On Integer Programming and Convolution

Klaus Jansen
Department of Computer Science, Kiel University, Kiel, Germany
kj@informatik.uni-kiel.de

Lars Rohwedder
Department of Computer Science, Kiel University, Kiel, Germany
lro@informatik.uni-kiel.de

Abstract

Integer programs with a constant number of constraints are solvable in pseudo-polynomial time. We give a new algorithm with a better pseudo-polynomial running time than previous results. Moreover, we establish a strong connection to the problem (min, +)-convolution. (min, +)-convolution has a trivial quadratic time algorithm and it has been conjectured that this cannot be improved significantly. We show that further improvements to our pseudo-polynomial algorithm for any fixed number of constraints are equivalent to improvements for (min, +)-convolution. This is a strong evidence that our algorithm's running time is the best possible. We also present a faster specialized algorithm for testing feasibility of an integer program with few constraints and for this we also give a tight lower bound, which is based on the SETH.

2012 ACM Subject Classification Theory of computation → Integer programming, Theory of computation → Dynamic programming

Keywords and phrases Integer programming, convolution, dynamic programming, SETH

Digital Object Identifier 10.4230/LIPIcs.ITCS.2019.43

Funding Research was supported by German Research Foundation (DFG) projects JA 612/20-1 and JA 612/16-1.

1 Introduction

Vectors $v^{(1)}, \dots, v^{(n)} \in \mathbb{R}^m$ that sum up to 0 can be seen as a circle in $\mathbb{R}^m$ that walks from 0 to $v^{(1)}$ to $v^{(1)} + v^{(2)}$, etc. until it reaches $v^{(1)} + \dots + v^{(n)} = 0$ again. The Steinitz Lemma [17] says that if each of the vectors is small with respect to some norm, we can reorder them in a way that each point in the circle is not far away from 0 w.r.t. the same norm.

Recently Eisenbrand and Weismantel found a beautiful application of this lemma in the area of integer programming [8]. They looked at ILPs of the form $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$, where $A \in \mathbb{Z}^{m \times n}, b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^n$ and obtained a pseudo-polynomial algorithm in Δ, the biggest absolute value of an entry in A, when m is treated as a constant. The running time they achieve is $n \cdot O(m\Delta)^{2m} \cdot \|b\|_1^2$ for finding the optimal solution and $n \cdot O(m\Delta)^m \cdot \|b\|_1$ for finding only a feasible solution. This improves on a classic algorithm by Papadimitriou, which has a running time of $O(n^{2m+2} \cdot (m\Delta + m\|b\|_\infty)^{(m+1)(2m+1)})$ [15]. The basic idea in [8] is that a solution x^* for the ILP above can be viewed as a walk in $\mathbb{Z}^m$ starting at 0 and ending at b. Every step is a column of the matrix A: For every $i \in \{1, \dots, n\}$ we step x^*_i times in the direction of A_i (see left picture in Figure 1). By applying the Steinitz Lemma they show that there is an ordering of these steps such that the walk never strays off far from the direct line between 0 and b (see right picture in Figure 1). They construct a directed graph with one vertex for every integer point near the line between 0 and b and create an

10th Innovations in Theoretical Computer Science (ITCS 2019).
Editor: Avrim Blum; Article No. 43; pp. 43:1–43:17
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

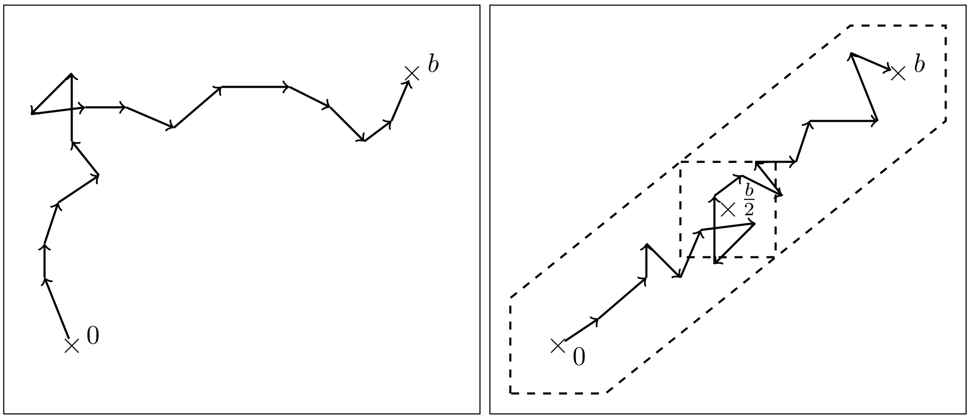

Figure 1 Steinitz Lemma in Integer Programming.

edge from u to v, if $v - u$ is a column in A. The weight of the edge is the same as the c-value of the column. An optimal solution to the ILP can now be obtained by finding a longest path from 0 to b. This can be done in the mentioned time, if one is careful with circles.

In this paper, we present an alternative way to apply the Steinitz Lemma to the same problem. Our approach does not reduce to a longest path problem, but rather solves the ILP in a divide and conquer fashion. Using the Steinitz Lemma and the intuition of a walk from 0 to b, we notice that this walk has to visit a vector b' near $b/2$ at some point. We guess this vector and solve the problem with $Ax = b'$ and $Ax = b - b'$ independently. Both results can be merged to a solution for $Ax = b$. In the sub-problems the norm of b and the norm of the solution are roughly divided in half. We use this idea in a dynamic program and speed up the process of merging solutions using algorithms for convolution problems. This approach gives us better running times for both the problem of finding optimal solutions and for testing feasibility only. We complete our study by giving (almost) tight conditional lower bounds on the running time in which such ILPs can be solved.

1.1 Detailed description of results

In the running times we give, we frequently use logarithmic factors like $\log(k)$ for some parameter k. To handle the values $k \in \{0, 1\}$ formally correct, we would need to write $\log(k + 1) + 1$ instead of $\log(k)$ everywhere. This is ignored for simplicity of notation. We are assuming the word RAM model with word size $O(m \log(m\Delta) + \log(\|b\|_\infty) + \log(\|c\|_\infty))$ (see Preliminaries for details).

Optimal solutions for ILPs

We show that a solution to $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ can be found in time $O(m\Delta)^{2m} \cdot \log(\|b\|_\infty) + O(nm)$. If given a vertex solution to the fractional relaxation, we can even get to $O(m\Delta)^{2m} + O(nm)$. The running time can be improved if there exists a truly sub-quadratic algorithm for (min, +)-convolution (see Section 4.1 for details on the problem). However, it has been conjectured that no such algorithm exists and this conjecture is the base of several lower bounds in fine-grained complexity [7, 14, 3]. We show that for every m the running time above is essentially the best possible unless the (min, +)-convolution conjecture is false. More formally, for every m there exists no algorithm that solves ILP in time $f(m) \cdot (n^{2-\delta} + (\Delta + \|b\|_\infty)^{2m-\delta})$ for some $\delta > 0$ and an arbitrary computable function f, unless there exists a truly sub-quadratic algorithm for (min, +)-convolution. Indeed, this means there is an equivalence between improving algorithms for (min, +)-convolution and for ILPs of fixed number of constraints. It is notable that this also rules out improvements when both Δ and $\|b\|_\infty$ are small. Our lower bound does leave open some trade-off between n and $O(m\Delta)^m$ like for example $n \cdot O(m\Delta)^m \cdot \log(\|b\|_\infty)$, which would be an interesting improvement for sparse instances, i.e., when $n \ll (2\Delta+1)^m$. A running time of $n^{f(m)} \cdot (m\Delta + m\|b\|_\infty)^{m-\delta}$, however, is not possible (see feasibility below).

Feasibility of ILPs

Testing only the feasibility of an ILP is easier than finding an optimal solution. It can be done in time $O(m\Delta)^m \cdot \log(\Delta) \cdot \log(\Delta + \|b\|_\infty) + O(nm)$ by solving a Boolean convolution problem that has a more efficient algorithm than the (min, +)-convolution problem that arises in the optimization version. Under the Strong Exponential Time Hypothesis (SETH) this running time is tight except for logarithmic factors. If this conjecture holds, there is no $n^{f(m)} \cdot (m\Delta + m\|b\|_\infty)^{m-\delta}$ time algorithm for any $\delta > 0$ and any computable function f.

1.2 Other related work

The case where the number of variables n is fixed and not m as in this paper behaves somewhat differently. There is a $2^{O(n\log(n))} \cdot |I|^{O(1)}$ time algorithm ($|I|$ being the encoding length of the input), whereas an algorithm of the kind $f(m) \cdot |I|^{O(1)}$ (or even $|I|^{f(m)}$) is impossible for any computable function f, unless P = NP. This can be seen with a trivial reduction from Unbounded Knapsack (where $m = 1$). The $2^{O(n\log(n))} \cdot |I|^{O(1)}$ time algorithm is due to Kannan [12] improving over a $2^{O(n^2)} \cdot |I|^{O(1)}$ time algorithm by Lenstra [11]. It is a long open question whether $2^{O(n)} \cdot |I|^{O(1)}$ is possible instead [8].

Another intriguing question is whether a running time like $(m\Delta + m\|b\|_\infty)^{O(m)} \cdot n^{O(1)}$ is still possible when upper bounds on variables are added to the ILP. In [8] an algorithm for this extension is given, but the exponent of Δ is $O(m^2)$.

As for other lower bounds on pseudo-polynomial algorithms for integer programming, the only result we are aware of is a bound of $n^{o(m/\log(m))} \cdot \|b\|_\infty^{o(m)}$ due to Fomin et al. [9], which is based on the ETH (a weaker conjecture than the SETH). Their reduction implies that there is no algorithm with running time $n^{o(m/\log(m))} \cdot (\Delta + \|b\|_\infty)^{o(m)}$, since in their construction the matrix A is non-negative and therefore columns with entries larger than $\|b\|_\infty$ can be discarded; thus leading to $\Delta \leq \|b\|_\infty$. As opposed to our bounds, theirs does not give a precise value for the constant in the exponent.

2 Preliminaries

In this paper we are assuming a word size of $O(m\log(m\Delta) + \log(\|b\|_\infty) + \log(\|c\|_\infty))$ in the word RAM model, that is to say, arithmetic operations on numbers of this encoding size take constant time. When considering m to be a constant, this makes perfect sense. Also, since we are going to use algorithms with space roughly $O(m\Delta)^m$, it is only natural to assume that a single pointer fits into a word.

In the remainder of the paper we will assume that A has no duplicate columns. Note that we can completely ignore a column i, if there is another identical column i' with $c_{i'} \geq c_i$. This implies that in time $O(nm) + O(\Delta)^m$ we can reduce to an instance without duplicate columns and, in particular, with $n \leq (2\Delta+1)^m$. The running time can be achieved as follows.

We create a new matrix for the ILP with all $(2\Delta + 1)^m$ possible columns (in lexicographic order) and objective value $c_i = -\infty$ for all columns i. Now we iterate over all n old columns and compute in time $O(m)$ the index of the new column corresponding to the same entries. We then replace its objective value with the current one if this is bigger. In the upcoming running times we will omit the additive term $O(nm)$ and assume the duplicates are already eliminated ($(O(\Delta)^m$ is always dominated by actual algorithms running time).

▶ **Theorem 1** (Steinitz Lemma). *Let $\|\cdot\|$ be a norm in $\mathbb{R}^m$ and let $v^{(1)}, \dots, v^{(t)} \in \mathbb{R}^m$ such that $\|v^{(i)}\| \leq 1$ for all i and $v^{(1)} + \cdots + v^{(t)} = 0$. Then there exists a permutation $\pi \in S_t$ such that for all $j \in \{1, \dots, t\}$*

$$\|\sum_{i=1}^{j} v^{(\pi(i))}\| \leq m.$$

The proof for bound m is due to Sevast'janov [16] (see also [8] for a good overview). Eisenbrand and Weismantel observed that the Steinitz Lemma implies the following.

▶ **Corollary 2** ([8]). *Let $v^{(1)}, \dots, v^{(t)}$ denote columns of A with $\sum_{i=1}^{t} v^{(i)} = b$. Then there exists a permutation $\pi \in S_t$ such that for all $j \in \{1, \dots, t\}$*

$$\|\sum_{i=1}^{j} v^{(\pi(i))} - \frac{j}{t} \cdot b\|_\infty \leq 2m\Delta.$$

This can be obtained by inserting $(v^{(i)} - b/t)/(2\Delta)$, $i \in \{1, \dots, t\}$, in the Steinitz Lemma. Note that $\|v^{(i)} - b/t\|_\infty \leq 2\Delta$.

▶ **Lemma 3.** *Let $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ be bounded and feasible. Then there exists an optimal solution x^* with $\|x^*\|_1 \leq O(m\Delta)^m(\|b\|_\infty + 1)$.*

A similar bound is proved for example in [15]. However, we can also give a proof via the Steinitz Lemma.

Proof. Let x^* be an optimal solution of minimal 1-norm. Let $v^{(1)}, \dots, v^{(t)}$ denote the multiset of columns of A that represent x^*. Assume w.l.o.g. these vectors are ordered as in the previous corollary. There cannot be a circle of positive value in $v^{(1)}, \dots, v^{(t)}$ or else the ILP would be unbounded. By circle we mean a non-empty subset that sums up to 0 and we consider the value of the columns with regard to c. In fact, there cannot be a circle of nonpositive value either, since the 1-norm of the solution is minimal. Hence, each vector in $\mathbb{Z}^m$ is visited at most once by the walk $v^{(1)}, v^{(1)} + v^{(2)}, \dots, v^{(1)} + \cdots + v^{(t)} = b$. The number of integer points a with

$$\|a - \gamma b\|_\infty \leq 2m\Delta \tag{1}$$

for some $\gamma \in [0, 1]$ is at most $O(m\Delta)^m \cdot (\|b\|_\infty + 1)$ and this upper bounds the 1-norm of x^*: Assume w.l.o.g. $\|b\|_\infty > 0$ as the case $b = 0$ is trivial. Take $\|b\|_\infty + 1$ many points evenly distributed along the line from 0 to b, i.e., $b \cdot 0/\|b\|_\infty$, $b \cdot 1/\|b\|_\infty$,..., $b \cdot \|b\|_\infty/\|b\|_\infty$. Then the distance between two consecutive points is small:

$$\left\| b \cdot \frac{j+1}{\|b\|_\infty} - b \cdot \frac{j}{\|b\|_\infty} \right\|_\infty = \left\| \frac{b}{\|b\|_\infty} \right\|_\infty = 1.$$

In particular, for every vector of the form γb, $\gamma \in [0,1]$, there is a point $b \cdot j/\|b\|_\infty$ that is not further away than $1/2$. Thus, for every a that satisfies (1), we have a point $b \cdot j/\|b\|_\infty$ with

$$\left\| a - b \cdot \frac{j}{\|b\|_\infty} \right\|_\infty \leq \|a - \gamma b\|_\infty + \left\| \gamma b - b \cdot \frac{j}{\|b\|_\infty} \right\|_\infty \leq 2m\Delta + 1/2.$$

To upper bound the number of vectors of type (1), we count the number of vectors within distance at most $2m\Delta + 1/2$ to each of the $\|b\|_\infty + 1$ points. This number is at most $(\|b\|_\infty + 1) \cdot (4m\Delta + 2)^m$. This concludes the proof. ◀

▶ **Corollary 4.** *By adding a zero column we can assume w.l.o.g., if the ILP is feasible and bounded, then there exists an optimal solution x^* with $\|x^*\|_1 = U$ where U is the upper bound for $\|x^*\|_1$. By scaling the bound of Lemma 3 to the next power of 2, we can assume that $\|x^*\|_1 = 2^K$ where $K \in \mathbb{N}$ and $K \leq O(m \log(m\Delta) + \log(\|b\|_\infty))$.*

3 Dynamic Program

In this section we will show how to compute the best solution x^* to an ILP with the additional constraint $\|x^*\|_1 = 2^K$. If the ILP is bounded, then with $K = O(m \log(m\Delta) + \log(\|b\|_\infty))$ and an extra zero column this is the optimum to the ILP (Corollary 4). In Section 3.2 we discuss how to cope with unbounded ILPs. For every $i = K, K-1, \ldots, 0$ and every b' with

$$\|b' - 2^{-i} \cdot b\|_\infty \leq 4m\Delta$$

we solve $\max\{c^T x : Ax = b', \|x\|_1 = 2^{K-i}, x \in \mathbb{Z}^n_{\geq 0}\}$. We start by computing these for $i = K$ and then iteratively derive solutions for smaller values of i using only the bigger ones. Ultimately, we will compute a solution for $i = 0$ and $b' = b$.

If $i = K$, then every solution must consist of exactly one column ($\|x\|_1 = 1$). We can compute this solution by finding the column that equals b' should there exist one and set $-\infty$ otherwise.

Fix some $i < K$ and b' and let $v^{(1)}, \ldots, v^{(t)}$ be columns of A that correspond to an optimal solution to $\max\{c^T x : Ax = b', \|x\|_1 = 2^{K-i}, x \in \mathbb{Z}^n_{\geq 0}\}$. In particular, $v^{(1)} + \cdots + v^{(t)} = b'$ and $t = 2^{K-i}$. Assume w.l.o.g. that the $v^{(i)}$ are ordered such that for all $j \in \{0, \ldots, t\}$

$$\|\sum_{i=1}^{j} v^{(i)} - \frac{j}{t} \cdot b'\|_\infty \leq 2m\Delta.$$

Note that $v^{(1)}, \ldots, v^{(t/2)}$ is an optimal solution to $\max\{c^T x : Ax = b'', \|x\|_1 = 2^{K-(i+1)}, x \in \mathbb{Z}^n_{\geq 0}\}$ where $b'' = v^{(1)} + \cdots + v^{(t/2)}$. Likewise, $v^{(t/2+1)}, \ldots, v^{(t)}$ is an optimal solution to $\max\{c^T x : Ax = b' - b'', \|x\|_1 = 2^{K-(i+1)}, x \in \mathbb{Z}^n_{\geq 0}\}$. We claim that $\|b'' - 2^{-(i+1)} \cdot b\|_\infty \leq 4m\Delta$ and $\|(b' - b'') - 2^{-(i+1)} \cdot b\|_\infty \leq 4m\Delta$. This implies that we can look up solutions for b'' and $b' - b''$ in the dynamic table and their union is a solution for b'. Clearly it is also optimal. We do not know b'', but we can guess it: There are only $(8m\Delta + 1)^m$ candidates. To compute an entry, we therefore enumerate all possible b'' and take the two partial solutions (for b'' and $b' - b''$), where the sum of both values is maximized.

Proof of claim

We have that,

$$\|\sum_{i=1}^{t/2} v^{(i)} - 2^{-(i+1)} \cdot b\|_\infty = \|\sum_{i=1}^{t/2} v^{(i)} - \frac{1}{2} \cdot b' + \frac{1}{2} \cdot b' - 2^{-(i+1)} \cdot b\|_\infty$$

$$\leq \|\sum_{i=1}^{t/2} v^{(i)} - \frac{1}{2} \cdot b'\|_\infty + \|\frac{1}{2} \cdot b' - 2^{-(i+1)} \cdot b\|_\infty \leq 2m\Delta + \frac{1}{2}\|b' - 2^{-i} \cdot b\|_\infty \leq 4 \cdot m\Delta.$$

In a similar way, we can show that

$$\|\sum_{i=t/2+1}^{t} v^{(i)} - 2^{-(i+1)} \cdot b\|_\infty = \|\sum_{i=t/2+1}^{t} v^{(i)} - \sum_{i=1}^{t} v^{(i)} + b' - 2^{-(i+1)} \cdot b\|_\infty$$

$$= \|\frac{1}{2} \cdot b' - \sum_{i=1}^{t/2} v^{(i)} + \frac{1}{2} \cdot b' - 2^{-(i+1)} \cdot b\|_\infty$$

$$\leq \|\sum_{i=1}^{t/2} v^{(i)} - \frac{1}{2} \cdot b'\|_\infty + \|\frac{1}{2} \cdot b' - 2^{-(i+1)} \cdot b\|_\infty \leq 4 \cdot m\Delta.$$

3.1 Naive running time

The dynamic table has $(K+1) \cdot O(m\Delta)^m$ entries. To compute an entry, $O(n \cdot m) \leq O(m\Delta)^m$ operations are necessary during initialization and $O(m\Delta)^m$ in the iterative calculations. This gives a total running time of

$$O(m\Delta)^{2m} \cdot (K+1) = O(m\Delta)^{2m} \cdot (m \log(m\Delta) + \log(\|b\|_\infty)) = O(m\Delta)^{2m} \cdot (\log(\Delta) + \log(\|b\|_\infty)).$$

Note that $O(m\Delta)^{2m} = O(m\Delta)^{2m} \cdot 2^m$ hides factors polynomial in m.

3.2 Unbounded solutions

In the previous dynamic program there is no mechanism for detecting when the ILP is unbounded. We follow the approach from [8] to handle unbounded ILPs. The ILP $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ is unbounded, if and only if $\{x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ has a solution and $\max\{c^T x : Ax = 0, x \in \mathbb{Z}^n_{\geq 0}\}$ has any solution with positive objective value. After running the dynamic program - thereby verifying that there exists any solution - we have to check if the latter condition holds. We can simply run the algorithm again on $\max\{c^T x : Ax = 0, x \in \mathbb{Z}^n_{\geq 0}\}$ with $K = m \cdot \lceil \log(2m\Delta + 1) \rceil$. If it returns a positive value, the ILP is unbounded. Let us argue why this is enough. We need to understand that when there is a positive solution to $\max\{c^T x : Ax = 0, x \in \mathbb{Z}^n_{\geq 0}\}$, then there is also a positive solution with 1-norm at most $(2m\Delta + 1)^m \leq 2^K$. Let x^* be a positive solution to the former ILP with minimal 1-norm, i.e., $c^T x^* > 0$ and $\|x^*\|_1$ minimal. Let $v^{(1)}, \dots, v^{(t)}$ be the multiset of columns representing x^*. We assume that they are ordered as in Corollary 2. If $t > (2m\Delta + 1)^m$, then there must be two identical partial sums $\sum_{i=1}^{j} v^{(i)} = \sum_{i=1}^{k} v^{(i)}$ with $j < k$. In other words, the circle can be decomposed into two circles $v^{(1)}, \dots, v^{(j)}, v^{(k+1)}, \dots, v^{(t)}$ and $v^{(j+1)}, \dots, v^{(k)}$. One of these must be a positive solution or else their sum would be negative. This means the 1-norm of x^* is not minimal. We conclude that $t \leq (2m\Delta + 1)^m$.

4 Improvements to the running time

4.1 Applying convolution

Can we speed up the computation of entries in the dynamic table? Let D_i be the set of vectors b' with $\|b' - 2^{-i} \cdot b\|_\infty \leq 4m\Delta$. Recall, the dynamic programs computes values for each element in $D_K, D_{K-1}, \dots, D_1$. More precisely for the value of $b' \in D_i$ we consider vectors b'' such that $b'', b' - b'' \in D_{i+1}$ and take the maximum sum of the values for $b'', b' - b''$ among all. First consider only the case of $m = 1$. Here we have that $b' \in D_i$ is equivalent to $-4\Delta \leq b' - 2^{-i} \cdot b \leq 4\Delta$. This problem is well studied. It is a variant of (min, +)-convolution.

(MIN, +)-CONVOLUTION
Input: $r_1, \dots, r_n$ and $s_1, \dots, s_n$.
Output: $t_1, \dots, t_n$, where $t_k = \min_{i+j=k} r_i + s_j$.

(max, +)-convolution is the counterpart where the maximum is taken instead of the minimum. The two problems are equivalent. Each of them can be transformed to the other by negating the elements. We construct an instance of (max, +)-convolution of size $12\Delta + 2$. We set r_j and s_j, $j \in \{1, \dots, 8\Delta + 1\}$ both to the value for $b/2^{i+1} - (4\Delta + 1) + j \in D_{i+1}$ in the dynamic table. Set the remaining values of r and s to $-\infty$. Then for $b' = b/2^i - (4\Delta + 1) + k \in D_i$, the correct result will be at $t_{4\Delta+1+k}$.

(min, +)-convolution admits a trivial $O(n^2)$ time algorithm and it has been conjectured that there exists no truly sub-quadratic algorithm [7]. There does, however, exist an $O(n^2/\log(n))$ time algorithm [4], which we are going to use. In fact, there is a slightly faster algorithm with running time $n^2/2^{\Omega(\sqrt{\log(n)})}$ [6].

We can reduce the problem for arbitrary m to a (max, +)-convolution instance of size $O(m\Delta)^m$. To do so, project a vector $b' \in D_i$ to

$$f_i(b') = \sum_{j=1}^{m} (16m\Delta + 3)^{j-1} \underbrace{(4m\Delta + 1 + b'_j - b_j/2^i)}_{\in [1,\ 8m\Delta+1]} . \tag{2}$$

The value $16m\Delta + 3$ is chosen because it is always greater than the sum of two values of the form $4m\Delta + 1 + b'_j - b_j/2^i$. For all $a, a' \in D_{i+1}, b' \in D_i$, it holds that $f_{i+1}(a) + f_{i+1}(a') = f_i(b')$, if and only if $a + a' = b' - (4m\Delta + 1, \dots, 4m\Delta + 1)^T$:

Proof ⇒. Let $f_{i+1}(a) + f_{i+1}(a') = f_i(b')$. Then in particular,

$$f_{i+1}(a) + f_{i+1}(a') \equiv f_i(b') \mod 16m\Delta + 3$$

Since all but the first element of the sum (2) are multiples of $16m\Delta + 3$, i.e., they are equal 0 modulo $16m\Delta + 3$, we can omit them in the equation. Hence,

$$(4m\Delta + 1 + a_1 - b_1/2^{i+1}) + (4m\Delta + 1 + a'_1 - b_1/2^{i+1}) \equiv (4m\Delta + 1 + b'_1 - b_1/2^i) \mod 16m\Delta + 3.$$

We even have equality (without modulo) here, because both sides are smaller than $16m\Delta + 3$. Simplifying the equation gives $a_1 + a'_1 = b'_1 - (4m\Delta + 1)$. Now consider again the equation $f_{i+1}(a) + f_{i+1}(a') = f_i(b')$. In the sums leave out the first element. The equation still holds, since by the elaboration above this changes the left and right hand-side by the same value. We can now repeat the same argument to obtain $a_2 + a'_2 = b'_2 - (4m\Delta + 1)$ and the same for all other dimensions. ◀

Proof $\Leftarrow$. Let $a + a' = b' - (4m\Delta + 1, \dots, 4m\Delta + 1)^T$. Then for every j,

$$(4m\Delta + 1 + a_j - b_j/2^{i+1}) + (4m\Delta + 1 + a'_j - b_j/2^{i+1}) = 4m\Delta + 1 + b'_j - b_j/2^i.$$

It directly follows that $f_{i+1}(a) + f_{i+1}(a') = f_i(b')$. ◀

This means when we write the value of each $b'' \in D_{i+1}$ to r_j and s_j, where $j = f_{i+1}(b'')$, the correct solutions will be in t. More precisely, we can read the result for some $b' \in D_i$ at t_k where $k = f_i(b' + (4m\Delta + 1, \dots, 4m\Delta + 1)^T)$.

With an algorithm for (min, +)-convolution with running time $T(n)$ we get an algorithm with running time $T(O(m\Delta)^m) \cdot (m \log(m\Delta) + \log(\|b\|_\infty))$. Inserting $T(n) = n^2/\log(n)$ we get:

▶ **Theorem 5.** *There exists an algorithm that finds the optimum of* $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$*, in time* $O(m\Delta)^{2m} \cdot (1 + \log(\|b\|_\infty)/\log(\Delta))$.

Clearly, a sub-quadratic algorithm, where $T(n) = n^{2-\delta}$ for some $\delta > 0$, would directly improve the exponent. Next, we will consider the problem of only testing feasibility of an ILP. Since we only record whether or not there exists a solution for a particular right-hand side, the convolution problem reduces to the following.

Boolean convolution
Input: $r_1, \dots, r_n \in \{0, 1\}$ and $s_1, \dots, s_n \in \{0, 1\}$.
Output: $t_1, \dots, t_n \in \{0, 1\}$, where $t_k = \bigvee_{i+j=k} r_i \wedge s_j$.

This problem can be solved very efficiently via fast Fourier transform. We compute the $(+, \cdot)$-convolution of the input. It is well known that this can be done using FFT in time $O(n \log(n))$. The $(+, \cdot)$-convolution of r and s is the vector t, where $t_k = \sum_{i+j=k} r_i \cdot s_j$. To get the Boolean convolution instead, we simply replace each $t_k > 0$ by 1. Using $T(n) = O(n \log(n))$ for the convolution algorithm we obtain the following.

▶ **Theorem 6.** *There exists an algorithm that finds an element in* $\{x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$*, if there is one, in time* $O(m\Delta)^m \cdot \log(\Delta) \cdot \log(\Delta + \|b\|_\infty)$.

This can be seen from the calculation below. First we scrape off factors polynomial in m:

$$O(m\Delta)^m \cdot m \log(m\Delta) \cdot (m \log(m\Delta) + \log(\|b\|_\infty)) \leq O(m\Delta)^m \cdot \log(\Delta) \cdot (\log(\Delta) + \log(\|b\|_\infty))$$

Next, we use that $\log(\Delta) + \log(\|b\|_\infty) = \log(\Delta \cdot \|b\|_\infty) \leq \log((\Delta + \|b\|_\infty)^2) = O(\log(\Delta + \|b\|_\infty))$.

4.2 Use of proximity

Eisenbrand and Weismantel gave the following bound on the proximity between continuous and integral solutions.

▶ **Theorem 7** ([8]). *Let* $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\geq 0}\}$ *be feasible and bounded. Let* x^* *be an optimal vertex solution of the fractional relaxation. Then there exists an optimal solution* z^* *with*

$$\|z^* - x^*\|_1 \leq m(2m\Delta + 1)^m.$$

We briefly explain, how they use this theorem to reduce the right-hand side b at the expense of computing the optimum of the fractional relaxation: Note that $z^*_i \geq \ell_i := \max\{0, \lceil x^*_i \rceil - m(2m\Delta + 1)^m\}$. Since x^* is a vertex solution, it has at most m non-zero components. By setting $y = x - \ell$ we obtain the equivalent ILP $\max\{c^T y : Ay = b - A\ell, y \in \mathbb{Z}^n_{\geq 0}\}$. Indeed, this ILP has a bounded right-hand side:

$$\|b - A\ell\|_\infty = \|A(x^* - \ell)\|_\infty \leq \Delta m^2 (2m\Delta + 1)^m = O(m\Delta)^{m+1}.$$

Here, we use that x^* and ℓ differ only in non-zero components of x^* and in those by at most $m(2m\Delta+1)^m$. Like in earlier bounds, the O-notation hides polynomial terms in m. Using the $n \cdot O(m\Delta)^{2m} \cdot \|b\|_1^2$ time algorithm from [8], this gives a running time of $n \cdot O(m\Delta)^{4m+2} + \mathrm{LP}$, where LP is the time to solve the relaxation. The logarithmic dependence on $\|b\|_\infty$ in our new algorithm leads to a much smaller exponent: Using Theorem 5 and the construction above, the ILP can be solved in time $O(m\Delta)^{2m} + \mathrm{LP}$. Feasibility can be tested in time $O(m\Delta)^m \cdot \log^2(\Delta) + \mathrm{LP}$ using Theorem 6.

4.3 Heterogeneous matrices

Let $\Delta_1, \dots, \Delta_m \leq \Delta$ denote the largest absolute values of each row in A. When some of these values are much smaller than Δ, the maximum among all, we can do better than $O(m\Delta)^{2m} \cdot \log(\|b\|_\infty)$. An example for a highly heterogeneous matrix is UNBOUNDED KNAPSACK with cardinality constraints. Consider the norm $\|v\| = 1/2 \cdot \max_k |v_k/\Delta_k|$ and let $v^{(1)}, \dots, v^{(t)} \in \mathbb{Z}^m$ be the multiset of columns corresponding to an optimal solution of the ILP. Using the Steinitz Lemma on this norm, it follows that there exists a permutation π such that for all $j \in \{1, \dots, t\}$ and $k \in \{1, \dots, k\}$

$$|\sum_{i=1}^{j} v_k^{(\pi(j))} - \frac{j}{t} \cdot b_k| \leq 2m\Delta_k.$$

This means the number of states we have to consider reduces from $O(m\Delta)^m$ to $\prod_{k=1}^m O(m\Delta_k)$ at each level of the dynamic program. Hence, we obtain the running time $\prod_{k=1}^m O(m\Delta_k)^2 \cdot \log(\|b\|_\infty)$. When the objective function has small coefficients, it is more efficient to perform a binary search for the optimum and encode the objective function as an additional constraint. We can bound the optimum by $O(m\Delta)^m \cdot (\|b\|_\infty + 1) \cdot \|c\|_\infty$ using the bound on the 1-norm of the solution. Hence, the binary search takes at most $O(m \log(m\Delta \cdot \|c\|_\infty \cdot \|b\|_\infty)) = O(m \log(m\Delta + \|c\|_\infty + \|b\|_\infty))$ iterations. For a guess τ the following feasibility ILP tests if there is a solution of value at least τ.

$$\begin{pmatrix} c_1 & \dots & c_n & -1 \\ & & & 0 \\ & A & & \vdots \\ & & & 0 \end{pmatrix} x = \begin{pmatrix} \tau \\ b_1 \\ \vdots \\ b_n \end{pmatrix}$$
$$x \in \mathbb{Z}^{n+1}_{\geq 0}$$

We can solve the ILP above in time

$$T(\|c\|_\infty \cdot \prod_{k=1}^m O((m+1)\Delta_k)) \cdot \log(\|b\|_\infty + \tau) \leq T(\|c\|_\infty \cdot \prod_{k=1}^m O(m\Delta_k)) \cdot m \log(m\Delta + \|c\|_\infty + \|b\|_\infty),$$

where $T(n) = O(n \log(n))$ is the running time of Boolean convolution. By adding the time for the binary search and by hiding polynomials in m, we get the total running time of

$$\|c\|_\infty \cdot \prod_{k=1}^m [O(m\Delta_k)] \cdot \log(\Delta + \|c\|_\infty) \cdot \log^2(\Delta + \|c\|_\infty + \|b\|_\infty).$$

5 Lower bounds

5.1 Optimization problem

We use an equivalence between Unbounded Knapsack and (min, +)-convolution regarding sub-quadratic algorithms.

Unbounded Knapsack
Input: $C \in \mathbb{N}$, $w_1, \dots, w_n \in \mathbb{N}$, and $p_1, \dots, p_n \in \mathbb{N}$.
Output: Multiplicities $x_1, \dots, x_n$, such that $\sum_{i=1}^n x_i \cdot w_i \le C$ and $\sum_{i=1}^n x_i \cdot p_i$ is maximized.

Note that when we instead require $\sum_{i=1}^n x_i \cdot w_i = C$ in the problem above, we can transform it to this form by adding an item of profit zero and weight 1.

▶ **Theorem 8** ([7]). *For any $\delta > 0$ there exists no $O((n+C)^{2-\delta})$ time algorithm for* Unbounded Knapsack *unless there exists a truly sub-quadratic algorithm for (min, +)-convolution.*

When using this theorem, we assume that the input already consists of the at most C relevant items only, $n \le C$, and $w_i \le C$ for all i. This preprocessing can be done in time $O(n+C)$.

▶ **Theorem 9.** *For every fixed m there does not exist an algorithm that solves ILPs with m constraints in time $f(m) \cdot (n^{2-\delta} + (\Delta + \|b\|_\infty)^{2m-\delta})$ for some $\delta > 0$ and a computable function f, unless there exists a truly sub-quadratic algorithm for (min, +)-convolution.*

Proof. Let $\delta > 0$ and $m \in \mathbb{N}$. Assume that there exists an algorithm that solves ILPs of the form $\max\{c^T x : Ax = b, x \in \mathbb{Z}^n_{\ge 0}\}$ where $A \in \mathbb{Z}^{m \times n}$, $b \in \mathbb{Z}^m$, and $c \in \mathbb{Z}^n$ in time $f(m) \cdot (n^{2-\delta} + (\Delta + \|b\|_\infty)^{2m-\delta})$, where Δ is the greatest absolute value in A. We will show that this implies an $O((n+C)^{2-\delta'})$ time algorithm for the Unbounded Knapsack Problem for some $\delta' > 0$. Let $(C, (w_i)_{i=1}^n, (p_i)_{i=1}^n)$ be an instance of this problem. Let us first observe that the claim holds for $m = 1$. Clearly the Unbounded Knapsack Problem (with equality) can be written as the following ILP (UKS1).

$$\max \sum_{i=1}^n p_i \cdot x_i$$
$$\sum_{i=1}^n w_i \cdot x_i = C$$
$$x \in \mathbb{Z}^n_{\ge 0}$$

Since $w_i \le C$ for all i (otherwise the item can be discarded), we can solve this ILP by assumption in time $f(1) \cdot (n^{2-\delta} + (2C)^{2-\delta}) \le O((n+C)^{2-\delta})$. Now consider the case where $m > 1$. We want to reduce Δ by exploiting the additional rows. Let $\Delta = \lfloor C^{1/m} \rfloor + 1 > C^{1/m}$. We write C in base-Δ notation, i.e.,

$$C = C^{(0)} + \Delta C^{(1)} + \dots + \Delta^{m-1} C^{(m-1)},$$

where $0 \le C^{(k)} < \Delta$ for all k. Likewise, write $w_i = w_i^{(0)} + \Delta w_i^{(1)} + \dots + \Delta^{m-1} w_i^{(m-1)}$ with

$0 \le w_i^{(k)} < \Delta$ for all k. We claim that (UKS1) is equivalent to the following ILP (UKSm).

$$\max \sum_{i=1}^{n} p_i \cdot x_i$$

$$\sum_{i=1}^{n} [w_i^{(0)} \cdot x_i] - \Delta \cdot y_1 = C^{(0)} \tag{3}$$

$$\sum_{i=1}^{n} [w_i^{(1)} \cdot x_i] + y_1 - \Delta \cdot y_2 = C^{(1)} \tag{4}$$

$$\vdots$$

$$\sum_{i=1}^{n} [w_i^{(m-2)} \cdot x_i] + y_{m-2} - \Delta \cdot y_{m-1} = C^{(m-2)} \tag{5}$$

$$\sum_{i=1}^{n} [w_i^{(m-1)} \cdot x_i] + y_{m-1} = C^{(m-1)} \tag{6}$$

$$x \in \mathbb{Z}^n_{\ge 0}$$
$$y \in \mathbb{Z}^m_{\ge 0}$$

Claim $x \in$ (USK1) $\Rightarrow x \in$ (USKm)

Let x be a solution to (UKS1). Then for all $1 \le \ell \le m$,

$$\sum_{i=1}^{n} \sum_{k=0}^{\ell-1} \Delta^k w_i^{(k)} \cdot x_i \equiv \sum_{i=1}^{n} w_i \cdot x_i \equiv C \equiv \sum_{k=0}^{\ell-1} \Delta^k C^{(k)} \mod \Delta^\ell.$$

This is because all $\Delta^\ell w_i^{(\ell)}, \dots, \Delta^{m-1} w_i^{(m-1)}$ and $\Delta^\ell C^{(\ell)}, \dots, \Delta^{m-1} C^{(m-1)}$ are multiples of Δ^ℓ. It follows that there exists an $y_\ell \in \mathbb{Z}$ such that

$$\sum_{i=1}^{n} \sum_{k=0}^{\ell-1} [\Delta^k w_i^{(k)} \cdot x_i] - \Delta^\ell \cdot y_\ell = \sum_{k=0}^{\ell-1} \Delta^k C^{(k)}.$$

Furthermore, y_ℓ is non-negative, because otherwise

$$\begin{aligned}\sum_{k=0}^{\ell-1} \Delta^k C^{(k)} \le \sum_{k=0}^{\ell-1} \Delta^k (\Delta - 1) < \Delta^{\ell-1}(\Delta - 1) \sum_{k=0}^{\infty} \Delta^{-k} \\ = \Delta^{\ell-1} \frac{\Delta - 1}{1 - \frac{1}{\Delta}} = \Delta^\ell \le -\Delta^\ell y_\ell \le \sum_{i=1}^{n} \sum_{k=0}^{\ell-1} [\Delta^k w_i^{(k)} \cdot x_i] - \Delta^\ell y_\ell.\end{aligned}$$

We choose $y_1, \dots, y_m$ exactly like this. The first constraint (3) follows directly. Now let $\ell \in \{2, \dots, m\}$. By choice of $y_{\ell-1}$ and y_ℓ we have that

$$\sum_{i=1}^{n} \Bigg[\underbrace{\left(\sum_{k=0}^{\ell-1} \Delta^k w_i^{(k)} - \sum_{k=0}^{\ell-2} \Delta^k w_i^{(k)} \right)}_{= \Delta^{\ell-1} w_i^{(\ell-1)}} \cdot x_i \Bigg] + \Delta^{\ell-1} \cdot y_{\ell-1} - \Delta^\ell \cdot y_\ell = \underbrace{\sum_{k=0}^{\ell-1} \Delta^k C^{(k)} - \sum_{k=0}^{\ell-2} \Delta^k C^{(k)}}_{= \Delta^{\ell-1} C^{(\ell-1)}}. \tag{7}$$

Dividing both sides by $\Delta^{\ell-1}$ we get every constraint (4) - (5) for the correct choice of ℓ. Finally, consider the special case of the last constraint (6). By choice of y_m we have that

$$\sum_{i=1}^{n} \underbrace{\sum_{k=0}^{m-1} \Delta^k w_i^{(k)}}_{=w_i} \cdot x_i - \Delta^m \cdot y_m = \underbrace{\sum_{k=0}^{m-1} \Delta^k C^{(k)}}_{=C}.$$

Thus, $y_m = 0$ and (7) implies the last constraint (with $\ell = m$).

Claim $x \in$ (USKm) $\Rightarrow x \in$ (USK1)

Let $x_1, \ldots, x_n, y_1, \ldots, y_{m-1}$ be a solution to (UKSm) and set $y_m = 0$. We show by induction that for all $\ell \in \{1, \ldots, m\}$

$$\sum_{i=1}^{n} \sum_{k=0}^{\ell-1} \Delta^k w_i^{(k)} \cdot x_i - \Delta^\ell y_\ell = \sum_{k=0}^{\ell-1} \Delta^k C^{(k)}.$$

With $\ell = m$ this implies the claim as $y_m = 0$ by definition. For $\ell = 1$ the equation is exactly the first constraint (3). Now let $\ell > 1$ and assume that the equation above holds. We will show that it also holds for $\ell + 1$. From (USKm) we have

$$\sum_{i=1}^{n} [w_i^{(\ell)} \cdot x_i] + y_\ell - \Delta \cdot y_{\ell+1} = C^{(\ell)}.$$

Multiplying each side by Δ^ℓ we get

$$\sum_{i=1}^{n} [\Delta^\ell w_i^{(\ell)} \cdot x_i] + \Delta^\ell y_\ell - \Delta^{\ell+1} \cdot y_{\ell+1} = \Delta^\ell C^{(\ell)}.$$

By adding and subtracting the same elements, it follows that

$$\sum_{i=1}^{n} \left[\left(\sum_{k=0}^{\ell} \Delta^k w_i^{(k)} - \sum_{k=0}^{\ell-1} \Delta^k w_i^{(k)} \right) \cdot x_i \right] + \Delta^\ell \cdot y_\ell - \Delta^{\ell+1} \cdot y_{\ell+1} = \sum_{k=0}^{\ell} \Delta^k C^{(k)} - \sum_{k=0}^{\ell-1} \Delta^k C^{(k)}.$$

By inserting the induction hypothesis we conclude

$$\sum_{i=1}^{n} \sum_{k=0}^{\ell} [\Delta^k w_i^{(k)} \cdot x_i] - \Delta^{\ell+1} y_{\ell+1} = \sum_{k=0}^{\ell} \Delta^k C^{(k)}.$$

Constructing and solving the ILP

The ILP (UKSm) can be constructed easily in $O(Cm + nm) \leq O((n + C)^{2-\delta/m})$ operations (recall that m is a constant). We obtain $\Delta = \lfloor C^{1/m} \rfloor + 1$ by guessing: More precisely, we iterate over all numbers $\Delta_0 \leq C$ and find the one where $(\Delta_0 - 1)^m < C \leq \Delta_0^m$. There are of course more efficient, non-trivial ways to compute the rounded m-th root. The base-Δ representation for $w_1, \ldots, w_n$ and C can be computed with $O(m)$ operations for each of these numbers.

All entries of the matrix in (UKSm) and the right-hand side are bounded by $\Delta = O(C^{1/m})$. Therefore, by assumption this ILP can be solved in time

$$f(m) \cdot (n^{2-\delta} + O(C^{1/m})^{2m-\delta}) \leq f(m) \cdot O(1)^{2m-\delta} \cdot (n + C)^{2-\delta/m} = O((n + C)^{2-\delta/m})$$

This would therefore yield a truly sub-quadratic algorithm for the Unbounded Knapsack Problem. ◀

5.2 Feasibility problem

We will show that our algorithm for solving feasibility of ILPs is optimal (except for log factors). We use a recently discovered lower bound for k-SUM based on the SETH.

k-SUM
Input: $T \in \mathbb{N}_0$ and $Z_1, \dots, Z_k \subset \mathbb{N}_0$ where $|Z_1| + |Z_2| + \cdots + |Z_k| = n \in \mathbb{N}$.
Output: $z_1 \in Z_1, z_2 \in Z_2, \dots, z_k \in Z_k$ such that $z_1 + z_2 + \cdots + z_k = T$.

▶ **Theorem 10** ([1]). *If the SETH holds, then for every $\delta > 0$ there exists a value $\gamma > 0$ such that k-SUM cannot be solved in time $O(T^{1-\delta} \cdot n^{\gamma k})$.*

This implies that for every $p \in \mathbb{N}$ there is no $O(T^{1-\delta} \cdot n^p)$ time algorithm for k-SUM if $k \geq p/\gamma$.

▶ **Theorem 11.** *If the SETH holds, for every fixed m there does not exist an algorithm that solves feasibility of ILPs with m constraints in time $n^{f(m)} \cdot (\Delta + \|b\|_\infty)^{m-\delta}$.*

Proof. Like in the previous reduction we start with the case of $m = 1$. For higher values of m the result can be shown in the same way as before.

Suppose there exists an algorithm for solving feasibility of ILPs with one constraint in time $n^{f(1)} \cdot (\Delta + \|b\|_\infty)^{1-\delta}$ for some $\delta > 0$ and $f(1) \in \mathbb{N}$. Set $k = \lceil f(1)/\gamma \rceil$ with γ as in in Theorem 10 and consider an instance $(T, Z_1, \dots, Z_k)$ of k-SUM. We will show that this can be solved in time $O(T^{1-\delta} \cdot n^{f(1)})$, which contradicts the SETH. For every $i \leq k$ and every $z \in Z_i$ we use a binary variable $x_{i,z}$ that describes whether z is used. We can easily model k-SUM as the following ILP:

$$\sum_{i=1}^{k} \sum_{z \in Z_i} z \cdot x_{i,z} = T$$

$$\sum_{z \in Z_i} x_{i,z} = 1 \qquad \forall i \in \{1, \dots, k\}$$

$$x_{i,z} \in \mathbb{Z}_{\geq 0} \qquad \forall i \in \{1, \dots, k\}, z \in Z_i$$

However, since we want to reduce to an ILP with one constraint, we need a slightly more sophisticated construction. We will show that the cardinality constraints can be encoded into the k-SUM instance by increasing the numbers by a factor of $2^{O(k)}$, which is in $O(1)$ since k is some constant depending on $f(1)$ and γ only. We will use this to obtain an ILP with only one constraint and values of size at most $O(T)$. A similar construction is also used in [1].

Our goal is to construct an instance $(T', Z'_k, \dots, Z'_k)$ such that for every x^* it holds that x^* is a solution to the first ILP if and only if $x^* \in \{x : \sum_{i=1}^{k} \sum_{z \in Z'_i} z \cdot x_{i,z} = T', x \in \mathbb{Z}^n_{\geq 0}\}$ $(*)$. We will use one element to represent each element in the original instance. Consider the binary representation of numbers in $Z'_1 \cup \cdots \cup Z'_k$ and of T'. The numbers in the new instance will consist of three parts and $\lceil \log(k) \rceil$ many 0s between them to prevent interference. For an illustration of the construction see Figure 2. The $\lceil \log(k) \rceil$ most significant bits ensure that exactly k elements are selected; the middle part are k bits that ensure of every set Z'_i exactly one element is selected; the least significant $\lceil \log(T) \rceil$ bits represent the original values of the elements. Set the values in the first part of the numbers to 1 for all elements $Z'_1 \cup \cdots \cup Z'_k$ and to k in T'. Clearly this ensures that at most k elements are chosen. The sum of at most k elements cannot be larger than $k \leq 2^{\lceil \log(k) \rceil}$ times the biggest element. This implies that the buffers of $\lceil \log(k) \rceil$ zeroes cannot overflow and we can consider each of the three parts independently. It follows that exactly k elements must be chosen by any

$$Z_i' \ni z' = \overbrace{\underbrace{0\dots0001}_{\lceil\log(k)\rceil}}^{\mathrm{bin}(1)} \mid \underbrace{0\dots0}_{\lceil\log(k)\rceil} \mid \overbrace{\underbrace{0\dots010\dots0}_{k}}^{\mathrm{bin}(2^i)} \mid \underbrace{0\dots0}_{\lceil\log(k)\rceil} \mid \overbrace{\underbrace{0110\dots}_{\lceil\log(T)\rceil}}^{\mathrm{bin}(z)}$$

$$T' = \overbrace{\underbrace{0\dots1011}_{\lceil\log(k)\rceil}}^{\mathrm{bin}(k)} \mid \underbrace{0\dots0}_{\lceil\log(k)\rceil} \mid \overbrace{\underbrace{1111\dots1111}_{k}}^{\mathrm{bin}(2^{k+1}-1)} \mid \underbrace{0\dots0}_{\lceil\log(k)\rceil} \mid \overbrace{\underbrace{1011\dots}_{\lceil\log(T)\rceil}}^{\mathrm{bin}(T)}$$

Figure 2 Construction of Z_i' and T'.

feasible solution. The system $\{x : \sum_{i=1}^{k} 2^i x_i = 2^{k+1} - 1, \|x\|_1 = k, \mathbb{Z}_{\geq 0}^k\}$ has exactly one solution and this solution is $(1, 1, \dots, 1)$: Consider summing up k powers of 2 and envision the binary representation of the partial sums. When we add some 2^i to the partial sum, the number of ones in the binary representation increases by one, if the i'th bit of the current sum is zero. Otherwise, it does not increase. However, since in the binary representation of the final sum there are k ones, it has to increase in each addition. This means no power of two can be added twice and therefore each has to be added exactly once.

It follows that the second part of the numbers enforces that of every Z_i' exactly one element is chosen. We conclude that $(*)$ solves the initial k-SUM instance. By assumption this can be done in time $n^{f(1)} \cdot (\Delta + \|b\|_\infty)^{1-\delta} = n^{f(1)} \cdot O(T')^{1-\delta} = O(n^{f(1)} \cdot T^{1-\delta})$. Here we use that $T' \leq 2^{3\log(k)+k+\log(T)+4} = O(k^3 2^k T) = O(T)$, since k is a constant.

For $m > 1$ we can use the same construction as in the reduction for the optimization problem: Suppose there is an algorithm that finds feasible solutions to ILPs with m constraints in time $n^{f(m)} \cdot (\Delta + \|b\|_\infty)^{m-\delta}$. Choose γ such that there is no algorithm for k-SUM with running time $O(T^{1-\delta/m} \cdot n^{\gamma k})$ (under SETH). We set $k = \lceil f(m)/\gamma \rceil$. By splitting the one constraint of $(*)$ into m constraints we can reduce the upper bound on elements from $O(T)$ to $O(T^{1/m})$. This means the assumed running time for solving ILPs can be used to solve k-SUM in time

$$n^{f(m)} \cdot O(T^{1/m})^{m-\delta} \leq n^{\gamma k} \cdot O(1)^{m-\delta} \cdot T^{1-\delta/m} = O(n^{\gamma k} \cdot T^{1-\delta/m}). \qquad \blacktriangleleft$$

6 Applications

We describe the implications of our results on a couple of well-known problems, which can be formulated using ILPs with few constraints and small entries. In particular, we give an example, where the reduction of the running time by a factor n improves on the state-of-the-art and one where the logarithmic dependence on $\|b\|_\infty$ proves useful.

6.1 Unbounded Knapsack and Unbounded Subset-Sum

Unbounded Knapsack with equality constraint is simply an ILP with $m = 1$ and positive entries and objective function:

$$\max\{\sum_{i=1}^{n} p_i \cdot x_i : \sum_{i=1}^{n} w_i \cdot x_i = C, x \in \mathbb{Z}_{\geq 0}^n\}$$

where $p_i \geq 0$ are called the profits and $w_i \geq 0$ the weights of the items $1, \dots, n$. More common is to let C be only an upper bound on $\sum_{i=1}^{n} w_i \cdot x_i$, but that variant easily reduces to the problem above by adding a slack variable. Unbounded Subset-Sum is the same problem without an objective function, i.e., the problem of finding a multiset of items whose weights sum up to exactly C. We assume that no two items have the same weight. Otherwise in time $O(n+\Delta)$ we can remove all duplicates by keeping only the most valuable ones. The fractional solutions to both problems are of a very simple structure: For Unbounded Knapsack choose only the item i of maximal efficiency, that is p_i/w_i, and select it C/w_i times. For Unbounded Subset-Sum choose an arbitrary item. This gives algorithms with running time $O(\Delta^2)$ and $O(\Delta \log^2(\Delta))$ for Unbounded Knapsack and Unbounded Subset-Sum, respectively, where Δ is the maximum weight among all items (using the results from Section 4.2). The previously best pseudo-polynomial algorithms for Unbounded Knapsack, have running times $O(nC)$ (standard dynamic programming; see e.g. [13]), $O(n\Delta^2)$ [8], or very recently $O(\Delta^2 \log(C))$ [2]. We note that the algorithm from the last paper, which was discovered independently and concurrently to our results, also uses (min, +)-convolution. It could probably be improved to the same running time as our general algorithm using the proximity ideas. For Unbounded Subset-Sum the state-of-the-art algorithm has a running time $O(C \log(C))$ [5]. Hence, our algorithm is preferable when $\Delta \ll C$.

6.2 Scheduling on Identical Machines

The problem Scheduling on Identical Machines asks for the distribution of N jobs onto $M \leq N$ machines. Each job j has a processing time p_j and the objective is to minimize the makespan, i.e., the maximum sum of processing times on a single machine. Since an exact solution cannot be computed unless P = NP, we are satisfied with a $(1+\epsilon)$-approximation, where $\epsilon > 0$ is part of the input. We will outline how this problem can be solved using our algorithm. More details on many of the techniques involved can be found in [10].

We consider here the variant, in which a makespan τ is given and we have to find a schedule with makespan at most $(1+\epsilon)\tau$ or prove that there exists no schedule with makespan at most τ. This suffices by using a standard dual approximation framework. It is easy to see that one can discard all jobs of size at most $\epsilon \cdot \tau$ and add them greedily after a solution for the other jobs is found. The big jobs can each be rounded to the next value of the form $\epsilon \cdot \tau \cdot (1+\epsilon)^i$ for some i. This reduces the number of different processing times to $O(1/\epsilon \log(1/\epsilon))$ many and increases the makespan by at most a factor of $1+\epsilon$. We are now ready to write this problem as an ILP. A configuration is a way to use a machine. It describes how many jobs of each size are assigned to this machine. Since we aim for a makespan of $(1+\epsilon) \cdot \tau$, the sum of these sizes must not exceed this value. The configuration ILP has a variable for every valid configuration and it describes how many machines use this configuration. Let $\mathcal{C}$ be the set of valid configurations and C_k the multiplicity of size k in a configuration $C \in \mathcal{C}$. The following ILP solves the rounded instance. We note that there is no objective function in it.

$$\begin{aligned}
\sum_{C \in \mathcal{C}} x_C &= M && \\
\sum_{C \in \mathcal{C}} C_k \cdot x_C &= N_k && \forall k \in \mathcal{K} \\
x_C &\in \mathbb{Z}_{\geq 0} && \forall C \in \mathcal{C}
\end{aligned}$$

Here $\mathcal{K}$ are the rounded sizes and N_k the number of jobs with rounded size $k \in \mathcal{K}$. The first constraint enforces that the correct number of machines is used, the next $|\mathcal{K}|$ many enforce that for each size the correct number of jobs is scheduled.

It is notable that this ILP has only few constraints (a constant for a fixed choice of ϵ) and also the entries of the matrix are small. More precisely, they are at most $1/\epsilon$, since every size is at least $\epsilon \cdot \tau$ and therefore no more than $1/\epsilon$ jobs fit in one configuration. The ILP can be solved with our algorithm. Note that $\Delta \leq 1/\epsilon$, $m = O(1/\epsilon \log(1/\epsilon))$, $\|b\|_\infty \leq N$, and $n \leq (1/\epsilon)^{O(1/\epsilon \log(1/\epsilon))}$. Including the rounding in time $O(N + 1/\epsilon \log(1/\epsilon))$ the running time for the ILP is

$$O(m\Delta)^m \cdot \log(\Delta) \cdot \log(\Delta + \|b\|_\infty) + O(nm) + O(N + 1/\epsilon \log(1/\epsilon))$$
$$\leq 2^{O(1/\epsilon \log^2(1/\epsilon))} \log(N) + O(N + 1/\epsilon \log(1/\epsilon)) \leq 2^{O(1/\epsilon \log^2(1/\epsilon))} + O(N).$$

The trick in the bound above is to distinguish between the cases $2^{O(1/\epsilon \log^2(1/\epsilon))} \leq \log(N)$ and $2^{O(1/\epsilon \log^2(1/\epsilon))} > \log(N)$. The same running time (except for a higher constant in the exponent) could be obtained with [8]. However, in order to avoid a multiplicative factor of N, one would have to solve the LP relaxation first and then use proximity. Our approach gives an easier, purely combinatorial algorithm. The crucial feature of our algorithm is the lower dependence on $\|b\|_\infty$.

References

1 Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-Based Lower Bounds for Subset Sum and Bicriteria Path. *CoRR*, abs/1704.04546, 2017. arXiv:1704.04546.

2 Kyriakos Axiotis and Christos Tzamos. Capacitated Dynamic Programming: Faster Knapsack and Graph Algorithms. *CoRR*, abs/1802.06440, 2018. arXiv:1802.06440.

3 Arturs Backurs, Piotr Indyk, and Ludwig Schmidt. Better Approximations for Tree Sparsity in Nearly-Linear Time. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 2215–2229, 2017. doi:10.1137/1.9781611974782.145.

4 David Bremner, Timothy M. Chan, Erik D. Demaine, Jeff Erickson, Ferran Hurtado, John Iacono, Stefan Langerman, Mihai Patrascu, and Perouz Taslakian. Necklaces, Convolutions, and X+Y. *Algorithmica*, 69(2):294–314, 2014. doi:10.1007/s00453-012-9734-3.

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, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1073–1084, 2017. doi:10.1137/1.9781611974782.69.

6 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, Portland, OR, USA, June 14-17, 2015*, pages 31–40, 2015. doi:10.1145/2746539.2746568.

7 Marek Cygan, Marcin Mucha, Karol Wegrzycki, and Michal Wlodarczyk. On Problems Equivalent to (min, +)-Convolution. In *44th International Colloquium on Automata, Languages, and Programming, ICALP 2017, July 10-14, 2017, Warsaw, Poland*, pages 22:1–22:15, 2017. doi:10.4230/LIPIcs.ICALP.2017.22.

8 Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for Integer Programming using the Steinitz Lemma. In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 808–816, 2018. doi:10.1137/1.9781611975031.52.

9 Fedor V. Fomin, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. On the Optimality of Pseudo-polynomial Algorithms for Integer Programming. In *26th Annual European Symposium on Algorithms, ESA 2018, August 20-22, 2018, Helsinki, Finland*, pages 31:1–31:13, 2018. doi:10.4230/LIPIcs.ESA.2018.31.

10 Klaus Jansen, Kim-Manuel Klein, and José Verschae. Closing the Gap for Makespan Scheduling via Sparsification Techniques. In *43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy*, pages 72:1–72:13, 2016. doi:10.4230/LIPIcs.ICALP.2016.72.

11 Hendrik W. Lenstra Jr. Integer Programming with a Fixed Number of Variables. *Math. Oper. Res.*, 8(4):538–548, 1983. doi:10.1287/moor.8.4.538.

12 Ravi Kannan. Minkowski's Convex Body Theorem and Integer Programming. *Math. Oper. Res.*, 12(3):415–440, 1987. doi:10.1287/moor.12.3.415.

13 Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer, 2004.

14 Eduardo Sany Laber, Wilfredo Bardales Roncalla, and Ferdinando Cicalese. On lower bounds for the Maximum Consecutive Subsums Problem and the (min, +)-convolution. In *2014 IEEE International Symposium on Information Theory, Honolulu, HI, USA, June 29 - July 4, 2014*, pages 1807–1811, 2014. doi:10.1109/ISIT.2014.6875145.

15 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981. doi:10.1145/322276.322287.

16 Sergey Vasil'evich Sevast'janov. Approximate solution of some problems in scheduling theory. *Metody Diskret. Analiz*, 32:66–75, 1978. in Russian.

17 Ernst Steinitz. Bedingt konvergente Reihen und konvexe Systeme. *Journal für die reine und angewandte Mathematik*, 143:128–176, 1913.