Convolution and Knapsack in Higher Dimensions

Kilian Grage
Kiel University, Germany

Klaus Jansen
Kiel University, Germany

Björn Schumacher
Kiel University, Germany

Abstract

In the Knapsack problem, one is given the task of packing a knapsack of a given size with items in order to gain a packing with a high profit value. As one of the most classical problems in computer science, research for this problem has gone a long way. One important connection to the $(\max, +)$-convolution problem has been established, where knapsack solutions can be combined by building the convolution of two sequences. This observation has been used in recent years to give conditional lower bounds but also parameterized algorithms.

In this paper we carry these results into higher dimensions. We consider Knapsack where items are characterized by multiple properties – given through a vector – and a knapsack that has a capacity vector. The packing must not exceed any of the given capacity constraints. In order to show a similar sub-quadratic lower bound we consider a multidimensional version of $(\max, +)$-convolution. We then consider variants of this problem introduced by Cygan et al. and prove that they are all equivalent in terms of algorithms that allow for a running time sub-quadratic in the number of entries of the array.

We further develop a parameterized algorithm to solve higher dimensional Knapsack. The techniques we apply are inspired by an algorithm introduced by Axiotis and Tzamos. We will show that even for higher dimensional Knapsack, we can reduce the problem to convolution on one-dimensional, concave sequences, leading to an $\mathcal{O}(dn + dD \cdot \max\{\Pi_{i=1}^{d} t_i, t_{\max} \log t_{\max}\})$ algorithm, where D is the number of different weight vectors, t the capacity vector and d is the dimension of the problem. Then, we use the techniques to improve the approach of Eisenbrand and Weismantel to obtain an algorithm for Integer Linear Programming with upper bounds with running time $\mathcal{O}(dn) + D \cdot \mathcal{O}(d\Delta)^{d(d+1)} + T_{\text{LP}}$.

Finally, we give an divide-and-conquer algorithm for ILP with running time $n^{d+1} \cdot \mathcal{O}(\Delta)^d \cdot \log(\|u - \ell\|_\infty)$.

2012 ACM Subject Classification Theory of computation → Complexity classes; Theory of computation → Parameterized complexity and exact algorithms

Keywords and phrases Knapsack, Convolution, Integer Linear Programming

Digital Object Identifier 10.4230/LIPIcs.WADS.2025.30

Related Version *Full Version*: `https://arxiv.org/abs/2403.16117` [12]

Funding *Klaus Jansen*: Supported by the German Research Foundation (DFG) project JA 612/25-1.
Björn Schumacher: Supported by the German Research Foundation (DFG) project JA 612/25-1.

Acknowledgements We want to thank the anonymous referees for their time and helpful feedback.

1 Introduction

The Knapsack problem is one of the core problems in computer science. The task of finding a collection of items that fits into a knapsack but also maximizes some profit is NP-hard and as such the aim is to find approximation algorithms, parameterized algorithms, and determine lower bounds for the running time of exact algorithms.

Cygan et al. [9] and Künnemann et al. [16] among others have used the following relationship between Knapsack and the $(\max,+)$-convolution problem. Assume we are given two disjoint item sets A and B and a knapsack of size t. If we additionally know the optimal profits for all knapsack sizes $t' \leq t$ we then can calculate the maximum profits for all sizes t' for $A \cup B$ by using convolution. This connection was used in order to show that the existence of a sub-quadratic (in terms of capacity) algorithm for Knapsack is equivalent to the existence of a sub-quadratic algorithm for $(\max,+)$-convolution.

In this work, we consider these problems in higher dimensions. The Knapsack problem is simply generalized by replacing the single value weight and capacity by vectors. We then look for a collection of items whose summed up vectors do not exceed the capacity vector in any position. The natural question arises whether a similar quadratic time lower bound exists for this problem. We answer this question positively by giving a generalization of convolution in higher dimensions as well. Using this generalization and techniques introduced by Bringmann [4] such as color-coding we are able to achieve similar subquadratic lower bounds as in the one-dimensional case.

1.1 Problem Definitions and Notations

We define $[k] \coloneqq \{i \in \mathbb{N} : 1 \leq i \leq k\}$, $[k]_0 \coloneqq [k] \cup \{0\}$, and $[a,b] \coloneqq \{i \in \mathbb{Z} : a \leq i \leq b\}$ for $k, a, b \in \mathbb{Z}$. In the following, we write for two vectors $v, u \in \mathbb{R}^d$ that $v \leq u$ (resp. $v < u$) if for all $i \in [d]$ we have that $v_i \leq u_i$ (resp. $v_i < u_i$). Further we denote with $v_{\max} = \max_{i \in [d]} v_i$ for any vector $v \in \mathbb{R}^d$. With $\vec{\mathbf{k}} \in \mathbb{R}^d$ we denote the vector that has $k \in \mathbb{R}$ in every position.

▶ **Definition 1.** *Let $L \in \mathbb{N}^d$ be a d-dimensional vector and $A = (A_{i_1 i_2 \cdots i_d})$ be a $L_1 \times L_2 \times \cdots \times L_d$* array*. We call the vector L the* size *of A and denote the number of entries in A with $\Pi(L) \coloneqq \prod_{i=1}^{d} L_i$.*

We call a vector $v \in \mathbb{Z}$ with $\vec{\mathbf{0}} \leq v \leq L - \vec{\mathbf{1}}$ position of array A. For ease of notation, we will denote for any array position v of A the respective array entry $A_{v_1 v_2 \cdots v_d}$ with A_v.

We note that in this definition, array positions lie in between $\vec{\mathbf{0}}$ and $L - \vec{\mathbf{1}}$. This makes it easier to work with positions for convolution and also for formulating time complexity bounds, as $\Pi(L)$ will be the main parameter we consider.

▶ **Definition 2** (Maximum Convolution)**.** *Given two d-dimensional arrays A, B with equal size L, the $(\max,+)$-convolution of A and B denoted as $A \oplus B$ is defined as an array C of size L with $C_v \coloneqq \max_{u \leq v} A_u + B_{v-u}$ for any $v < L$.*

d-MaxConv	
Input:	Two d-dimensional arrays A, B with equal size L.
Problem:	Compute the array $C \coloneqq A \oplus B$ of size L.

Note that in the following we will refer to this problem as "Convolution". We specifically limit ourselves to the special truncated case where both input arrays and the output array have the same size. In a more general setting, we could allow arrays of sizes $L^{(1)}, L^{(2)}$ as input and compute an array of size $L^{(1)} + L^{(2)} - \vec{\mathbf{1}}$.

To measure the running time of our algorithms, we mainly consider the size or rather the number of entries from the resulting array because we need to calculate a value for every position. Therefore, in terms of theoretical performance, working with different sizes or calculating an array of combined size will not make a difference. By only considering arrays of the same size, we avoid many unnecessary cases and the dummy values of $-\infty$.

There is a quadratic time algorithm for d-MaxConv like in the one-dimensional case. This algorithm simply iterates through all pairs of positions in time $\mathcal{O}(\Pi(L)^2) \subseteq \mathcal{O}(L_{\max}^{2d})$.

Further we define an upper bound test for the convolution problem. In this problem, we are given a third input array and need to decide whether its entries are upper bounds for the entries of the convolution.

d-MaxConv UpperBound	
Input:	Three d-dimensional arrays A, B, C with equal size L.
Problem:	Decide whether $(A \oplus B)_v \leq C_v$ for all $v < L$.

We further generalize the notion of superadditivity to multidimensional arrays.

d-SuperAdditivity Testing	
Input:	One d-dimensional array A of size L.
Problem:	Decide whether A is superadditive, i.e., $A_v \geq (A \oplus A)_v$ for all $v < L$.

The next problem class we consider is the d-Knapsack problem. In the one-dimensional case, one is given a set of items with weights and profits and one knapsack with a certain weight capacity. The goal is now to pack the knapsack such that the profit is maximized and the total weight of packed items does not exceed the weight capacity.

The natural higher dimensional generalization arises when we have more constraints to fulfill. When going on a journey by plane, one for example has several further requirements such as a maximum amount of allowed liquid or number of suitcases. By imposing more similar requirements, we can simply identify each item by a vector and also define the knapsack by a capacity vector. This leads to the following generalization of Knapsack into higher dimensions. We further differentiate between two problems 0/1 d-Knapsack and Unbounded d-Knapsack, depending on whether we allow items to be only used one time or an arbitrary number of times.

0/1 d-Knapsack	
Input:	Set I of n items each defined by a profit $p_i \in \mathbb{R}_{\geq 0}$ and weight vector $w^{(i)} \in \mathbb{N}^d$, along with a knapsack of capacity $t \in \mathbb{N}^d$.
Problem:	Find subset $S \subseteq I$ such that $\sum_{i \in S} w^{(i)} \leq t$ and $\sum_{i \in S} p_i$ is maximal.

Unbounded d-Knapsack	
Input:	Set I of n items each defined by a profit $p_i \in \mathbb{R}_{\geq 0}$ and weight vector $w^{(i)} \in \mathbb{N}^d$, along with a knapsack of capacity $t \in \mathbb{N}^d$.
Problem:	Find a multi-set $S \subseteq I$ such that $\sum_{i \in S} w^{(i)} \leq t$ and $\sum_{i \in S} p_i$ is maximal.

The running time for these problems is mainly dependent on the dimension d, the number of items n of an instance and the number of feasible capacities $\Pi(t + \vec{\mathbf{1}})$ and we will further study the connection of these in regards to the convolution problems.

1.2 Related Work

Cygan et al. [9] as well as Künnemann et al. [16] initiated the research and were the first to introduce this class of problems and convolution hardness. In particular, Cygan et al. showed for $d = 1$ that all these problems are equivalent in terms of whether they allow for a subquadratic algorithm. This allows to formulate a conditional lower bound for all these problems under the hypothesis that no subquadratic algorithm for d-MaxConv exists.

▶ **Conjecture 3** (MaxConv-hypothesis). *There exists no $\mathcal{O}(\Pi(L)^{2-\varepsilon})$ time algorithm for any $\varepsilon > 0$ for d-MaxConv with $d = 1$.*

(max, +)-Convolution. The best known algorithm to solve convolution on sequences without any further assumption takes time $n^2/2^{\Omega(\sqrt{\log n})}$. This result was achieved by Williams [21], who gave an algorithm for APSP in conjunction with a reduction by Bremner et al. [3]. However, the existence of a truly subquadratic algorithm remains open.

Research therefore has taken a focus on special cases of convolution where one or both input sequences is required to have certain structural properties such as monotony, concavity or linearity. Chan and Lewenstein [6] gave a subquadratic $\mathcal{O}(n^{1.864})$ algorithm for instances where both sequences are monotone increasing and the values are bound by $\mathcal{O}(n)$. Chi et al. [8] improved this further with a randomized $\widetilde{\mathcal{O}}(n^{1.5})$ algorithm.

Axiotis and Tzamos [1] showed that Convolution with only one concave sequence can be solved in linear time. Gribanov et al. [13] studied multiple cases and gave subquadratic algorithms when one sequence is convex, piece-wise linear or polynomial.

Knapsack. Convolution has proven to be a useful tool to solve other problems as well, in particular the Knapsack problem. In fact one of the reductions of Cygan et al. [9] was an adapted version of Bringmann's algorithm for subset sum [4]. Bringmann's algorithm works by constructing sub-instances, solving these and then combining the solutions via Fast Fourier Transformation (FFT). The algorithm by Cygan et al. to solve Knapsack works the same way, but uses Convolution instead of FFT.

Axiotis and Tzamos follow a similar approach but choose their sub-instances more carefully, by grouping items with the same weight. That way, the solutions make up concave profit sequences which can be combined in linear time [1]. This yields in total an $\mathcal{O}(n + Dt)$ algorithm, where D is the number of different item weights.

Polak et al. [19] gave an $\mathcal{O}(n + \min\{w_{\max}, p_{\max}\}^3)$ algorithm, where $w_{\max}, p_{\max}$ denote the maximum weight and profit respectively. They achieved this by combining the techniques from Axiotis and Tzamos with proximity results from Eisenbrand and Weismantel [11]. Further, Chen et al. [7] improved this to a time of $\widetilde{\mathcal{O}}(n + w_{\max}^{2.4})$. Recently, Ce Jin [14] gave an improved $\widetilde{\mathcal{O}}(n + w_{\max}^2)$ algorithm. Independently to these results for 0/1 Knapsack, Bringmann gave an $\widetilde{\mathcal{O}}(n + w_{\max}^2)$ algorithm for Bounded Knapsack.

Doron-Arad et al. [10] showed that there are constants $\zeta, d_0 > 0$, such that for every integer $d > d_0$ there is no algorithm that solves 0/1 d-Knapsack in time $\mathcal{O}((n + t_{\max})^{\zeta \frac{d}{\log d}})$.

Integer Linear Program. d-dimensional Knapsack problems are a special case of integer linear programs.

Integer Linear Program (ILP)	
Input:	A matrix $A \in \mathbb{Z}^{d \times n}$, a target vector $b \in \mathbb{Z}^d$, a profit vector $c \in \mathbb{Z}^n$ and an upper bound vector $u \in \mathbb{N}^d$.
Problem:	Find $x \in \mathbb{Z}^n$ with $Ax = b$, $0 \leq x \leq u$ and such that $c^\mathsf{T} x$ is maximal.

Lower bounds $\ell \in \mathbb{N}^d$ may be present but can easily be removed by changing the right side to $b - A\ell$ and the upper bound to $u - \ell$. If we limit A to be a matrix with only non-negative entries and $u \coloneqq \vec{\mathbf{1}}$ then solving the above defined ILP is equivalent to 0/1 d-Knapsack. If we omit the $x \leq u$ constraint, the resulting problem is Unbounded d-Knapsack.

A very important part in research of ILPs has been on *proximity*. The proximity of an ILP is the distance between an optimal solution and an optimal solution of its relaxation. The relaxation of an ILP is given by allowing the solution vector x to be fractional.

Eisenbrand and Weismantel [11] have proven that for an optimal solution of the LP-relaxation $x^* \in \mathbb{R}^n$ there is an optimal integral solution $z^* \in \mathbb{Z}^n$ with $||x^* - z^*||_1 \leq d(2d\Delta+1)^d$ with Δ being the largest absolute entry in A. They used this result to give an algorithm that solves ILPs without upper bounds in time $(d\Delta)^{\mathcal{O}(m)} \cdot \|b\|_\infty^2$ and ILPs as defined above in time $\mathcal{O}(n \cdot \mathcal{O}(d)^{(d+1)^2} \cdot \mathcal{O}(\Delta)^{d(d+1)} \cdot \log^2(d\Delta))$. They additionally extended their result to Bounded Knapsack and obtained the running time $\mathcal{O}(n^2 \cdot \Delta^2)$.

There have been further results on proximity such as Lee et al. [17], who gave a proximity bound of $3d^2 \log(2\sqrt{d} \cdot \Delta_m^{1/m}) \cdot \Delta_m$ where Δ_m is the largest absolute value of a minor of order m of matrix A. Celaya et al. [5] also gave proximity bounds for certain modular matrices.

Rohwedder and Węgrzycki [20] conjecture that there is no $2^{\mathcal{O}(m^{2-\varepsilon})}\text{poly}(n)$ time algorithm for ILP with $\Delta = \mathcal{O}(1)$. They also show that there are several problems that are equivalent with respect to this conjecture.

1.3 Our Results

We begin by expanding the results of Cygan et al. [9] into higher dimensions. The natural question is whether similar relations shown in their work also exist in higher dimensions and in fact they do. In the first part of this paper we show that the same equivalence – regarding existing subquadratic time algorithms – holds among the higher dimensional problems.

▶ **Theorem 4.** *For any fixed d and any $\varepsilon > 0$, the following statements are equivalent:*

1. *There exists an $\mathcal{O}(\Pi(L)^{2-\varepsilon})$-time algorithm for d-MaxConv.*
2. *There exists an $\mathcal{O}(\Pi(L)^{2-\varepsilon})$-time algorithm for d-MaxConv UpperBound.*
3. *There exists an $\mathcal{O}(\Pi(L)^{2-\varepsilon})$-time algorithm for d-SuperAdditivity Testing.*
4. *There exists an $\mathcal{O}(n + \Pi(t)^{2-\varepsilon})$-time algorithm for Unbounded d-Knapsack.*
5. *There exists an $\mathcal{O}(n + \Pi(t)^{2-\varepsilon})$-time algorithm for 0/1 d-Knapsack.*

Some of these reduction incur a multiplicative factor of $\mathcal{O}(2^d)$. This is a natural consequence due to the exponentially larger amount of entries that our arrays hold and that we need to process. As an example, where it was sufficient in the one-dimensional case to split a problem in two sub-problems, we may now need to consider 2^d sub-problems. For this reason we require d to be fixed, so we can omit these factors. We will prove this statement through a ring of reductions. We note that one of the reductions uses an algorithm or a generalization of it from Bringmann [4, 9] that is randomized. Part of this algorithm, involving so-called color-coding can be derandomized, but a full derandomization is still an open problem.

We note that under the MaxConv-hypothesis, there does not exist an $\mathcal{O}(\Pi(L)^{2-\varepsilon})$-time algorithm for 1-MaxConv. If there exists any d such that d-MaxConv admits a sub-quadratic algorithm, then it would also solve the problem in sub-quadratic time for any $d' \leq d$ and especially $d' = 1$, hence contradicting the MaxConv-hypothesis, because we can extend any d'-dimensional array to a d-dimensional one by adding dimensions with size one.

We also discuss how to use lower order improvements for 1-dimensional convolution for the d-dimensional convolution via a standard argument using linearization of the vector and adding appropriate padding. As this is a standard trick we omit the proof in the main body but we state it in Section A for completeness.

▶ **Lemma 5.** *Given an algorithm for 1-dimensional convolution with running time $T(n)$ and two d-dimensional arrays A, B with size L, we can calculate $A \oplus B$ in time $T(2^d \prod(L))$.*

Together with the discussion above, Lemma 5 shows that 1-dimensional and d-dimensional convolution are equivalent for fixed d or up to a factor $2^{\mathcal{O}(d)}$.

In the second part of our paper, we complement our conditional lower bound with a parameterized algorithm. To achieve this, we generalize an algorithm by Axiotis and Tzamos [1]. Our algorithm will also have a running time dependent on the number of different item weights, that we denote with D and the largest item weight $\Delta \coloneqq \max_{i \in I} \|w^{(i)}\|_\infty$. This algorithm will also group items by weight vector and solve the respective sub-instances. The resulting partial solutions are then combined via d-MaxConv. However, solving general d-MaxConv needs quadratic time in the number of entries. We can overcome this barrier by reducing our problem to convolution on one-dimensional, concave sequences.

▶ **Theorem 6.** *There is an algorithm for* Bounded d-EqualityKnapsack *with running time* $\mathcal{O}(dn + d \cdot D \cdot \max\{\prod(t+\vec{\mathbf{1}}), t_{\max} \log t_{\max}\}) \subseteq \mathcal{O}(dn + d \cdot \min\{n, (\Delta+1)^d\} \cdot \max\{\prod(t+\vec{\mathbf{1}}), t_{\max} \log t_{\max}\}))$.

In the general case our algorithm will achieve a running time of $\mathcal{O}(dn + D \cdot \Pi(t))$ as the $t_{\max} \log t_{\max}$ part only becomes relevant when we have a slim knapsack, that is very large in one component, but comparatively small in the other. We note that our algorithm achieves the lower bound proposed in Theorem 4 since D is also upper bounded by $\Pi(t)$.

We use the techniques for Theorem 6 to improve the approach of Eisenbrand and Weismantel [11] to obtain the following running time for ILP.

▶ **Theorem 7.** *There is an algorithm for ILP with running time* $\mathcal{O}(dn) + D \cdot \mathcal{O}(d\Delta)^{d(d+1)} + T_{\text{LP}}$.

Our approach reduces the dependency on the number of dimensions ($\mathcal{O}(d)^{d(d+1)}$ instead of $\mathcal{O}(d)^{(d+1)^2}$) and removes the logarithmic factor $\log^2(d\Delta)$. Further, our algorithm is mostly independent on n but relies on the number of different columns of the given ILP.

In addition to their conjecture Rohwedder and Węgrzycki [20] showed that the quadratic dependence on the number of constraints in the exponent can be avoided if the term n^d is allowed in the runtime. We improve their algorithm by removing d from the base.

▶ **Theorem 8.** *There is an algorithm that can solve ILP in time* $n^{d+1} \cdot \mathcal{O}(\Delta)^d \cdot \log(\|u - \ell\|_\infty)$.

The algorithm is a divide and conquer algorithm which repeatedly halves the upper bounds.

1.4 Organization of the Paper

In Section 2 we present an overview of the reductions to proof Theorem 4. However, the concrete proofs are in the full version [12] as the proofs are similar to the ones by Cygan et al. [9]. Next we give the parameterized algorithm as well as the generalization to ILPs in Section 3. The proof of the divide-and-conquer algorithm for Theorem 8 is given in Section 4.

2 Reductions

For the Convolution problems, we will formulate the running time via $T(\Pi(L), d)$, where d is the dimension and L is the size of the result array - meaning the first parameter resembles the number of entries in the array. For the Knapsack problems, we will add the number of items n as parameter and denote the running time of an algorithm via $T(n, \Pi(t+\vec{\mathbf{1}}), d)$, again using $\Pi(t+\vec{\mathbf{1}})$ to denote the number of possible knapsack capacities.

Similar to Cygan et al. [9], we mainly look at functions satisfying $T(\Pi(L), d) = c^{\mathcal{O}(d)} \cdot \Pi(L)^\alpha$ for some constants $c, \alpha > 0$. Therefore, we remark that for all constant $c' > 1$ we can write $T(\Pi(c' \cdot L), d) \in \mathcal{O}(T(\Pi(L), d))$ since d is fixed and we then have $\Pi(c'L)^\alpha = c'^{d \cdot \alpha} \cdot \Pi(L)^\alpha$.

For all the mentioned problems we assume that all inputs, that is also any value in any vector, consist of integers in the range of $[-W, W]$ for some $W \in \mathbb{Z}$. This W is generally omitted as a running time parameter and polylog(W) are omitted or hidden in T functions. We remark that – unavoidably – during the reductions we may have to deal with larger values than W. We generally will multiply a factor of polylog(λ) in cases where the values we have to handle increase up to λW. Note that if λ is a constant, we again omit it.

We follow the same order as Cygan et al. [9], because that makes the reduction increasingly more complex. For the first reduction from Unbounded d-Knapsack to $0/1$ d-Knapsack, we use the same reduction as in the one-dimensional case. The idea is to simply encode taking a multitude of items via binary encoding.

▶ **Theorem 9** (Unbounded d-Knapsack $\to$ $0/1$ d-Knapsack)**.** *A $T(n, \Pi(t), d)$ algorithm for $0/1$ d-Knapsack implies an $\mathcal{O}(T(n \log(t_{\max}), \Pi(t), d))$ algorithm for Unbounded d-Knapsack, where $t \in \mathbb{N}^d$ is the capacity of the knapsack.*

For the next reduction we reduce d-SuperAdditivity Testing to a special case where each array entry is non-negative and values fulfill a monotony property.

▶ **Theorem 10** (d-SuperAdditivity Testing $\to$ Unbounded d-Knapsack)**.** *A $T(n, \Pi(t), d)$ algorithm for Unbounded d-Knapsack implies the existence an algorithm that solves d-SuperAdditivity Testing in time $\mathcal{O}(T(2\Pi(L), \Pi(2L), d) \operatorname{polylog}(dL_{\max}))$ for an array A with size L.*

The next reduction differs from Cygan et al. [9]. We also combine our input arrays together, but in the context of arrays we need to handle a number of different combinations more than in the one-dimensional case. We therefore add a block of negative values in an initial dummy block. This way, any combination that is not relevant to the actual upper bound test, will result positively when tested in the upper bound test.

▶ **Theorem 11** (d-MaxConv UpperBound $\to$ d-SuperAdditivity Testing)**.** *A $T(\Pi(L), d)$ algorithm for d-SuperAdditivity Testing implies an $\mathcal{O}(T(4\Pi(L), d))$ algorithm for d-MaxConv UpperBound.*

For the next reduction from d-MaxConv to d-MaxConv UpperBound, we will prove that we can use an algorithm for d-MaxConv UpperBound to also identify a position that violates the upper bound property.

▶ **Theorem 12** (d-MaxConv $\to$ d-MaxConv UpperBound)**.** *A $T(\Pi(L), d)$ algorithm for d-MaxConv UpperBound implies an $\mathcal{O}(2^d \Pi(L) \cdot T(2^d \sqrt{\Pi(L)}, d) \cdot d \cdot \log(L_{\max}))$ algorithm for d-MaxConv.*

The last reduction is based on Cygan et al. [9] and Bringmann [4]. To make their approach work even in higher dimensional cases, we require a new more refined distribution of items into layers. With this new partition of items many used techniques such as color-coding naturally translate into higher dimensions.

▶ **Theorem 13** ($0/1$ d-Knapsack $\to$ d-MaxConv)**.** *If d-MaxConv can be solved in time $T(\Pi(L), d)$ then $0/1$ d-Knapsack can be solved with a probability of at least $1 - \delta$ in time $\mathcal{O}(T(\Pi(12t), d) \log(d \cdot t_{\max}) \log^3(\log n/\delta) \cdot d \cdot \log n)$ for any $\delta \in (0, 1)$.*

We remark that this also yields a respective algorithm for $0/1$ d-Knapsack with the $\Pi(L)^2$ algorithm for d-MaxConv. Lower order improvements like the results of Bremner [3] or Chan and Lewenstein [6] can be applied by calculating the convolution using Lemma 5.

3 Parameterized Algorithm for Multi-Dimensional Knapsack

In this part, we will consider solving Knapsack such that the capacity is completely utilized. This enables a concise presentation and is not a restriction as discussed after the concrete problem definition.

Bounded d-EqualityKnapsack	
Input:	Set I of n items each defined by a profit $p_i \in \mathbb{R}_{\geq 0}$, weight vector $w^{(i)} \in \mathbb{N}^d$, and upper bound $u_i \in \mathbb{N}$ along with a knapsack of capacity $t \in \mathbb{N}^d$.
Problem:	Find a vector $x \in \mathbb{N}^n$ such that $x \leq u$, $\sum_{i=1}^{n} x_i w^{(i)} = t$, and $\sum_{i=1}^{n} x_i p_i$ is maximal.

We will not only solve this problem for the given capacity t but for all smaller capacities. In detail, we will construct a solution array A with size $t + \vec{\mathbf{1}}$ such that for any $v \leq t$ we have that A_v is the maximum profit of an item set whose total weight is exactly v. Note that we can construct a solution for Bounded d-Knapsack after solving Bounded d-EqualityKnapsack by remembering the highest profit achieved in any position.

The algorithm we will show is based on the algorithm of Axiotis and Tzamos [1]. They solved one-dimensional Knapsack by splitting all items into subsets with equal weight. They proceed then to solve these resulting sub-instances. These sub-instances can be easily solved by gathering the highest profit items that fit into the knapsack. Finally, they combine these resulting partial solutions via convolution. The profits of one such partial solution forms a concave sequence, which allows each convolution to be calculated in linear time.

▶ **Definition 14** (Concave Sequences). *Let $a = (a_i)_{i \leq n}$ be a sequence of numbers of length n. We call the sequence a* concave *if for all $i \leq n-2$ we have $a_{i+1} - a_i \geq a_{i+2} - a_{i+1}$.*

▶ **Lemma 15** ([1, Lemma 9]). *Given an arbitrary sequence of length m and a concave sequence of length n we can compute the $(\max, +)$ convolution of a and b in time $\mathcal{O}(m+n)$.*

We achieve a similar result and calculate higher dimensional convolutions in linear time. In fact, we will reduce our problem to calculating convolution of one-dimensional concave sequences. This way we can calculate the convolution of our sub-solutions in linear time in the number of entries in our array.

▶ **Theorem 6.** *There is an algorithm for Bounded d-EqualityKnapsack with running time $\mathcal{O}(dn + d \cdot D \cdot \max\{\prod(t+\vec{\mathbf{1}}), t_{\max} \log t_{\max}\}) \subseteq \mathcal{O}(dn + d \cdot \min\{n, (\Delta+1)^d\} \cdot \max\{\prod(t+\vec{\mathbf{1}}), t_{\max} \log t_{\max}\}))$.*

We remark, that in most cases $V := \prod(t + \vec{\mathbf{1}}) > t_{\max} \log t_{\max}$ so generally this algorithm will have a running time of $\mathcal{O}(n + DV)$. However, in the special case of a slim array that has size t with one large value t_j and other values being very small $t_i \ll t_j$ for $i \neq j$ our algorithm might have a slightly worse running time of $\mathcal{O}(n + Dt_{\max} \log t_{\max})$. Note that is algorithm also works for d-Knapsack by setting every upper bound to $t_{\max}$.

Proof of Theorem 6. Since we have D different weights, let $w(1), \ldots, w(D)$ be the different weights. We partition the items I into D sets of the same weight, $I_{w(i)} := \{j \in I : w(i) = w^{(j)}\}$ for all $i \in [D]$. We start with an empty solution array R^0 of size $t + \vec{\mathbf{1}}$ such that $R^0_{\vec{\mathbf{0}}} = 0$ and $R^0_v = -\infty$ for $v \neq \vec{\mathbf{0}}$. We will calculate solution arrays R^i for $i \in [D]$ where R^i is the solution array for instance restricted to the items $\bigcup_{j=1}^{i} I_{w(j)}$. Next we describe how we can calculate R^i from R^{i-1}.

Consider an optimal solution for some position v in R^i, i.e., containing only items from $\bigcup_{j=1}^{i} I_{w(j)}$. The solution contains a certain number of items of weight $w(i)$. Let this number be k. We have $v' \coloneqq v - kw(i) \geq \vec{\mathbf{0}}$. Thus, the other items in the solution are from $\bigcup_{j=1}^{i-1} I_{w(j)}$ and have combined profit $R^{i-1}_{v'}$ otherwise our initial solution is not optimal or R^{i-1} is not correct. Also, the k items with weight $w(i)$ in the optimal solution have the highest profit possible, otherwise the solution is not optimal. Therefore, we have

$$R^i_v = \max_{\substack{k \geq 0 \\ v - k \cdot w(i) \geq \vec{\mathbf{0}}}} R^{i-1}_{v - k \cdot w} + f_w(k) \tag{1}$$

where $f_{w(i)}(k)$ is the largest profit for taking k items of weight $w(i)$. Hence, only position pairs v, u such that their difference is an integer multiple of $w(i)$ ($v - u \in w(i)\mathbb{Z}$) are relevant for the calculation of R^i_v or R^i_u. This condition forms an equivalence relation and thus we can calculate the values of R^i per equivalence class. Every equivalence class has the structure $\{v, v + w(i), v + 2 \cdot w(i), v + k_v \cdot w(i)\}$ with $v - w(i) \not\geq \vec{\mathbf{0}}$. Define the sequences $(r_j)_{j \leq k_v}$ with $r_j \coloneqq R^{i-1}_{v + j \cdot w(i)}$ and $(a_j)_{j \leq k_v}$ with $a_j \coloneqq f_{w(i)}(j)$. Thus, we can calculate R^i by $R^i_{v + j \cdot w(i)} \coloneqq (r \oplus a)_j$ for every $j \in [k_v]$. This is sufficient by Equation (1). Note that, we only have to calculate the convolution once as we can use the result for every element of the equivalence class. We always have $k_v \leq t_{\max}$ and $(a_j)_{j \leq k_v}$ is a concave sequence.

The items can be partitioned in the sets $I_{w(i)}$ in time $\mathcal{O}(dn + \Delta)$. In every set the $t_{\max}$ most profitable items can be calculated in total time $\mathcal{O}(n)$ by using [2]. Calculating all values for $f_{w(i)}(j)$ with $i \in [D]$ and $j \in t_{\max}$ takes time $\mathcal{O}(Dt_{\max} \log(t_{\max}))$. Every convolution can be calculated in linear time by Lemma 15. Since the equivalence classes partition the set of positions, we can calculate R^i from R^{i-1} in time $d \cdot \mathcal{O}(\max\{V, t_{\max} \log(t_{\max})\})$ where the d factor is due to multidimensional indices. We may assume $\Delta \leq t_{\max}$ which shows the claimed running time. ◀

3.1 Generalisation to ILPs

As we mentioned before, Knapsack is a special case of ILP. We now want to discuss how to extend our results to ILPs. In the case that all matrix entries are non-negative ILP is equivalent to Bounded d-EqualityKnapsack. We discuss later how to handle negative entries in the constraint matrix. Negative item profits are sensible in Bounded d-EqualityKnapsack as those may be required to reach the capacity. The algorithm presented in Theorem 6 also works with negative profits. The running time of Theorem 6 depends on the capacity of the knapsack. To bound this value for ILPs we use the proximity bound by Eisenbrand and Weismantel [11] and improve their algorithmic approach. We start by giving a short summary of their approach.

For an ILP Instance with constraint matrix $A \in \mathbb{Z}^{d \times n}$, right side $b \in \mathbb{Z}^d$, and upper bounds $u \in \mathbb{N}^n$, let $\Delta \coloneqq \|A\|_{\max}$ be the biggest absolute value of an entry in the constraint matrix. Eisenbrand and Weismantel [11] calculate an optimal solution x^* for the LP-relaxation, which is possible in polynomial time. They proved that there exists an integral optimal solution z^*, such that the distance in ℓ_1-norm is small, more precisely $\|z^* - x^*\|_1 \leq d \cdot (2d\Delta + 1)^d$. They then proceed to take $\lfloor x^* \rfloor$ as an integral solution ($\|z^* - \lfloor x^* \rfloor\|_1 \leq d \cdot (2d\Delta + 1)^d + d =: L$) and construct the following ILP to find an optimal solution.

$$\begin{aligned} \max c^\mathsf{T} x \text{ subject to} \\ Ax &= b - A\lfloor x^* \rfloor \\ \|x\|_1 &\leq L \\ \ell' \leq x &\leq u' \\ x &\in \mathbb{Z}^n \end{aligned}$$

Note that we may have $\ell'_i < 0$ for some $i \in [n]$. Combining an optimal solution to this ILP with $\lfloor x^* \rfloor$ yields an optimal solution to the original ILP. For any $x \in \mathbb{Z}^n$ with $\|x\|_1 \leq L$ we have $\|Ax\|_\infty \leq \Delta L \leq \mathcal{O}(d\Delta)^{d+1}$ due to the bound on the ℓ_1-norm. Eisenbrand and Weismantel construct an acyclic directed graph and find the longest path in that graph which is equivalent to an optimal solution to the ILP above. They bound the size of the graph using the bound on the ℓ_∞-norm above. We avoid such a graph construction for our algorithm which allows us to improve the running time. For two $i, j \in [n]$ with $A_i = A_j$ and $(c_i, i) > (c_j, j)$ we may assume that $x^*_j > 0$ implies $x^*_i = u_i$. Otherwise, we can modify x^* to obtain a solution that adheres to this property with at least the same value. This structure allows us to reformulate the ILP by grouping variables together with the same column in A as follows

$$\begin{aligned} \max \sum\nolimits_{i\in[D]} f_i(x_i) \text{ subject to} \\ A'x &= b - A\lfloor x^* \rfloor \\ \|x\|_1 &\leq L \\ \ell'' \leq x &\leq u'' \\ x &\in \mathbb{Z}^D \end{aligned}$$

where f_i is a concave function for all $i \in [D]$. We calculate an optimal solution to this ILP using the convolution approach from Theorem 6.

▶ **Theorem 7.** *There is an algorithm for ILP with running time $\mathcal{O}(dn)+D\cdot\mathcal{O}(d\Delta)^{d(d+1)}+T_{\mathrm{LP}}$.*

Proof. For $i \in [D]_0$ let U^i be the best possible values using the variables x_j for $j \leq i$. The size of U^i in every dimension is $2\Delta L+1$. To calculate U^i_v from U^{i-1} for some position $v \in \mathbb{Z}^d$ such that $\|v\|_\infty \leq \Delta L$ we need to calculate

$$U^i_v = \max_{\substack{\ell''_i \leq j \leq u''_i \\ \|v - jA'_i\|_\infty \leq \Delta L}} U^{i-1}_{v-jA'_i} + f_i(j).$$

As in the proof of Theorem 6 we can consider the equivalence classes. Let S be the equivalence class of v, it has the structure $\{v', v'+A'_i, v'+2A'_i, \ldots, v'+(|S|-1)A'_i\}$. Define the sequence $(a_j)_{j\leq 2|S|-2}$ with $a_j \coloneqq U^{i-1}_{v'+jA'_i}$ for $j < |S|$ and $a_j \coloneqq \infty$ for $j \geq |S|$. Further we define the sequence $(b_j)_{j\leq 2|S|-2}$ with $b_j \coloneqq f_i(j-|S|+1)$. We calculate the convolution of the sequences $c \coloneqq a \oplus b$ in time $\mathcal{O}(|S|)$ using Lemma 15. Then we get

$$\begin{aligned} U^i_{v'+jA'_i} &= \max_{\substack{\ell''_i \leq k \leq u''_i \\ \|v'+jA'_i-kA'_i\|_\infty \leq \Delta L}} U^{i-1}_{v'+jA'_i-kA'_i} + f_i(k) \\ &= \max_{j-|S|+1\leq k\leq j} U^{i-1}_{v'+(j-k)A'_i} + f_i(k) \\ &= \max_{j\leq k\leq j+|S|-1} U^{i-1}_{v'+(j-k+|S|-1)A'_i} + f_i(k-|S|+1) \\ &= \max_{0\leq k\leq |S|-1} U^{i-1}_{v'+(j-k-j+|S|-1)A'_i} + f_i(k+j-|S|+1) \\ &= \max_{0\leq k\leq |S|-1} U^{i-1}_{v'+(|S|-1-k)A'_i} + f_i(j+k-|S|+1) \\ &= \max_{0\leq k\leq |S|-1} U^{i-1}_{v'+kA'_i} + f_i(j-k) \\ &= \max_{0\leq k\leq |S|-1} U^{i-1}_{v'+kA'_i} + f_i(j-k+|S|-1-|S|+1) \end{aligned}$$

$$
\begin{aligned}
&= \max_{0 \le k \le |S|-1} a_k + b_{j-k+|S|-1} \\
&= \max_{0 \le k \le j+|S|-1} a_k + b_{j-k+|S|-1} && (a_k = -\infty \text{ for } k \ge |S|) \\
&= c_{j+|S|-1}
\end{aligned}
$$

for $j \in [|S|-1]_0$. Next discuss how to obtain the running time of $\mathcal{O}(d\Delta)^{d(d+1)}$ instead of the expected $d\mathcal{O}(d\Delta)^{d(d+1)}$ as in the proof of Theorem 6. To do this we add buffer zones of size Δ at the start and the end of every dimension. Then the size for every dimension is still bounded by $\mathcal{O}(d\Delta)^{d+1}$. Next we save the d-dimensional array as a 1-dimensional array by concatenating the dimensions. We mark the entries in the buffer zones. For a fixed column we can calculate a step size in the 1-dimensional array that corresponds to a step with the column in the d-dimensional array. Using the marked buffer zones we can check if we stepped outside the valid area which allows us to find the equivalence class of a position in linear time of the size of the class. Therefore, we can find all equivalence classes in time $\mathcal{O}(d\Delta)^{d(d+1)}$, and we can calculate the convolution in the same time.

First we need to solve the linear relaxation of the ILP. Denote the necessary time by T_{LP}. Next we partition the columns by weight in time $\mathcal{O}(dn)+\Delta$. We can find the L most profitable picked and unpicked items for every column type in total time $\mathcal{O}(n)$ using [2]. With this we can calculate all relevant values for every f_i in time $D \cdot L \log L$. As explained above every convolution can be calculated in time $\mathcal{O}(d\Delta)^{d(d+1)}$ which yields the claimed running time. An optimal solution vector can be calculated by backtracking in time $D \cdot L \le D \cdot d \cdot \mathcal{O}(d\Delta)^{d+1}$ ◀

4 Divide and Conquer Algorithm

In this section we show Theorem 8. We start by showing the central structural property.

▶ **Lemma 16.** *Let $A \in \mathbb{Z}^{d \times n}$, $b \in \mathbb{Z}^d$, and $x, u \in \mathbb{N}^d$. If $Ax = b$ and $x \le u$, there exists $x' \in \mathbb{N}^d$ such that*

$$\|2Ax' - b\|_\infty \le 2n\Delta, \quad x' \le u' := \left\lfloor \frac{u-1}{2} \right\rfloor \quad \text{and} \quad 0 \le x_i - 2x'_i \le u_i - 2u'_i \le 2 \text{ for } i \in [n].$$

Proof. Let $0 \le x \le u$ with $Ax = b$. We define the new solution x' component wise as follows.

$$x'_i := \begin{cases} 0 & \text{if } x_i = 0, \\ \lfloor \frac{x_i}{2} \rfloor & \text{if } u_i \text{ is odd}, \\ \lfloor \frac{x_i - 1}{2} \rfloor & \text{otherwise} \end{cases}$$

for $i \in [n]$. We start by showing $x' \le u'$. Let $i \in [n]$. If u_i is odd, we have

$$x'_i = \left\lfloor \frac{x_i}{2} \right\rfloor \le \frac{x_i}{2} \le \frac{u_i}{2} = \frac{2u'_i + 1}{2} = u'_i + \frac{1}{2} \implies x'_i \le u'_i$$

and

$$0 \le x_i - 2x'_i = x_i - 2\left\lfloor \frac{x_i}{2} \right\rfloor \le x_i - 2\frac{x_i - 1}{2} = 1 = u_i - 2u'_i \le 2.$$

Otherwise, if u_i is even, we have

$$x'_i = \left\lfloor \frac{x_i - 1}{2} \right\rfloor \le \frac{x_i - 1}{2} \le \frac{u_i - 1}{2} = \frac{2u'_i + 1}{2} = u'_i + \frac{1}{2} \implies x'_i \le u'_i$$

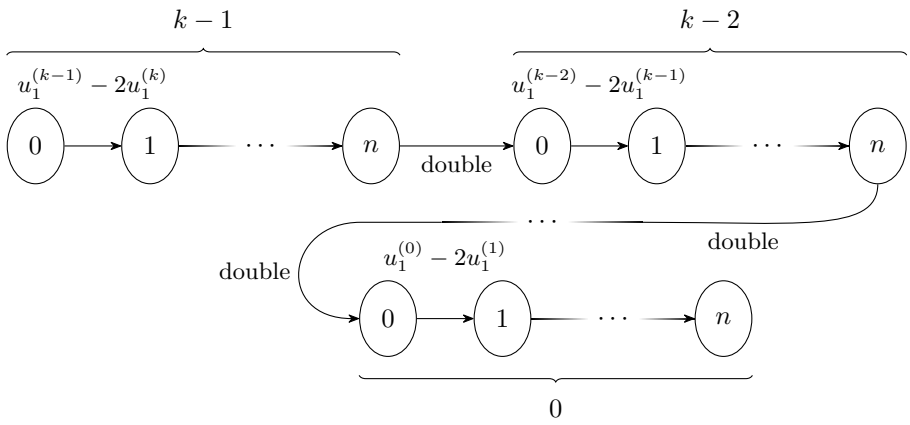

Figure 1 Structure of the graph in the proof of Theorem 8.

and

$$0 \leq x_i - 2x'_i = x_i - 2\left\lfloor \frac{x_i - 1}{2} \right\rfloor \leq x_i - 2\frac{x_i - 2}{2} = 2 = u_i - 2u'_i \leq 2.$$

Next, let $b' \coloneqq Ax'$. We have

$$|2b'_j - b_j| = \left|\sum_{i=1}^{n} A_{j,i}(2x'_i - x_i)\right| \leq \sum_{i=1}^{n} |A_{j,i}(2x'_i - x_i)| = \sum_{i=1}^{n} |A_{j,i}||2x'_i - x_i| \leq 2n\Delta$$

for every $j \in [d]$ and thus $\|2b' - b\|_\infty \leq 2n\Delta$. ◀

This lemma implies that *every* solution $x \in \mathbb{N}^n$ to $Ax = b$ with $x \leq u$ can be decomposed into $x', x'' \in \mathbb{N}^n$ such that $x = 2x' + x''$, x' is a solution to $Ax = b'$ and $x'' \leq u - 2u'$. Note that u' does not depend on the concrete solution but is the same for all.

We can iterate Lemma 16 to obtain a series of solutions and upper bounds until we reach a point where all upper bounds are zero. Define $x^{(j)} \in \mathbb{N}^n$ and $u^{(j)} \in \mathbb{N}^n$ after applying Lemma 16 j times, i.e., $x^{(0)} = x$ and $u^{(0)} = u$. Let $k \in \mathbb{N}$ be smallest value such that $u^{(k)} = 0$.

▶ **Corollary 17.** *We have* $\left\|Ax^{(j)} - \frac{b}{2^j}\right\|_\infty \leq n\Delta \sum_{i=1}^{j} \frac{1}{2^{i-1}} \leq 2n\Delta$ *for* $j \leq k$.

Proof. By induction. $j = 1$ is Lemma 16. Let $j > 1$. Then, we have

$$\begin{aligned}
\left\|Ax^{(j)} - \frac{b}{2^j}\right\|_\infty &= \left\|Ax^{(j)} - \frac{Ax^{(j-1)}}{2} + \frac{Ax^{(j-1)}}{2} - \frac{b}{2^j}\right\|_\infty \\
&\leq \left\|Ax^{(j)} - \frac{Ax^{(j-1)}}{2}\right\|_\infty + \frac{1}{2}\left\|Ax^{(j-1)} - \frac{b}{2^{j-1}}\right\|_\infty \\
&\leq n\Delta + \frac{n\Delta}{2}\sum_{i=1}^{j-1}\frac{1}{2^{i-1}} = n\Delta + n\Delta\sum_{i=2}^{j}\frac{1}{2^{i-1}} = n\Delta\sum_{i=1}^{j}\frac{1}{2^{i-1}}.
\end{aligned}$$

◀

With this we can give the algorithm and proof Theorem 8.

▶ **Theorem 8.** *There is an algorithm that can solve ILP in time* $n^{d+1} \cdot \mathcal{O}(\Delta)^d \cdot \log(\|u - \ell\|_\infty)$.

Proof. We find an optimal solution to the bounded ILP by finding the longest path in a graph that is constructed based on Lemma 16. We define the graph as follows.

$$V \coloneqq \left\{(b', j, i) : j \in [k]_0, \left\|b' - \frac{b}{2^j}\right\|_\infty \leq 2(2n - i)\Delta, i \in [n]_0\right\}$$

The intuition here is that we have a layer for every application of Lemma 16. We can jump from one layer to the next by doubling the corresponding right side (the first component) and length of the path to the current vertex. Then, to reconstruct the solution prior to an application of Lemma 16 we may need to increase variables by up to two. For this we have the third component. While going to a vertex with $i \in [n]$ in the last component we may use the ith column up to the number of times it was removed to get the new upper bound. This is also an upper bound for the number of times it was removed from a solution as stated in Lemma 16. More precisely, we associate the ILP

$$\begin{aligned} \max\, & c^\mathsf{T} x \\ & Ax = b' \\ & x_\ell \leq u_\ell^{(j)} && \text{(for } \ell \in [i]\text{)} \\ & x_\ell \leq 2u_\ell^{(j+1)} && \text{(for } \ell > i\text{)} \\ & x \in \mathbb{N}^n \end{aligned}$$

with a vertex $(b', j, i) \in V$ and refer to it by $\mathrm{ILP}_{(b',j,i)}$. A path from $(0, k-1, 0)$ to (b', j, i) describes a solution to $\mathrm{ILP}_{(b',j,i)}$ where the value is equal to the length of the path.

Next we give the precise definitions for the edges in the graph. For a vertex with $j < k$ and $i < n$ we define the following edges

$$(b', j, i) \xrightarrow{xc_{i+1}} (b' + xA_{i+1}, j, i+1)$$

with $x \in [u_{i+1}^{(j)} - 2u_{i+1}^{(j+1)}]_0 \subseteq [2]_0$ and for vertices with $j > 0$ and $i = n$ we define the edge

$$(b', j, n) \xrightarrow{\text{double the length}} (2b', j-1, 0).$$

Because of the changes in the second and third component the graph is acyclic, and thus the longest path can be calculated in linear time by utilizing a topological order of the graph. Note that the second type of edge does not have a classic length, but this doubling of the current length works with the aforementioned algorithm. The out degree of every vertex is bounded by 3 and thus the size of graph is linear in the number vertices which is $n^{d+1} \cdot \mathcal{O}(\Delta)^d \cdot \log(\|u\|_\infty)$. Now we find the longest path from $(0, k-1, 0)$ to $(b, 0, n)$. The only solution to $\mathrm{ILP}_{(0,k-1,0)}$ is $\vec{\mathbf{0}}$. The structure of the graph is depicted in Figure 1.

Let $x \in \mathbb{N}^n$ be a solution to the ILP. Further, let $x^{(j)} \in \mathbb{N}^n$ be the solutions obtained by iterating Lemma 16. Define

$$x^{(j,i)} := \begin{cases} 2x^{(j+1)} & \text{if } i = 0, \\ x^{(j,i-1)} + \left(x_i^{(j)} - 2x_i^{(j+1)}\right) e_i & \text{otherwise} \end{cases}$$

for $j \in [k-1]_0$ and $i \in [n]_0$ where e_i is the ith unit vector. We have $\left\| Ax^{(j,i)} - \frac{b}{2^j} \right\|_\infty \leq 2(2n-i)\Delta$ by Lemma 16 and Corollary 17. Thus,

$$\begin{aligned} & (0 = Ax^{(k-1,0)}, k-1, 0), (Ax^{(k-1,1)}, k-1, 1), \ldots, \\ & \quad (Ax^{(k-1,n)}, k-1, n), (Ax^{(k-2,0)}, k-2, 0), \ldots, (b = Ax^{(0,n)}, 0, n) \end{aligned}$$

is a path in the graph with length $c^\mathsf{T} x$. Therefore, ILP has a solution if and only if there is a path in the graph from $(0, k-1, 0)$ to $(b, 0, n)$. Also the value of the optimal solution to the ILP is equal to the length of the longest path in the graph. ◀

5 Conclusion

First, we have proven that the relationship between Knapsack and Convolution in the one-dimensional case also prevails in higher dimensional variants. A natural follow-up question is how many more problems can be shown to be equivalent to d-dimensional convolution.

We developed a parameterized algorithm for multidimensional knapsack that generalized techniques for one-dimensional knapsack. Further, we used these techniques to obtain a faster algorithm for Integer Linear Programming with upper bounds.

Finally, we gave an improved algorithm for Integer Linear Programming with upper bounds that avoids the quadratic dependency on the dimension by increasing the dependency on the number of variables.

References

1 Kyriakos Axiotis and Christos Tzamos. Capacitated Dynamic Programming: Faster Knapsack and Graph Algorithms. In *46th International Colloquium on Automata, Languages, and Programming (ICALP)*, volume 132, pages 19:1–19:13, Dagstuhl, Germany, 2019. doi:10.4230/LIPICS.ICALP.2019.19.

2 Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, and Robert Endre Tarjan. Time bounds for selection. *J. Comput. Syst. Sci.*, 7(4):448–461, 1973. doi:10.1016/S0022-0000(73)80033-9.

3 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, and Perouz Taslakian. Necklaces, convolutions, and x + y. In *Algorithms – ESA 2006*, pages 160–171, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg. doi:10.1007/11841036_17.

4 Karl Bringmann. *A Near-Linear Pseudopolynomial Time Algorithm for Subset Sum*, pages 1073–1084. Society for Industrial and Applied Mathematics, 2017. doi:10.1137/1.9781611974782.69.

5 Marcel Celaya, Stefan Kuhlmann, Joseph Paat, and Robert Weismantel. Improving the cook et al. proximity bound given integral valued constraints. In *Integer Programming and Combinatorial Optimization - 23rd International Conference, IPCO 2022, Eindhoven, The Netherlands, June 27-29, 2022, Proceedings*, volume 13265 of *Lecture Notes in Computer Science*, pages 84–97. Springer, 2022. doi:10.1007/978-3-031-06901-7_7.

6 Timothy M. Chan and Moshe Lewenstein. Clustered integer 3sum via additive combinatorics. In *Proceedings of the Forty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '15, pages 31–40, New York, NY, USA, 2015. Association for Computing Machinery. doi:10.1145/2746539.2746568.

7 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, Alexandria, VA, USA, January 7-10, 2024*, pages 4828–4848. SIAM, 2024. doi:10.1137/1.9781611977912.171.

8 Shucheng Chi, Ran Duan, Tianle Xie, and Tianyi Zhang. Faster min-plus product for monotone instances. In *Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing*, STOC 2022, pages 1529–1542, New York, NY, USA, 2022. Association for Computing Machinery. doi:10.1145/3519935.3520057.

9 Marek Cygan, Marcin Mucha, Karol Węgrzycki, and Michał Włodarczyk. On problems equivalent to $(min, +)$-convolution. *ACM Trans. Algorithms*, 15(1), January 2019. doi:10.1145/3293465.

10 Ilan Doron-Arad, Ariel Kulik, and Pasin Manurangsi. Fine grained lower bounds for multidimensional knapsack, 2024. doi:10.48550/arXiv.2407.10146.

11 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for integer programming using the steinitz lemma. *ACM Trans. Algorithms*, 16(1), November 2019. doi:10.1145/3340322.

12 Kilian Grage, Klaus Jansen, and Björn Schumacher. Convolution and knapsack in higher dimensions, 2024. arXiv:2403.16117.

13 Dmitry V. Gribanov, Ivan A. Shumilov, and Dmitriy S. Malyshev. Structured $(\min, +)$-convolution and its applications for the shortest/closest vector and nonlinear knapsack problems. *Optim. Lett.*, 18(1):73–103, 2024. doi:10.1007/S11590-023-02017-5.

14 Ce Jin. 0-1 knapsack in nearly quadratic time. In *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.

15 L. Kronecker. Grundzüge einer arithmetischen theorie der algebraische grössen. *Journal für die reine und angewandte Mathematik*, 1882(92):1–122, 1882. doi:10.1515/crll.1882.92.1.

16 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, July 10-14, 2017, Warsaw, Poland*, volume 80 of *LIPIcs*, pages 21:1–21:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICS.ICALP.2017.21.

17 Jon Lee, Joseph Paat, Ingo Stallknecht, and Luze Xu. Improving proximity bounds using sparsity. In *Combinatorial Optimization - 6th International Symposium, ISCO 2020, Montreal, QC, Canada, May 4-6, 2020, Revised Selected Papers*, volume 12176 of *Lecture Notes in Computer Science*, pages 115–127. Springer, 2020. doi:10.1007/978-3-030-53262-8_10.

18 Victor Y. Pan. Simple multivariate polynomial multiplication. *J. Symb. Comput.*, 18(3):183–186, 1994. doi:10.1006/JSCO.1994.1042.

19 Adam Polak, Lars Rohwedder, and Karol Węgrzycki. Knapsack and Subset Sum with Small Items. In *48th International Colloquium on Automata, Languages, and Programming (ICALP 2021)*, volume 198 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 106:1–106:19, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ICALP.2021.106.

20 Lars Rohwedder and Karol Węgrzycki. Fine-Grained Equivalence for Problems Related to Integer Linear Programming. In *16th Innovations in Theoretical Computer Science Conference (ITCS 2025)*, volume 325 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 83:1–83:18, Dagstuhl, Germany, 2025. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.ITCS.2025.83.

21 Ryan Williams. Faster all-pairs shortest paths via circuit complexity. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, pages 664–673, New York, NY, USA, 2014. Association for Computing Machinery. doi:10.1145/2591796.2591811.

A Calculate d-dimensional Convolution with 1-dimensional Convolution

We can interpret $(\max, +)$-convolution as multiplication of polynomials over the $(\max, +)$-semiring, also called the Arctic semiring. d-dimensional convolution is multiplication of polynomials with d variables $x_1, \ldots, x_d$. More precisely, let A, B be two d-dimensional arrays of size L. Then, the corresponding polynomial to A is given by

$$p(A) := \sum_{0 \leq v < L} A[v] \prod_{i=1}^{d} x_i^{v_i}.$$

The Kronecker map [15] as described in [18] maps x_{k+1} to $x^{D(1)\cdots D(k)}$ for every $k \in [d-1]_0$ where $D(i) = 2L_i - 1$ for every $i \in [d]$. This map is generalized to polynomials accordingly and allows us to obtain an one-dimensional array this way by adding dummy values. An example for this construction is depicted in Figure 2. Let $\tilde{p}(A)$ and $\tilde{p}(B)$ be the results of this transformation.

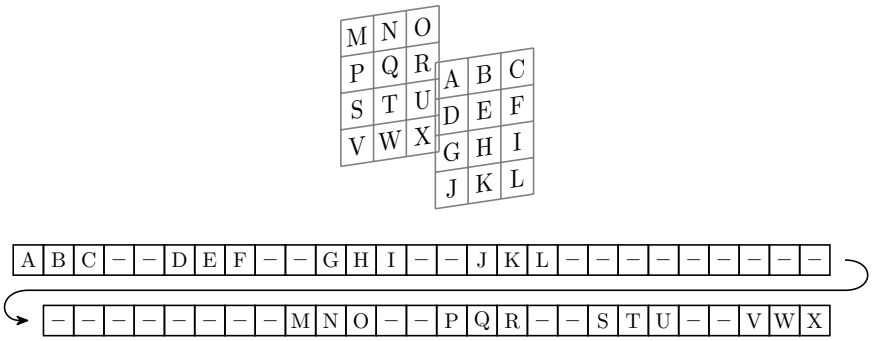

Figure 2 Example for the transformation in 3 dimensions with $L = (3, 4, 2)^\mathsf{T}$.

Now, we can calculate $\tilde{p}(A) \cdot \tilde{p}(B)$ with an one-dimensional $(\max, +)$-convolution algorithm and transform the result back according to the transformations above to obtain an d-dimensional array. The result is correct since the polynomial multiplication is correct due to the added padding in the definition of $D(i)$ [18].

To show Lemma 5 is remains to show that $\deg(\tilde{p}(A)) + 1 \leq 2^d \prod(L)$ since $\deg(\tilde{p}(A))$ is the length of the resulting arrays for the one-dimensional convolution. We have

$$
\begin{aligned}
2\deg(\tilde{p}(A)) &= 2\deg\Big(A[L - \vec{\mathbf{1}}] \prod_{i=1}^{d} (x^{D(1)\cdots D(i-1)})^{L_i - 1}\Big) \quad (L - \vec{\mathbf{1}} \text{ is the maximum index.}) \\
&= 2\sum_{\ell=1}^{d} (L_\ell - 1) \prod_{m=1}^{\ell-1} D(m) \\
&\leq \sum_{\ell=1}^{d} \Big(\prod_{m=1}^{\ell-1} 2L_m\Big) 2L_\ell \\
&= 2L_1 + 2L_1 \sum_{\ell=2}^{d} \Big(\prod_{m=1}^{\ell-1} 2L_m\Big) 2L_\ell \\
&= \sum_{\ell=1}^{d} 2^\ell \prod_{m=1}^{\ell} L_m \qquad \text{(With induction)} \\
&= \Big(\sum_{\ell=1}^{d} 2^\ell\Big) \prod(L) \\
&\leq 2^{d+1} \prod(L) \\
\implies \deg(\tilde{p}(A)) &\leq 2^d \prod(L)
\end{aligned}
$$

In conclusion we have show the following lemma.

▶ **Lemma 5.** *Given an algorithm for 1-dimensional convolution with running time $T(n)$ and two d-dimensional arrays A, B with size L, we can calculate $A \oplus B$ in time $T(2^d \prod(L))$.*