in \mathbb{N}^n$. The more standard 0-1 Knapsack problem is defined in the same way, but the solution x is constrained to $x \in \{0, 1\}^n$.

Given an instance $(\mathcal{I}, W)$, we denote by $x \in \mathbb{N}^n$ a multiset of items, where x_i is the number of copies of the i-th item. We sometimes refer to x as a *solution*. We write $p_{\mathcal{I}}(x)$ for the total profit of x, i.e., $p_{\mathcal{I}}(x) := \sum_i x_i \cdot p_i$. Similarly, we write $w_{\mathcal{I}}(x) := \sum_i x_i \cdot w_i$ for the weight of x. When the item set $\mathcal{I}$ is clear from context, we drop the subscript and simply write $p(x)$ and $w(x)$. We denote the number of items contained in a solution x by $\|x\|_1 := \sum_i x_i$. A solution x is *feasible* if it satisfies the constraint $w(x) \le W$. We denote by OPT the maximum profit $p(x)$ of any feasible solution x. We denote by $p_{\max} := \max_{(p,w) \in \mathcal{I}} p$ the maximum profit of any input item and by $w_{\max} := \max_{(p,w) \in \mathcal{I}} w$ the maximum weight of any input item.

Notions of Approximation. We say that an algorithm gives a *strong* $(1+\varepsilon)$-approximation for Unbounded Knapsack if it returns a solution $x \in \mathbb{N}^n$ with weight $w(x) \le W$ and profit $p(x) \ge (1 - \varepsilon) \cdot \mathrm{OPT}$. We say that an algorithm gives a *weak* $(1 + \varepsilon)$-approximation for Unbounded Knapsack if it returns a solution $x \in \mathbb{N}^n$ with profit $p(x) \ge (1 - \varepsilon) \cdot \mathrm{OPT}$ and weight $w(x) \le (1 + \varepsilon) \cdot W$. We stress that here OPT still denotes the optimum value with weight at most W, i.e., $\mathrm{OPT} = \max\{p(x) \mid x \in \mathbb{N}^n,\ w(x) \le W\}$.

Profit Sequences. Given an item set $\mathcal{I}$ and capacity W, we define the array $\mathcal{P}_{\mathcal{I}}[0, \dots, W]$, where $\mathcal{P}_{\mathcal{I}}[j]$ is the maximum profit achievable with capacity j, i.e.,

$$\mathcal{P}_{\mathcal{I}}[j] := \max\{p_{\mathcal{I}}(x) \colon x \in \mathbb{N}^n, w_{\mathcal{I}}(x) \le j\}.$$

Note that $\mathcal{P}_{\mathcal{I}}[0] = 0$. A textbook way to compute $\mathcal{P}_{\mathcal{I}}[0, \dots, W]$ is by dynamic programming:

▶ **Fact 6.** *$\mathcal{P}_{\mathcal{I}}[0, \dots, W]$ can be computed using dynamic programming in time $O(n \cdot W)$.*

We will also consider the array $\mathcal{P}_{\mathcal{I},k}[0, \dots, W]$, where we restrict to solutions with at most 2^k items, for any non-negative integer k, i.e., for any $j \in [W]$ we set

$$\mathcal{P}_{\mathcal{I},k}[j] := \max\{p_{\mathcal{I}}(x) \colon x \in \mathbb{N}^n, w(x) \le j, \|x\|_1 \le 2^k\}.$$

When $\mathcal{I}$ is clear from context, we will drop the subscript and write $\mathcal{P}[0, \dots, W]$ and $\mathcal{P}_k[0, \dots, W]$. When we work with 0-1 Knapsack instead of Unbounded Knapsack, we will use the same notation $\mathcal{P}_{\mathcal{I}}$ and $\mathcal{P}_{\mathcal{I},k}$, where we restrict to $x \in \{0, 1\}^n$ instead of $x \in \mathbb{N}^n$.

MaxConv. The $(\max,+)$-convolution $A \oplus B$ of two sequences $A[0,\dots,n], B[0,\dots,n] \in \mathbb{Z}^{n+1}$ is a sequence of length $2n+1$ where $(A \oplus B)[k] := \max_{i+j=k} A[i] + B[j]$. We call MaxConv the task of computing the $(\max,+)$-convolution of two given sequences.

We will use the following handy notation: Given sequences $A[0,\dots,n], B[0,\dots,n]$ and intervals $I, J \subseteq [n]$ and $K \subseteq [2n]$, we denote by $C[K] := A[I] \oplus B[J]$ the computation of $C[k] := \max\{A[i] + B[j] : i \in I, j \in J, i+j=k\}$ for each $k \in K$. The following proposition shows that this can be computed efficiently. We defer the proof to Appendix A.

▶ **Proposition 7.** *If MaxConv can be solved in time $T(n)$, then $C[K] = A[I] \oplus B[J]$ can be computed in time $O(T(|I|+|J|) + |K|)$.*

Sometimes we will refer to the $(\min,+)$-convolution, where we replace max by a min. The two problems MinConv and MaxConv are equivalent after negating the sequences.

BMMaxConv. In the BMMaxConv problem, we compute the $(\max,+)$-convolution of sequences of length n which are monotone non-decreasing and have bounded values. For this setting Chan and Lewenstein [16] gave the first subquadratic algorithm, and recently Chi, Duan, Xie and Zhang gave the following remarkable result:

▶ **Theorem 8** (BMMaxConv [18])**.** *Given monotone non-decreasing sequences $A[0,\dots,n]$ and $B[0,\dots,n]$ with entries $A[i], B[i] \in [O(n)] \cup \{-\infty\}$ for all $i \in [n]$, their $(\max,+)$-convolution $A \oplus B$ can be computed in expected time $\widetilde{O}(n^{1.5})$.*

Note that Chi, Duan, Xie and Zhang phrase their result for $(\min,+)$-convolution of monotone increasing sequences with entries in $[O(n)]$. We prove in Appendix A that both statements are equivalent, so their result also works for $(\max,+)$-convolution with entries in $[O(n)] \cup \{-\infty\}$.

Witnesses. Let $A[0,\dots,n], B[0,\dots,n]$ be an instance of MaxConv. Let $C := A \oplus B$. Given $k \in [2n]$, we say that $i \in [n]$ is a *witness* for $C[k]$ if $C[k] = A[i] + B[k-i]$. We say that an array $M[0.\dots,2n]$ is a *witness array*, if each entry $M[k]$ contains some witness for $C[k]$.

For the general case of MaxConv it is well known (e.g. [35, 1]) that computing the witness array has the same time complexity as $(\max,+)$-convolution, up to a $\text{polylog}(n)$ overhead. This reduction does not immediately apply to BMMaxConv because the sequences might not remain monotone. However, we make it work with some extra care, see Appendix B for the proof.

▶ **Lemma 9** (Witness Finding)**.** *If BMMaxConv can be computed in time $T(n)$, then a witness array $M[0,\dots,2n]$ can be computed in time $\widetilde{O}(T(n))$.*

Niceness asumptions on time bounds. For all time bounds $T(n)$ in this paper, we make the following niceness assumptions: (1) $T(\widetilde{O}(n)) \le \widetilde{O}(T(n))$, and (2) $k \cdot T(n) \le O(T(kn))$ for any $k, n \ge 1$. This is satisfied for all natural time bounds of polynomial-time or pseudopolynomial-time algorithms, in particular it holds for all functions of the form $T(n) = \Theta(n^\alpha \log^\beta n)$ for any constants $\alpha \ge 1, \beta \ge 0$.

3 Exact algorithm for Unbounded Knapsack

In this section we prove the following Theorem:

▶ **Theorem 10.** *If* BMMaxConv *on length-n sequences can be solved in time $T(n)$, then* Unbounded Knapsack *can be solved in time $\widetilde{O}(n + T(p_{\max} + w_{\max}))$, where $p_{\max}$ is the largest profit of any item and $w_{\max}$ is the largest weight of any item.*

Note that Theorem 1 follows as an immediate corollary of Theorem 10 by plugging in Chi, Duan, Xie and Zhang's algorithm (Theorem 8).

For the entire section, fix an instance $(\mathcal{I}, W)$ of the Unbounded Knapsack problem. Recall that $\mathcal{P}_i[0, \dots, W]$ is defined as $\mathcal{P}_i[j] := \max\{p(x) \colon w(x) \leq j, \|x\|_1 \leq 2^i\}$, and set $\Delta := p_{\max} + w_{\max}$. Suppose we know that the optimal solution consists of at most 2^k items. Then, our goal is to compute the value $\mathcal{P}_k[W]$. The natural approach is to use dynamic programming: if we have computed $\mathcal{P}_{i-1}$, then $\mathcal{P}_i = \mathcal{P}_{i-1} \oplus \mathcal{P}_{i-1}$. To get our desired running time, we will show that we only need to convolve $O(\Delta)$ entries of $\mathcal{P}_{i-1}$ and that we can enforce that all of these fall in a range of $O(\Delta)$ values. By monotonicity of $\mathcal{P}_{i-1}$, we end up with a BMMaxConv instance, which can be solved in time $O(T(\Delta))$. The resulting total time to compute $\mathcal{P}_i[W]$ is $O(n + k \cdot T(\Delta))$. With additional preprocessing we ensure that $k = O(\log \Delta)$, turning the running time into $\widetilde{O}(n + T(\Delta))$.

3.1 Preparations

We need to show that when computing the optimal answer for some entry $\mathcal{P}_i[j]$, we can split it in such a way that both its total profit and its total weight are roughly halved. Our main tool to show this is the Steinitz Lemma [22, 36]. A beautiful proof for it can be found in [31].

▶ **Lemma 11** ([22, 36, Steinitz Lemma]). *Let $\|.\|$ be a norm in $\mathbb{R}^m$ and let M be an arbitrary collection of t vectors in $\mathbb{R}^m$ such that $\|v\| \leq 1$ for every $v \in M$ and $\sum_{v \in M} v = 0$. Then, it is possible to permute the vectors in M into a sequence $(v_1, \dots, v_t)$ such that $\|v_1 + \dots + v_k\| \leq m$ holds for every $k \in [t]$.*

We use the Steinitz Lemma to argue that the items in a solution can be split in two parts in such a way that both the total profit and the total weight are roughly halved:

▶ **Lemma 12** (Splitting Lemma). *Let $i \geq 1$ and consider a solution $x \in \mathbb{N}^n$ with $\|x\|_1 \leq 2^i$. Then there is a partition of x into two solutions $x_1, x_2 \in \mathbb{N}^n$ with the following properties:*
1. *(Splitting of Items) $\|x_1\|_1, \|x_2\|_1 \leq 2^{i-1}$ and $x = x_1 + x_2$,*
2. *(Approximate Splitting of Weight) $|w(x_1) - \frac{1}{2}w(x)| \leq 2\Delta$ and $|w(x_2) - \frac{1}{2}w(x)| \leq 2\Delta$,*
3. *(Approximate Splitting of Value) $|p(x_1) - \frac{1}{2}p(x)]| \leq 2\Delta$ and $|p(x_2) - \frac{1}{2}p(x)]| \leq 2\Delta$.*

Proof. Let $t := \|x\|_1 \leq 2^i$. First assume that t is even; we will remove this assumption later. Write $x = \sum_{j=1}^{t} x^{(j)}$ where each $x^{(j)}$ corresponds to one copy of some item, i.e. $\|x^{(j)}\|_1 = 1$, and set $v^{(j)} = \binom{w(x^{(j)})}{p(x^{(j)})}$. Note that $\|v^{(j)}\|_\infty \leq \Delta$. By applying the Steinitz Lemma on the vectors $v^{(j)} - \frac{1}{t}\binom{w(x)}{p(x)}$ (after normalizing by Δ), we can assume that the $v^{(j)}$'s are ordered such that

$$\left\| \sum_{j=1}^{t/2} v^{(j)} - \frac{1}{2}\binom{w(x)}{p(x)} \right\|_\infty \leq 2\Delta. \tag{1}$$

Fix this ordering, and let $x_1 = x^{(1)} + \ldots + x^{(t/2)}$, corresponding to $v^{(1)}, \ldots, v^{(t/2)}$, and let $x_2 = x^{(t/2+1)} + \ldots + x^{(t)}$, corresponding to the remaining vectors $v^{(t/2+1)}, \ldots, v^{(t)}$. We now check that x_1, x_2 satisfy the properties of the lemma:

- Property 1 is clearly satisfied by construction.
- For property 2, note that (1) implies $|w(x_1) - \frac{1}{2}w(x)| \leq 2\Delta$. Since $w(x_2) = w(x) - w(x_1)$, we have that $|w(x_2) - \frac{1}{2}w(x)| = |\frac{1}{2}w(x) - w(x_1)| \leq 2\Delta$.
- Property 3 follows in the same way as property 2.

If t is odd, then $t+1 \leq 2^i$, so we can add a dummy vector $x^{(t+1)} = 0$ with corresponding $v^{(t+1)} := \binom{0}{0}$ and repeat the same argument with $t := t+1$. ◀

When we apply Lemma 12 to an optimal solution corresponding to an entry of the array $\mathcal{P}_i$, we obtain the following lemma.

▶ **Lemma 13.** *Let $\beta > 0$. For any index $j \in [\beta \pm 8\Delta] \cap [W]$ there are indices $j_1, j_2 \in [\frac{\beta}{2} \pm 8\Delta] \cap [W]$ with the following properties:*

(i) $j_1 + j_2 = j$,
(ii) $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2]$,
(iii) $|\mathcal{P}_{i-1}[j_1] - \frac{1}{2}\mathcal{P}_i[j]| \leq 2\Delta$ *and* $|\mathcal{P}_{i-1}[j_2] - \frac{1}{2}\mathcal{P}_i[j]| \leq 2\Delta$.

Proof. Let $x \in \mathbb{N}^n$ be an optimal solution for $\mathcal{P}_i[j]$, that is, we have $p(x) = \mathcal{P}_i[j]$, $w(x) \leq j$, and $\|x\|_1 \leq 2^i$. We apply Lemma 12 to x and obtain $x_1, x_2 \in \mathbb{N}^n$ such that $x_1 + x_2 = x$. We do a case distinction based on $w(x_1), w(x_2)$:

- $w(x_1), w(x_2) \in [\frac{j}{2} \pm 4\Delta]$: Let $j_1 := w(x_1)$ and $j_2 := j - j_1$; note that $j_1, j_2 \in [\frac{j}{2} \pm 4\Delta] \subseteq [\frac{\beta}{2} \pm 8\Delta]$. We argue that $p(x_1) = \mathcal{P}_{i-1}[j_1]$ and $p(x_2) = \mathcal{P}_{i-1}[j_2]$. Indeed, since $w(x_1) = j_1$ the solution x_1 is feasible for weight j_1, so $p(x_1) \leq \mathcal{P}_{i-1}[j_1]$. Similarly, since $w(x_2) = w(x) - w(x_1) \leq j - w(x_1) = j_2$ the solution x_2 is feasible for weight j_2, so $p(x_2) \leq \mathcal{P}_{i-1}[j_2]$. Moreover, by optimality of x we have $p(x_1) + p(x_2) = p(x) = \mathcal{P}_i[j] \geq \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2]$, so we obtain $p(x_1) = \mathcal{P}_{i-1}[j_1]$ and $p(x_2) = \mathcal{P}_{i-1}[j_2]$. Using these equations together with $p(x) = \mathcal{P}_i[j]$, property (ii) follows from $p(x) = p(x_1) + p(x_2)$, property (iii) follows from Property 3 of Lemma 12, and property (i) holds by definition of j_2.
- $w(x_1) < \frac{j}{2} - 4\Delta$: Property 2 of Lemma 12 implies that $|w(x_1) - w(x_2)| \leq 4\Delta$, and thus $w(x_2) \leq \frac{j}{2}$. Therefore, x_1 and x_2 are feasible for weights $j_1 := \lfloor \frac{j}{2} \rfloor$ and $j_2 := \lceil \frac{j}{2} \rceil$, respectively. Note that $j_1, j_2 \in [\frac{\beta}{2} \pm (4\Delta + 1)] \subseteq [\frac{\beta}{2} \pm 8\Delta]$. Property (i) is obvious, and properties (ii) and (iii) now follow as in the first case.
- $w(x_1) > \frac{j}{2} + 4\Delta$: Similarly as the previous case, property 2 of Lemma 12 implies that $w(x_2) \geq \frac{j}{2}$. Therefore, we have $w(x) = w(x_1) + w(x_2) > j + 4\Delta$, which contradicts the assumption $w(x) \leq j$.
- $w(x_2) < \frac{j}{2} - 4\Delta$ or $w(x_2) > \frac{j}{2} + 4\Delta$: Symmetric to the previous two cases. ◀

3.2 The algorithm

We are now ready to present our algorithm. The idea is to use the Splitting Lemma 12 to convolve smaller sequences which are bounded and monotone.

Let $\mathcal{I} = \{(w_i, p_i)\}_{i=1}^n$ with capacity constraint W be an instance of UNBOUNDED KNAPSACK. Since any item has $w_i \geq 1$, we know that any solution $x \in \mathbb{N}^n$ consists of at most W items. Thus, to compute the value of the optimal solution it suffices to compute $\mathcal{P}_k[W]$ where $k := \lceil \log W \rceil$.

Our approach is as follows. We do binary search for OPT in the range $[p_{\max} \cdot W]$. Suppose we have the current guess α. Instead of computing the arrays $\mathcal{P}_i$, we compute *clipped versions*, i.e., C_i which has the property that $C_i[j] \geq \alpha$ if and only if $\mathcal{P}_i[j] \geq \alpha$.

We compute C_i as follows: At every step, we only compute the values for $O(\Delta)$ weights $C_i[W \cdot 2^{i-k} \pm 8\Delta]$. For the base case $i = 0$, we simply set $C_0[0, \dots, 8\Delta] := \mathcal{P}_0[0, \dots, 8\Delta]$. Note that this can be done in time $O(n + \Delta)$ by doing one pass over the item set, since $\mathcal{P}_0$ only considers solutions with at most one item. Moreover, observe that $C_0[0, \dots, 8\Delta]$ is monotone non-decreasing by definition of $\mathcal{P}_0$.

For the general case $i > 0$ we first compute an array $A_i[W \cdot 2^{i-k} \pm 8\Delta]$ by taking the $(\max, +)$-convolution of $C_{i-1}[W \cdot 2^{i-1-k} \pm 8\Delta]$ with itself. To obtain $C_i[W \cdot 2^{i-k} \pm 8\Delta]$, we clip the values in A_i which are too large, and set to $-\infty$ the values which are to small. This ensures that all values in C_i lie within a range of $O(\Delta)$, except for values that are $-\infty$. Algorithm 1 contains the pseudocode.

Algorithm 1 Given an instance $(\mathcal{I}, W)$ of Unbounded Knapsack and a guess $\alpha \in [p_{\max} \cdot W]$, the algorithm computes a value $C_k[W]$ satisfying the guarantee in Lemma 15.

```
1: $k := \lceil \log W \rceil$
2: Initialize $C_0[0, \dots, 8\Delta] := \mathcal{P}_0[0, \dots, 8\Delta]$ by iterating over the item set $\mathcal{I}$ once
3: for $i = 1, \dots, k$ do
4:     $A_i[W \cdot 2^{i-k} \pm 8\Delta] := C_{i-1}[W \cdot 2^{i-1-k} \pm 8\Delta] \oplus C_{i-1}[W \cdot 2^{i-1-k} \pm 8\Delta]$
5:     $C_i[j] := \begin{cases} \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta & \text{if } A_i[j] > \alpha \cdot 2^{i-k} + 24\Delta \\ -\infty & \text{if } A_i[j] < \alpha \cdot 2^{i-k} - 40\Delta \\ A_i[j] & \text{otherwise} \end{cases}$
   return $C_k[W]$
```

Due to the clipping, at every step we compute a $(\max, +)$-convolution of sequences of length $O(\Delta)$ and values in $[O(\Delta)] \cup \{-\infty\}$ (after shifting the indices and values appropriately). Furthermore, note that all convolutions involve monotone non-decreasing sequences. Indeed, as noted above the starting sequence C_0 is monotone non-decreasing. Convolving it with itself produces a monotone non-decreasing sequence again, and the clipping in line 5 of Algorithm 1 preserves monotonicity. The same argument applies for further iterations. Thus, the running time of Algorithm 1 is $O(n + T(\Delta) \log W)$, where $T(\Delta)$ is the running time to compute BMMaxConv on sequences of length Δ.

Regarding correctness, we claim the following:

▷ Claim 14. For every $i \in [k]$ and every index $j \in [W \cdot 2^{i-k} \pm 8\Delta] \cap [W]$ the following holds:
- If $\mathcal{P}_i[j] \in [\alpha \cdot 2^{i-k} - 40\Delta, \alpha \cdot 2^{i-k} + 24\Delta]$, then $C_i[j] = \mathcal{P}_i[j]$.
- If $\mathcal{P}_i[j] > \alpha \cdot 2^{i-k} + 24\Delta$, then $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$.
- If $\mathcal{P}_i[j] < \alpha \cdot 2^{i-k} - 40\Delta$, then $C_i[j] = -\infty$.

Intuitively, the claim says that entries "close" to the (scaled) guess $\alpha \cdot 2^{i-k}$ get computed exactly, while entries below and above get clipped appropriately.

Proof. We prove the claim by induction on i. In the base case $i = 0$, note that since $\alpha \in [p_{\max} \cdot W]$ and $k = \lceil \log W \rceil$ we have $\alpha \cdot 2^{-k} \le p_{\max} \le \Delta$. Thus, $[\alpha \cdot 2^{-k} - 40\Delta, \alpha \cdot 2^{-k} + 24\Delta]$ contains the whole interval $[0, \Delta]$ of possible values of $\mathcal{P}_0[j] = C_0[j]$ (for any $0 \le j \le 8\Delta$).

Now we show that the claim holds for any $1 \le i \le k$ assuming it holds for $i - 1$. Fix any $j \in [W \cdot 2^{i-k} \pm 8\Delta]$. Note that the thresholding in line 4 of Algorithm 1 does not increasy any of the entries in A_i, so $C_i[j] \le A_i[j]$. Moreover, since inductively $C_{i-1}[j'] \le \mathcal{P}_{i-1}[j']$ for all j', by definition of A_i we have $A_i[j] \le \mathcal{P}_i[j]$. Hence, we obtain $C_i[j] \le \mathcal{P}_i[j]$. We use this observation to obtain the claim, by showing an appropriate lower bound for $C_i[j]$ in the following.

Pick indices j_1, j_2 as guaranteed by Lemma 13. Property (i) of Lemma 13 guarantees that the computation of $A_i[j]$ in line 3 of Algorithm 1 looks at the entries j_1, j_2 in C_{i-1}. Hence, $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2]$.

We proceed by a case distinction on the values of the entries $\mathcal{P}_{i-1}[j_1]$ and $\mathcal{P}_{i-1}[j_2]$:

Case 1: $\boldsymbol{\mathcal{P}_{i-1}[j_1], \mathcal{P}_{i-1}[j_2] \in [\alpha \cdot 2^{i-1-k} - 40\Delta, \alpha \cdot 2^{i-1-k} + 24\Delta]}$. By the induction hypothesis, both values are computed exactly, that is, $C_{i-1}[j_1] = \mathcal{P}_{i-1}[j_1]$ and $C_{i-1}[j_2] = \mathcal{P}_{i-1}[j_2]$. Thus, $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] = \mathcal{P}_i[j]$, using property (ii) of Lemma 13. Since we observed above that $A_i[j] \leq \mathcal{P}_i[j]$, we obtain $A_i[j] = \mathcal{P}_i[j]$. The thresholding in line 4 of Algorithm 1 now yields the claim for this case.

Case 2: $\boldsymbol{\mathcal{P}_{i-1}[j_1] > \alpha \cdot 2^{i-1-k} + 24\Delta}$. Property (iii) of Lemma 13 implies that $|\mathcal{P}_{i-1}[j_1] - \mathcal{P}_{i-1}[j_2]| \leq 4\Delta$, and hence $\mathcal{P}_{i-1}[j_2] \geq \alpha \cdot 2^{i-1-k} + 20\Delta$. Thus, property (ii) of Lemma 13 implies $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] > \alpha \cdot 2^{i-k} + 24\Delta$. Therefore, we want to show that $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$.

By the induction hypothesis, we have $C_{i-1}[j_1] = \lceil \alpha \cdot 2^{i-1-k} \rceil + 24\Delta$ and $C_{i-1}[j_2] \geq \alpha \cdot 2^{i-1-k} + 20\Delta$. Hence, $A_i[j] \geq C_{i-1}[j_1] + C_{i-1}[j_2] > \alpha \cdot 2^{i-k} + 40\Delta$. Due to the thresholding, we conclude that $C_i[j] = \lceil \alpha \cdot 2^{i-k} \rceil + 24\Delta$, as desired.

Case 3: $\boldsymbol{\mathcal{P}_{i-1}[j_1] < \alpha \cdot 2^{i-k} - 40\Delta}$. Similarly as in case 2, property (iii) of Lemma 13 implies that $\mathcal{P}_{i-1}[j_2] \leq \alpha \cdot 2^{i-1-k} - 36\Delta$. Thus, $\mathcal{P}_i[j] = \mathcal{P}_{i-1}[j_1] + \mathcal{P}_{i-1}[j_2] < \alpha \cdot 2^{i-k} - 76\Delta$. Since $C_i[j] \leq \mathcal{P}_i[j]$, but $C_i[j]$ takes values in $\{-\infty\} \cup [\alpha \cdot 2^{i-k} \pm 40\Delta]$ it follows that $C_i[j] = -\infty$. $\triangleleft$

Given the claim, it is easy to see that $C_k[W] \geq \alpha$ if and only if $\mathcal{P}_k[W] \geq \alpha$. Along with the running time analysis argued earlier, we obtain the following lemma.

▶ **Lemma 15.** *Algorithm 1 runs in time $O(n + T(\Delta) \log W)$, where $T(\Delta)$ is the time complexity of* BMMaxConv *on sequences of length Δ, and computes a value $C_k[W]$ which satisfies $C_k[W] \geq \alpha$ if and only if* $\mathrm{OPT} = \mathcal{P}_k[W] \geq \alpha$.

Given Lemma 15, we can do binary search to find the optimal value. This gives an algorithm for Unbounded Knapsack in time $O((n + T(\Delta)) \log W \log \mathrm{OPT})$. To shave the $\mathrm{polylog}(W, \mathrm{OPT})$ factors and obtain the running time $\widetilde{O}(n + T(\Delta))$ claimed in Theorem 10, we make use of the following lemma. It allows us to reduce the capacity of the instance by repeatedly adding copies of the item with maximum profit-to-weight ratio. Similar results have been shown for general ILPs [20], for Unbounded Knapsack [5] and for the Coin Change problem [15]. For completeness, we include the proof by Chan and He [15, Lemma 4.1].[7]

▶ **Lemma 16.** *Let $(p_{i^*}, w_{i^*}) := \mathrm{argmax}_{(p,w) \in \mathcal{I}} \frac{p}{w}$. If $W \geq 2w_{\max}^3$, then there exists an optimal solution containing (p_{i^*}, w_{i^*}).*

Proof. Consider an optimal solution x that does not contain item (p_{i^*}, w_{i^*}). If there is an item (p_j, w_j) that appears at least w_{i^*} times in x, then we can replace w_{i^*} of the copies of item (p_j, w_j) by w_j copies of item (p_{i^*}, w_{i^*}). By definition of (p_{i^*}, w_{i^*}), this does not decrease the total profit of the solution, so by optimality of x the new solution x' is also optimal. Therefore, some optimal solution contains (p_{i^*}, w_{i^*}).

[7] Both Chan and He [15] and Bateni et al. [5] show that the same conclusion of the lemma holds if $W > w_{i^*}^2$, with a slightly more involved argument. For our purposes, this simple variant is enough.

It remains to consider the case that x contains less than w_{i^*} copies of every item, so its total weight is at most $n \cdot w_{i^*} \cdot w_{\max}$. Note that $n \le w_{\max}$, because without loss of generality there is at most one item per distinct weight (otherwise we can keep only the item with the largest profit for each weight). Thus, the total weight of x is at most $w_{\max}^3$. It follows that $W < w_{\max}^3 + w_{i^*} \le 2w_{\max}^3$, since otherwise we could add at least one copy of (p_{i^*}, w_{i^*}) to x, contradicting its optimality. ◀

We now put all pieces together: As a preprocessing step we repeatedly add the item (p_{i^*}, w_{i^*}) and decrease W by w_{i^*}, as long as $W > 2w_{\max}^3$. After this preprocessing, we have $W = O(w_{\max}^3) = O(\Delta^3)$, and thus $\mathrm{OPT} \le W \cdot p_{\max} = O(\Delta^4)$. The we do binary search for OPT, using Algorithm 1 as a decision procedure. By Lemma 15, the overall running time is $O((n + T(\Delta)) \log^2 \Delta) = \widetilde{O}(n + T(\Delta))$. This completes the proof of Theorem 10.

3.3 Solution Reconstruction

The algorithm we described gives us the value OPT of the optimal solution. In this section we will describe how to use witness arrays (Lemma 9) to reconstruct a feasible solution $x \in \mathbb{N}^n$ such that $p(x) = \mathrm{OPT}$ with only a polylogarithmic overhead in the overall running time.

▶ **Lemma 17.** *A optimal solution x can be reconstructed in time $\widetilde{O}(n + T(p_{\max} + w_{\max}))$.*

Proof Sketch. Let $k = \lceil \log W \rceil$ be as in Algorithm 1. After determining the value of OPT, run Algorithm 1 again with the guess $\alpha = \mathrm{OPT}$. For every BMMaxConv in line 4 compute the witness array M_i corresponding to A_i via Lemma 9. This takes time $\widetilde{O}(n+T(p_{\max}+w_{\max}))$. Now, the idea is to start from $C_k[W]$ and traverse the *computation tree* of Algorithm 1 backwards. That is, we look at the pair of entries $C_{k-1}[M_k[W]], C_{k-1}[W - M_k[W]]$ which define the value of $C_k[W]$ and recursively obtain the pair of entries in C_{k-2} determining the value of $C_{k-1}[M_k[W]]$, etc. By proceeding in this way, we eventually hit the leaves, i.e., the entries of $C_0[0, \dots, 8\Delta] = \mathcal{P}_0[0, \dots, 8\Delta]$, which correspond to the items in an optimal solution. A naive implementation of this idea takes time $O(\sum_{i \le k} 2^i) = O(2^k) = O(W)$, which is too slow.

Now we describe an efficient implementation of the same idea. For each $i \in [k]$ construct an array $Z_i[W \cdot 2^{i-k} \pm 8\Delta]$ initialized to zeros. Set $Z_k[W] := 1$. We will maintain the invariant that $Z_i[j]$ stores the number of times we arrive at $C_i[j]$ by traversing the computation tree starting at $C_k[W]$. This clearly holds for $Z_k[W] = 1$ by definition. Now we describe how to fill the entries for the levels below. Iterate over $i = k, k-1, \dots, 1$. For each entry $j \in [W \cdot 2^{i-k} \pm 8\Delta] \cap [W]$ add $Z_i[j]$ to its witness entries in the level below, i.e., increase $Z_{i-1}[M_i[j]]$ by $Z_i[j]$ and $Z_{i-1}[j - M_i[j]]$ by $Z_i[j]$. The invariant is maintained by definition of the witnesses, and because Algorithm 1 guarantees that $M_i[j], j - M_i[j] \in [2 \cdot 2^{i-1-k} \pm 8\Delta] \cap [W]$. Note that this procedure takes time $O(k\Delta) = \widetilde{O}(\Delta)$.

Finally, note that for the base case we have that each $Z_0[j]$ for $j \in [8\Delta]$ counts the number of times that we hit the entry $C_0[j] = \mathcal{P}_0[j]$ in the computation tree starting from $C_k[W]$. Recall that by definition, $\mathcal{P}_0[j]$ is the maximum profit of an item in $\mathcal{I}$ with weight at most j. Hence, every entry $\mathcal{P}_0[j]$ corresponds to a unique item in $\mathcal{I}$. Therefore, we can read off from $Z_0[0, \dots, 8\Delta]$ the multiplicity of each item included in an optimal solution. The overall time of the procedure is $\widetilde{O}(n + T(p_{\max} + w_{\max}))$, as claimed. ◀

4 Exact Algorithm for 0-1 Knapsack

Cygan et al. [19] showed the following reduction from 0-1 Knapsack to BMMaxConv:

▶ **Theorem 18** ([19, Theorem 13]). *If MaxConv on length-n sequences can be solved in time $T(n)$, then 0-1 Knapsack can be solved in time $O(T(W\log W)\log^3(n/\delta)\log n)$ with probability at least $1-\delta$.*

Their reduction is a generalization of Bringmann's algorithm for Subset Sum [8], replacing Boolean convolutions by $(\max,+)$-convolutions. We observe that essentially the same reduction yields sequences of length $O(W)$ which are monotone and have entries bounded by OPT. In particular, these are BMMaxConv instances. This yields the following:

▶ **Theorem 19** (0-1 Knapsack $\rightarrow$ BMMaxConv). *If BMMaxConv on length-n sequences can be solved in time $T(n)$, then 0-1 Knapsack can be solved in time $\widetilde{O}(n+T(W+\mathrm{OPT}))$ with high probability.*

Proof. The proof is virtually the same as [19, Theorem 13], so we omit some details. In particular, we emphasize how the constructed instances can be seen to be monotone and bounded, but we omit some details of the correctness argument. The idea of the algorithm is the following: split the item set $\mathcal{I}$ into groups $G_{(a,b)}\subseteq\mathcal{I}$ such that all items $(p,w)\in\mathcal{I}$ with $p\in[2^{a-1},2^a)$ and $w\in[2^{b-1},2^b)$ are in group $G_{(a,b)}$. That is, all items within each group have weights and profits within a factor of 2 of each other, and thus there are $O(\log W\log\mathrm{OPT})$ many groups. We will describe how to compute $\mathcal{P}_{G_{(a,b)}}[0,\dots,W]$ for each $G_{(a,b)}$. Having that, we simply combine all the profit arrays into $\mathcal{P}_{\mathcal{I}}[0,\dots,W]$ using $(\max,+)$-convolutions. Since we have $O(\log W\log\mathrm{OPT})$ groups, and each profit array is a monotone non-decreasing sequence of length W with entries bounded by OPT, the combination step takes time $\widetilde{O}(T(W+\mathrm{OPT}))$.

Fix some group $G_{(a,b)}$. Since every $(p,w)\in G_{(a,b)}$ has $w\in[2^{a-1},2^a)$ and $p\in[2^{b-1},2^b)$, any feasible solution from $G_{(a,b)}$ consists of at most $z:=\lceil\min\{W/2^{a-1},\mathrm{OPT}/2^{b-1}\}\rceil$ items. Thus, by splitting the items in $G_{(a,b)}$ randomly into z subgroups $G_{(a,b),1},\dots,G_{(a,b),z}$, any fixed feasible solution has at most $O(\log z)$ items in each subgroup $G_{(a,b),k}$ with high probability. To see this, fix a solution x and note that,

$$\mathbf{Pr}[\text{at least } r \text{ items from } x \text{ fall in } G_{(a,b),k}]\leq\binom{z}{r}\left(\frac{1}{z}\right)^r\leq\left(\frac{e\cdot z}{r}\right)^r\left(\frac{1}{z}\right)^r=\left(\frac{e}{r}\right)^r,$$

where the first inequality follows due to a union bound over all subsets of items of size r. By setting $r=O(\log z)$, we can bound this probability by z^{-c} for any constant c. So by a union bound, none of the z groups $G_{(a,b),1},\dots,G_{(a,b),z}$ has more than $\kappa:=O(\log z)$ elements from the fixed solution x with probability at least $1-1/\operatorname{poly}(z)$.

Therefore, to obtain the value of any fixed solution it suffices to compute the optimal solution consisting of at most κ items from $G_{(a,b),k}$ for every target weight $\leq O(2^a\kappa)$, and then merge the results. More precisely, for every $1\leq i\leq z$ we compute the array $\mathcal{P}_{G_{(a,b),i},\log(\kappa)}[0,\dots,O(2^a\kappa)]$. Recall that this is defined as

$$\mathcal{P}_{G_{(a,b),i},\log(\kappa)}[j]:=\max\{p(x)\colon x \text{ is a solution from } G_{(a,b),i} \text{ with } w(x)\leq j, \|x\|_1\leq\kappa\}$$

for each $j\in[O(2^a\kappa)]$. For ease of notation, we denote the array by $\mathcal{P}_{G_i,\kappa}:=\mathcal{P}_{G_{(a,b),i},\log\kappa}$.

Then, we merge the $\mathcal{P}_{G_i,\kappa}$'s using $(\max,+)$-convolutions. Now we describe these two steps in more detail:

Computing $\mathcal{P}_{G_i,\kappa}[0,\ldots,O(2^a\kappa)]$. Since we only care about solutions with at most κ items, we use randomization again[8]: split the items in $G_{(a,b),i}$ into κ^2 buckets $A_1,\ldots,A_{\kappa^2}$. By the birthday paradox, with constant probability it holds that any fixed solution is shattered among the buckets, i.e., each bucket contains at most 1 item of the solution. Thus, for each bucket A_k we construct the array $\mathcal{P}_{A_k,0}[0,\ldots,2^a]$. Recall that this is defined as

$$\mathcal{P}_{A_k,0}[j] := \max\{p(x)\colon x \text{ is a solution from } A_k \text{ with } w(x) \le j, \|x\|_1 \le 1\}$$

for each entry $j \in [2^a]$. To combine the results, we compute $\mathcal{P}_{A_1,0} \oplus \mathcal{P}_{A_2,0} \oplus \cdots \oplus \mathcal{P}_{A_{\kappa^2},0}$. By definition, every $\mathcal{P}_{A_i,0}$ is a monotone non-decreasing sequence of length 2^a with entries bounded by 2^b. Thus, the merging step takes time $O(T((2^a+2^b)\cdot\kappa^2)\cdot\kappa^2)$.

Each entry of the resulting array has the correct value $\mathcal{P}_{i,\kappa}[j]$ with constant probability, since a corresponding optimal solution is shattered with constant probability. By repeating this process $O(\log z)$ times and keeping the entrywise maximum among all repetitions, we boost the success probability to $1-1/\operatorname{poly}(z)$. Thus, by a union bound over the z subgroups $G_{(a,b),1},\ldots,G_{(a,b),z}$, we get that any z fixed entries $\mathcal{P}_{G_1,\kappa}[j_1],\ldots,\mathcal{P}_{G_z,\kappa}[j_z]$ corresponding to a solution which is partitioned among the z subgroups get computed correctly with probability at least $1-1/\operatorname{poly}(z)$. This adds an extra $O(\log z) = O(\kappa)$ factor to the running time.

Merging $\mathcal{P}_{G_1,\kappa} \oplus \cdots \oplus \mathcal{P}_{G_z,\kappa}$. This computation is done in a binary tree-like fashion. That is, in the first level we compute $(\mathcal{P}_{G_1,\kappa} \oplus \mathcal{P}_{G_2,\kappa}), (\mathcal{P}_{G_3,\kappa} \oplus \mathcal{P}_{G_4,\kappa}), \ldots, (\mathcal{P}_{G_{z-1},\kappa} \oplus \mathcal{P}_{G_z,\kappa})$. In the second level we merge the results from the first level in a similar way. We proceed in the same way with further levels. Since we merge z sequences, we have $\lceil \log z \rceil$ levels of computation. In the j-th level, we compute the $(\max,+)$-convolution of $z/2^j$ many monotone non-decreasing sequences of length $O(2^j \cdot 2^a \cdot \kappa)$ with entries bounded by $O(2^j \cdot 2^b \cdot \kappa)$. Therefore, overall the merging takes time

$$O\left(\sum_{j=1}^{\lceil \log z \rceil} \frac{z}{2^j} \cdot T((2^a+2^b)\cdot 2^j \cdot \kappa)\right) \le \widetilde{O}(T((2^a+2^b)\cdot z)),$$

where we used both of our niceness assumptions $k \cdot T(n) \le O(T(k\cdot n))$ for any $k > 1$ and $T(\widetilde{O}(n)) \le \widetilde{O}(T(n))$. Since $z = \lceil \min\{W/2^{a-1}, \mathrm{OPT}/2^{b-1}\}\rceil$, we have $\widetilde{O}(T((2^a+2^b)\cdot z) = \widetilde{O}(T(W+\mathrm{OPT}))$.

Wrapping up. To recap, the algorithm does the following steps:

1. Split the items into $O(\log W \log \mathrm{OPT})$ groups $G_{(a,b)}$. This takes time $O(n)$.
2. Randomly split each group $G_{(a,b)}$ into $z := \lceil \min\{W/2^{a-1}, \mathrm{OPT}/2^{b-1}\}\rceil$ subgroups $G_{(a,b),i}$ for $i \in [z]$.
3. For each $G_{(a,b),i}$ compute the array $\mathcal{P}_{G_i,\kappa}[0,\ldots,O(2^a\kappa)]$ in time $O(T((2^a+2^b)\kappa^2)\cdot\kappa^3)$. Since $\kappa = O(\log z)$, the total time over all $i \in [z]$ is

   $$O(z\cdot T((2^a+2^b)\kappa^2)\cdot\kappa^3) \le O(T((2^a+2^b)\cdot z\cdot\kappa^2)\kappa^3) \le \widetilde{O}(T((2^a+2^b)\cdot z)) \le \widetilde{O}(T(W+\mathrm{OPT})).$$

   Note that here we use the niceness assumptions on $T(n)$. In particular, first we used that $k\cdot T(n) \le O(T(k\cdot n))$ for any $k>1$, and then that $T(\widetilde{O}(n)) \le \widetilde{O}(T(n))$.
4. Merge the arrays $\mathcal{P}_{G_1,\kappa} \oplus \cdots \oplus \mathcal{P}_{G_z,\kappa}$ in time $\widetilde{O}(T(W+\mathrm{OPT}))$ to obtain $\mathcal{P}_{G_{(a,b)}}[0,\ldots,W]$.
5. Merge the arrays $\mathcal{P}_{G_{(a,b)}}$ using $O(\log W \log \mathrm{OPT})$ convolutions in total time $\widetilde{O}(T(W+\mathrm{OPT}))$.

[8] This step is called "Color Coding" in [8, 19].

Thus, the overall time of the algorithm is $\widetilde{O}(n+T(W+\mathrm{OPT}))$. Note that as mentioned earlier in the proof, the algorithm succeeds in computing any *fixed* entry $\mathcal{P}_{\mathcal{I}}[j]$ with probability at least $1-1/\operatorname{poly}(z)$. In particular, this is sufficient to compute the optimal solution $\mathcal{P}_{\mathcal{I}}[W]$ with good probability. As described, the algorithm only returns the value of the optimal solution. In the full version of the paper, we show how to reconstruct an optimal solution. ◀

References

1 Noga Alon, Zvi Galil, Oded Margalit, and Moni Naor. Witnesses for Boolean matrix multiplication and for shortest paths. In *FOCS*, pages 417–426. IEEE Computer Society, 1992.

2 Noga Alon and Moni Naor. Derandomization, witnesses for boolean matrix multiplication and construction of perfect hash functions. *Algorithmica*, 16(4/5):434–449, 1996.

3 Kyriakos Axiotis and Christos Tzamos. Capacitated dynamic programming: Faster Knapsack and graph algorithms. In *ICALP*, volume 132 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

4 Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. Better approximations for tree sparsity in nearly-linear time. In *SODA*, pages 2215–2229. SIAM, 2017.

5 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for Knapsack via convolution and prediction. In *STOC*, pages 1269–1282. ACM, 2018.

6 Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, 1957.

7 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014.

8 Karl Bringmann. A near-linear pseudopolynomial time algorithm for Subset Sum. In *SODA*, pages 1073–1084. SIAM, 2017.

9 Karl Bringmann, Fabrizio Grandoni, Barna Saha, and Virginia Vassilevska Williams. Truly subcubic algorithms for language edit distance and RNA folding via fast bounded-difference min-plus product. *SIAM J. Comput.*, 48(2):481–512, 2019.

10 Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating Subset sum and Partition. In *SODA*, pages 1797–1815. SIAM, 2021.

11 Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In *SODA*, pages 1777–1796. SIAM, 2021.

12 Michael R. Bussieck, Hannes Hassler, Gerhard J. Woeginger, and Uwe T. Zimmermann. Fast algorithms for the maximum convolution problem. *Oper. Res. Lett.*, 15(3):133–141, 1994.

13 Timothy M. Chan. Approximation schemes for 0-1 Knapsack. In *SOSA*, volume 61 of *OASICS*, pages 5:1–5:12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018.

14 Timothy M. Chan and Sariel Har-Peled. Smallest k-enclosing rectangle revisited. In *SoCG*, volume 129 of *LIPIcs*, pages 23:1–23:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

15 Timothy M. Chan and Qizheng He. More on change-making and related problems. *J. Comput. Syst. Sci.*, 124:159–169, 2022.

16 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via Additive Combinatorics. In *STOC*, pages 31–40. ACM, 2015.

17 Timothy M. Chan and R. Ryan Williams. Deterministic APSP, orthogonal vectors, and more: Quickly derandomizing Razborov-Smolensky. *ACM Trans. Algorithms*, 17(1):2:1–2:14, 2021.

18 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC*. ACM, 2022.

19 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On problems equivalent to (min, +)-convolution. *ACM Trans. Algorithms*, 15(1):14:1–14:25, 2019.

20 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the Steinitz lemma. *ACM Trans. Algorithms*, 16(1):5:1–5:14, 2020.

21 Zvi Galil and Oded Margalit. An almost linear-time algorithm for the dense subset-sum problem. *SIAM J. Comput.*, 20(6):1157–1189, 1991.

22 V. S. Grinberg and S. V. Sevastyanov. Value of the Steinitz constant. *Funktsional. Anal. i Prilozhen.*, 14(2):56–57, 1980.

23 Oscar H. Ibarra and Chul E. Kim. Fast approximation algorithms for the Knapsack and Sum of Subset problems. *J. ACM*, 22(4):463–468, 1975.

24 Klaus Jansen and Stefan Erich Julius Kraft. A faster FPTAS for the Unbounded Knapsack problem. *Eur. J. Comb.*, 68:148–174, 2018.

25 Klaus Jansen and Lars Rohwedder. On integer programming and convolution. In *ITCS*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

26 Ce Jin and Hongxun Wu. A simple near-linear pseudopolynomial time randomized algorithm for Subset Sum. In *SOSA*, volume 69 of *OASICS*, pages 17:1–17:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

27 Hans Kellerer and Ulrich Pferschy. Improved dynamic programming in connection with an FPTAS for the knapsack problem. *J. Comb. Optim.*, 8(1):5–11, 2004.

28 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *ICALP*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

29 Eduardo Sany Laber, Wilfredo Bardales Roncalla, and Ferdinando Cicalese. On lower bounds for the maximum consecutive subsums problem and the (min, +)-convolution. In *ISIT*, pages 1807–1811. IEEE, 2014.

30 Eugene L. Lawler. Fast approximation algorithms for Knapsack problems. *Math. Oper. Res.*, 4(4):339–356, 1979.

31 Jiří Matoušek. *Thirty-three miniatures: Mathematical and Algorithmic applications of Linear Algebra*. American Mathematical Society Providence, RI, 2010.

32 Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. A subquadratic approximation scheme for Partition. In *SODA*, pages 70–88. SIAM, 2019.

33 David Pisinger. Linear time algorithms for Knapsack problems with bounded weights. *J. Algorithms*, 33(1):1–14, 1999.

34 Adam Polak, Lars Rohwedder, and Karol Wegrzycki. Knapsack and Subset Sum with small items. In *ICALP*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

35 Raimund Seidel. On the All-Pairs-Shortest-Path problem in unweighted undirected graphs. *J. Comput. Syst. Sci.*, 51(3):400–403, 1995.

36 Ernst Steinitz. Bedingt konvergente Reihen und konvexe Systeme. *Journal für die reine und angewandte Mathematik*, 143:128–176, 1913. URL: `http://eudml.org/doc/149403`.

37 R. Ryan Williams. Faster All-Pairs Shortest Paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018.

A Missing Proofs from Section 2

Proof of Proposition 7. By shifting the indices, we can assume that $A[I]$ and $B[J]$ are sequences $A'[0,\dots,|I|-1]$ and $B'[0,\dots,|J|-1]$. Compute $C' := A' \oplus B'$ in time $T(|A|+|B|)$. By shifting the indices back, we can infer the values of the entries $C[I+J] = A[I] \oplus B[J]$. Thus, we can simply read off the entries in $C[K]$ from the array C'. ◀

Equivalence between variants of BMMaxConv

As stated in Section 2, the algorithm of Chi, Duan, Xie and Zhang [18] computes the $(\min, +)$-convolution of sequences which are monotone increasing and have values in $[O(n)]$. The following proposition shows that this is equivalent to solving MaxConv on monotone non-decreasing sequences with values in $[O(n)] \cup \{-\infty\}$, which justifies Theorem 8.

▶ **Proposition 20.** *MaxConv on monotone non-decreasing sequences of length n and values in $[O(n)] \cup \{-\infty\}$ is equivalent to MinConv on monotone increasing sequences of length n and values in $[O(n)]$, in the sense that if one can be solved in time $T(n)$ then the other can be solved in time $O(T(n))$.*

Proof. We first describe how to reduce MaxConv on monotone non-decreasing sequences and values in $[O(n)] \cup \{-\infty\}$ to MinConv on monotone increasing sequences and values in $[O(n)]$ via a simple chain of reductions:

- *Removing $-\infty$:* Let $A[0, \dots, n], B[0, \dots, n]$ be an instance of MaxConv where A, B are monotone non-decreasing and $A[i], B[i] \in [O(n)] \cup \{-\infty\}$. We start by reducing it to an equivalent instance of MaxConv on monotone non-decreasing sequences and values in $[O(n)]$ (i.e. we remove the $-\infty$ entries). Let Δ be the maximum entry of A and B. Construct a new sequence $A'[0, \dots, n]$ where $A'[i] := 0$ if $A[i] = -\infty$, and $A'[i] := A[i] + 2\Delta$ otherwise. Construct $B'[0, \dots, n]$ from B in the same way. Note that A' and B' are monotone non-decreasing and have values in $[O(n)]$. Moreover, we can infer the values of any entry $(A \oplus B)[k]$ from C': if $C'[k] \le 3\Delta$ then $(A \oplus B)[k] = -\infty$ and otherwise $(A \oplus B)[k] = C'[k] - 4\Delta$.
- *Reducing to MaxConv on non-increasing sequences:* Now we reduce an instance $A[0, \dots, n]$, $B[0, \dots, n]$ of MaxConv on monotone non-decreasing sequences and values in $[O(n)]$ to an instance of MinConv on monotone non-increasing sequences and values in $[O(n)]$. Let Δ be the maximum entry of A and B. Construct two new sequences A' and B' by setting $A'[i] := \Delta - A[i]$ and $B'[i] := \Delta - B[i]$. Then A' and B' are monotone non-increasing and given their $(\min, +)$-convolution we can easily infer the $(\max, +)$-convolution of A and B.
- *Reducing to MinConv on increasing sequences:* Next, we reduce an instance $A[0, \dots, n]$, $B[0, \dots, n]$ of MinConv on monotone non-increasing sequences and values in $[O(n)]$ to an instance of MinConv on increasing sequences and values in $[O(n)]$. Construct two new sequences A' and B' by reversing and adding a linear function to A and B, i.e., set $A'[i] := A[n - i] + i$ and $B'[i] := B[n - i] + i$ for every $i \in [n]$. Note that A' and B' are monotone increasing sequences, and given their $(\min, +)$-convolution we can infer the $(\min, +)$-convolution of A and B.

Combining the reductions above, we conclude that MaxConv on monotone non-decreasing sequences with values in $[O(n)] \cup \{-\infty\}$ can be reduced in linear time to MinConv on monotone increasing sequences with values in $[O(n)]$. To show the reduction in the other direction, we can apply the same ideas: we first negate the entries and shift them to make them non-negative, then reverse the resulting sequences. We omit the details. ◀

B Witnesses for BMMaxConv

In this section we give the proof of Lemma 9. For the remainder of this section, fix an instance $A[0, \dots, n], B[0, \dots, n]$ of BMMaxConv and let $C := A \oplus B$. To compute the witness array $M[0, \dots, 2n]$, we first show that if an entry $C[k]$ has a unique witness then we can easily find it. Then we reduce to the unique witness case with randomization. This approach is well known [35, 1], but some extra care is needed to ensure that the instances remain monotone and bounded. The standard derandomization for this approach [2] also works in our setting.

▶ **Lemma 21.** *If* BMMAXCONV *on length-n sequences can be computed in time $T(n)$, then in time $\widetilde{O}(T(n))$ we can compute an array $U[0,\dots,2n]$ such that for every $k \in [2n]$ if $C[k]$ has a unique witness $M[k]$ then $U[k] = M[k]$. The output $U[k]$ is undefined otherwise.*

Proof. We will describe how to compute the unique witnesses bit by bit. For each bit position $b \in [\lceil \log n \rceil]$ let $i_b \in \{0,1\}$ be the b-th bit of i. Construct sequences A_b, B_b defined as $A_b[i] := 2A[i] + i + i_b$ and $B_b[i] := 2B[i] + i$ for $i \in [n]$. Note that A_b, B_b are still monotone non-decreasing and have entries bounded by $O(n)$. Compute $C_b := A_b \oplus B_b$. For each $k \in [2n]$, we set the b-th bit of $U[k]$ to $C_b[k] \bmod 2$. It is not hard to see that for those entries $C[k]$ which have unique witnesses $U[k]$, this procedure indeed gives the b-th bit of $U[k]$. Indeed, note that because we double every entry in A_b, B_b and add a linear function, adding i_b does not change the maximizer. Therefore, if $C[k]$ has a unique witness then $C'[k]$ has a unique witness whose b-th bit can be read from the least significant bit of $C'[k]$. Thus, by repeating this over all bit positions $b \in [\lceil \log n \rceil]$, we compute the entire array of unique witnesses U using $O(\log n)$ invocations to BMMAXCONV, as desired. ◀

Proof of Lemma 9. Fix a set $S \subseteq [n]$. We say that an entry $C[k]$ gets *isolated* by S if the number of witnesses of $C[k]$ in S is exactly one. We will now describe how to find the witnesses of all entries isolated by S (the idea and argument is similar as in the proof of Lemma 21). Construct sequences A', B' where for each $i \in [n]$ we set

$$A'[i] := \begin{cases} 2A[i] + i + 1 & \text{if } i \in S \\ 2A[i] + i & \text{otherwise} \end{cases}$$

and $B'[i] := 2B[i] + i$. Note that these sequences are monotone non-decreasing and have entries bounded by $O(n)$. Let $C' := A' \oplus B'$. We claim that if an entry $C[k]$ is isolated by S, then $C'[k]$ has a unique witness. To see this, note that if no witness of $C[k]$ gets included in S, then we have that $C'[k] = 2C[k] + k$. If at least one witness gets included in S, then $C'[k] = 2C[k] + k + 1$. In particular, if a witness gets isolated then $C'[k]$ will have a unique witness, as claimed. Thus, by Lemma 21 we can compute in time $\widetilde{O}(T(n))$ an array $U[0,\dots,2n]$ which contains the witnesses of all entries that are isolated by S. Note that some entries of U might be undefined, but we can simply check in time $O(n)$ which entries of U are true witnesses, by iterating over $k \in [2n]$ and checking whether the equality $C[k] = A[U[k]] + B[k - U[k]]$ holds.

Now we show how to select appropriate sets S. Fix an entry $C[k]$ and denote by $R \in [n]$ its number of witnesses. We sample $S \subseteq [n]$ by including each element $i \in [n]$ independently with probability $p := 2^{-\alpha}$ where $\alpha \in \mathbb{N}$ is chosen such that $2^{\alpha-2} \le R \le 2^{\alpha-1}$. Let X be the random variable counting the number of witnesses of $C[k]$ that get sampled in S. By keeping the first two terms in the inclusion-exclusion formula we have that $\mathbf{Pr}[X \ge 1] \ge p \cdot R - \binom{R}{2} p^2$, and by a union bound $\mathbf{Pr}[X \ge 2] \le \binom{R}{2} p^2$. Thus,

$$\mathbf{Pr}[X = 1] = \mathbf{Pr}[X \ge 1] - \mathbf{Pr}[X \ge 2] \ge p \cdot R(1 - p \cdot R) \ge 1/8$$

where the last inequality holds because $1/8 \le p \cdot R \le 1/4$ due to the choice of α. In particular, S isolates $C[k]$ with probability at least $1/8$.

We now put the pieces together. Iterate over the $O(\log n)$ possible values for α. Sample a set S and find all witnesses of entries isolated by S as described earlier in time $\widetilde{O}(T(n))$. As we argued above, if $C[k]$ has R witnesses and $2^{\alpha-2} \leq R \leq 2^{\alpha-1}$, then $C[k]$ gets isolated with constant probability. Thus, by repeating this step with the same α for $O(\log n)$ freshly sampled sets S we find a witness for all such entries $C[k]$ with probability at least $1-1/\operatorname{poly}(n)$. Combining the results across iterations we obtain the array of witnesses $M[0,\dots,2n]$ in time $\widetilde{O}(T(n))$, as desired.

Finally, we note that this procedure can be derandomized with standard techniques [2]. ◀