Faster 0-1-Knapsack via Near-Convex Min-Plus-Convolution

Karl Bringmann
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Alejandro Cassis
Saarland University, Saarland Informatics Campus, Saarbrücken, Germany
Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We revisit the classic 0-1-Knapsack problem, in which we are given n items with their weights and profits as well as a weight budget W, and the goal is to find a subset of items of total weight at most W that maximizes the total profit. We study pseudopolynomial-time algorithms parameterized by the largest profit of any item $p_{\max}$, and the largest weight of any item $w_{\max}$. Our main result are algorithms for 0-1-Knapsack running in time $\widetilde{O}(nw_{\max}p_{\max}^{2/3})$ and $\widetilde{O}(np_{\max}w_{\max}^{2/3})$, improving upon an algorithm in time $O(np_{\max}w_{\max})$ by Pisinger [J. Algorithms '99]. In the regime $p_{\max} \approx w_{\max} \approx n$ (and $W \approx \mathrm{OPT} \approx n^2$) our algorithms are the first to break the cubic barrier n^3.

To obtain our result, we give an efficient algorithm to compute the min-plus convolution of *near-convex* functions. More precisely, we say that a function $f\colon [n] \mapsto \mathbf{Z}$ is Δ-near convex with $\Delta \geq 1$, if there is a convex function $\breve{f}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta$ for every i. We design an algorithm computing the min-plus convolution of two Δ-near convex functions in time $\widetilde{O}(n\Delta)$. This tool can replace the usage of the *prediction technique* of Bateni, Hajiaghayi, Seddighin and Stein [STOC '18] in all applications we are aware of, and we believe it has wider applicability.

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

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

Digital Object Identifier 10.4230/LIPIcs.ESA.2023.24

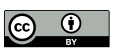
Related Version *Full Version*: https://arxiv.org/abs/2305.01593

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 the 0-1-Knapsack problem, we are given a set of n items $\mathcal{I} = \{ (p_1, w_1), \dots, (p_n, w_n) \}$, where item i has a profit $p_i \in \mathbf{N}$ and a weight $w_i \in \mathbf{N}$, as well as a weight budget $W \in \mathbf{N}$. The goal is to compute $\mathrm{OPT} := \max \sum_{i=1}^n p_i x_i$ subject to the contraints $\sum_{i=1}^n w_i x_i \leq W$ and $x \in \{0,1\}^n$. This classic and fundamental problem in computer science and operations research has been studied for decades (see e.g. [29] for a book on the topic and related problems). Knapsack is weakly NP-hard, and the textbook dynamic programming algorithm due to Bellman [5] solves it in time $O(n \cdot \min\{W, \mathrm{OPT}\})$.

Recent works have studied the fine-grained complexity of Knapsack and related problems, where the goal is to give best-possible pseudopolynomial-time algorithms with respect to different parameters, see Table 1 and [7, 27, 12, 2, 15, 22, 30, 23, 26]. In this work we study the complexity of 0-1-Knapsack in terms of two natural parameters: the largest weight among the items denoted by $w_{\max}$, and the largest profit denoted by $p_{\max}$. Note that we can assume



without loss of generality that $w_{\max} \leq W$ and $p_{\max} \leq \mathrm{OPT}$. Therefore, a small polynomial dependence on these parameters can lead to faster algorithms compared to the standard dynamic programming algorithm on certain instances.

This parameterization has been studied by several previous works, see Table 1. To compare these running times, note that since any feasible solution includes at most all items, we can assume without loss of generality that $W \leq nw_{\max}$ and $\mathrm{OPT} \leq np_{\max}$. Note that when $p_{\max} \approx w_{\max} \approx n$ (and $W \approx \mathrm{OPT} \approx n^2$), all known algorithms require time $\Omega(n^3)$. In particular, in this regime the algorithm in time $O(nw_{\max}p_{\max})$ of Pisinger from '99 [34] is still the best known. In this paper we *overcome this cubic barrier*:

▶ **Theorem 1.** *There is a randomized algorithm for 0-1-Knapsack that runs in time*[1] $\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max})$ *and succeeds with high probability. Using the bound* $W \leq nw_{\max}$*, this running time is at most* $\widetilde{O}(n\, w_{\max}\, p_{\max}^{2/3})$.

Symmetrically, we obtain the following:

▶ **Theorem 2.** *There is a randomized algorithm for 0-1-Knapsack that runs in time* $\widetilde{O}((w_{\max}\mathrm{OPT})^{2/3}(np_{\max})^{1/3} + np_{\max})$ *and succeeds with high probability. Using the bound* $\mathrm{OPT} \leq np_{\max}$*, this running time is at most* $\widetilde{O}(n\, p_{\max}\, w_{\max}^{2/3})$.

Table 1 Pseudopolynomial-time algorithms for 0-1 Knapsack.

Reference	Running Time
Bellman [5]	$O(n \cdot \min\{W, \mathrm{OPT}\})$
Pisinger [34]	$O(n \cdot p_{\max} \cdot w_{\max})$
Kellerer and Pferschy [28], also [4, 3]	$\widetilde{O}(n + w_{\max} \cdot W)$
Bateni, Hajiaghayi, Seddighin and Stein [4]	$\widetilde{O}(n + p_{\max} \cdot W)$
Axiotis and Tzamos [3]	$\widetilde{O}(n \cdot \min\{\, w_{\max}^2, p_{\max}^2 \,\})$
Bateni, Hajiaghayi, Seddighin and Stein [4]	$\widetilde{O}((n+W) \cdot \min\{\, w_{\max}, p_{\max} \,\})$
Polak, Rohwedder and Węgrzycki [35]	$O(n + \min\{\, w_{\max}^3, p_{\max}^3 \,\})$
Bringmann and Cassis [8]	$\widetilde{O}(n + (W + \mathrm{OPT})^{1.5})$
Theorem 1	$\widetilde{O}(n \cdot w_{\max} \cdot p_{\max}^{2/3})$
Theorem 2	$\widetilde{O}(n \cdot p_{\max} \cdot w_{\max}^{2/3})$

Min-Plus Convolution. Given functions $f, g\colon [\,n\,] \mapsto \mathbf{Z}$, their min-plus convolution is the function $h\colon [\,2n\,] \mapsto \mathbf{Z}$ defined as $h(x) = \min_{x'} f(x') + g(x - x')$ for $x \in [\,2n\,]$. This can be trivially computed in time $O(n^2)$, and the best known algorithm for it runs in time $n^2/2^{\Omega(\sqrt{\log n})}$ [6, 36, 17]. The lack of faster algorithms has led to the *Min-Plus Convolution Hypothesis*, which postulates that there is no truly subquadratic algorithm for this problem [20, 32]. Despite this hypothesis, there are structured instances of min-plus convolution that can be solved faster [1, 4, 13, 16, 18]. These improvements have been *key to obtain the Knapsack algorithms listed in Table 1* (the only exception being Bellman's and Pisinger's algorithms [5, 34]):

[1] We use $\widetilde{O}(\cdot)$ to supress polylogarithmic factors in the input size and the largest input number.

- When one of the functions is convex, their min-plus convolution can be computed in time $O(n)$ using the SMAWK algorithm [1]. This has been used for Knapsack indirectly[2] by Kellerer and Pferschy [28], and explicitly by Axiotis and Tzamos [3] and Polak, Rohwedder and Węgrzycki [35].
- When the functions are monotone and have bounded entries, their min-plus convolution can be computed in time $\widetilde{O}(n^{1.5})$ by an algorithm due to Chi, Duan, Xie and Zhang [18]. This has been used for Knapsack by Bringmann and Cassis [8].
- Bateni, Hajiaghayi, Seddighin and Stein [4] introduced the *prediction technique* to show that the min-plus convolution of certain instances arising from Knapsack can be computed efficiently. More precisely, let h be the min-plus convolution of two given functions $f, g\colon [\,n\,] \mapsto \mathbf{Z}$. They show that if one is given n intervals $[\,x_i \,..\, y_i\,]$ for $i \in [\,n\,]$ satisfying (i) $|h(i+j) - (f(i)+g(j))| \leq \Delta$ for every $i \in [\,n\,]$ and $j \in [\,x_i \,..\, y_i\,]$, (ii) for every output $h(k)$ there exists at least one i such that $f(i) + g(k-i) = h(k)$ and $k - i \in [\,x_i \,..\, y_i\,]$ and (iii) $0 \leq x_i, y_i < n$ for all intervals and $x_i \leq x_j, y_i \leq y_j$ for all $i < j$; then h can be computed in time $\widetilde{O}(n \cdot \Delta)$. They showed that this is applicable in the context of Knapsack.

Our Theorems 1 and 2 fall into the same category of improvements, as we design an efficient algorithm for a new class of structured instances of min-plus convolution, namely *near convex* functions: We say that $f\colon [\,n\,] \mapsto \mathbf{Z}$ is Δ-near convex, if there is a convex function $\breve{f}\colon [\,n\,] \mapsto \mathbf{Q}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta$ for all $i \in [\,n\,]$. Our theorem reads as follows:

▶ **Theorem 3** (Near Convex MinPlus Convolution). *Let $f\colon [\,n\,] \mapsto [\,-U \,..\, U\,]$, and $g\colon [\,m\,] \mapsto [\,-U \,..\, U\,]$ be given as inputs where $n, m, U \in \mathbf{N}$. Let $\Delta \geq 1$ such that both f and g are Δ-near convex. Then the min-plus convolution of f and g can be computed in time $\widetilde{O}((n+m) \cdot \Delta)$.*

We view our Theorem 3 as a replacement for the prediction technique by Bateni et al. [4]. Indeed, all uses of the prediction technique exploit near-convexity to ensure its preconditions, and thus all uses that we are aware of can be replaced by our Theorem 3. Since the prediction technique is both difficult to state and difficult to apply, we view our Theorem 3 as replacing the prediction technique by an *easily applicable tool with a concise statement*. Moreover, Theorem 3 provides a new tool for structured instances of min-plus convolution, which we use in this paper to make progress on 0-1-Knapsack, and which we believe has wider applicability.

Our Techniques. Our approach to prove Theorem 3 is as follows. Let $f, g\colon [\,n\,] \mapsto \mathbf{Z}$ be the input functions, and let h be their min-plus convolution, which we aim to compute. First we observe that we can obtain the convex approximations $\breve{f}, \breve{g}$ witnessing the Δ-near convexity of f and g, and compute their min-plus convolution $\breve{h}$ efficiently. By exploiting $\breve{h}$ and the convexity of $\breve{f}$ and $\breve{g}$, we identify a structured set $R \subseteq [\,n\,]^2$ with the property that any $(i,j) \in [\,n\,]^2 \setminus R$ satisfies $f(i) + g(j) > h(j)$. Then, we give a simple recursive algorithm to cover R with a collection $\mathcal{C}$ of disjoint dyadic boxes $I \times J$, where $(I, J) \in \mathcal{C}$ satisfies $I, J \subseteq [\,n\,]$ and $I \times J \subseteq R$. Thus, we can infer h by computing the *sumset* $A := \{\,(i, f(i)) \mid i \in I\,\} + \{\,(j, g(j)) \mid j \in J\,\}$ and taking $h(k) = \min\{\,y \mid (k, y) \in A\,\}$ for every $(I, J) \in \mathcal{C}$. To do this efficiently we observe that inside I and J, the functions $f[I]$ and $g[J]$ are close to linear functions with the same slope up to an additive error of $\pm O(\Delta)$ (which follows from their Δ-near convexity). This implies that their sumset is small; more

[2] Kellerer and Pferschy did not use SMAWK, but gave a different algorithm for computing the min-plus convolution of these instances in time $O(n \log n)$.

precisely it has size $O((|I|+|J|)\Delta)$. Finally, we make use of known tools that can compute a sumset in time proportional to its size. The idea of identifying a covering with small sumsets to efficiently compute the min-plus convolution is inspired by Chan and Lewenstein's [16] algorithm for bounded monotone sequences (in which they do not use convexity in any form). Our algorithm shares some similarities with the prediction technique by Bateni et al. [4]. In particular, the covering by dyadic boxes where functions are near-linear resembles the way in which they exploit the intervals $[\,x_i\,..\,y_i\,]$ required by their algorithm.

To obtain Theorems 1 and 2, we follow the *partition and convolve* paradigm that has been used in many recent algorithms for Subset Sum and Knapsack, see e.g. [7, 4, 8, 26, 14, 31, 21, 11]. Specifically, we randomly split the items into q groups. In each group, we use the standard dynamic programming algorithm to compute for each weight i, the maximum profit $f(i)$ attainable with weight at most i using items from that group. Then we combine the functions f over all groups by min-plus convolution. The crucial observation is that due to the random splitting we only need to compute the values $f(i)$ in a small weight interval.

Further Related Work. Cygan et al. [20] and Künnemann et al. [32] showed that under the Min-Plus Convolution Hypothesis, there is no truly subquadratic algorithm for Knapsack on instances with $w_{\max}, W = \Theta(n)$ and $p_{\max}, \mathrm{OPT} = \Omega(n^2)$, and symmetrically, on instances with $p_{\max}, \mathrm{OPT} = \Theta(n)$ and $w_{\max}, W = \Omega(n^2)$. This implies that Bellman's dynamic programming algorithm is conditionally optimal in these settings.

Pseudopolynomial-time algorithms parameterized by $p_{\max}$ and $w_{\max}$ have also been studied for the closely related Unbounded Knapsack problem. Here, the setup is the same as for 0-1 Knapsack but now a solution might include an arbitrary number of copies of each item. Chan and He [15] gave an algorithm for this problem in time $\widetilde{O}(n \cdot \min\{\, p_{\max}, w_{\max}\,\})$, which is optimal under the Min-Plus Convolution Hypothesis. Bringmann and Cassis [8] gave an algorithm in time $\widetilde{O}(n + (p_{\max}+w_{\max})^{1.5})$ which is better when $w_{\max} \approx p_{\max} \approx n$.

Outline. The paper is organized as follows. In Section 2 we give some formal preliminaries and establish some notation. In Section 3 we give our algorithm for Knapsack proving Theorems 1 and 2, assuming Theorem 3. In Section 4 we will then give our algorithm for min-plus convolution, proving Theorem 3.

2 Preliminaries

We write $\mathbf{N} = \{\,0, 1, 2, \dots\,\}$. For $t \in \mathbf{N}$, we define $[\,t\,] := \{\,0, 1, \dots, t\,\}$. Let $A \in \mathbf{Z}^{n+1}$ be an integer sequence, i.e., $A[i] \in \mathbf{Z}$ for $i \in [\,n\,]$. Sometimes we will refer to such a sequence as a *function* $A\colon [\,n\,] \mapsto \mathbf{Z}$. With this in mind, we use the notation $-A$ to denote the entry-wise negation of A. Given $a, b \in \mathbf{R}$ with $a \le b$, we define $[\,a\,..\,b\,] := \{\,\max(0, \lfloor a \rfloor), \max(0, \lfloor a \rfloor)+1, \dots, \lceil b \rceil - 1, \lceil b \rceil\,\}$. The non-standard rounding and capping at 0 in the definition of $[\,a\,..\,b\,]$ is useful to index a subsequence $A[\,a\,..\,b\,]$ when a and b might not be non-negative integers.

The max-plus convolution of two sequences $A[\,0\,..\,n\,] \in \mathbf{Z}^{n+1}, B[\,0\,..\,m\,] \in \mathbf{Z}^{m+1}$, denoted by $\textsc{MaxConv}(A, B)$, is a sequence of length $n+m+1$ where for each $k \in [\,n+m\,]$ we have $\textsc{MaxConv}(A,B)[k] := \max_{i+j=k} A[i] + B[j]$. The min-plus convolution $\textsc{MinConv}(A, B)$ is defined analogously, but replacing max by a min. Note that by negating the entries of the sequences, these two operations are equivalent.

▶ **Fact 4.** *For any $A \in \mathbf{Z}^{n+1}, B \in \mathbf{Z}^{m+1}$, we have $\textsc{MaxConv}(A,B) = -\textsc{MinConv}(-A,-B)$.*

We will use the following handy notation: Given sequences $A[0\,..\,n], B[0\,..\,n]$ and intervals $I, J \subseteq [n]$ and $K \subseteq [2n]$, we denote by $C[K] := \textsc{MaxConv}(A[I], B[J])$ the computation of $C[k] := \max\{A[i] + B[j] \colon i \in I, j \in J, i + j = k\}$ for each $k \in K$.

We say that a function $f \colon [n] \mapsto \mathbf{Q}$ is *convex* if $f(i) - f(i-1) \leq f(i+1) - f(i)$ holds for every $i \in [1\,..\,n-1]$. We say that f is *concave* if $-f$ is convex.

▶ **Definition 5** (Near Convex and Near Concave Functions). *For $\Delta \geq 0$, we say that a function $f \colon [n] \mapsto \mathbf{Z}$ is Δ-near convex, if there is a convex function $\breve{f} \colon [n] \mapsto \mathbf{Q}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta$. We say that f is Δ-near concave if $-f$ is Δ-near convex.*

If the input consists of N numbers in $[-U\,..\,U]$, we denote $\widetilde{O}(T) = \bigcup_{c \geq 0} O(T \log^c(NU))$.

3 Faster 0-1 Knapsack Algorithm

In this section we prove Theorem 1. Let $(\mathcal{I}, W)$ be a 0-1 Knapsack instance. Throughout, we denote the number of items by $n := |\mathcal{I}|$. We identify the item set $\mathcal{I}$ with $\{1, \ldots, n\}$. We represent a *solution* to the knapsack instance (i.e., a subset of $\mathcal{I}$), by an indicator vector $x \in \{0,1\}^n$. For a subset of the items $\mathcal{J} \subseteq \mathcal{I}$, we put $w_{\mathcal{J}}(x) := \sum_{i \in \mathcal{J}} w_i x_i$ and $p_{\mathcal{J}}(x) := \sum_{i \in \mathcal{J}} p_i x_i$. We define the profit sequence $\mathcal{P}_{\mathcal{I}}[\cdot]$, where for each $j \in \mathbf{N}$ we have

$$\mathcal{P}_{\mathcal{I}}[j] = \max\{\, p_{\mathcal{I}}(x) \mid x \in \{0,1\}^n, w_{\mathcal{I}}(x) \leq j \,\}.$$

Observe that $\mathcal{P}_{\mathcal{I}}$ is monotone non-decreasing, and that $\mathrm{OPT} = \mathcal{P}_{\mathcal{I}}[W]$. The textbook way to compute $\mathcal{P}_{\mathcal{I}}[0\,..\,j]$ is to use dynamic programming:

▶ **Fact 6.** *For any $j \in \mathbf{N}$ the sequence $\mathcal{P}_{\mathcal{I}}[0\,..\,j]$ can be computed in time $O(nj)$.*

Before presenting the algorithm, we make two simple observations about the given Knapsack instance $(\mathcal{I}, W)$. First, by ignoring items with weight larger than the capacity W, we can assume without loss of generality that $w_{\max} \leq W$. Now every single item is a feasible solution, so we have $p_{\max} \leq \mathrm{OPT}$. Second, observe that if $W \geq n \cdot w_{\max}$, then the instance is trivial since we can pack all items. Thus, we can assume without loss of generality that $W \leq n \cdot w_{\max}$. Moreover, since any feasible solution consists of at most all the n items, it follows that $\mathrm{OPT} \leq n \cdot p_{\max}$.

The Algorithm

We now describe the algorithm. Set parameters $q := \min\{(n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\}$ rounded down to the closest power of 2, $\Delta := w_{\max} W/q$ and $\eta := 11 \log n$. For each $\ell \in [\log q]$ we define the interval $J^\ell := [\,\frac{W}{q} 2^\ell - \sqrt{\Delta 2^\ell}\eta \,..\, \frac{W}{q} 2^\ell + \sqrt{\Delta 2^\ell}\eta\,]$.

We start by splitting the items $\mathcal{I}$ into q groups $\mathcal{I}_1^0, \ldots, \mathcal{I}_q^0$ uniformly at random. The idea will be to compute an array C_j^0 associated to each $\mathcal{I}_j^0$, and then combine them in a *tree-like* fashion. A crucial aspect for the running time is that we only compute $|J^\ell|$ entries of each array C_j^ℓ. In detail, we proceed as follows:

Base Case. For each $\mathcal{I}_j^0$, we use Fact 6 to compute $\mathcal{P}_{\mathcal{I}_j^0}[0\,..\,\frac{W}{q} + \sqrt{\Delta}\eta]$ and define the subarray $C_j^0[J^0] := \mathcal{P}_{\mathcal{I}_j^0}[J^0]$.

Combination. Iterate over the *levels* $\ell = 1, \ldots \log(q)$. For $j \in [1\,..\,q/2^\ell]$ we set $\mathcal{I}_j^\ell := \mathcal{I}_{2j-1}^{\ell-1} \cup \mathcal{I}_{2j}^{\ell-1}$. Then, compute the subarray $C_j^\ell[J^\ell]$ by taking the relevant entries of the max-plus convolution of $C_{2j-1}^{\ell-1}[J^{\ell-1}]$ and $C_{2j}^{\ell-1}[J^{\ell-1}]$.

Returning the answer. (Note that when $\ell = \log(q)$, it holds that $\mathcal{I}_1^{\log q} = \mathcal{I}$.) We return the value $C_1^{\log q}[W]$. See Algorithm 1 for the pseudocode.

Algorithm 1 Knapsack Algorithm. Given a set of items $\mathcal{I}$ and a weight budget W, the algorithm computes the maximum attainable profit.

$q \leftarrow \min\{ (n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max} \}$ rounded down to the closest power of 2
$\Delta \leftarrow w_{\max}W/q$
$\eta \leftarrow 11 \log n$
$\mathcal{I}_1^0, \dots, \mathcal{I}_q^0 \leftarrow$ random partitioning of $\mathcal{I}$ into q groups
for $i = 1 \dots q$ **do**
    Compute $\mathcal{P}_{\mathcal{I}_j^0}[\,0\,..\,\frac{W}{q} + \sqrt{\Delta}\eta\,]$ using standard dynamic programming (Fact 6)
    $J^0 \leftarrow [\,\frac{W}{q} - \sqrt{\Delta}\eta\,..\,\frac{W}{q} + \sqrt{\Delta}\eta\,]$
    $C_j^0[J^0] \leftarrow \mathcal{P}_{\mathcal{I}_j^0}[J^0]$
for $\ell = 1 \dots \log(q)$ **do**
    $J^\ell \leftarrow [\,\frac{W}{q}2^\ell - \sqrt{\Delta 2^\ell}\eta\,..\,\frac{W}{q}2^\ell + \sqrt{\Delta 2^\ell}\eta\,]$
    for $j = 1, \dots, q/2^\ell$ **do**
        $\mathcal{I}_j^\ell \leftarrow \mathcal{I}_{2j-1}^{\ell-1} \cup \mathcal{I}_{2j}^{\ell-1}$
        Compute $C_j^\ell[J^\ell] \leftarrow \textsc{MaxConv}(C_{2j-1}^{\ell-1}[J^{\ell-1}], C_{2j}^{\ell-1}[J^{\ell-1}])$ using Theorem 3
return $C_1^{\log q}[W]$

Correctness

We start by analyzing the correctness of the algorithm. The following lemma shows that the weight of any solution restricted to one of the sets $\mathcal{I}_j^\ell$ is concentrated around its expectation.

▶ **Lemma 7** (Concentration, proof deferred to the full version). *Let $x \in \{0,1\}^n$ be a solution to the given Knapsack instance. Fix a level $\ell \in [\,0\,..\,\log q\,]$ and $j \in [\,1\,..\,q/2^\ell\,]$. Then, with probability at least $1 - 1/n^4$ it holds that:*

$$\left| w_{\mathcal{I}_j^\ell}(x) - \frac{w_{\mathcal{I}}(x) \cdot 2^\ell}{q} \right| \leq \sqrt{\Delta 2^\ell} \cdot 10 \log n.$$

Using Lemma 7, we can argue that at level ℓ it suffices to compute a subarray of length $\widetilde{O}(\sqrt{\Delta 2^\ell})$ around $W2^\ell/q$. The following lemma makes this precise:

▶ **Lemma 8** (Proof deferred to the full version). *Let $x \in \{0,1\}^n$ be a solution to the given Knapsack instance satisfying $w_{\mathcal{I}}(x) \in [\,W - w_{\max}\,..\,W\,]$. With probability at least $1 - 1/n^2$, for all levels $\ell \in [\,0\,..\,\log q\,]$ and all $j \in [\,1\,..\,q/2^\ell\,]$ it holds that:*

- $w_{\mathcal{I}_j^\ell}(x) \in J^\ell = [\,\frac{W}{q}2^\ell - \sqrt{\Delta 2^\ell}\eta\,..\,\frac{W}{q}2^\ell + \sqrt{\Delta 2^\ell}\eta\,]$, *and*
- $C_j^\ell[w_{\mathcal{I}_j^\ell}(x)] \geq p_{\mathcal{I}_j^\ell}(x)$.

▶ **Lemma 9** (Correctness of Algorithm 1). *Let $x^* \in \{0,1\}^n$ be an optimal solution to the given Knapsack instance. Then, for every $i \in [\,w_{\mathcal{I}}(x^*)\,..\,W\,]$, it holds that $C_1^{\log q}[i] = \mathcal{P}_{\mathcal{I}}[i]$ with probability at least $1 - 1/n^2$.*

Proof. We can check in linear time $O(n)$ whether the optimal solution consists of all items, in which case the instance is trivial. Thus, we can assume without loss of generality that x^* does not include all items. In particular, x^* leaves at least one item out and therefore its weight satisfies $w_{\mathcal{I}}(x^*) \in [\,W - w_{\max}\,..\,W\,]$. By Lemma 8, it holds that $C_1^{\log q}[w_{\mathcal{I}}(x^*)] \geq p_{\mathcal{I}}(x^*) = \mathcal{P}_{\mathcal{I}}[w_{\mathcal{I}}(x^*)]$ with probability at least $1 - 1/n^2$. From now on we condition on this event. We will use the following auxiliary claim:

▷ Claim 10. The sequence $C_1^{\log q}[J^{\log q}]$ is monotone non-decreasing, and satisfies $C_1^{\log q}[i] \leq \mathcal{P}_{\mathcal{I}}[i]$ for all $i \in J^{\log q}$.

Proof. First we argue monotonicity by induction. Note that in the base case $\ell = 0$, the sequence $C_j^0[J^0] = \mathcal{P}_{\mathcal{I}_j^0}[J^0]$ is monotone non-decreasing due to the definition of $\mathcal{P}_{\mathcal{I}_j^0}$. For level $\ell > 0$, the sequence C_j^ℓ is computed by taking the max-plus convolution of sequences of level $\ell - 1$. The result follows by observing that the max-plus convolution of two monotone non-decreasing sequences is monotone non-decreasing.

The second part of the claim follows since (inductively) every entry $C_1^{\log q}[i]$ for $i \in J^{\log q}$ corresponds to the profit of a subset of items of $\mathcal{I}$ of weight at most i. ◁

Since x^* is an optimal solution, it holds that $\mathcal{P}_{\mathcal{I}}[i] = p_{\mathcal{I}}(x^*)$ for all $i \in [\, w_{\mathcal{I}}(x^*) \,..\, W\,]$. Thus Claim 10 yields that $C_1^{\log q}[i] = \mathcal{P}_{\mathcal{I}}[i]$ for all $i \in [\, w_{\mathcal{I}}(x^*) \,..\, W\,]$, completing the proof. ◀

Running Time

Now we analyze the running time of Algorithm 1. The key speedup comes from the computation in Algorithm 1, where we use Theorem 3 to perform the max-plus convolution. Since Theorem 3 is phrased in terms of min-plus convolution of near-convex functions, we will use the following corollary:

▶ **Corollary 11.** *Let $f\colon [\,n\,] \mapsto [\,-U \,..\, U\,]$ and $g\colon [\,m\,] \mapsto [\,-U \,..\, U\,]$ be given as inputs, where $U \in \mathbf{N}$. Let $\Delta \geq 1$ be such that both f and g are Δ-near concave. Then, $\textsc{MaxConv}(f, g)$ can be computed in time $\widetilde{O}((n+m)\Delta)$*

Proof. Noting that $-f$ and $-g$ are Δ-near convex (Definition 5), the result follows from Theorem 3 and Fact 4. ◀

▶ **Lemma 12** (Near Concavity, proof deferred to the full version)**.** *For every level $\ell \in [\,1 \,..\, q\,]$ and every $j \in [\,1 \,..\, q/2^\ell\,]$, it holds that $C_j^\ell[J^\ell]$ is $p_{\max}$-near concave.*

▶ **Lemma 13.** *Fix a level $\ell \in [\,1 \,..\, q\,]$ and an iteration $j \in [\,1 \,..\, q/2^\ell\,]$. The computation of C_j^ℓ in Algorithm 1 takes time $\widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell})$*

Proof. By Lemma 12, the sequences $C_{2j-1}^{\ell-1}[J^{\ell-1}], C_{2j}^{\ell-1}[J^{\ell-1}]$ are $p_{\max}$-near concave. Thus, by Corollary 11, their max-plus convolution can be computed in time $\widetilde{O}(p_{\max}|J^\ell|) = \widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell})$, where we used $\eta = \widetilde{O}(1)$. ◀

▶ **Lemma 14** (Running Time of Algorithm 1)**.** *Algorithm 1 runs in time*

$$\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max}).$$

Proof. Recall that $q = \min\{\, (n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\,\}$ (up to a factor of 2). Since $W \leq nw_{\max}$, we have that $q \leq n$. Moreover, since we assume without loss of generality that $w_{\max} \leq n$, note that $q < 1$ if and only if $q = (n/p_{\max})^{2/3}(W/w_{\max})^{1/3} < 1$. This implies that $p_{\max} > n\sqrt{W/w_{\max}}$. But in this case, the claimed running time is $\Omega(nW)$, so the standard $O(nW)$ dynamic programming algorithm (Fact 6) already achieves our time bound. Thus, we can assume without loss of generality that $1 \leq q \leq n$, i.e., q is a valid choice for the number of groups in which we split the item set $\mathcal{I}$.

We start bounding the running time of the base case, i.e., the computation of the arrays C_j^0 for $j \in [\,1\,..\,q\,]$ in Algorithm 1. By Fact 6, and the definition $\Delta = w_{\max}W/q$ this takes time

$$O\left(\sum_{j=1}^{q} |\mathcal{I}_j^0|(\tfrac{W}{q} + \sqrt{\Delta}\eta)\right) = O\left(n(\tfrac{W}{q} + \sqrt{\Delta}\eta)\right) = \widetilde{O}\left(n\tfrac{W}{q} + n\sqrt{\tfrac{w_{\max}W}{q}}\right). \tag{1}$$

Now we bound the time of the combination step done in Algorithms 1–1. At level $\ell \in [\,1\,..\,q\,]$ and iteration $j \in [\,1\,..\,q/2^\ell\,]$ the execution of Algorithm 1 takes time $\widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell})$ by Lemma 13. Thus, we can bound the overall time as

$$\sum_{\ell=1}^{\log q}\sum_{j=1}^{q/2^\ell} \widetilde{O}(p_{\max}\sqrt{\Delta 2^\ell}) = \sum_{\ell=1}^{\log q} \frac{q}{2^\ell}\widetilde{O}\left(p_{\max}\sqrt{\tfrac{w_{\max}W}{q}2^\ell}\right) = \sum_{\ell=1}^{\log q} \widetilde{O}\left(p_{\max}\sqrt{\tfrac{qw_{\max}W}{2^\ell}}\right),$$

since this is a geometric series, it is bounded by the first term $\widetilde{O}(p_{\max}\sqrt{qw_{\max}W})$. Combining this with (1), we obtain overall time

$$\widetilde{O}\left(p_{\max}\sqrt{qw_{\max}W} + n\tfrac{W}{q} + n\sqrt{\tfrac{w_{\max}W}{q}}\right).$$

Recalling that $q = \Theta(\min\{\,(n/p_{\max})^{2/3}(W/w_{\max})^{1/3}, W/w_{\max}\,\})$, we obtain overall time

$$\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max} + (p_{\max}W)^{1/3}(nw_{\max})^{2/3}).$$

Finally, using that $\sqrt{xy} \le (x+y)/2$ for all $x, y \ge 0$, we have that

$$\begin{aligned}(p_{\max}W)^{1/3}(nw_{\max})^{2/3} &= \sqrt{(p_{\max}W)^{2/3}(nw_{\max})^{1/3}nw_{\max}} \\ &\le O((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max}).\end{aligned}$$

Thus, the overall running time is $\widetilde{O}((p_{\max}W)^{2/3}(nw_{\max})^{1/3} + nw_{\max})$, as claimed. ◀

Proof of Theorem 1. Run Algorithm 1. By Lemma 9, we obtain that $\mathcal{I}_1^{\log q}[W] = \mathrm{OPT}$ with probability at least $1 - 1/n^2$, which proves correctness. The running time is immediate from Lemma 14. Observe that we can obtain success probability $1 - 1/n^c$ for any constant $c \ge 2$ by repeating the algorithm $c/2$ times. Finally, note that Algorithm 1 only computes the optimal profit of the given instance. In the full version of the paper we describe how to reconstruct the set of items in an optimal solution with no overhead in the running time. ◀

Proof Sketch of Theorem 2. Our presentation focused on proving Theorem 1. The proof of the symmetric variant stated in Theorem 2 is very similar, thus we only sketch the required changes. Essentially, we need to exchange profits with weights everywhere, which in turn means exchanging max-plus convolutions by min-plus convolutions. In more detail: Instead of working with the profit sequence $\mathcal{P}_\mathcal{I}$, we work with the weight sequence $\mathcal{W}_\mathcal{I}$, where the entry $\mathcal{W}_\mathcal{I}[j]$ stores the minimum weight of a solution with profit at least j. We do not know OPT, but we can compute an approximation $\tilde{V}$ satisfying $\tilde{V} - p_{\max} \le \mathrm{OPT} \le \tilde{V}$ in linear time (see e.g. [29, Theorem 2.5.4]). In the algorithm, we exchange all ocurrences of $w_{\max}$ by $p_{\max}$ and all ocurrences of W by $\tilde{V}$. With these changes, the functions C_j^ℓ are now $w_{\max}$-near convex (instead of $p_{\max}$-near concave) so we use Theorem 3 directly instead of Corollary 11. In this way, we obtain the array $C_1^{\log q}[\,\tilde{V} - p_{\max}\,..\,\tilde{V}\,] = \mathcal{W}_\mathcal{I}[\,\tilde{V} - p_{\max}\,..\,\tilde{V}\,]$. Then, we can infer OPT as the largest $i \in [\,\tilde{V} - p_{\max}\,..\,\tilde{V}\,]$ such that $\mathcal{W}_\mathcal{I}[i] \le W$. ◀

4 MinPlus Convolution for Near-Convex Sequences

In this section we prove Theorem 3.

▶ **Theorem 3** (Near Convex MinPlus Convolution). *Let $f\colon [\,n\,] \mapsto [\,-U\,..\,U\,]$, and $g\colon [\,m\,] \mapsto [\,-U\,..\,U\,]$ be given as inputs where $n, m, U \in \mathbf{N}$. Let $\Delta \geq 1$ such that both f and g are Δ-near convex. Then the min-plus convolution of f and g can be computed in time $\widetilde{O}((n+m)\cdot\Delta)$.*

4.1 Preparations

Throughout this section, fix the functions $f\colon [n] \mapsto [\,-U\,..\,U\,], g\colon [m] \mapsto [\,-U\,..\,U\,]$. Recall that we say that $f\colon [\,n\,] \mapsto \mathbf{Z}$ is Δ_f-near convex, if there is a convex function $\breve{f}\colon [\,n\,] \mapsto \mathbf{Q}$ such that $\breve{f}(i) \leq f(i) \leq \breve{f}(i) + \Delta_f$ for all $i \in [\,n\,]$ (see Definition 5). First observe that the lower convex hull of the points $\{\,(i, f(i)) \mid i \in [\,n\,]\,\}$ gives the pointwise maximal convex function $\breve{f}$ with $\breve{f} \leq f$. This can be computed in time $O(n)$ by Graham's scan [25], since the points are already sorted by x-coordinate. Then, we can infer $\Delta_f = \max\{\,1, \max_{i\in[\,n\,]} f(i) - \breve{f}(i)\,\}$. Thus, from now on we assume that we know $\breve{f}, \Delta_f, \breve{g}, \Delta_g$. Set $\Delta := \max\{\,\Delta_f, \Delta_g\,\}$. Let $\breve{h} := \textsc{MinConv}(\breve{f}, \breve{g})$ and $h := \textsc{MinConv}(f, g)$. The goal is to compute h.

We start by introducing some notation. We call $(i, j) \in [\,n\,] \times [\,m\,]$ a *point*. We visualize a point (i, j) as lying on the i-th row and j-th column of an $n \times m$ grid, where $(0, 0)$ is on the bottom-left corner and (n, m) on the top right corner. A point (i, j) lies on *diagonal* $i + j$. For any $\delta \geq 0$, a point (i, j) is *δ-relevant* if $\breve{f}(i) + \breve{g}(j) \leq \breve{h}(i + j) + \delta$. We denote by R_δ the set of all δ-relevant points.

Points that are 0-relevant are important because of the following observation: We call i a *witness* for $\breve{h}(k)$ if $\breve{f}(i) + \breve{g}(k - i) = \breve{h}(k)$. Thus, observe that i is a witness for $\breve{h}(k)$ if and only if $(i, k - i)$ is a 0-relevant point.

The importance of 2Δ-relevant points is captured by the following lemma:

▶ **Lemma 15.** *If $(i, j) \notin R_{2\Delta}$ then $f(i) + g(j) > h(i + j)$.*

That is, points that are not 2Δ-relevant can be ignored for the purpose of computing h.

Proof. Since (i, j) is not 2Δ-relevant, it holds that $f(i) + g(j) \geq \breve{f}(i) + \breve{g}(j) > \breve{h}(i + j) + 2\Delta$. Let $k := i + j$, and let i^* be a witness for $\breve{h}(k)$, i.e., $\breve{f}(i^*) + \breve{g}(k - i^*) = \breve{h}(k)$. Then,

$$h(k) \leq f(i^*) + g(k - i^*) \leq \breve{f}(i^*) + \Delta + \breve{g}(k - i^*) + \Delta = \breve{h}(k) + 2\Delta < f(i) + g(j). \qquad \blacktriangleleft$$

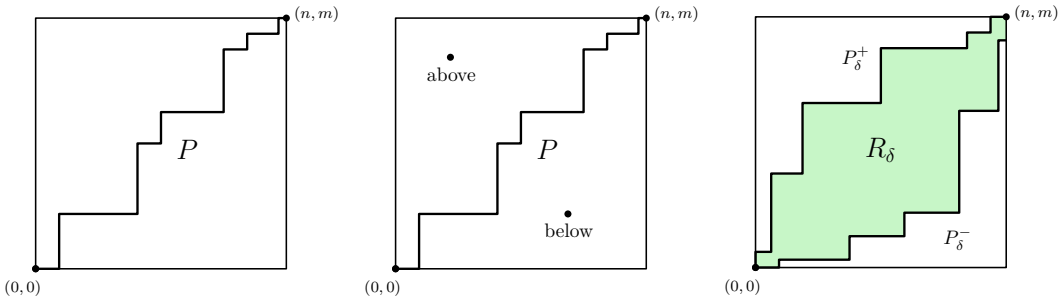
We say that a set of points P is a *monotone path* if for every $k \in [\,n + m\,]$ P contains exactly one point (i_k, j_k) on diagonal k, and we have $(i_{k+1}, j_{k+1}) \in \{\,(i_k + 1, j_k), (i_k, j_k + 1)\,\}$ for every $k \in [\,n + m - 1\,]$, see Figure 1a for an illustration. For any $\delta > 0$, we let

$$P_\delta^+ := \{\,(i, k - i) \mid k \in [\,n + m\,], i \in [\,n\,] \text{ is maximal s.t. } (i, k - i) \text{ is } \delta\text{-relevant}\,\},$$
$$P_\delta^- := \{\,(i, k - i) \mid k \in [\,n + m\,], i \in [\,n\,] \text{ is minimal s.t. } (i, k - i) \text{ is } \delta\text{-relevant}\,\}.$$

The next two lemmas show that P_δ^+, P_δ^- are monotone paths and that P_δ^+, P_δ^- form the boundary of R_δ, see Figure 1c for an illustration. This establishes structure of R_δ that we will be exploit later.

▶ **Lemma 16** (Monotone Paths, proof deferred to the full version). *For any $\delta \geq 0$, P_δ^-, P_δ^+ are monotone paths.*

Let (i, j) be a point and P a monotone path. Let $(a, b) \in P$ be the unique point on the same diagonal as (i, j), i.e., $a + b = i + j$. We say that (i, j) is *below* P if $i < a$, *above* P if $i > a$, and *on* P if $i = a$, see Figure 1b for an illustration.



(a) A monotone path P. **(b)** Points above and below P. **(c)** R_δ is between P_δ^+ and P_δ^-.

Figure 1 Visualizations for concepts used in Section 4.

▶ **Lemma 17.** *For any $\delta \geq 0$, R_δ consists of all points (i, j) that are on or below P_δ^+ and on or above P_δ^-.*

Proof. Fix $k \in [\, n + m \,]$ and let $(i^+, k - i^+), (i^-, k - i^-)$ be the point on diagonal k in P_δ^+ and P_δ^-, respectively. Consider any $(i, j) \in R_\delta$ on diagonal k. By maximality of i^+ we have $i \leq i^+$, and similarly $i \geq i^-$ by the minimality of i^-. Thus, no point in R_δ is above P_δ^+ or below P_δ^-. It remains to show that for any $i^- \leq i \leq i^+$ we have $(i, k - i) \in R_\delta$. Note that the function $r(i) := \breve{f}(i) + \breve{g}(k - i)$ is convex (since it is the sum of convex functions). Since $(i^+, k - i^+)$ is δ-relevant, we have $r(i^+) \leq \breve{h}(k) + \delta$. Similarly, since $(i^-, k - i^-)$ is δ-relevant, we have $r(i^-) \leq \breve{h}(k) + \delta$. By convexity of r, we obtain that $r(i) \leq \breve{h}(k) + \delta$ for all $i^- \leq i \leq i^+$. Hence, we conclude that for each $i^- \leq i \leq i^+$ we have $(i, k - i) \in R_\delta$. ◀

Finally, we need some background on *sumsets*. Given $A, B \subseteq [\, -U \,..\, U \,]^2$ where $U \in \mathbf{N}$, we define $A + B = \{\, a + b \mid a \in A, b \in B \,\}$ as their sumset, where the addition $a + b$ is done componentwise. The naive way to compute $A + B$ takes time $O(|A| \cdot |B|)$. For our application, we want to compute the sumset in time near linear in its size $|A + B|$. For this end, we will use the following tool to compute *sparse non-negative convolution*. Given vectors $P, Q \in \mathbf{N}^n$, their *convolution* $P \star Q \in \mathbf{N}^{2n-1}$ is defined coordinate-wise by $(P \star Q)[k] = \sum_{i+j=k} P[i] \cdot Q[j]$.

▶ **Theorem 18** (Deterministic Sparse Convolution [10])**.** *There is a deterministic algorithm to compute the convolution of two nonnegative vectors $A, B \in \mathbf{N}^n$ in time $O(t \operatorname{polylog}(n\Delta))$, where t is the number of non-zero entries in $A \star B$ and Δ is the largest entry in A and B.*

See also [9] for improvements in the log-factors at the cost of randomization and [19, 33, 24] for prior randomized algorithms with similar guarantees.

▶ **Corollary 19** (Output Sensitive Sumset Computation)**.** *Given $A, B \subseteq [\, -U \,..\, U \,]^2$, with $|A + B| \leq N$, $A + B$ can be computed in time $\widetilde{O}(N)$.*

Proof. Let $A' := \{\, (x+U) \cdot 5U + (y+U) \mid (x, y) \in A \,\}$ and similarly, let $B' := \{\, (x+U) \cdot 5U + (y+U) \mid (x, y) \in B \,\}$. Observe that this is a one-to-one embedding of $A, B \subseteq [\, -U \,..\, U \,]^2$ into $A', B' \subseteq [\, \Theta(U^2) \,]$. Moreover, one can check that given $C' := A' + B'$ we can infer $C := A + B$ (the choice of $5U$ prevents any interactions between coordinates when summing them up).

Thus, it suffices to compute $A' + B'$. To this end, construct their indicator vectors $P_{A'}, P_{B'} \in \mathbf{N}^{\Theta(U^2)}$ and compute the convolution $P_{C'} = P_{A'} \star P_{B'}$. The non-zero entries in $P_{C'}$ correspond to the elements of $A' + B'$. By Theorem 18, this runs in time $O(|A' + B'| \operatorname{polylog}(N, U)) = \widetilde{O}(N)$. ◀

4.2 Algorithm

We are ready to describe our algorithm. Recall that we have access to the functions $f, \breve{f}, g, \breve{g}$ and the value $\Delta = \max\{\Delta_f, \Delta_g\}$.

Computing $\breve{h} = \textsf{MinConv}(\breve{f}, \breve{g})$. Consider the pseudocode given in Algorithm 2.

Algorithm 2 Given convex functions $\breve{f}\colon [n] \mapsto \mathbf{Q}, \breve{g}\colon [m] \mapsto \mathbf{Q}$, the algorithm computes $\breve{h} = \textsc{MinConv}(\breve{f}, \breve{g})$.

1: $i_0^* \leftarrow 0, \breve{h}(0) \leftarrow \breve{f}(0) + \breve{g}(0)$
2: **for** $k = 1, \ldots, n+m$ **do**
3: $\quad i_k^* \leftarrow \operatorname{argmin}\{\breve{f}(i) + \breve{g}(k-i) + \frac{i}{2n} \mid i \in \{i_{k-1}^*, i_{k-1}^* + 1\} \cap [n]\}$
4: $\quad \breve{h}(k) \leftarrow \breve{f}(i_k^*) + \breve{g}(k - i_k^*)$

▶ **Lemma 20.** *Algorithm 2 computes $\breve{h} = \textsc{MinConv}(\breve{f}, \breve{g})$ in time $O(n+m)$.*

Proof. The running time is immediate. To see correctness, focus on i_k^* for $k \in [n+m]$ as computed in Algorithm 2. We claim that the path P_0^- equals $\{(i_k^*, k - i_k^*) \mid k \in [n+m]\}$. That is, we want to argue that i_k^* is the minimum witness of $\breve{h}(k)$ for each $k \in [n+m]$. Indeed, by Lemma 16, P_0^- is a monotone path. Thus, $i_k^* \in \{i_{k-1}^*, i_{k-1}^* + 1\}$. Observe that in Algorithm 2 we pick i_k^* as the minimizer of $\breve{f}(i) + \breve{g}(k-i) + \frac{i}{2n}$ where $i \in \{i_{k-1}^*, i_{k-1}^* + 1\}$. Therefore, the algorithm correctly computes i_k^* (the additive term $i/(2n)$ ensures that we choose the minimal i). Since i_k^* is a minimum witness of $\breve{h}(k)$, the algorithm correctly computes $\breve{h}(k)$ for all $k \in [n+m]$. ◀

We remark that $\textsc{MinConv}(\breve{f}, \breve{g})$ could be computed using the SMAWK algorithm [1]. The reason we give a direct algorithm, is that we will need the witness path P_0^- as computed by Algorithm 2.

Computing $h = \textsf{MinConv}(f, g)$. Recall that $f\colon [n] \mapsto \mathbf{Z}$ and $g\colon [m] \mapsto \mathbf{Z}$. As a final simplification, we argue that we can assume without loss of generality that $n = m$, and $n+1$ is a power of 2. To this end, let N be the smallest power of 2 greater than $\max\{n, m\}$. We pad the functions to length N by setting $f(n+j) := 2j \cdot W$ for $j \in [1\,..\,N-1-n]$ and $g(m+j) := 2j \cdot W$ for $j \in [1\,..\,N-1-m]$, where W is an integer larger than $\max_{i \in [n]} f(i) + \max_{j \in [m]} g(j)$. Observe that the entries $h(0), \ldots, h(n+m)$ of the result $h = \textsc{MinConv}(f, g)$ are unchanged (due to the choice of sufficiently large W), so we can read off the original result from the result of the padded functions. Moreover, observe that the padding does not change the parameters Δ_f and Δ_g.

Now we can describe the algorithm. After running Algorithm 2 we can assume that we have computed $\breve{h}$ and the witness path $P_0^- = \{(i_k^*, k - i_k^*) \mid k \in [n+m]\}$. We will make use of the following subroutines:

- $\textsc{Relevant}(i, j)$: returns $\breve{f}(i) + \breve{g}(j) \le \breve{h}(i+j) + 2\Delta$.
- $\textsc{BelowWitnessPath}(i, j)$: returns $i < i_{i+j}^*$
- $\textsc{AboveWitnessPath}(i, j)$: returns $i > i_{i+j}^*$

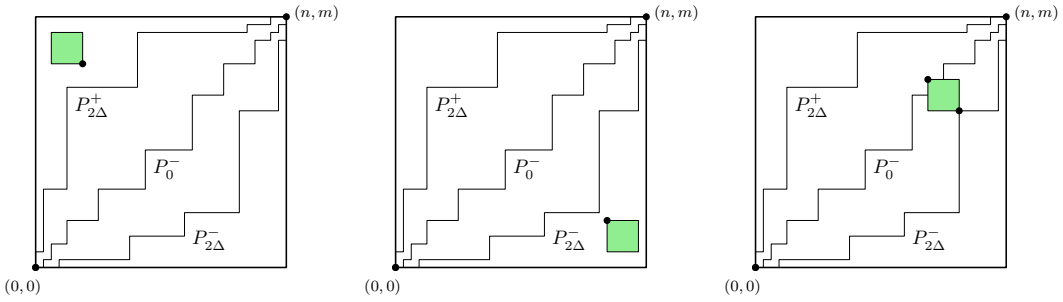
Now we can compute $h = \textsc{MinConv}(f, g)$ by calling $\textsc{RecMinConv}([0\,..\,n], [0\,..\,m])$. See Algorithm 3 for the pseudocode.

Algorithm 3 Given intervals $I = [\, i_A \,..\, i_B \,], J = [\, j_A \,..\, j_B \,]$, the algorithm computes the contribution of $f[I]$ and $g[J]$ to $\textsc{MinConv}(f, g)$.

procedure $\textsc{RecMinConv}(I = [\, i_A \,..\, i_B \,], J = [\, j_A \,..\, j_B \,])$
  if $\textsc{AboveWitnessPath}(i_A, j_B)$ and $\textsc{NotRelevant}(i_A, j_B)$ **then** ▷ Case 1
    return $\tilde{h}(k) = \infty$ for all $k \in [\, i_A + j_A \,..\, i_B + j_B \,]$
  if $\textsc{BelowWitnessPath}(i_A, j_B)$ and $\textsc{NotRelevant}(i_B, j_A)$ **then** ▷ Case 2
    return $\tilde{h}(k) = \infty$ for all $k \in [\, i_A + j_A \,..\, i_B + j_B \,]$
  if $\textsc{Relevant}(i_A, j_B)$ and $\textsc{Relevant}(i_B, j_A)$ **then** ▷ Case 3
    Compute $C \leftarrow \{\, (i, f(i)) \mid i \in I \,\} + \{\, (j, g(j)) \mid j \in J \,\}$ using Corollary 19
    Infer $\tilde{h}(k) \leftarrow \min\{\, y \mid (k, y) \in C \,\}$ for all $k \in [\, i_A + j_A \,..\, i_B + j_B \,]$
    return $\tilde{h}$
  else ▷ Case 4
    Split I into two intervals I_1, I_2 of equal length, similarly split J into J_1, J_2
    Recursively compute $\tilde{g}_{i,j} \leftarrow \textsc{RecMinConv}(I_i, J_j)$ for $i, j \in \{\, 1, 2 \,\}$
    return the pointwise minimum of the functions $\tilde{g}_{i,j}$ for $i, j \in \{\, 1, 2 \,\}$

Algorithm 3 recursively computes the contribution of $f[\, i_A \,..\, i_B \,]$ and $g[\, j_A \,..\, j_B \,]$ to $h = \textsc{MinConv}(f, g)$. We next discuss its four cases; see Figure 2 for illustrations of Cases 1-3. If (i_A, j_B) is above the witness path P_0^- and is not 2Δ-relevant (Case 1), then as we argue below no point in $I \times J$ contributes to the output h, so in this case we return a dummy function (which is $+\infty$ everywhere). Case 2 is symmetric, where (i_B, j_A) is above P_0^- and not 2Δ-relevant, and we again return a dummy function. Case 3 applies when (i_A, j_B) and (i_B, j_A) are both 2Δ-relevant. In this case, we explicitly compute $\tilde{h} = \textsc{MinConv}(f[\, i_A \,..\, i_B \,], g[\, j_A \,..\, j_B \,])$ by computing the sumset $C = \{\, (i, f(i)) \mid i \in I \,\} + \{\, (j, g(j)) \mid j \in J \,\}$ and infering $\tilde{h}(k)$ as the minimum y such that $(k, y) \in C$, which by definition of the sumset equals the minimum $f(i) + g(j)$ such that $i \in I, j \in J$ and $i + j = k$. Note that this step can be done for all $k \in [\, i_A + j_A \,..\, i_B + j_B \,]$ in total time $O(|C|)$ by once scanning over all elements of C.

Finally, if none of the above cases apply, then we split both intervals I and J into equal halves and recurse on all 4 combinations of halves. We combine them by taking the pointwise minimum of all computed functions. This case is essentially brute force.



(a) Case 1. **(b)** Case 2. **(c)** Case 3.

Figure 2 Visualization of Cases 1-3. The green box represents the current subproblem.

Correctness. We start by analyzing the correctness of the algorithm.

▶ **Lemma 21** (Correctness of Algorithm 3). *$\textsc{RecMinConv}([\, 0 \,..\, n \,], [\, 0 \,..\, m \,])$ (Algorithm 3) correctly computes $h = \textsc{MinConv}(f, g)$.*

Proof. Let $k \in [\, n+m \,]$ and consider a point (i^*, j^*) in diagonal k such that $f(i^*) + g(j^*) = h(k)$, i.e., a witness for $h(k)$. We argue that some recursive call computes $f(i^*) + g(j^*)$. This is clear in Case 4, as (i^*, j^*) is covered by one recursive subproblem. It is also clear in Case 3, since then $f(i^*) + g(j^*)$ is explicitly computed.

To finish correctness, we argue that (i^*, j^*) can never be in a subproblem to which Case 1 or 2 applies. Recall that Case 1 applies to a subproblem $I = [\, i_A \,..\, i_B \,], J = [\, j_A \,..\, j_B \,]$ if (i_A, j_B) is above P_0^- and (i_A, j_B) is not 2Δ-relevant. Since (i_A, j_B) is not 2Δ-relevant, by Lemma 17 (i_A, j_B) must be above $P_{2\Delta}^+$ or below $P_{2\Delta}^-$. Since (i_A, j_B) is above P_0^-, it can only be above $P_{2\Delta}^+$. Since (i_A, j_B) is the lower right corner of $I \times J$, it follows that all points in $I \times J$ are above $P_{2\Delta}^+$. Thus, by Lemma 17 all points in $I \times J$ are not 2Δ-relevant. If we assume for the sake of contradiction that $(i^*, j^*) \in I \times J$, then Lemma 15 implies $f(i^*) + g(j^*) > h(k)$, contradicting the choice of (i^*, j^*) as a witness for $h(k)$. Hence, (i^*, j^*) can never be in a Case 1 subproblem. Case 2 is symmetric. This finishes the correctness proof. ◀

Running Time. Next, we analyze the running time. The key insight is that in relevant regions both functions are essentially linear, with the same slope (see Lemma 22). This implies that the sumset computed in Case 3 is small (see Lemma 23), so it can be computed efficiently using Corollary 19. In the following two lemmas, let $I = [\, i_A \,..\, i_B \,] \subseteq [\, n \,]$ and $J = [\, j_A \,..\, j_B \,] \subseteq [\, m \,]$ be intervals of the same length $|I| = |J|$.

▶ **Lemma 22** (Near Linearity, proof deferred to the full version). *If $I \times J \subseteq R_{2\Delta}$ then there are $a, b, c \in \mathbf{R}$ such that $|f(i) - (a \cdot i + b)| \leq 2\Delta$ for all $i \in I$ and $|g(j) - (a \cdot j + c)| \leq 2\Delta$ for all $j \in J$.*

▶ **Lemma 23** (Relevant Regions have Small Sumsets). *If $I \times J \subseteq R_{2\Delta}$ then the sumset $\{\, (i, f(i)) \mid i \in I \,\} + \{\, (j, f(j)) \mid j \in J \,\}$ has size $O(\Delta \cdot (|I| + |J|))$.*

Proof. By Lemma 22, for any $(i, j) \in I \times J$ with $i + j = k$ we have

$$f(i) + g(j) = (a \cdot i + b) + (a \cdot j + c) \pm O(\Delta) = a \cdot k + b + c \pm O(\Delta).$$

Thus, for each of the $|I| + |J| - 1$ x-coordinates (i.e., choices of $i + j$), there are $O(\Delta)$ different y-coordinates (i.e., values $f(i) + g(j)$) in the sumset. ◀

▶ **Lemma 24** (Running Time of Algorithm 3). *RecMinConv$([\, 0 \,..\, n \,], [\, 0 \,..\, m \,])$ (Algorithm 3) runs in time $\widetilde{O}(n\Delta)$.*

Proof. We first analyze the running time of one recursive subproblem, ignoring the cost of recursive calls. Note that in Cases 1 and 2 it suffices to return a dummy value, i.e., we do not need to iterate over $k \in [\, i_A + j_A \,..\, i_B + j_B \,]$ to explitly return $\tilde{h}(k) = \infty$. Thus, Cases 1 and 2 run in time $O(1)$. We charge this time to the parent of the current subproblem, which is a Case 4-subproblem.

Consider Case 4. Ignoring the cost of the recursive subproblems, Case 4 runs in time $O(1)$, which also covers the charging from children which fall in Cases 1 and 2.

Consider Case 3, and let $s := i_B - i_A + 1 = j_B - j_A + 1$ be the current side length. By Lemma 23, the sumset computed in Algorithm 3 has size $O(\Delta s)$. Thus, it can be computed in time $\widetilde{O}(\Delta s)$ using Corollary 19, and the function $\tilde{h}$ can be inferred from it in time $O(\Delta s)$.

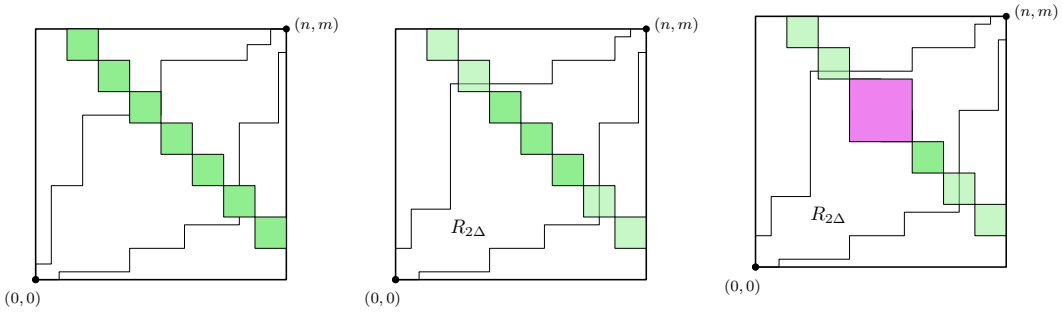
Now we bound the total running time across subproblems. Fix a side length s and consider all possible subproblems of side length s, i.e., all boxes

$$B_{x,y}^s := [\, x \cdot s \,..\, x \cdot s + s - 1 \,] \times [\, y \cdot s \,..\, y \cdot s + s - 1 \,], \text{ where } x, y \in [\, n/s \,].$$

Consider a diagonal $D_{s,d} := \{ B^s_{x,x+d} \mid x \in [\, n/s \,] \}$ of these boxes, see Figure 3a. Note that a box in $D_{s,d}$ that lies fully above P^+_Δ corresponds to a Case 1-subproblem. A box in $D_{s,d}$ that lies fully below corresponds to a Case 2-subproblem. A box that is below or on $P^+_{2\Delta}$ and above or on $P^-_{2\Delta}$ corresponds to a Case 3-subproblem. The remaining boxes intersect $P^+_{2\Delta}$ or $P^-_{2\Delta}$ and correspond to Case 4.

Note that by monotonicity of $P^+_{2\Delta}, P^-_{2\Delta}$, at most two boxes in $D_{s,d}$ are intersected by $P^+_{2\Delta}$ or $P^-_{2\Delta}$ and thus at most two boxes in $D_{s,d}$ can appear as Case 4-subproblems. Thus, Case 4 incurs time $O(1)$ per diagonal. We argue that among the boxes in $D_{s,d}$, at most two can appear as Case 3-subproblems. Indeed, if these would be at least three such boxes, then the parent of the middle box would also be between $P^+_{2\Delta}$ and $P^-_{2\Delta}$, and thus the parent would already be a Case 3-subproblem, see Figures 3b and 3c. Thus, the middle box would not get split, and it would not become a recursive subproblem. Hence per diagonal $D_{s,d}$, Case 3 incurs time $\widetilde{O}(\Delta s)$ for each of at most two boxes.

It remains to sum up over all side lengths $1 \leq s \leq n$ where $s = 2^\ell$ is a power of 2 (recall that at each recursive level we split the side length in two equal parts), and over all $O(n/s)$ diagonals d, to obtain total time $\sum_{\ell=1}^{\log n} O(n/2^\ell) \cdot \widetilde{O}(\Delta 2^\ell) = \widetilde{O}(\Delta n)$. Note that the sum over ℓ only adds another log-factor, which is hidden by the $\widetilde{O}$-notation. ◀



(a) A diagonal of boxes $D_{s,d}$.

(b) Three boxes inside $R_{2\Delta}$.

(c) The parent box is already contained in $R_{2\Delta}$.

Figure 3 Visualizations for the proof of Lemma 24.

References

1 Alok Aggarwal, Maria M. Klawe, Shlomo Moran, Peter W. Shor, and Robert E. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987. doi:10.1007/BF01840359.

2 Kyriakos Axiotis, Arturs Backurs, Karl Bringmann, Ce Jin, Vasileios Nakos, Christos Tzamos, and Hongxun Wu. Fast and simple modular subset sum. In *SOSA*, pages 57–67. SIAM, 2021. doi:10.1137/1.9781611976496.6.

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. doi:10.4230/LIPIcs.ICALP.2019.19.

4 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *STOC*, pages 1269–1282. ACM, 2018. doi:10.1145/3188745.3188876.

5 Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1957. doi:10.2307/j.ctv1nxcw0f.

6 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. doi:10.1007/s00453-012-9734-3.

7 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *SODA*, pages 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.

8 Karl Bringmann and Alejandro Cassis. Faster knapsack algorithms via bounded monotone min-plus-convolution. In *ICALP*, volume 229 of *LIPIcs*, pages 31:1–31:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.31.

9 Karl Bringmann, Nick Fischer, and Vasileios Nakos. Sparse nonnegative convolution is equivalent to dense nonnegative convolution. In *STOC*, pages 1711–1724. ACM, 2021. doi:10.1145/3406325.3451090.

10 Karl Bringmann, Nick Fischer, and Vasileios Nakos. Deterministic and las vegas algorithms for sparse nonnegative convolution. In *SODA*, pages 3069–3090. SIAM, 2022. doi:10.1137/1.9781611977073.119.

11 Karl Bringmann and Vasileios Nakos. A fine-grained perspective on approximating subset sum and partition. In *SODA*, pages 1797–1815. SIAM, 2021. doi:10.1137/1.9781611976465.108.

12 Karl Bringmann and Philip Wellnitz. On near-linear-time algorithms for dense subset sum. In *SODA*, pages 1777–1796. SIAM, 2021. doi:10.1137/1.9781611976465.107.

13 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. doi:10.1016/0167-6377(94)90048-5.

14 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. doi:10.4230/OASIcs.SOSA.2018.5.

15 Timothy M. Chan and Qizheng He. More on change-making and related problems. *J. Comput. Syst. Sci.*, 124:159–169, 2022. doi:10.1016/j.jcss.2021.09.005.

16 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *STOC*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.

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. doi:10.1145/3402926.

18 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.

19 Richard Cole and Ramesh Hariharan. Verifying candidate matches in sparse and wildcard matching. In *STOC*, pages 592–601. ACM, 2002. doi:10.1145/509907.509992.

20 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. doi:10.1145/3293465.

21 Mingyang Deng, Ce Jin, and Xiao Mao. Approximating knapsack and partition via dense subset sums. In *SODA*, pages 2961–2979. SIAM, 2023. doi:10.1137/1.9781611977554.ch113.

22 Mingyang Deng, Xiao Mao, and Ziqian Zhong. On problems related to unbounded subsetsum: A unified combinatorial approach. In *SODA*, pages 2980–2990. SIAM, 2023. doi:10.1137/1.9781611977554.ch114.

23 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. In *SODA*, pages 808–816. SIAM, 2018. doi:10.1137/1.9781611975031.52.

24 Pascal Giorgi, Bruno Grenet, and Armelle Perret du Cray. Essentially optimal sparse polynomial multiplication. In *ISSAC*, pages 202–209. ACM, 2020. doi:10.1145/3373207.3404026.

25 Ronald L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Inf. Process. Lett.*, 1(4):132–133, 1972. doi:10.1016/0020-0190(72)90045-2.

26 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. doi:10.4230/LIPIcs.ITCS.2019.43.

27 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. doi:10.4230/OASIcs.SOSA.2019.17.

28 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. doi:10.1023/B:JOCO.0000021934.29833.6b.

29 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004. doi:10.1007/978-3-540-24777-7.

30 Kim-Manuel Klein. On the fine-grained complexity of the unbounded subsetsum and the frobenius problem. In *SODA*, pages 3567–3582. SIAM, 2022. doi:10.1137/1.9781611977073.141.

31 Konstantinos Koiliaris and Chao Xu. Faster pseudopolynomial time algorithms for subset sum. *ACM Trans. Algorithms*, 15(3):40:1–40:20, 2019. doi:10.1145/3329863.

32 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. doi:10.4230/LIPIcs.ICALP.2017.21.

33 Vasileios Nakos. Nearly optimal sparse polynomial multiplication. *IEEE Trans. Inf. Theory*, 66(11):7231–7236, 2020. doi:10.1109/TIT.2020.2989385.

34 David Pisinger. Linear time algorithms for Knapsack problems with bounded weights. *J. Algorithms*, 33(1):1–14, 1999. doi:10.1006/jagm.1999.1034.

35 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. doi:10.4230/LIPIcs.ICALP.2021.106.

36 R. Ryan Williams. Faster All-Pairs Shortest Paths via circuit complexity. *SIAM J. Comput.*, 47(5):1965–1985, 2018. doi:10.1137/15M1024524.