Even Faster Knapsack via Rectangular Monotone Min-Plus Convolution and Balancing

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

Anita Dürr
Saarland University and Max Planck Institute for Informatics, Saarland Informatics Campus, Saarbrücken, Germany

Adam Polak
Bocconi University, Milan, Italy

Abstract

We present a pseudopolynomial-time algorithm for the Knapsack problem that has running time $\widetilde{O}(n+t\sqrt{p_{\max}})$, where n is the number of items, t is the knapsack capacity, and $p_{\max}$ is the maximum item profit. This improves over the $\widetilde{O}(n + t\,p_{\max})$-time algorithm based on the convolution and prediction technique by Bateni et al. (STOC 2018). Moreover, we give some evidence, based on a strengthening of the Min-Plus Convolution Hypothesis, that our running time might be optimal.

Our algorithm uses two new technical tools, which might be of independent interest. First, we generalize the $\widetilde{O}(n^{1.5})$-time algorithm for bounded monotone min-plus convolution by Chi et al. (STOC 2022) to the *rectangular* case where the range of entries can be different from the sequence length. Second, we give a reduction from general knapsack instances to *balanced* instances, where all items have nearly the same profit-to-weight ratio, up to a constant factor.

Using these techniques, we can also obtain algorithms that run in time $\widetilde{O}(n + \mathrm{OPT}\sqrt{w_{\max}})$, $\widetilde{O}(n + (nw_{\max}p_{\max})^{1/3}t^{2/3})$, and $\widetilde{O}(n + (nw_{\max}p_{\max})^{1/3}\mathrm{OPT}^{2/3})$, where OPT is the optimal total profit and $w_{\max}$ is the maximum item weight.

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

Keywords and phrases 0-1-Knapsack problem, bounded monotone min-plus convolution, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ESA.2024.33

Related Version *Full Version*: `https://arxiv.org/abs/2404.05681`

Funding *Karl Bringmann and Anita Dürr:* 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).
Adam Polak: Part of this work was done when Adam Polak was at Max Planck Institute for Informatics.

Acknowledgements The authors thank Alejandro Cassis for many fruitful discussions.

1 Introduction

In the Knapsack problem[1] the input consists of a set of n items, where item i has weight $w_i \in \mathbb{N}$ and profit $p_i \in \mathbb{N}$, as well as a weight budget $t \in \mathbb{N}$ (also referred to as *knapsack capacity*). The task is to compute the maximum total profit of any subset of items with total weight

[1] Some related works refer to this problem as 0-1-Knapsack to distinguish it from its variants that allow picking an item multiple times in a solution, e.g., Multiple Knapsack or Unbounded Knapsack. In this paper we consider only the 0-1-Knapsack variant, hence we write Knapsack for short.

at most t, i.e., we want to compute $\mathrm{OPT} := \max\{\sum_{i=1}^{n} p_i x_i \mid x \in \{0,1\}^n, \sum_{i=1}^{n} w_i x_i \leqslant t\}$. Knapsack is one of the most fundamental problems in the intersection of computer science, mathematical optimization, and operations research. Since Knapsack is one of Karp's original 21 NP-complete problems [17], we cannot hope for polynomial-time algorithms. However, when the input integers are small, we can consider pseudopolynomial-time algorithms where the running time depends polynomially on n and the input integers. A well-known example is Bellman's dynamic programming algorithm that runs in time $O(n \cdot t)$, or alternatively in time $O(n \cdot \mathrm{OPT})$ [4].

Cygan et al. [12] and Künnemann et al. [20] showed that under the Min-Plus Convolution Hypothesis there is no algorithm solving Knapsack in time $\widetilde{O}((n+t)^{2-\varepsilon})$ or $\widetilde{O}((n+\mathrm{OPT})^{2-\varepsilon})$ for any constant $\varepsilon > 0$. Hence in the regimes $t = \Theta(n)$ or $\mathrm{OPT} = \Theta(n)$ Bellman's dynamic programming algorithms are near-optimal. To overcome this barrier, recent works study the complexity of Knapsack in terms of two additional parameters: the maximum weight $w_{\max}$ and the maximum profit $p_{\max}$ of the given items. Note that we can assume without loss of generality that $w_{\max} \leqslant t$ and $p_{\max} \leqslant \mathrm{OPT}$. Clearly, by the same lower bounds as above there is no algorithm solving Knapsack in time $\widetilde{O}((n + w_{\max})^{2-\varepsilon})$ or $\widetilde{O}((n + p_{\max})^{2-\varepsilon})$ for any $\varepsilon > 0$. However, in certain regimes small polynomial dependencies on $w_{\max}$ and $p_{\max}$ can lead to faster algorithms compared to the standard dynamic programming algorithm. Table 1 lists the results of prior work with this parameterization. To compare these running times, observe that we can assume without loss of generality that $t \leqslant n \cdot w_{\max}$ and $\mathrm{OPT} \leqslant n \cdot p_{\max}$, since any feasible solution includes at most all n items. We remark that most of the cited algorithms, including our contributions, are randomized.

Table 1 Pseudopolynomial-time algorithms for Knapsack.

Reference	Running Time
Bellman [4]	$O(n \cdot \min\{t, \mathrm{OPT}\})$
Pisinger [21]	$O(n \cdot w_{\max} \cdot p_{\max})$
Kellerer and Pferschy [18], also [3, 2]	$\widetilde{O}(n + \min\{t \cdot w_{\max}, \mathrm{OPT} \cdot p_{\max}\})$
Bateni, Hajiaghayi, Seddighin and Stein [3]	$\widetilde{O}(n + t \cdot p_{\max})$
Axiotis and Tzamos [2]	$\widetilde{O}(n \cdot \min\{w_{\max}^2, p_{\max}^2\})$
Polak, Rohwedder and Węgrzycki [22]	$\widetilde{O}(n + \min\{w_{\max}^3, p_{\max}^3\})$
Bringmann and Cassis [7]	$\widetilde{O}(n + (t + \mathrm{OPT})^{1.5})$
Bringmann and Cassis [8]	$\widetilde{O}(n \cdot \min\{w_{\max} \cdot p_{\max}^{2/3}, p_{\max} \cdot w_{\max}^{2/3}\})$
Jin [15] and He and Xu [14]	$\widetilde{O}(n + \min\{w_{\max}^{5/2}, p_{\max}^{5/2}\})$
Jin [15]	$\widetilde{O}(n \cdot \min\{w_{\max}^{3/2}, p_{\max}^{3/2}\})$
Chen, Lian, Mao and Zhang [10]	$\widetilde{O}(n + \min\{w_{\max}^{12/5}, p_{\max}^{12/5}\})$
Bringmann [6] and Jin [16]	$\widetilde{O}(n + \min\{w_{\max}^2, p_{\max}^2\})$
He and Xu [14]	$\widetilde{O}(n^{3/2} \cdot \min\{w_{\max}, p_{\max}\})$
Theorem 1, this work	$\widetilde{O}(n + t\sqrt{p_{\max}})$
Theorem 3, this work	$\widetilde{O}(n + \mathrm{OPT}\sqrt{w_{\max}})$
Theorem 2, this work	$\widetilde{O}(n + (nw_{\max}p_{\max})^{1/3} \cdot t^{2/3})$
Theorem 4, this work	$\widetilde{O}(n + (nw_{\max}p_{\max})^{1/3} \cdot \mathrm{OPT}^{2/3})$

1.1 Our results

Our main contribution is an $\widetilde{O}(n + t\sqrt{p_{\max}})$-time algorithm for Knapsack.

▶ **Theorem 1.** *There is a randomized algorithm for Knapsack that is correct with high probability and runs in time $\widetilde{O}(n + t\sqrt{p_{\max}})$.*

Let us put this result in context. Bellman's algorithm and many other Knapsack algorithms in Table 1 run in pseudopolynomial time with respect to *either* weights or profits. The first exception is Pisinger's $O(n \cdot w_{\max} \cdot p_{\max})$-time algorithm [21], which offers an improvement in the regime where both weights and profits are small. Later, Bateni et al. [3] introduced the *convolution and prediction* technique, which enabled them to improve over Pisinger's running time to $\widetilde{O}(n + tp_{\max})$. Prior to our work, this was the best known pseudopolynomial upper bound in terms of n, t and $p_{\max}$. In Theorem 1 we improve this running time by a factor $\sqrt{p_{\max}}$. We will discuss below that further improvements in terms of this parameterization seem difficult to obtain (see Theorem 6).

Further upper bounds. A long line of research [2, 22, 15, 14, 10, 6, 16] recently culminated into an $\widetilde{O}(n + w_{\max}^2)$-time algorithm for Knapsack [6, 16], which matches the conditional lower bound ruling out time $\widetilde{O}((n + w_{\max})^{2-\varepsilon})$ for any $\varepsilon > 0$ [12, 20]. The biggest remaining open problem in this line of research is whether Knapsack can be solved in time $\widetilde{O}(n \cdot w_{\max})$, which again would match the conditional lower bound and would be favourable if n is smaller than $w_{\max}$. Our next result is a step in this direction: We design a Knapsack algorithm whose running time is the weighted geometric mean (with weights 1/3 and 2/3) of $\widetilde{O}(n \cdot w_{\max})$ and the running time $\widetilde{O}(t\sqrt{p_{\max}})$ of Theorem 1 (ignoring additive terms $\widetilde{O}(n)$).

▶ **Theorem 2.** *There is a randomized algorithm for Knapsack that is correct with high probability and runs in time* $\widetilde{O}(n + (nw_{\max}p_{\max})^{1/3} \cdot t^{2/3})$.

We also show that one can change our previous two algorithms to obtain symmetric running times where weight and profit parameters are exchanged.

▶ **Theorem 3.** *There is a randomized algorithm for Knapsack that is correct with high probability and runs in time* $\widetilde{O}(n + \mathrm{OPT}\sqrt{w_{\max}})$.

▶ **Theorem 4.** *There is a randomized algorithm for Knapsack that is correct with high probability and runs in time* $\widetilde{O}(n + (nw_{\max}p_{\max})^{1/3} \cdot \mathrm{OPT}^{2/3})$.

Lower bound? Finally, we give some argument why it might be difficult to improve upon any of our running times by a factor polynomial in any of the five parameters n, $w_{\max}$, $p_{\max}$, t and OPT. Specifically, we present a fine-grained reduction from the following variant of min-plus convolution.

▶ **Definition 5** (Bounded Min-Plus Convolution Verification Problem)**.** *Given sequences* $A[0 \dots n-1], B[0 \dots n-1]$, *and* $C[0 \dots 2n-2]$ *with entries in* $\{0, 1, \dots, n\}$, *determine whether for all* k *we have* $C[k] \leqslant \min_{i+j=k} A[i] + B[j]$.

Min-plus convolution can be naively solved in time $O(n^2)$, and the Min-Plus Convolution Hypothesis postulates that this time cannot be improved to $O(n^{2-\varepsilon})$ for any $\varepsilon > 0$ even for integer entries bounded by $M = \mathrm{poly}(n)$. For small M, min-plus convolution can be solved in time $\widetilde{O}(nM)$ using Fast Fourier Transform (FFT). Thus, $M = \Theta(n)$ is the smallest bound for which min-plus convolution conceivably might require quadratic time (although this is not asserted or implied by any standard hypothesis). This situation can be compared to the Strong 3SUM Hypothesis, which asserts hardness of the 3SUM problem with the smallest universe size that is not solved in subquadratic time by FFT.

Our reduction is not from the problem of computing the convolution, but only from the problem of verifying whether a given third sequence lower bounds the convolution element-wise. These two variants – computation and verification – are equivalent for the general

unbounded min-plus convolution [12], but no such equivalence is known for the bounded version (because the relevant reduction blows up the entries). We can show a reduction from the (potentially easier) verification problem.

▶ **Theorem 6.** *If Knapsack can be solved faster than the running time of any of Theorems 1–4 by at least a factor polynomial in any of n, $w_{\max}$, $p_{\max}$, t, or* OPT*, then Bounded Min-Plus Convolution Verification can be solved in time $O(n^{2-\varepsilon})$ for some $\varepsilon > 0$.*

Specifically, we show that if Knapsack with parameters $w_{\max}, t = \Theta(n)$ and $p_{\max}, \mathrm{OPT} = \Theta(n^2)$ can be solved in time $O(n^{2-\varepsilon})$, then so can Bounded Min-Plus Convolution Verification. The same holds for Knapsack with parameters $w_{\max}, t = \Theta(n^2)$ and $p_{\max}, \mathrm{OPT} = \Theta(n)$.

This gives some evidence that our running times achieved in Theorems 1–4 are near-optimal. While this lower bound is not assuming a standard hypothesis from fine-grained complexity, it still describes a barrier that needs to be overcome by any improved algorithm.

1.2 Technical overview

The algorithms in Theorems 1–4 follow the *convolve and partition* paradigm used in many recent algorithms for Knapsack and Subset Sum (see, e.g., [5, 7, 8, 3]). Our general setup follows [8]: We split the items at random into 2^q groups. In the base case, for each group and each target weight j we compute the maximum profit attainable with weight at most j using items from that group. These groups are then combined in a tree-like fashion by computing max-plus convolutions. A key observation is that those sequences are monotone non-decreasing with non-negative entries, and one can bound the range of entries. We deviate from [8] in the algorithms for solving the base case and for combining subproblems by max-plus convolution: For the base case, we use improved variants of the Knapsack algorithms of [7] or [14] to obtain Theorems 1 and 3 or Theorems 2 and 4, respectively. For the combination by max-plus convolution, we use the specialized max-plus convolution algorithm that we discuss next.

Rectangular Monotone Max-Plus Convolution

The max-plus convolution of two sequences $A, B \in \mathbb{Z}^n$ is defined as the sequence $C \in \mathbb{Z}^{2n-1}$ such that $C[k] = \max_{i+j=k}\{A[i] + B[j]\}$. This is well-known to be equivalent to min-plus convolution, and is more relevant for Knapsack applications, therefore from now on we only consider max-plus convolution. Despite the quadratic time complexity of max-plus convolution on general instances, there are algorithms running in strongly subquadratic time if we assume some structure on the input sequences, see [9, 11]. In fact, fast algorithms for structured max-plus convolution are exploited in multiple Knapsack algorithms: Kellerer and Pferschy [18], Axiotis and Tzamos [2] and Polak, Rohwedder, and Węgrzycki [22] use the SMAWK algorithm [1], which can be used to compute in linear time the max-plus convolution of two sequences where one is concave. Bringmann and Cassis [8] develop a subquadratic algorithm to compute the max-plus convolution between two near-concave sequences, and use this algorithm to solve Knapsack in time $\widetilde{O}(n \cdot \min\{w_{\max} \cdot p_{\max}^{2/3}, p_{\max} \cdot w_{\max}^{2/3}\})$. Finally, another Knapsack algorithm of Bringmann and Cassis in [7] uses the algorithm due to Chi, Duan, Xie and Zhang [11] that computes the max-plus convolution between monotone sequences of non-negative entries bounded by $O(n)$ in time $\widetilde{O}(n^{1.5})$.

To obtain our Theorems 1–4 we exploit a modification of the algorithm of Chi, Duan, Xie and Zhang [11]. In particular, the following theorem generalizes the result of [11] to monotone sequences with non-negative entries bounded by an arbitrary parameter M.

▶ **Theorem 7** (Slight modification of [11]). *The min-plus or max-plus convolution of two monotone (either both non-decreasing or both non-increasing) sequences of length at most n with entries in $\{0, 1, \ldots, M\}$ can be computed by a randomized algorithm that is correct with high probability and runs in time $\widetilde{O}(n\sqrt{M})$.*

As a side result that might be of independent interest, we show that the assumption in Theorem 7, that both input sequences are monotone, can be replaced without loss of generality by the assumption that at least one input sequence is monotone, see Theorem 8.

▶ **Theorem 8.** *Suppose that there is an algorithm computing the max-plus convolution of two monotone non-decreasing sequences $A, B \in \{0, 1, \ldots, M\}^n$ in time $T_2(n, M)$, and assume that $T_2(n, M)$ is monotone in n. Then there also is an algorithm computing the max-plus convolution of a monotone non-decreasing sequence $A \in \{0, 1, \ldots, M\}^n$ and an arbitrary (i.e., not necessarily monotone) sequence $B \in \{0, 1, \ldots, M\}^n$ in time $T_1(n, M)$ which satisfies the recurrence $T_1(n, M) \leqslant 2\,T_1(n/2, M) + O(T_2(n, M))$.*

The same statement holds with "non-decreasing" replaced by "non-increasing", or with "max-plus" replaced by "min-plus", or both.

Balancing

In the above described Knapsack algorithm of Theorems 1–4, the sequences for which we want to compute the max-plus convolution are monotone non-decreasing and contain non-negative entries. To use Theorem 7 for the computation of their max-plus convolution, we need to ensure that the entries also have bounded values. We will show that under the *balancedness* assumption $t/w_{\max} = \Theta(\mathrm{OPT}/p_{\max})$, and due to the random splitting, it suffices to consider entries in a small weight interval and in a small profit interval. In order to use this observation, the algorithms of Theorems 1–4 first reduce a Knapsack instance $(\mathcal{I}, t)$ to another instance $(\mathcal{I}', t')$ where the balancedness assumption is satisfied, and then solve Knapsack on this balanced instance $(\mathcal{I}', t')$.

▶ **Lemma 9.** *Solving Knapsack can be reduced, in randomized time $\widetilde{O}(n + w_{\max}\sqrt{p_{\max}})$ (respectively $\widetilde{O}(n + p_{\max}\sqrt{w_{\max}})$), to solving a Knapsack instance for $O(w_{\max})$ consecutive capacities (respectively $O(p_{\max})$ consecutive profits), where the reduced instance satisfies $t/w_{\max} = \Theta(\mathrm{OPT}/p_{\max})$ and consists of a subset of the items of the original instance; in particular, all relevant parameters n, $w_{\max}$, $p_{\max}$, t, and OPT of the reduced instance are no greater than those of the original instance.*

1.3 Outline

After preliminaries in Section 2, in Section 3 we focus on the core ideas behind Theorem 1 by presenting the corresponding Knapsack algorithm that assumes the instance is balanced.

Due to space constraints, all the remaining results are deferred to the full version of this paper. In particular, the full version contains a justification of the balancedness assumption (Lemma 9), which together with the algorithm of Section 3 proves Theorem 1. Additionally, in the full version, we explain how the result of [11] generalizes to Theorem 7, and we prove the conditional lower bound (Theorem 6), as well as the side result that in Theorem 7 the assumption that both input sequences are monotone can be replaced without loss of generality by the assumption that at least one input sequence is monotone (Theorem 8). Finally, the full version also includes variations of the Knapsack algorithm of Section 3 corresponding to Theorems 2–4, as well as a discussion that shows how to derive from [7] the algorithm of Theorem 12 used in Section 3.

2 Preliminaries

We use the notation $\mathbb{N} = \{0, 1, 2, \dots\}$ and define $[n] := \{1, 2, \dots, n\}$ for $n \in \mathbb{N}$. Let $A[i_s \dots i_f]$ be an integer array of length $i_f - i_s + 1$ with start index i_s and end index i_f. We interpret out-of-bound entries as $-\infty$, and thus, when it is clear from context, simply denote the array $A[i_s \dots i_f]$ by A. Then $-A$ is the entry-wise negation of A. We call A *monotone non-decreasing* (respectively *non-increasing*), if for every i, j such that $i_s \leqslant i \leqslant j \leqslant i_f$ we have $A[i] \leqslant A[j]$ (resp. $A[i] \geqslant A[j]$). A is *monotone* if it is either monotone non-decreasing or monotone non-increasing.

▶ **Definition 10** (Restriction to index and entry interval). *Suppose that A is monotone and consider intervals $I \subseteq \mathbb{N}$ and $V \subseteq \mathbb{Z}$. We define the operation $D \leftarrow A[I; V]$ as follows. If there exist no index $i \in I$ with $A[i] \in V$, then set D to the empty array. Otherwise, let $i_{\min} := \min\{i \in I : A[i] \in V\}$ and $i_{\max} := \max\{i \in I : A[i] \in V\}$, and set D to the subarray $A[i_{\min} \dots i_{\max}]$. Note that since A is monotone, for every $i \in \{i_{\min}, \dots, i_{\max}\}$ we have $A[i] \in V$. Thus $A[I; V]$ returns the subarray of A with indices in I and values in V.*

We sometimes abbreviate $A[\{0, \dots, i\}; \{0, \dots, v\}]$ by $A[0 \dots i; 0 \dots v]$.

Max-plus convolution. Let $A[i_s \dots i_f]$ and $B[j_s \dots j_f]$ be two integer arrays of length $n := i_f - i_s + 1$ and $m := j_f - j_s + 1$, respectively. Assume without loss of generality that $n \geqslant m$. The max-plus convolution problem on instance (A, B) asks to compute the finite values of the array $C := \textsc{MaxConv}(A, B)$, which is defined as $C[k] := \max_{i+j=k}\{A[i]+B[j]\}$ for every $k \in \mathbb{N}$; here i, j range over all integers with $i + j = k$. Note that $C[k]$ is finite only for $k \in \{i_s + j_s, \dots, i_f + j_f\}$.

The min-plus convolution problem is defined analogously by replacing max by min. Note that the two operations are equivalent since $\textsc{MinConv}(A, B) = -\textsc{MaxConv}(-A, -B)$. In the context of min-plus convolution, we interpret out-of-bound entries as ∞ instead of $-\infty$.

When the sequences $A[i_s \dots i_f]$ and $B[j_s \dots j_f]$ are either both monotone non-decreasing or both monotone non-increasing, and with values contained in $\{0, 1, \dots, M\}$, for some integer M, then the problem of computing $\textsc{MaxConv}(A, B)$ is called the *bounded monotone* max-plus convolution problem. We call the general case with arbitrary M *rectangular*, as opposed to the *square* bounded monotone max-plus convolution where $M = \Theta(n)$. Chi et al. [11] showed that square bounded monotone max-plus convolution can be solved in time $\widetilde{O}(n^{1.5})$. By slightly adapting their algorithm, we show in the full version of this paper that rectangular bounded monotone max-plus convolution can be solved in time $\widetilde{O}(n\sqrt{M})$.

▶ **Theorem 7** (Slight modification of [11]). *The min-plus or max-plus convolution of two monotone (either both non-decreasing or both non-increasing) sequences of length at most n with entries in $\{0, 1, \dots, M\}$ can be computed by a randomized algorithm that is correct with high probability and runs in time $\widetilde{O}(n\sqrt{M})$.*

Knapsack. The Knapsack problem is defined as follows. Let $\mathcal{I} = \{(w_1, p_1), (w_2, p_2), \dots, (w_n, p_n)\}$ be a (multi-)set of n items, where item i has weight w_i and profit p_i. Let $t \in \mathbb{N}$ be a weight capacity. The goal is to compute $\text{OPT} = \max \sum_{i=1}^{n} p_i x_i$ subject to the constraints $x \in \{0, 1\}^n$ and $\sum_{i=1}^{n} w_i x_i \leqslant t$. We denote by $w_{\max} = \max_i w_i$ the maximum weight and by $p_{\max} = \max_i p_i$ the maximum profit of the items in $\mathcal{I}$. Note that by removing items that have weight larger than t we can assume without loss of generality that $w_{\max} \leqslant t$. Then every single item is a feasible solution, so $p_{\max} \leqslant \text{OPT}$. If $t \geqslant n w_{\max}$ then all items can be picked in a solution and the result is $\text{OPT} = \sum_i p_i$, so the instance is trivial; therefore we

can assume without loss of generality that $t < nw_{\max}$. Since any feasible solution contains at most all n items we also have $\mathrm{OPT} \leqslant n \cdot p_{\max}$. We can also assume that $n \geqslant 10$, since for $n = O(1)$ a standard $O(2^n)$-time algorithm runs in time $O(1)$.

We identify each item $(w_i, p_i) \in \mathcal{I}$ with its index $i \in [n]$ so that any subset of items $\mathcal{J} \subseteq \mathcal{I}$ can be identified with the set of indices $S \subseteq [n]$ such that $\mathcal{J} = \{(w_i, p_i) : i \in S\}$. With slight abuse of notation we sometimes write $\mathcal{J} = S$. We define the partial weight and partial profit functions $w_{\mathcal{J}}(x) := \sum_{i \in \mathcal{J}} w_i x_i$ and $p_{\mathcal{J}}(x) := \sum_{i \in \mathcal{J}} p_i x_i$ for $\mathcal{J} \subseteq \mathcal{I}$. We also define the profit sequence $\mathcal{P}_{\mathcal{J}}[\cdot]$ such that

$$\mathcal{P}_{\mathcal{J}}[k] = \max\{p_{\mathcal{J}}(x) \mid x \in \{0,1\}^n, w_{\mathcal{J}}(x) \leqslant k\}$$

for any $k \in \mathbb{N}$. Note that the task is to compute $\mathrm{OPT} = \mathcal{P}_{\mathcal{I}}[t]$.

Computing $\mathcal{P}_{\mathcal{I}}$. A standard way to compute (part of) the profit sequence $\mathcal{P}_{\mathcal{I}}$ is to use dynamic programming:

▶ **Theorem 11** (Bellman [4]). *Given a Knapsack instance $(\mathcal{I}, t)$ and $k \in \mathbb{N}$, the sequence $\mathcal{P}_{\mathcal{I}}[0 \dots k]$ can be computed in time $O(|\mathcal{I}| \cdot k)$.*

Bringmann and Cassis exploit in [7] the fact that $\mathcal{P}_{\mathcal{I}}$ is monotone non-decreasing. They show that one can compute $\mathcal{P}_{\mathcal{I}}[0 \dots j; 0 \dots j]$ in roughly the same time as it takes to compute a square bounded monotone max-plus convolution of length j. In the full version of this paper we slightly generalize their algorithm so that it computes the entries of $\mathcal{P}_{\mathcal{I}}[0 \dots j; 0 \dots v]$. The modified algorithm uses rectangular instead of square bounded monotone max-plus convolutions. Combining the result with Theorem 7, we prove the following theorem in the full version of this paper.

▶ **Theorem 12** (Slight modification of [7]). *Given a Knapsack instance $(\mathcal{I}, t)$ and $v \in \mathbb{N}$, the sequence $\mathcal{P}_{\mathcal{I}}[0 \dots t; 0 \dots v]$ can be computed by a randomized algorithm that is correct with high probability and runs in time $\widetilde{O}(n + t\sqrt{v})$.*

Approximating OPT. We use the following variant of the greedy algorithm for Knapsack.

▶ **Lemma 13** (e.g. [19, Theorem 2.5.4]). *Given a Knapsack instance $(\mathcal{I}, t)$, one can compute $\widetilde{\mathrm{OPT}} \in \mathbb{N}$ such that $\mathrm{OPT} \leqslant \widetilde{\mathrm{OPT}} \leqslant \mathrm{OPT} + p_{\max}$ and $p_{\max} \leqslant \widetilde{\mathrm{OPT}} \leqslant n \cdot p_{\max}$ in $\widetilde{O}(n)$ time.*

Proof. The greedy algorithm works as follows. Sort and relabel the elements in decreasing order of profit-to-weight ratio such that $p_1/w_1 \geqslant p_2/w_2 \geqslant \cdots \geqslant p_n/w_n$. Select the maximum prefix of items $\{1, 2, \dots, i^*\}$ such that $\sum_{i=1}^{i^*} w_i \leqslant t$. We define $\widetilde{\mathrm{OPT}} := \sum_{i=1}^{\min\{i^*+1, n\}} p_i$.

The fractional solution x^{LP} which fully selects the items in $\{1, 2, \dots, i^*\}$ and selects a $(t - w_{\mathcal{I}}(x))/w_{i^*+1}$-fraction of item $i^* + 1$ is the optimal solution to the linear programming relaxation of the Knapsack problem (see [19, Theorem 2.2.1]). Thus, $\mathrm{OPT} \leqslant p_{\mathcal{I}}(x^{\mathrm{LP}}) \leqslant \widetilde{\mathrm{OPT}}$. Since we assumed without loss of generality that $w_{\max} \leqslant t$, each single item fits into the knapsack, which implies $p_{\max} \leqslant \mathrm{OPT} \leqslant \widetilde{\mathrm{OPT}}$. Since the solution $\{1, 2, \dots, i^*\}$ is feasible and item $i^* + 1$ has profit at most $p_{\max}$, we have $\widetilde{\mathrm{OPT}} \leqslant \mathrm{OPT} + p_{\max}$. Finally, we have $\widetilde{\mathrm{OPT}} \leqslant \sum_{i=1}^{n} p_i \leqslant n \cdot p_{\max}$. ◀

Pareto optimum of $\mathcal{P}_{\mathcal{I}}$. The sequence $\mathcal{P}_{\mathcal{I}}$ is monotone non-decreasing, so we can define the *break points* of $\mathcal{P}_{\mathcal{I}}$ as the integers $k \in \mathbb{N}$ such that $\mathcal{P}_{\mathcal{I}}[k-1] < \mathcal{P}_{\mathcal{I}}[k]$. In particular, $\mathcal{P}_{\mathcal{I}}$ is constant between two break points, and thus it is enough to focus on the values taken at break points of $\mathcal{P}_{\mathcal{I}}$. For every break point $k \in \mathbb{N}$, there exists $x \in \{0,1\}^n$ with $w_{\mathcal{I}}(x) = k$

and $\mathcal{P}_{\mathcal{I}}[k] = p_{\mathcal{I}}(x)$. We call such a vector a *Pareto optimum* of $\mathcal{P}_{\mathcal{I}}$. Indeed, by the definition of $\mathcal{P}_{\mathcal{I}}$, if a vector $y \in \{0,1\}^n$ has higher profit $p_{\mathcal{I}}(y) > p_{\mathcal{I}}(x) = \mathcal{P}_{\mathcal{I}}[w_{\mathcal{I}}(x)]$ then it necessarily has higher weight $w_{\mathcal{I}}(y) > w_{\mathcal{I}}(x)$. We observe the following property of Pareto optima.

▶ **Lemma 14.** *Let $x \in \{0,1\}^n$ be a Pareto optimum of $\mathcal{P}_{\mathcal{I}}$ and let $\mathcal{J} \subseteq \mathcal{I}$. Consider a vector $y \in \{0,1\}^n$ such that $w_{\mathcal{J}}(y) \leqslant w_{\mathcal{J}}(x)$ and $p_{\mathcal{J}}(y) \geqslant p_{\mathcal{J}}(x)$. Then necessarily $p_{\mathcal{J}}(y) = p_{\mathcal{J}}(x)$.*

Proof. Suppose for the sake of contradiction that $p_{\mathcal{J}}(y) > p_{\mathcal{J}}(x)$. Consider the vector y' that is equal to y on $\mathcal{J}$ and equal to x on $\mathcal{I} \setminus \mathcal{J}$. Then $p_{\mathcal{I}}(y') = p_{\mathcal{J}}(y) + p_{\mathcal{I}\setminus\mathcal{J}}(x) > p_{\mathcal{J}}(x) + p_{\mathcal{I}\setminus\mathcal{J}}(x) = p_{\mathcal{I}}(x)$. We also have $w_{\mathcal{I}}(y') = w_{\mathcal{J}}(y) + w_{\mathcal{I}\setminus\mathcal{J}}(x) \leqslant w_{\mathcal{J}}(x) + w_{\mathcal{I}\setminus\mathcal{J}}(x) = w_{\mathcal{I}}(x)$. This contradicts x being a Pareto optimum. ◀

We use $\widetilde{O}$-notation to hide poly-logarithmic factors in the input size n and the largest input number U, i.e., $\widetilde{O}(T) := \bigcup_{c \geqslant 0} O(T \log^c(n \cdot U))$. In particular, for Knapsack we hide polylogarithmic factors in $n, w_{\max}, p_{\max}$. Many subroutines that we use throughout the paper are randomized and compute the correct output with probability at least $1 - 1/n$. Standard boosting improves the success probability to $1 - 1/n^{10}$ at the cost of only a constant factor increase in running time. We can therefore assume that these subroutines have success probability $1 - 1/n^{10}$.

3 Knapsack algorithm for balanced instances

In this section we focus on balanced Knapsack instances, i.e., instances satisfying $t/w_{\max} = \Theta(\mathrm{OPT}/p_{\max})$. We call this the *balancedness assumption*. In the full version of this paper we show that any Knapsack instance can be reduced to a balanced instance (see Lemma 9). Combined with the following Lemma 15, this proves Theorem 1.

▶ **Lemma 15.** *For any Knapsack instance $(\mathcal{I}, t)$ satisfying $t/w_{\max} = \Theta(\mathrm{OPT}/p_{\max})$ the sequence $\mathcal{P}_{\mathcal{I}}[T; P]$ for $T := [t - \sqrt{t \cdot w_{\max}}, t + \sqrt{t \cdot w_{\max}}]$, $P := [\widetilde{\mathrm{OPT}} - \sqrt{\widetilde{\mathrm{OPT}} \cdot p_{\max}}, \widetilde{\mathrm{OPT}} + \sqrt{\widetilde{\mathrm{OPT}} \cdot p_{\max}}]$ and $\mathrm{OPT} \leqslant \widetilde{\mathrm{OPT}} \leqslant \mathrm{OPT} + p_{\max}$ can be computed by a randomized algorithm in time $\widetilde{O}(n + t\sqrt{p_{\max}})$.*

We prove Lemma 15 in the remainder of this section. Observe that, with the notation of Lemma 15, we have $\mathcal{P}_{\mathcal{I}}[t] = \mathrm{OPT}$, $t \in T$ and $\mathrm{OPT} \in P$, since $p_{\max} \leqslant \mathrm{OPT}$. Hence the algorithm in Lemma 15 computes in particular the value $\mathcal{P}_{\mathcal{I}}[t] = \mathrm{OPT}$.

Idea. The idea of the algorithm is to randomly split the items of $\mathcal{I}$ into 2^q groups $\mathcal{I}_1^q, \ldots, \mathcal{I}_{2^q}^q$, for some parameter q which we define later. Using the $\widetilde{O}(n + t\sqrt{v})$ time Knapsack algorithm (Theorem 12), we compute a subarray of $\mathcal{P}_{\mathcal{I}_q^j}$ for every $j \in [2^q]$. The arrays are then combined in a tree-like fashion by taking their max-plus convolution. A key observation is that, with high probability, it suffices to compute a subarray of $\mathcal{P}_{\mathcal{I}_q^j}$ for a small range of indices and a small range of values. The same will hold for the intermediate arrays resulting from the max-plus convolutions. Since the sequences are monotone non-decreasing, we can use the rectangular bounded monotone max-plus convolution algorithm of Theorem 7 to accelerate the computation. We explain the algorithm in more details below before proving its correctness and analyzing its running time.

Algorithm. The algorithm of Lemma 15 is presented in pseudocode in Algorithm 1. Let $\widetilde{\mathrm{OPT}}$ be the approximation of OPT from Lemma 13, i.e. $\widetilde{\mathrm{OPT}}$ satisfies $\mathrm{OPT} \leqslant \widetilde{\mathrm{OPT}} \leqslant \mathrm{OPT} + p_{\max}$ and $p_{\max} \leqslant \widetilde{\mathrm{OPT}} \leqslant n \cdot p_{\max}$. Note that since $p_{\max} \leqslant \mathrm{OPT}$, we have $\widetilde{\mathrm{OPT}} = \Theta(\mathrm{OPT})$. Set the parameters $\eta := 17 \log n$ and q to be the largest integer such that $2^q \leqslant \min\{t/w_{\max}, \widetilde{\mathrm{OPT}}/p_{\max}\}$. We also define $\Delta_w := t \cdot w_{\max}$ and $\Delta_p := \widetilde{\mathrm{OPT}} \cdot p_{\max}$, as well as the weight and profit intervals for $\ell \in \{0, \dots, q\}$

$$W^\ell := \left[\frac{t}{2^\ell} - \sqrt{\frac{\Delta_w}{2^\ell}}\eta,\ \frac{t}{2^\ell} + \sqrt{\frac{\Delta_w}{2^\ell}}\eta\right] \quad \text{and} \quad P^\ell := \left[\frac{\widetilde{\mathrm{OPT}}}{2^\ell} - \sqrt{\frac{\Delta_p}{2^\ell}}\eta,\ \frac{\widetilde{\mathrm{OPT}}}{2^\ell} + \sqrt{\frac{\Delta_p}{2^\ell}}\eta\right].$$

Algorithm 1 The $\widetilde{O}(n + t\sqrt{p_{\max}})$-time algorithm of Lemma 15. The input $(\mathcal{I}, t)$ is a Knapsack instance such that $t/w_{\max} = \Theta(\mathrm{OPT}/p_{\max})$.

$w_{\max} \leftarrow \max_{i \in [n]} w_i$
$p_{\max} \leftarrow \max_{i \in [n]} p_i$
Compute an approximation $\widetilde{\mathrm{OPT}}$ of OPT using Lemma 13.
$q \leftarrow$ largest integer such that $2^q \leqslant \min\{t/w_{\max}, \widetilde{\mathrm{OPT}}/p_{\max}\}$
$\eta \leftarrow 17 \log n$
$\Delta_w \leftarrow t \cdot w_{\max}$
$\Delta_p \leftarrow \widetilde{\mathrm{OPT}} \cdot p_{\max}$
$\mathcal{I}_1^q, \dots, \mathcal{I}_{2^q}^q \leftarrow$ random partitioning of $\mathcal{I}$ into 2^q groups
$W^q \leftarrow \left[\frac{t}{2^q} - \sqrt{\frac{\Delta_w}{2^q}}\eta,\ \frac{t}{2^q} + \sqrt{\frac{\Delta_w}{2^q}}\eta\right]$
$P^q \leftarrow \left[\frac{\widetilde{\mathrm{OPT}}}{2^q} - \sqrt{\frac{\Delta_p}{2^q}}\eta,\ \frac{\widetilde{\mathrm{OPT}}}{2^q} + \sqrt{\frac{\Delta_p}{2^q}}\eta\right]$
$W^* \leftarrow \left[0, \frac{t}{2^q} + \sqrt{\frac{\Delta_w}{2^q}}\eta\right]$
$P^* \leftarrow \left[0, \frac{\widetilde{\mathrm{OPT}}}{2^q} + \sqrt{\frac{\Delta_p}{2^q}}\eta\right]$
for $j = 1, \dots, 2^q$ **do**
    Compute $D_j^q \leftarrow \mathcal{P}_{\mathcal{I}_j^q}[W^*; P^*]$ using Theorem 12
    $C_j^q \leftarrow D_j^q[W^q; P^q]$
for $\ell = q - 1, \dots, 0$ **do**
    $W^\ell \leftarrow \left[\frac{t}{2^\ell} - \sqrt{\frac{\Delta_w}{2^\ell}}\eta,\ \frac{t}{2^\ell} + \sqrt{\frac{\Delta_w}{2^\ell}}\eta\right]$
    $P^\ell \leftarrow \left[\frac{\widetilde{\mathrm{OPT}}}{2^\ell} - \sqrt{\frac{\Delta_p}{2^\ell}}\eta,\ \frac{\widetilde{\mathrm{OPT}}}{2^\ell} + \sqrt{\frac{\Delta_p}{2^\ell}}\eta\right]$
    for $j = 1, \dots, 2^\ell$ **do**
        $D_j^\ell \leftarrow \textsc{MaxConv}(C_{2j-1}^{\ell+1}, C_{2j}^{\ell+1})$ using Theorem 7
        $C_j^\ell \leftarrow D_j^\ell[W^\ell; P^\ell]$
$T \leftarrow [t - \sqrt{t \cdot w_{\max}}, t + \sqrt{t \cdot w_{\max}}]$
$P \leftarrow [\widetilde{\mathrm{OPT}} - \sqrt{\widetilde{\mathrm{OPT}} \cdot p_{\max}}, \widetilde{\mathrm{OPT}} + \sqrt{\widetilde{\mathrm{OPT}} \cdot p_{\max}}]$
return $C_1^0[T; P]$

Algorithm 1 starts by splitting the items of $\mathcal{I}$ into 2^q groups $\mathcal{I}_1^q, \dots, \mathcal{I}_{2^q}^q$ uniformly at random. For each group $\mathcal{I}_j^q$ it computes the sequence $D_j^q := \mathcal{P}_{\mathcal{I}_j^q}[W^*; P^*]$ using Theorem 12, where $W^* := \left[0, \frac{t}{2^q} + \sqrt{\frac{\Delta_w}{2^q}}\eta\right]$ and $P^* := \left[0, \frac{\widetilde{\mathrm{OPT}}}{2^q} + \sqrt{\frac{\Delta_p}{2^q}}\eta\right]$. Then it extracts the entries

corresponding to weights in W^q and profits in P^q, i.e., $C^j_q := D^q_j[W^q; P^q]$. Next, the algorithm iterates over the levels $\ell = q-1, \dots, 0$. For every iteration $j \in [2^\ell]$, the set of items in group j on level ℓ is $\mathcal{I}^\ell_j = \mathcal{I}^{\ell+1}_{2j-1} \cup \mathcal{I}^{\ell+1}_{2j}$ and the algorithm computes the max-plus convolution D^ℓ_j of the arrays $C^{\ell+1}_{2j-1}$ and $C^{\ell+1}_{2j}$. It extracts the relevant entries of weights in W^ℓ and profits in P^ℓ, i.e., $C^\ell_j := D^\ell_j[W^\ell; P^\ell]$. Finally, observe that when $\ell = 0$ then $\mathcal{I}^0_1 = \mathcal{I}$. The algorithm returns the sequence $C^0_1[T; P]$, for the intervals $T := [t - \sqrt{t \cdot w_{\max}}, t + \sqrt{t \cdot w_{\max}}]$ and $P := [\widetilde{\mathrm{OPT}} - \sqrt{\widetilde{\mathrm{OPT}} \cdot p_{\max}}, \widetilde{\mathrm{OPT}} + \sqrt{\widetilde{\mathrm{OPT}} \cdot p_{\max}}]$.

3.1 Correctness of Algorithm 1

Let us analyze the correctness of the algorithm. For the rest of this section, fix a Knapsack instance $(\mathcal{I}, t)$ with $n := |\mathcal{I}|$ and such that $t/w_{\max} = \Theta(\mathrm{OPT}/p_{\max})$. First, recall that we defined q to be the largest integer such that $2^q \leqslant \min\{t/w_{\max}, \widetilde{\mathrm{OPT}}/p_{\max}\}$. In particular, since $t \leqslant n w_{\max}$, we have $2^q \leqslant t/w_{\max} \leqslant n$. Moreover since $w_{\max} \leqslant t$ and $p_{\max} \leqslant \widetilde{\mathrm{OPT}}$, we have $2^q \geqslant 1$. So 2^q is a valid choice for the number of groups in which we split the item set $\mathcal{I}$. Also note that $2^q = \Theta(t/w_{\max}) = \Theta(\widetilde{\mathrm{OPT}}/p_{\max})$. Next, we argue that the subarray C^ℓ_j constructed in Lines 1.15 and 1.21 is monotone non-decreasing.

▶ **Lemma 16.** *For every level $\ell \in \{0, \dots, q\}$ and iteration $j \in [2^\ell]$, the sequence C^ℓ_j is monotone non-decreasing.*

Proof. For $\ell = q$ and $j \in [2^q]$, D^q_j is a subarray of $\mathcal{P}_{\mathcal{I}^q_j}$, which is monotone non-decreasing by definition. Hence D^q_j is monotone non-decreasing, and since W^q and P^q are intervals, the array $C^q_j = D^q_j[W^q; P^q]$ is also monotone non-decreasing. The statement follows from induction by noting that the max-plus convolution of two monotone non-decreasing sequences is a monotone non-decreasing sequence. ◀

The above lemma justifies the use of Theorem 7 to compute the max-plus convolutions in Line 1.20. We now explain why it is enough to restrict the entries of D^ℓ_j corresponding to indices in W^ℓ and values in P^ℓ. The following lemma shows that, for any fixed subset of items, the weight and profit of that subset restricted to $\mathcal{I}^\ell_j$ are concentrated around their expectations.

▶ **Lemma 17.** *Let $x \in \{0,1\}^n$. Fix $\ell \in \{0, \dots, q\}$ and $j \in [2^\ell]$. Then with probability at least $1 - 1/n^7$ the following holds:*

$$\left| w_{\mathcal{I}^\ell_j}(x) - w_{\mathcal{I}}(x)/2^\ell \right| \leqslant \sqrt{\Delta_w/2^\ell} \cdot 16 \log n \quad \text{and} \quad \left| p_{\mathcal{I}^\ell_j}(x) - p_{\mathcal{I}}(x)/2^\ell \right| \leqslant \sqrt{\Delta_p/2^\ell} \cdot 16 \log n.$$

Proof. By construction, $\mathcal{I}^\ell_j$ is a random subset of $\mathcal{I}$ where each item is included with probability $p := 1/2^\ell$. For each item $i \in [n]$, let Z_i be a random variable taking value $w_i x_i$ with probability p, and 0 with probability $1-p$. Then $Z := \sum_{i=1}^n Z_i$ has the same distribution as $w_{\mathcal{I}^\ell_j}(x)$ and $\mathbb{E}(Z) = w_{\mathcal{I}}(x)p$.

Using Bernstein's inequality (see, e.g., [13, Theorem 1.2]) we get that for any $\lambda > 0$:

$$\begin{aligned}
\mathbb{P}(|Z - \mathbb{E}(Z)| \geqslant \lambda) &\leqslant 2 \exp\left(- \frac{\lambda^2}{2 \cdot \mathrm{Var}(Z) + \frac{2}{3}\lambda \cdot w_{\max}} \right) \\
&\leqslant 2 \exp\left(- \min\left\{ \frac{\lambda^2}{4 \cdot \mathrm{Var}(Z)}, \frac{\lambda}{2 w_{\max}} \right\} \right)
\end{aligned}$$

Set $\lambda := \sqrt{p \cdot \Delta_w} \cdot 16 \log n$. We can bound the variance of Z as follows:

$$\begin{aligned}\mathrm{Var}(Z) = \sum_{i=1}^{n} p(1-p) w_i^2 x_i^2 &\leqslant p \cdot w_{\max} \sum_{i=1}^{n} w_i x_i \\ &= p \cdot w_{\max} \cdot w_{\mathcal{I}}(x) \leqslant p \cdot w_{\max} \cdot t = p \cdot \Delta_w.\end{aligned}$$

Hence $\lambda^2/(4 \cdot \mathrm{Var}(Z)) \geqslant 16 \log n$. To bound $\lambda/(2w_{\max})$, note that $2^q \leqslant t/w_{\max}$ so $p = \frac{1}{2^\ell} \geqslant \frac{1}{2^q} \geqslant \frac{w_{\max}}{t}$. Thus,

$$\frac{\lambda}{2w_{\max}} = \frac{\sqrt{p \cdot \Delta_w} \cdot 16 \log n}{2w_{\max}} \geqslant 8 \log n.$$

Combining all the above we obtain that

$$|w_{\mathcal{I}_j^\ell}(x) - w_{\mathcal{I}}(x)/2^\ell| = |Z - \mathbb{E}(Z)| \leqslant \lambda = \sqrt{\Delta_w/2^\ell} \cdot 16 \log n$$

holds with probability at least $1 - 2/n^8$.

We can apply a similar reasoning on $p_{\mathcal{I}_j^\ell}(x)$ and get the analogous result that $|p_{\mathcal{I}_j^\ell}(x) - p_{\mathcal{I}}(x)/2^\ell| \leqslant \sqrt{\Delta_p/2^\ell} \cdot 16 \log n$ holds with probability at least $1 - 2/n^8$. To this end, we define a random variable Y, analogous to Z, with respect to profits and set the constant in Bernstein's inequality to $\lambda = \sqrt{p \cdot \Delta_p} \cdot 16 \log n$. Then to bound $\mathrm{Var}(Y)$ we use $p_{\mathcal{I}}(x) \leqslant \mathrm{OPT} \leqslant \widetilde{\mathrm{OPT}}$, and to bound $\lambda/(2p_{\max})$ we use the fact that $2^q \leqslant \widetilde{\mathrm{OPT}}/p_{\max}$ so that $p \geqslant p_{\max}/\widetilde{\mathrm{OPT}}$. By a union bound, both events hold with probability at least $1 - 4/n^8 \geqslant 1 - 1/n^7$ (recall that we can assume $n \geqslant 10$). ◀

In the next lemma, we show that, as a consequence of Lemma 17, at level ℓ the weights and profits of solutions of interest restricted to $\mathcal{I}_j^q$ lie with sufficiently high probability in W^ℓ and P^ℓ.

▶ **Lemma 18.** *Let $x \in \{0,1\}^n$ such that $|w_{\mathcal{I}}(x) - t| \leqslant 2\sqrt{\Delta_w}$ and $|p_{\mathcal{I}}(x) - \widetilde{\mathrm{OPT}}| \leqslant 2\sqrt{\Delta_p}$. Fix a level $\ell \in \{0, \ldots, q\}$ and an iteration $j \in [2^\ell]$. Then with probability at least $1 - 1/n^7$ we have $w_{\mathcal{I}_j^\ell}(x) \in W^\ell$ and $p_{\mathcal{I}_j^\ell}(x) \in P^\ell$.*

Proof. By Lemma 17 we have with probability at least $1 - 1/n^7$

$$\left|w_{\mathcal{I}_j^\ell}(x) - w_{\mathcal{I}}(x)/2^\ell\right| \leqslant \sqrt{\Delta_w/2^\ell} 16 \log n \quad \text{and} \quad \left|p_{\mathcal{I}_j^\ell}(x) - p_{\mathcal{I}}(x)/2^\ell\right| \leqslant \sqrt{\Delta_p/2^\ell} \cdot 16 \log n.$$

We condition on that event. Since $|w_{\mathcal{I}}(x) - t| \leqslant 2\sqrt{\Delta_w}$, we have:

$$\begin{aligned}|w_{\mathcal{I}_j^\ell}(x) - t/2^\ell| &\leqslant |w_{\mathcal{I}_j^\ell}(x) - w_{\mathcal{I}}(x)/2^\ell| + \frac{1}{2^\ell}|w_{\mathcal{I}}(x) - t| \\ &\leqslant \sqrt{\Delta_w/2^\ell} \cdot 16 \log n + 2\sqrt{\Delta_w}/2^\ell \leqslant \sqrt{\Delta_w/2^\ell} \cdot 17 \log n.\end{aligned}$$

Here the last step follows from $2^\ell \geqslant 1$ and $n \geqslant 10$. Since we set $\eta = 17 \log n$, the above implies that $w_{\mathcal{I}_j^\ell}(x) \in W^\ell$. Similarly, we can deduce from $|p_{\mathcal{I}}(x) - \widetilde{\mathrm{OPT}}| \leqslant 2\sqrt{\Delta_p}$ that $p_{\mathcal{I}_j^\ell}(x) \in P^\ell$. ◀

Using Lemma 18 we can argue that at level ℓ it suffices to compute the subarray of D_j^ℓ corresponding to indices in W^ℓ and values in P^ℓ. We make this idea precise in Lemma 19.

▶ **Lemma 19.** *Let $x \in \{0,1\}^n$ be a Pareto optimum of $\mathcal{P}_\mathcal{I}$ satisfying $|w_\mathcal{I}(x) - t| \leqslant 2\sqrt{\Delta_w}$ and $|p_\mathcal{I}(x) - \widetilde{\mathrm{OPT}}| \leqslant 2\sqrt{\Delta_p}$. Then with probability at least $1 - 1/n^5$ we have for all $\ell \in \{0,\dots,q\}$ and all $j \in [2^\ell]$ that $w_{\mathcal{I}_j^\ell}(x) \in W^\ell$, $p_{\mathcal{I}_j^\ell}(x) \in P^\ell$ and $C_j^\ell[w_{\mathcal{I}_j^\ell}(x)] = p_{\mathcal{I}_j^\ell}(x)$.*

Proof. By Lemma 18, for fixed $\ell \in \{0,\dots,q\}$ and $j \in [2^\ell]$ we have $w_{\mathcal{I}_j^\ell}(x) \in W^\ell$ and $p_{\mathcal{I}_j^\ell}(x) \in P^\ell$ with probability at least $1 - 1/n^7$. Since $2^q \leqslant n$ we can afford a union bound and deduce that $w_{\mathcal{I}_j^\ell}(x) \in W^\ell$ and $p_{\mathcal{I}_j^\ell}(x) \in P^\ell$ holds *for all* $\ell \in \{0,\dots,q\}$ and *for all* $j \in [2^\ell]$ with probability at least $1 - 1/n^5$. We condition on that event and prove by induction that $C_j^\ell[w_{\mathcal{I}_j^\ell}(x)] = p_{\mathcal{I}_j^\ell}(x)$ for all $\ell \in \{0,\dots,q\}$ and all $j \in [2^\ell]$.

For the base case, fix $\ell = q$ and $j \in [2^\ell]$. Recall that $\mathcal{P}_{\mathcal{I}_j^q}[k]$ is the maximum profit of a subset of items of $\mathcal{I}_j^q$ of weight at most k. So if y is such that $\mathcal{P}_{\mathcal{I}_j^q}[w_{\mathcal{I}_j^q}(x)] = p_{\mathcal{I}_j^q}(y)$ and $w_{\mathcal{I}_j^q}(y) \leqslant w_{\mathcal{I}_j^q}(x)$, then $p_{\mathcal{I}_j^q}(y) \geqslant p_{\mathcal{I}_j^q}(x)$. By Lemma 14, since x is a Pareto optimum of $\mathcal{P}_\mathcal{I}$, we deduce $p_{\mathcal{I}_j^q}(y) = p_{\mathcal{I}_j^q}(x)$. We have $w_{\mathcal{I}_j^q}(x) \in W^q$ and $\mathcal{P}_{\mathcal{I}_j^q}[w_{\mathcal{I}_j^q}(x)] = p_{\mathcal{I}_j^q}(x) \in P^q$, so by the construction in Line 1.15 $C_j^q[w_{\mathcal{I}_j^q}(x)] = p_{\mathcal{I}_j^q}(x)$.

In the inductive step, fix $\ell < q$ and $j \in [2^\ell]$. We want to prove that $D_j^\ell[w_{\mathcal{I}_j^\ell}] = p_{\mathcal{I}_j^\ell}(x)$. Indeed, since $w_{\mathcal{I}_j^\ell}(x) \in W^\ell$ and $p_{\mathcal{I}_j^\ell}(x) \in P^\ell$, this shows that $C_j^\ell[w_{\mathcal{I}_j^\ell}(x)] = D_j^\ell[w_{\mathcal{I}_j^\ell}] = p_{\mathcal{I}_j^\ell}(x)$. By induction, $D_j^\ell[w_{\mathcal{I}_j^\ell}(x)]$ is the profit of some subset of items of $\mathcal{I}_j^\ell$ of weight at most $w_{\mathcal{I}_j^\ell}(x)$. So there exists $y \in \{0,1\}^n$ such that $D_j^\ell[w_{\mathcal{I}_j^\ell}(x)] = p_{\mathcal{I}_j^\ell}(y)$ and $w_{\mathcal{I}_j^\ell}(y) \leqslant w_{\mathcal{I}_j^\ell}(x)$. Then

$$\begin{aligned} p_{\mathcal{I}_j^\ell}(y) = D_j^\ell[w_{\mathcal{I}_j^q}(x)] &= \max\left\{ C_{2j-1}^{\ell+1}[k] + C_{2j}^{\ell+1}[k'] \;:\; k + k' = w_{\mathcal{I}_j^\ell}(x) \right\} \\ &\geqslant C_{2j-1}^{\ell+1}[w_{\mathcal{I}_{2j-1}^{\ell+1}}(x)] + C_{2j}^{\ell+1}[w_{\mathcal{I}_{2j}^{\ell+1}}(x)] \\ &= p_{\mathcal{I}_{2j-1}^{\ell+1}}(x) + p_{\mathcal{I}_{2j}^{\ell+1}}(x) = p_{\mathcal{I}_j^\ell}(x) \end{aligned}$$

where we use the induction hypothesis and the fact that $\mathcal{I}_j^\ell = \mathcal{I}_{2j-1}^{\ell+1} \cup \mathcal{I}_{2j}^{\ell+1}$ is a partitioning. Recall that we interpret out-of-bound entries of arrays as $-\infty$. Since x is a Pareto optimum of $\mathcal{P}_\mathcal{I}$, we obtain $p_{\mathcal{I}_j^\ell}(y) = p_{\mathcal{I}_j^\ell}(x)$ by Lemma 14, and thus $D_j^\ell[w_{\mathcal{I}_j^\ell}(x)] = p_{\mathcal{I}_j^\ell}(x)$. This implies $C_j^\ell[w_{\mathcal{I}_j^q}(x)] = p_{\mathcal{I}_j^q}(x)$ as argued above. ◀

Finally, we can prove that Algorithm 1 correctly computes $\mathcal{P}_\mathcal{I}[T;P]$ as defined in Lemma 15 with high probability. Note that we can boost the success probability to any polynomial by repeating this algorithm and taking the entry-wise maximum of each computed sequence C_1^0.

▶ **Lemma 20** (Correctness of Algorithm 1). *Let $T := [t - \sqrt{\Delta_w}, t + \sqrt{\Delta_w}]$ and $P := [\widetilde{\mathrm{OPT}} - \sqrt{\Delta_p}, \widetilde{\mathrm{OPT}} + \sqrt{\Delta_p}]$. Then with probability at least $1 - 1/n$ we have $C_1^0[T;P] = \mathcal{P}_\mathcal{I}[T;P]$.*

Proof. First, observe that $T \subseteq W^0$ and $P \subseteq P^0$. Let K^0 be the set of indices of C_1^0, i.e., $K^0 := \{k \mid k \in W^0, D_1^0[k] \in P^0\}$. Let K be the interval such that $C_1^0[K] = C_1^0[T;P]$, i.e., $K := \{k \mid k \in T, C_1^0[k] \in P\}$. We want to show that $C_1^0[k] = \mathcal{P}_\mathcal{I}[k]$ for every $k \in K$ with high probability. Since C_1^0 and $\mathcal{P}_\mathcal{I}$ are monotone non-decreasing (see Lemma 16), to compare C_1^0 and $\mathcal{P}_\mathcal{I}$ it is enough to focus on break points. Recall that $k \in \mathbb{N}$ is a break point of $\mathcal{P}_\mathcal{I}$ if $\mathcal{P}_\mathcal{I}[k-1] < \mathcal{P}_\mathcal{I}[k]$, and that each break point k corresponds to a Pareto optimum $x \in \{0,1\}^n$ such that $w_\mathcal{I}(x) = k$ and $\mathcal{P}_\mathcal{I}[w_\mathcal{I}(x)] = p_\mathcal{I}(x)$. We claim that for any $k \in K$ and $k' \leqslant k$ maximal such that k' is a break point of $\mathcal{P}_\mathcal{I}$ we have $C_1^0[k'] = \mathcal{P}_\mathcal{I}[k']$. Together with monotonicity this proves that $C_1^0[k] = \mathcal{P}_\mathcal{I}[k]$ for all $k \in K$ as desired.

To prove the claim, we first need to establish that every $k \in K$ has a break point $k' \leqslant k$ that is not too far, specifically $k' \geqslant t - 2\sqrt{\Delta_w}$. We prove that $[t - 2\sqrt{\Delta_w}, t - \sqrt{\Delta_w}]$ contains a break point of $\mathcal{P}_\mathcal{I}$. Let $y \in \{0,1\}^n$ be such that $w_\mathcal{I}(y) \leqslant t - 2\sqrt{\Delta_w}$ and $\mathcal{P}_\mathcal{I}[t - 2\sqrt{\Delta_w}] = p_\mathcal{I}(y)$.

Let y' be y with an additional item. This is always possible since we can assume without loss of generality that the total weight of all items in $\mathcal{I}$ exceeds t, i.e., any subset of items of weight at most t leaves at least one item out. The additional item has weight at most $w_{\max}$ and profit at least 1. So $w_{\mathcal{I}}(y') \leqslant w_{\mathcal{I}}(y) + w_{\max} \leqslant t - 2\sqrt{\Delta_w} + w_{\max} \leqslant t - \sqrt{\Delta_w}$ and $p_{\mathcal{I}}(y) < p_{\mathcal{I}}(y')$. In particular, we have $p_{\mathcal{I}}(y') \leqslant \mathcal{P}_{\mathcal{I}}[t - \sqrt{\Delta_w}]$. We obtain $\mathcal{P}_{\mathcal{I}}[t - 2\sqrt{\Delta_w}] = p_{\mathcal{I}}(y) < p_{\mathcal{I}}(y') \leqslant \mathcal{P}_{\mathcal{I}}[t - \sqrt{\Delta_w}]$. Therefore, $[t - 2\sqrt{\Delta}_w, t - \sqrt{\Delta}_w]$ contains a break point.

Recall that our goal is to show that for any $k \in K$ and $k' \leqslant k$ maximal such that k' is a break point of $\mathcal{P}_{\mathcal{I}}$ it holds that $C_1^0[k'] = \mathcal{P}_{\mathcal{I}}[k']$. Since we showed that $[t - 2\sqrt{\Delta_w}, t - \sqrt{\Delta_w}]$ contains a break point, we define $T' := T \cup [t - 2\sqrt{\Delta_w}, t - \sqrt{\Delta_w}] = [t - 2\sqrt{\Delta_w}, t + \sqrt{\Delta_w}]$, and K' such that $C_1^0[K'] = C_1^0[T'; P]$, i.e., $K' := \{k \mid k \in T', C_1^0[k] \in P\}$. Then all it remains to show is that $C_1^0[k] = \mathcal{P}_{\mathcal{I}}[k]$ for every break point $k \in K'$. Fix a break point $k \in K'$ and let $x \in \{0,1\}^n$ be the Pareto optimum such that $w_{\mathcal{I}}(x) = k$ and $\mathcal{P}_{\mathcal{I}}[k] = p_{\mathcal{I}}(x)$. Then in particular $w_{\mathcal{I}}(x) \in T'$ and $p_{\mathcal{I}}(x) \in P$, and thus $|w_{\mathcal{I}}(x) - t| \leqslant 2\sqrt{\Delta_w}$ and $|p_{\mathcal{I}}(x) - \widetilde{\mathrm{OPT}}| \leqslant 2\sqrt{\Delta_p}$. By Lemma 19, this implies $C_1^0[k] = p_{\mathcal{I}}(x) = \mathcal{P}_{\mathcal{I}}[k]$ with probability at least $1 - 1/n^5$. Since $|K'| \leqslant |P| \leqslant 2p_{\max}\sqrt{n}$, by a union bound over all break points $k \in K'$, we obtain that $C_1^0[T; P] = \mathcal{P}_{\mathcal{I}}[T; P]$ with probability at least $1 - 2p_{\max}\sqrt{n}/n^5 \geqslant 1 - 2p_{\max}/n^4$. Note that if $n^3 \leqslant 2p_{\max}$, then in particular $n^2 \leqslant 2p_{\max}$ and we can use Bellman's dynamic program to compute the profit sequence in time $O(n \cdot t) = O(t\sqrt{p_{\max}})$ (see Theorem 11). Hence, we can assume that $n^3 \geqslant 2p_{\max}$. Thus, with probability at least $1 - 2p_{\max}/n^4 \geqslant 1 - 1/n$ we have $C_1^0[T; P] = \mathcal{P}_{\mathcal{I}}[T; P]$. ◀

3.2 Running time of Algorithm 1

▶ **Lemma 21.** *For a fixed level $\ell \in \{0, \dots, q-1\}$ and iteration $j \in [2^\ell]$, the computation of D_j^ℓ in Line 1.20 takes time $\widetilde{O}((t/2^\ell)^{3/4} p_{\max}^{1/2} w_{\max}^{1/4})$.*

Proof. By Lemma 16, the sequences $C_{2j-1}^{\ell+1}$ and $C_{2j}^{\ell+1}$ are bounded monotone. Additionally, they have length at most $|W^{\ell+1}| = \widetilde{O}(\sqrt{\Delta_w/2^\ell})$ and the values are in a range of length at most $|P^\ell| = \widetilde{O}(\sqrt{\Delta_p/2^\ell})$. So their max-plus convolution can be computed using the algorithm of Theorem 7 in time $\widetilde{O}((\Delta_w/2^\ell)^{1/2}(\Delta_p/2^\ell)^{1/4})$. We apply the definitions of $\Delta_w = t w_{\max}$ and $\Delta_p = \widetilde{\mathrm{OPT}} p_{\max}$ and the balancedness assumption $t/w_{\max} = \Theta(\widetilde{\mathrm{OPT}}/p_{\max})$, which yields $\Delta_p = O(t p_{\max}^2 / w_{\max})$, to bound the running time by $\widetilde{O}((t/2^\ell)^{3/4} p_{\max}^{1/2} w_{\max}^{1/4})$. ◀

▶ **Lemma 22.** *Algorithm 1 runs in time $\widetilde{O}(n + t\sqrt{p_{\max}})$.*

Proof. We first bound the running time of the base case, i.e., the computations of Lines 1.13–1.15. For each $j \in [2^q]$, the array D_j^q is obtained by computing the sequence $\mathcal{P}_{\mathcal{I}_j^q}[W^*; P^*]$, where $W^* := \left[0, \frac{t}{2^q} + \sqrt{\frac{\Delta_w}{2^q}\eta}\right]$ and $P^* := \left[0, \frac{\widetilde{\mathrm{OPT}}}{2^q} + \sqrt{\frac{\Delta_p}{2^q}\eta}\right]$. Since $\Delta_w = t w_{\max}$, $\eta = O(\log n)$ and $2^q = \Theta(t/w_{\max})$, we can bound $\frac{t}{2^q} + \sqrt{\frac{\Delta_w}{2^q}\eta} = \widetilde{O}(w_{\max})$, and analogously $\frac{\widetilde{\mathrm{OPT}}}{2^q} + \sqrt{\frac{\Delta_p}{2^q}\eta} = \widetilde{O}(p_{\max})$. Using Theorem 12, we can therefore compute D_j^q in time $\widetilde{O}(|\mathcal{I}_j^q| + w_{\max}\sqrt{p_{\max}})$. Hence, the total running time of the base case is:

$$\sum_{j=1}^{2^q} \widetilde{O}\left(|\mathcal{I}_j^q| + w_{\max} \cdot \sqrt{p_{\max}}\right) = \widetilde{O}\left(n + 2^q \cdot w_{\max} \cdot \sqrt{p_{\max}}\right) = \widetilde{O}\left(n + t \cdot \sqrt{p_{\max}}\right)$$

where we again used $2^q = \Theta(t/w_{\max})$.

Using Lemma 21, we bound the running time of the combination step, i.e., the computations of Lines 1.16–1.21, as follows:

$$\sum_{\ell=0}^{q-1}\sum_{j=1}^{2^\ell}\widetilde{O}\left((t/2^\ell)^{3/4}p_{\max}^{1/2}w_{\max}^{1/4}\right)=\sum_{\ell=0}^{q-1}\widetilde{O}\left(t^{3/4}p_{\max}^{1/2}(w_{\max}\cdot 2^\ell)^{1/4}\right)$$

This is a geometric series, so it is bounded by $\widetilde{O}(t^{3/4}p_{\max}^{1/2}(w_{\max}\cdot 2^q)^{1/4})$. Since $2^q \leqslant t/w_{\max}$ we obtain a running time of $\widetilde{O}(t\sqrt{p_{\max}})$. Hence, in total Algorithm 1 takes time $\widetilde{O}(n+t\sqrt{p_{\max}})$. ◀

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 and Christos Tzamos. Capacitated dynamic programming: Faster knapsack and graph algorithms. In *46th International Colloquium on Automata, Languages, and Programming, ICALP 2019*, volume 132 of *LIPIcs*, pages 19:1–19:13. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPICS.ICALP.2019.19.

3 MohammadHossein Bateni, MohammadTaghi Hajiaghayi, Saeed Seddighin, and Cliff Stein. Fast algorithms for knapsack via convolution and prediction. In *Proceedings of the 50th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2018*, pages 1269–1282. ACM, 2018. doi:10.1145/3188745.3188876.

4 Richard Bellman. Notes on the theory of dynamic programming IV - maximization over discrete sets. *Naval Research Logistics Quarterly*, 3(1-2):67–70, March 1956. doi:10.1002/nav.3800030107.

5 Karl Bringmann. A near-linear pseudopolynomial time algorithm for subset sum. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017*, pages 1073–1084. SIAM, 2017. doi:10.1137/1.9781611974782.69.

6 Karl Bringmann. Knapsack with small items in near-quadratic time. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 259–270. ACM, 2024. doi:10.1145/3618260.3649719.

7 Karl Bringmann and Alejandro Cassis. Faster knapsack algorithms via bounded monotone min-plus-convolution. In *49th International Colloquium on Automata, Languages, and Programming, ICALP 2022*, volume 229 of *LIPIcs*, pages 31:1–31:21. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICALP.2022.31.

8 Karl Bringmann and Alejandro Cassis. Faster 0-1-knapsack via near-convex min-plus-convolution. In *31st Annual European Symposium on Algorithms, ESA 2023*, volume 274 of *LIPIcs*, pages 24:1–24:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.ESA.2023.24.

9 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3SUM via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015*, pages 31–40. ACM, 2015. doi:10.1145/2746539.2746568.

10 Lin Chen, Jiayi Lian, Yuchen Mao, and Guochuan Zhang. Faster algorithms for bounded knapsack and bounded subset sum via fine-grained proximity results. In *Proceedings of the 2024 ACM-SIAM Symposium on Discrete Algorithms, SODA 2024*, pages 4828–4848. SIAM, 2024. doi:10.1137/1.9781611977912.171.

11 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *STOC '22: 54th Annual ACM SIGACT Symposium on Theory of Computing*, pages 1529–1542. ACM, 2022. doi:10.1145/3519935.3520057.

12 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.

13 Devdatt P. Dubhashi and Alessandro Panconesi. *Concentration of Measure for the Analysis of Randomized Algorithms*. Cambridge University Press, 2009. URL: http://www.cambridge.org/gb/knowledge/isbn/item2327542/.

14 Qizheng He and Zhean Xu. Simple and faster algorithms for knapsack. In *2024 Symposium on Simplicity in Algorithms, SOSA 2024*, pages 56–62. SIAM, 2024. doi:10.1137/1.9781611977936.6.

15 Ce Jin. Solving knapsack with small items via L0-proximity. *CoRR*, abs/2307.09454, 2023. arXiv:2307.09454, doi:10.48550/arXiv.2307.09454.

16 Ce Jin. 0-1 knapsack in nearly quadratic time. In Bojan Mohar, Igor Shinkar, and Ryan O'Donnell, editors, *Proceedings of the 56th Annual ACM Symposium on Theory of Computing, STOC 2024, Vancouver, BC, Canada, June 24-28, 2024*, pages 271–282. ACM, 2024. doi:10.1145/3618260.3649618.

17 Richard M. Karp. Reducibility among combinatorial problems. In *Proceedings of a symposium on the Complexity of Computer Computations, 1972*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972. doi:10.1007/978-1-4684-2001-2_9.

18 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.

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

20 Marvin Künnemann, Ramamohan Paturi, and Stefan Schneider. On the fine-grained complexity of one-dimensional dynamic programming. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ICALP.2017.21.

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

22 Adam Polak, Lars Rohwedder, and Karol Wegrzycki. Knapsack and subset sum with small items. In *48th International Colloquium on Automata, Languages, and Programming, ICALP 2021*, volume 198 of *LIPIcs*, pages 106:1–106:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021. doi:10.4230/LIPICS.ICALP.2021.106.