Tight Complexity Lower Bounds for Integer Linear Programming with Few Constraints

Dušan Knop
Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Germany
Department of Theoretical Computer Science, Faculty of Information Technology, Czech Technical University in Prague, Czech Republic
knop.dusan@fit.cvut.cz

Michał Pilipczuk
Institute of Informatics, University of Warsaw, Poland
michal.pilipczuk@mimuw.edu.pl

Marcin Wrochna
Institute of Informatics, University of Warsaw, Poland
University of Oxford, UK
m.wrochna@mimuw.edu.pl

Abstract

We consider the standard ILP Feasibility problem: given an integer linear program of the form $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$, where A is an integer matrix with k rows and ℓ columns, $\mathbf{x}$ is a vector of ℓ variables, and $\mathbf{b}$ is a vector of k integers, we ask whether there exists $\mathbf{x} \in \mathbb{N}^\ell$ that satisfies $A\mathbf{x} = \mathbf{b}$. Each row of A specifies one linear *constraint* on $\mathbf{x}$; our goal is to study the complexity of ILP Feasibility when both k, the number of constraints, and $\|A\|_\infty$, the largest absolute value of an entry in A, are small.

Papadimitriou [29] was the first to give a fixed-parameter algorithm for ILP Feasibility under parameterization by the number of constraints that runs in time $((\|A\|_\infty + \|\mathbf{b}\|_\infty) \cdot k)^{\mathcal{O}(k^2)}$. This was very recently improved by Eisenbrand and Weismantel [9], who used the Steinitz lemma to design an algorithm with running time $(k\|A\|_\infty)^{\mathcal{O}(k)} \cdot \|\mathbf{b}\|_\infty^2$, which was subsequently improved by Jansen and Rohwedder [17] to $\mathcal{O}(k\|A\|_\infty)^k \cdot \log \|\mathbf{b}\|_\infty$. We prove that for $\{0,1\}$-matrices A, the running time of the algorithm of Eisenbrand and Weismantel is probably optimal: an algorithm with running time $2^{o(k \log k)} \cdot (\ell + \|\mathbf{b}\|_\infty)^{o(k)}$ would contradict the Exponential Time Hypothesis (ETH). This improves previous non-tight lower bounds of Fomin et al. [10].

We then consider integer linear programs that may have many constraints, but they need to be structured in a "shallow" way. Precisely, we consider the parameter *dual treedepth* of the matrix A, denoted $\mathrm{td}_D(A)$, which is the treedepth of the graph over the rows of A, where two rows are adjacent if in some column they simultaneously contain a non-zero entry. It was recently shown by Koutecký et al. [24] that ILP Feasibility can be solved in time $\|A\|_\infty^{2^{\mathcal{O}(\mathrm{td}_D(A))}} \cdot (k + \ell + \log \|\mathbf{b}\|_\infty)^{\mathcal{O}(1)}$. We present a streamlined proof of this fact and prove that, again, this running time is probably optimal: even assuming that all entries of A and $\mathbf{b}$ are in $\{-1, 0, 1\}$, the existence of an algorithm with running time $2^{2^{o(\mathrm{td}_D(A))}} \cdot (k + \ell)^{\mathcal{O}(1)}$ would contradict the ETH.

2012 ACM Subject Classification Theory of computation → Integer programming; Theory of computation → Fixed parameter tractability

Keywords and phrases integer linear programming, fixed-parameter tractability, ETH

Digital Object Identifier 10.4230/LIPIcs.STACS.2019.44

Related Version The full version of this paper is available as an arXiv preprint [23], `http://arxiv.org/abs/1811.01296`.

Funding This work is a part of projects CUTACOMBS, PowAlgDO (M. Wrochna) and TOTAL (M. Pilipczuk) that have received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreements No. 714704, No 714532, and No. 677651). Dušan Knop is supported by DFG, project "MaMu", NI 369/19. Marcin Wrochna is supported by Foundation for Polish Science (FNP) via the START stipend.

1 Introduction

Integer linear programming (ILP) is a powerful technique used in countless algorithmic results of theoretical importance, as well as applied routinely in thousands of instances of practical computational problems every day. Despite the problem being NP-hard in general, practical ILP solvers excel in solving real-life instances with thousands of variables and constraints. This can be partly explained by applying a variety of subroutines, often based on heuristic approaches, that identify and exploit structure in the input in order to apply the best suited algorithmic strategies. A theoretical explanation of this phenomenon would of course be hard to formulate, but one approach is to use the paradigm of *parameterized complexity*. Namely, the idea is to design algorithms that perform efficiently when certain relevant structural parameters of the input have moderate values.

In this direction, probably the most significant is the classic result of Lenstra [25], who proved that ILP Optimization is *fixed-parameter tractable* when parameterized by the number of variables ℓ. That is, it can be solved in time $f(\ell) \cdot |I|^{\mathcal{O}(1)}$, where f is some function and $|I|$ is the total bitsize of the input; we shall use the previous notation throughout the whole manuscript. Subsequent work in this direction [11, 18] improved the dependence of the running time on ℓ to $f(\ell) \leqslant 2^{\mathcal{O}(\ell \log \ell)}$.

In this work we turn to a different structural aspect and study ILPs that have few *constraints*, as opposed to few *variables* as in the setting considered by Lenstra. Formally, we consider the parameterization by k, the number of *constraints* (rows of the input matrix A), and $\|A\|_\infty$, the maximum absolute value over all entries in A. The situation when the number of constraints is significantly smaller than the number of variables appears naturally in many relevant settings. For instance, to encode Subset Sum as an instance of ILP Feasibility it suffices to introduce a $\{0,1\}$-variable x_i for every input number s_i, and then set only one constraint: $\sum_{i=1}^{n} s_i x_i = t$, where t is the target value. Note that the fact that Subset Sum is NP-hard for the binary encoding of the input and polynomial-time solvable for the unary encoding, explains why $\|A\|_\infty$ is also a relevant parameter for the complexity of the problem. Integer linear programs with few constraints and many variables arise most often in the study of knapsack-like and scheduling problems via the concept of so-called *configuration ILPs*, in the context of approximation and parameterized algorithms.

Parameterization by the number of constraints. Probably the first to study the complexity of integer linear programming with few constraints was Papadimitriou [29], who already in 1981 observed the following. Consider an ILP of the standard form $\{A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geqslant 0\}$, where A is an integer matrix with k rows (constraints) and ℓ columns (variables), $\mathbf{x}$ is a vector of integer variables, and $\mathbf{b}$ is a vector of integers. Papadimitriou proved that assuming such an ILP is feasible, it admits a solution with all variables bounded by $B = \ell \cdot ((\|A\|_\infty + \|\mathbf{b}\|_\infty) \cdot k)^{2k+1}$, which in turn can be found in time $\mathcal{O}((\ell B)^{k+1} \cdot |I|)$ using simple dynamic programming. Noting that by removing duplicate columns one can assume that $\ell \leqslant (2\|A\|_\infty + 1)^k$, this yields an algorithm with running time $((\|A\|_\infty + \|\mathbf{b}\|_\infty) \cdot k)^{\mathcal{O}(k^2)}$. The approach can be lifted to give an algorithm with a similar running time bound also for the ILP Optimization problem, where instead of finding any feasible solution $\mathbf{x}$, we look for one that maximizes the value $\mathbf{w}^\intercal\mathbf{x}$ for a given optimization goal vector $\mathbf{w}$.

The result of Papadimitriou was recently improved by Eisenbrand and Weismantel [9], who used the Steinitz Lemma to give an amazingly elegant algorithm solving the ILP Optimization problem (and thus also the ILP Feasibility problem) for a given instance $\{\max \mathbf{w}^\intercal\mathbf{x}\colon A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ with k constraints in time $(k\|A\|_\infty)^{\mathcal{O}(k)} \cdot \|b\|_\infty^2$. This running time has been subsequently refined by Jansen and Rohwedder [17] to $\mathcal{O}(k\|A\|_\infty)^{2k} \cdot \log\|\mathbf{b}\|_\infty$ in the case of ILP Optimization, and to $\mathcal{O}(k\|A\|_\infty)^{k} \cdot \log\|\mathbf{b}\|_\infty$ in the case of ILP Feasibility.

From the point of view of fine-grained parameterized complexity, this raises the question of whether the parametric factor $\mathcal{O}(k\|A\|_\infty)^k$ is the best possible. Jansen and Rohwedder [17] studied this question under the assumption that k is a fixed constant and $\|A\|_\infty$ is the relevant parameter. They proved that assuming the Strong Exponential Time Hypothesis (SETH), for every fixed k there is no algorithm with running time $(k \cdot (\|A\|_\infty + \|\mathbf{b}\|_\infty))^{k-\delta} \cdot |I|^{\mathcal{O}(1)}$, for any $\delta > 0$. Note that as k is considered a fixed constant, this essentially shows that the degree of $\|A\|_\infty$ needs to be at least k, but does not exclude algorithms with running time of the form $\|A\|_\infty^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$, or $2^{\mathcal{O}(k)} \cdot |I|^{\mathcal{O}(1)}$ when all entries in the input matrix A are in $\{-1, 0, 1\}$. On the other hand, the algorithms of [9, 17] provide only an upper bound of $2^{\mathcal{O}(k \log k)} \cdot |I|^{\mathcal{O}(1)}$ in the latter setting. As observed by Fomin et al. [10], a trivial encoding of 3SAT as an ILP shows a lower bound of $2^{o(k)} \cdot |I|^{\mathcal{O}(1)}$ for instances with A having entries only in $\{0, 1\}$, $\mathbf{b}$ having entries only in $\{0, 1, 2, 3\}$, and $\ell = \mathcal{O}(k)$. This still leaves a significant gap between the $2^{o(k)} \cdot |I|^{\mathcal{O}(1)}$ lower bound and the $2^{\mathcal{O}(k \log k)} \cdot |I|^{\mathcal{O}(1)}$ upper bound.

Parameterization by the dual treedepth. A related, recent line of research concerns ILPs that may have many constraints, but these constraints need to be somehow organized in a structured, "shallow" way. It started with a result of Hemmecke et al. [13], who gave a fixed-parameter tractable algorithm for solving the so-called *n-fold ILPs*. An n-fold ILP is an ILP where the constraint matrix is of the form

$$A = \begin{pmatrix} B & B & \dots & B \\ C & 0 & \cdots & 0 \\ 0 & C & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & C \end{pmatrix},$$

and the considered parameters are the dimensions of matrices B and C, as well as $\|A\|_\infty$. The running time obtained by Hemmecke et al. is $\|A\|_\infty^{\mathcal{O}(k^3)} \cdot |I|^{\mathcal{O}(1)}$ when all these dimensions are bounded by k. See [13] and the recent improvements of Eisenbrand et al. [8] for more refined running time bounds expressed in terms of particular dimensions.

The result of Hemmecke et al. [13] quickly led to multiple improvements in the best known upper bounds for several parameterized problems, where the technique of configuration ILPs is applicable [20, 21, 22]. Recently, the technique was also applied to improve the running times of several approximation schemes for scheduling problems [16]. Chen and Marx [6] introduced a more general concept of *tree-fold ILPs*, where the "star-like" structure of an n-fold ILP is generalized to any bounded-depth rooted tree, and they showed that it retains relevant fixed-parameter tractability results. This idea was followed on by Eisenbrand et al. [8] and by Koutecký et al. [24], whose further generalizations essentially boil down to considering a structural parameter called the *dual treedepth* of the input matrix A. This parameter, denoted $\mathrm{td}_D(A)$, is the smallest number h such that the rows of A can be organized into a rooted forest of height h with the following property: whenever two rows have non-zero entries in the same column, one is the ancestor of the other in the forest. As shown explicitly by Koutecký et al. [24] and somewhat implicitly by Eisenbrand et al. [8], ILP Optimization can be solved in fixed-parameter time when parameterized by $\|A\|_\infty$ and $\mathrm{td}_D(A)$.

Our results. For the parameterization by the number of constraints k, we close the above mentioned complexity gap by proving the following optimality result.

▶ **Theorem 1.** *Assuming ETH, there is no algorithm that would solve any* ILP feasibility *instance* $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ *with* $A \in \{0,1\}^{k\times\ell}$*,* $\mathbf{b} \in \mathbb{N}^k$*, and* $\ell, \|\mathbf{b}\|_\infty = \mathcal{O}(k \log k)$ *in time* $2^{o(k \log k)}$.

This shows that the algorithms of [9, 17] have the essentially optimal running time of $2^{\mathcal{O}(k\log k)} \cdot |I|^{\mathcal{O}(1)}$ also in the regime where $\|A\|_\infty$ is a constant and the number of constraints k is the relevant parameter.

The main ingredient of the proof of Theorem 1 is a certain quaint combinatorial construction – *detecting matrices* introduced by Lindström [26] – that provides a general way for compressing a system $A\mathbf{x} = \mathbf{b}$ with k equalities and bounded targets $\|\mathbf{b}\|_\infty \leqslant d$ into $\mathcal{O}(k/\log_d k)$ equalities (with unbounded targets). Each new equality is a linear combination of the original ones; in fact, just taking $\mathcal{O}(k/\log_d k)$ sums of random subsets of the original equalities suffices, but we also provide a deterministic construction taking $\mathcal{O}(dk/\log_d k)$ such subsets. By composing such a compression procedure for $d = 4$ with a standard reduction from (3,4)SAT – a variant of 3SAT where every variable occurs at most 4 times – to ILP Feasibility, we obtain a reduction that given an instance of (3,4)SAT with n variables and m clauses, produces an equivalent instance of ILP Feasibility with $k = \mathcal{O}((n + m)/\log(n + m))$ constraints. Since $2^{o(k\log k)} = 2^{o(n+m)}$, we would obtain a $2^{o(n+m)}$-time algorithm for (3,4)SAT, which is known to contradict ETH. We note that detecting matrices were recently used by two of the authors in the context of different lower bounds based on ETH [3].

For the parameterization by the dual treedepth, we first streamline the presentation of the approach of Koutecký et al. [24] and clarify that the parametric factor in the running time is doubly-exponential in the treedepth. The key ingredient here is the upper bound on ℓ_1-norms of the elements of the *Graver basis* of the input matrix A, expressed in terms of $\|A\|_\infty$ and $\mathrm{td}_D(A)$. Using standard textbook bounds for Graver bases and the recursive definition of treedepth, we prove that these ℓ_1-norms can be bounded by $(2\|A\|_\infty + 1)^{2^{\mathrm{td}_D(A)}-1}$. This, combined with the machinery developed by Koutecký et al. [24], implies the following.

▶ **Theorem 2.** *There is an algorithm that solves any given* ILP Optimization *instance* $I = \{\max \mathbf{w}^\intercal\mathbf{x} \colon A\mathbf{x} = \mathbf{b}, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}\}$ *in time* $\|A\|_\infty^{2^{\mathcal{O}(\mathrm{td}_D(A))}} \cdot |I|^{\mathcal{O}(1)}$.

We remark that the running time as outlined above also follows from a fine analysis of the reasoning presented in [24], but the intermediate step of using tree-fold ILPs in [24] makes tracking parametric dependencies harder to follow.

We next show that the running time provided by Theorem 2 is probably optimal. Namely, we have the following lower bound.

▶ **Theorem 3.** *Assuming ETH, there is no algorithm that would solve any* ILP Feasibility *instance* $I = \{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$*, where all entries of A and* $\mathbf{b}$ *are in* $\{-1, 0, 1\}$*, in time* $2^{2^{o(\mathrm{td}_D(A))}} \cdot |I|^{\mathcal{O}(1)}$.

To prove Theorem 3 we reduce from the Subset Sum problem. The key idea is that we are able to “encode” any positive integer s using an ILP with dual treedepth $\mathcal{O}(\log \log s)$.

2 Parameterization by the number of constraints

2.1 Detecting matrices

Our main tool is the usage of so-called *detecting matrices*, first studied by Lindström [26]. They can be explained via the following coin-weighing puzzle: given m coins with weights in $\{0,1,\ldots,d-1\}$, we want the deduce the weight of each coin with as few weighings as possible. We have a spring scale, so in one weighing we can exactly determine the sum of weights of any subset of the coins. While the naive strategy – weigh coins one by one – yields m weighings, it is actually possible to find a solution using $\mathcal{O}(m/\log_d m)$ weighings. This number is asymptotically optimal, as each weighing provides $\Theta(\log m)$ bits of information, so fewer weighings would not be enough to distinguish all d^m possible weight functions.

Probably the easiest way to construct such a strategy is using the probabilistic method. It turns out that querying $\mathcal{O}(m/\log_d m)$ random subsets of coins with high probability provides enough information to determine the weight of each coin. This is because a random subset distinguishes any of the $\mathcal{O}(d^m \cdot d^m)$ non-equal pairs of weight functions with probability at least $\frac{1}{2}$, but pairs of weight functions that are close to each other are few, while pairs of weight functions that are far from each other have a significantly better probability than $\frac{1}{2}$ of being distinguished. Note that thus we construct a *non-adaptive* strategy: the subsets of coins to be weighed can be determined and fixed at the very start. We refer the reader to e.g. [12, Corollary 2] for full details, and we remark that the last two authors recently used detecting matrices in the context of algorithmic lower bounds for the MULTICOLORING problem [3].

Viewing each tuple of coin weights as a vector $\mathbf{v} \in \{0,\ldots,d-1\}^m$, each weighing returns the value $\mathbf{a}^\intercal\mathbf{v}$ for the characteristic vector $\mathbf{a} \in \{0,1\}^m$ of some subset of coins. Thus k weighings give the vectors of values $M\mathbf{v}$ for some $\{0,1\}$-matrix M with k rows and m columns. An equivalent formulation is then to ask for a $\{0,1\}$-matrix M with m columns, such that knowing the vector $M\mathbf{v}$ uniquely determines any $\mathbf{v} \in \{0,\ldots,d-1\}^m$. Such an M is called a *d-detecting matrix* and we seek to minimize the number of rows/weighings k it can have. Lindström gave a deterministic construction and proved the bound on k to be tight. See also Bshouty [4] for a more direct and general construction using Fourier analysis.

▶ **Theorem 4** ([26]). *For all $d, m \in \mathbb{N}$, there is a $\{0,1\}$-matrix M with m columns and $k \leqslant \frac{2m\log d}{\log m}(1+o(1))$ rows such that for any $\mathbf{u},\mathbf{v} \in \{0,\ldots,d-1\}^m$, if $M\mathbf{u} = M\mathbf{v}$ then $\mathbf{u} = \mathbf{v}$. Moreover, such matrix M can be constructed in time polynomial in dm.*

In other words, this allows us to check m equalities between values in $\{0,\ldots,d-1\}$ (i.e., corresponding coordinates of vectors $\mathbf{u}$ and $\mathbf{v}$) using only $\mathcal{O}(m/\log_d m)$ comparisons of sums of certain subsets of these values (i.e., coordinates of vectors $M\mathbf{u}$ and $M\mathbf{v}$). For an ILP instance $A\mathbf{x} = \mathbf{b}$ with $\|\mathbf{b}\|_\infty \leqslant d$ and m constraints, we may use this idea to check the equality on each of the m coordinates of $A\mathbf{x}$ using only $\mathcal{O}(m/\log_d m)$ constraints. Indeed, the intuition is that if M is a d-detecting matrix, then we can rewrite $A\mathbf{x} = \mathbf{b}$ as $MA\mathbf{x} = M\mathbf{b}$ and check the latter – which involves $\mathcal{O}(m/\log_d m)$ $\{0,1\}$-combinations of the original constraints.

This is the core of our approach. However, there is one subtle caveat: in order to claim that the assertions $A\mathbf{x} = \mathbf{b}$ and $MA\mathbf{x} = M\mathbf{b}$ are equivalent, we would need to ensure that $\|A\mathbf{x}\|_\infty \leqslant d$ for an arbitrary vector $\mathbf{x} \in \mathbb{N}^n$. One solution is to use the fact that a uniformly random $\{0,1\}$-matrix has a stronger "detecting" property: it will, with high probability, distinguish all vectors of low ℓ_1-norm, as shown by Grebinski and Kucherov [12].

▶ **Lemma 5** ([12]). *For all $d, m \in \mathbb{N}$, there exists a $\{0,1\}$-matrix M with m columns and $k \leqslant \frac{4m\log(d+1)}{\log m}(1+o(1))$ rows such that for any $\mathbf{u}, \mathbf{v} \in \mathbb{N}^m$ satisfying $\|\mathbf{u}\|_1, \|\mathbf{v}\|_1 \leqslant dm$, if $M\mathbf{u} = M\mathbf{v}$ then $\mathbf{u} = \mathbf{v}$. Moreover, such matrix M can be computed in randomized polynomial time (in dm).*

Note that in Lemma 5, we do not actually have to assume bounds on one of the two vectors: it suffices to assume $\mathbf{u} \in \mathbb{N}^m$ and $\|\mathbf{v}\|_1 \leqslant dm$, because simply adding a single row full of ones to M guarantees $\|\mathbf{u}\|_1 = \|\mathbf{v}\|_1$. Therefore as long as A is non-negative and $\|\mathbf{b}\|_\infty \leqslant d$, it suffices to check $MA\mathbf{x} = M\mathbf{b}$. Unfortunately, to the best of our knowledge, no deterministic construction is known for Lemma 5. We remark that Bshouty gave a deterministic, but adaptive detecting strategy [4]; that is, in terms of coin weighing, consecutive queries on coins may depend on results of previous weighings.

Instead, we show that a different, recursive construction by Cantor and Mills [5] for 2-detecting matrices can be adapted so that no bounds (other than non-negativity) are assumed for one of the vectors, while the other must have all coefficients in $\{0, 1, \ldots, d-1\}$. The proof is deferred to the full version [23], which we mark with (♠).

▶ **Lemma 6** (♠). *For all $d, m \in \mathbb{N}$, there exists a $\{0,1\}$-matrix M with m columns and $k \leqslant \frac{md\log d}{\log m}(1+o(1))$ rows such that for any $\mathbf{u} \in \mathbb{N}^m$ and $\mathbf{v} \in \{0, 1, \ldots, d-1\}^m$, if $M\mathbf{u} = M\mathbf{v}$ then $\mathbf{u} = \mathbf{v}$. Moreover, such matrix M can be computed in time polynomial in dm.*

We remark that the bounds in Theorem 4 and Lemma 5 were also shown to be tight. Lemma 6 gives matrices that are also d-detecting, in particular, hence the bound is tight for $d = 2$ (and tight up to an $\mathcal{O}(d)$ factor in general).

Note also that we can relax the non-negativity constraint to requiring that $\mathbf{u} \in \mathbb{Z}^m$ is any integer with all entries lower bounded by $-\lfloor \frac{d}{2} \rfloor$ and $\mathbf{v} \in \{-\lfloor \frac{d}{2} \rfloor, \ldots, \lfloor \frac{d}{2} \rfloor\}^m$. This is because $M\mathbf{u} = M\mathbf{v}$ is equivalent to $M(\mathbf{u}+\mathbf{c}) = M(\mathbf{v}+\mathbf{c})$ where $\mathbf{c}$ is the constant $\lfloor \frac{d}{2} \rfloor$ vector. This allows to use the same detecting matrix for such pairs of vectors as well. However, note that some lower bound on the coefficients of $\mathbf{u}$ is necessary, since even if we fix $\mathbf{v} = 0$, the matrix M has a non-trivial kernel, giving many non-zero vectors $\mathbf{u} \in \mathbb{Z}^m$ satisfying $M\mathbf{u} = M\mathbf{v}$.

2.2 Coefficient reduction

In further constructions, we will need a way to reduce coefficients in a given ILP Feasibility instance with a nonnegative constraint matrix A to $\{0,1\}$. We now prove that this can be done in a standard way by replacing each constraint with $\mathcal{O}(\log \|A\|_\infty)$ constraints that check the original equality bit by bit. Here and throughout this paper we use the convention that for a vector $\mathbf{x}$, by x_i we denote the i-th entry of $\mathbf{x}$.

▶ **Lemma 7** (Coefficient Reduction). *Consider an instance $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ of* ILP Feasibility*, where $\mathbf{b} \in \mathbb{N}^k$ and A is a nonnegative integer matrix with k rows and ℓ columns. In polynomial time, this instance can be reduced to an equivalent instance $\{A'\mathbf{x} = \mathbf{b}', \mathbf{x} \geqslant 0\}$ of* ILP Feasibility *where A' is a $\{0,1\}$-matrix with $k' = \mathcal{O}(k \log \|A\|_\infty)$ rows and $\ell' = \ell + \mathcal{O}(k \log \|A\|_\infty)$ columns, and $\mathbf{b}' \in \mathbb{N}^{k'}$ is a vector with $\|\mathbf{b}'\|_\infty = \mathcal{O}(\|\mathbf{b}\|_\infty)$.*

Proof. Denote $\delta = \lceil \log(1 + \|A\|_\infty) \rceil = \mathcal{O}(\log \|A\|_\infty)$. Consider a single constraint $\mathbf{a}^\intercal \mathbf{x} = b$, where $\mathbf{a} \in \mathbb{N}^\ell$ is a row of A and $b \in \mathbb{N}$ is an entry of $\mathbf{b}$. Let $a_i[j]$ be the j-th bit of a_i, the i-th entry of vector $\mathbf{a}$; similarly for b. By choice of δ, $\|\mathbf{a}\|_\infty \leqslant 2^\delta - 1$, so each entry of $\mathbf{a}$ has up to δ binary digits. Now, for $\mathbf{x} \in \mathbb{Z}^n$, the constraint $\mathbf{a}^\intercal \mathbf{x} = b$ is equivalent to

$$\sum_{j=0}^{\delta-1} 2^j \left(\sum_{i=1}^{n} a_i[j] x_i \right) = b\,.$$

We rewrite this equation into δ equations, each responsible for verifying one bit. For this, we introduce $\delta - 1$ carry variables $y_0, y_1, \ldots, y_{\delta-2}$ and emulate the standard algorithm for adding binary numbers by writing equations

$$y_{j-1} + \sum_{i=1}^{n} a_i[j]x_i = b[j] + 2y_j \qquad \text{for } j = 0, \ldots, \delta - 1,$$

where y_{-1} and $y_{\delta-1}$ are replaced with 0 and $b[\delta-1]$ is replaced with the number whose binary digits are (from the least significant): $b[\delta-1], b[\delta], b[\delta+1], \ldots$ (we do this because b may have more than δ digits). To get rid of the variable y_j on the right-hand side, we let $B = 2^{\lceil \log b \rceil}$ and introduce two new variables y'_j, y''_j for each carry variable y_j, with constraints

$$y_j + y'_j = B \quad \text{and} \quad y_j + y''_j = B \quad \text{for } j = 0, \ldots, \delta - 2,$$

which is equivalent to $y'_j = y''_j = B - y_j$. Hence the previous equations can be replaced by

$$y_{j-1} + \sum_{i=1}^{n} a_i[j]x_i + y'_j + y''_j = b[j] + 2B \qquad \text{for } j = 0, \ldots, \delta - 1.$$

We thus replace each row of A with $2(\delta - 1) + \delta$ rows and $3(\delta - 1)$ auxiliary variables. ◀

2.3 Proof of Theorem 1

The Exponential Time Hypothesis states that for some $0 < c < 1$, 3SAT with n variables cannot be solved in time $\mathcal{O}^\star(2^{cn})$ (the $\mathcal{O}^\star$ notation hides polynomial factors). It was introduced by Impagliazzio, Paturi, and Zane [15] and developed by Impagliazzo and Paturi [14] to become a central conjecture for proving tight lower bounds for the complexity of various problems. While the original statement considers the parameterization by the number of variables, the *Sparsification Lemma* [14] allows us to assume that the number of clauses is linear in the number of variables, and hence we have the following.

▶ **Theorem 8** (see e.g. Theorem 14.4 in [7]). *Unless ETH fails, there is no algorithm for* 3SAT *that runs in time* $2^{o(n+m)}$*, where* n *and* m *denote the numbers of variables and clauses, respectively.*

We now proceed to the proof of Theorem 1. Our first step is to decrease the number of occurrences of each variable. The (3,4)SAT is the variant of 3SAT where each clause uses exactly 3 different variables and every variable occurs in at most 4 clauses. Tovey [30] gave a linear reduction from 3SAT to (3,4)SAT, i.e., an algorithm that, given an instance of 3SAT with n variables and m clauses, in linear time constructs an equivalent instance of (3,4)SAT with $\mathcal{O}(n + m)$ variables and clauses. In combination with Theorem 8 this yields:

▶ **Corollary 9.** *Unless ETH fails, there is no algorithm for* (3,4)SAT *that runs in time* $2^{o(n+m)}$*, where* n *and* m *denote the numbers of variables and clauses, respectively.*

We now reduce (3,4)SAT to ILP FEASIBILITY. A $(3,4)$SAT instance φ with n variables and m clauses can be encoded in a standard way as an ILP FEASIBILITY instance with $\mathcal{O}(n + m)$ variables and constraints as follows. For each formula variable v we introduce two ILP variables x_v and $x_{\neg v}$ with a constraint $x_v + x_{\neg v} = 1$ (hence exactly one of them should be 1, the other 0). For each clause c we introduce two auxiliary slack variables y_c, z_c and two constraints: $y_c + z_c = 2$ and $x_{\ell_1} + x_{\ell_2} + x_{\ell_3} + y_c = 3$, where ℓ_1, ℓ_2, ℓ_3 are the three literals in c. Since y_c, z_c will not appear in any other constraints, the first constraint is equivalent to ensuring that $y_c \leqslant 2$, so the second constraint is equivalent to $x_{\ell_1} + x_{\ell_2} + x_{\ell_3} \geqslant 1$. This way, one can reduce in polynomial time a $(3,4)$SAT instance φ with n variables and m clauses into an equivalent instance $\{\mathbf{x} \in \mathbb{Z}^\ell \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ of ILP FEASIBILITY where:

- the constraint matrix A has $k := n + 2m$ rows and $\ell := 2n + 2m$ columns;
- each entry in A is zero or one;
- each row and column of A contains at most 4 non-zero entries; and
- the target vector $\mathbf{b}$ has all entries equal to 1, 2, or 3;

We now reduce the obtained instance to another ILP FEASIBILITY instance containing only $\mathcal{O}((n + m)/\log(n + m))$ constraints. Let M be the detecting matrix given by Lemma 6 for $d = 4$ and the required number of columns (m in the notation of the statement of Lemma 6) equal to the number or rows (constraints) in A, which is k. Then for any $\mathbf{x} \in \mathbb{N}^\ell$, we have $A\mathbf{x} \in \mathbb{N}^k$ (since A is non-negative) and $\mathbf{b} \in \{0, \ldots, d-1\}^k$, hence by Lemma 6 we have that $A\mathbf{x} = \mathbf{b}$ if and only if $MA\mathbf{x} = M\mathbf{b}$. We conclude that the ILP FEASIBILITY instance $\{\mathbf{x} \in \mathbb{Z}^\ell \mid A'\mathbf{x} = \mathbf{b}', \mathbf{x} \geqslant 0\}$ with $A' = MA$ and $\mathbf{b}' = M\mathbf{b}$ is equivalent to the previous instance $\{\mathbf{x} \in \mathbb{Z}^\ell \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$.

The new instance has the same number $\ell' = \ell = 2n + 2m$ of variables, but only $k' = \mathcal{O}(k/\log k) = \mathcal{O}((n + m)/\log(n + m))$ constraints. The entries of $\mathbf{b}' = M\mathbf{b}$ are non-negative and bounded by $k \cdot \|\mathbf{b}\|_\infty = \mathcal{O}(n + m)$. Similarly, the entries of $A' = MA$ are non-negative, and since every column of A has at most 4 non-zero entries, we get $\|A'\|_\infty \leqslant 4$.

To further reduce $\|A'\|_\infty$, we apply Lemma 7, replacing each row of A' by a constant number of $\{0,1\}$-rows and auxiliary variables. This way, we reduced in polynomial time a $(3,4)$SAT instance φ with n variables and m clauses into an equivalent ILP FEASIBILITY instance $\{\mathbf{x} \in \mathbb{Z}^{\ell''} \mid A''\mathbf{x} = \mathbf{b}'', \mathbf{x} \geqslant 0\}$, where A'' is a $\{0,1\}$-matrix with $\ell'' = \ell' + \mathcal{O}(k') = \mathcal{O}(n+m)$ columns and $k'' = \Theta(k') = \Theta((n+m)/\log(n+m))$ rows, while $\|\mathbf{b}''\|_\infty = \mathcal{O}(n+m)$. Hence $\ell'', \|\mathbf{b}''\|_\infty = \mathcal{O}(k'' \log k'')$.

We are now in position to finish the proof of Theorem 1. Suppose there is an algorithm for ILP FEASIBILITY that works in time $2^{o(k'' \log k'')}$ on instances with $A \in \{0,1\}^{k'' \times \ell''}$ and $\ell'', \|\mathbf{b}''\|_\infty = \mathcal{O}(k'' \log k'')$. Then applying the above reduction would solve (3,4)SAT instances with $N = n + m$ variables and clauses in time $2^{o((N/\log N) \cdot \log(N/\log N))} = 2^{o(N)}$, which contradicts ETH by Corollary 9. This concludes the proof of Theorem 1.

3 Parameterization by the dual treedepth

3.1 Preliminaries

Treedepth and dual treedepth. For a graph G, the *treedepth* of G, denoted $\mathrm{td}(G)$, can be defined recursively as follows:

$$\mathrm{td}(G) = \begin{cases} 1 & \text{if } G \text{ has one vertex;} \\ \max(\mathrm{td}(G_1), \ldots, \mathrm{td}(G_p)) & \text{if } G \text{ is disconnected and } G_1, \ldots, G_p \\ & \text{are its connected components;} \\ 1 + \min_{u \in V(G)} \mathrm{td}(G - u) & \text{if } G \text{ has more than one vertex} \\ & \text{and is connected.} \end{cases} \tag{1}$$

See e.g. [28]. Equivalently, treedepth is the smallest possible height of a rooted forest F on the same vertex set as G such that whenever uv is an edge in G, then u is an ancestor of v in F or vice versa.

Since we focus on constraints, we consider, for a matrix A, the *constraint graph* or *dual graph* $G_D(A)$, defined as the graph with rows of A as vertices where two rows are adjacent if and only if in some column they simultaneously contain a non-zero entry. The *dual treedepth* of A, denoted $\mathrm{td}_D(A)$, is the treedepth of $G_D(A)$.

The recursive definition (1) is elegantly reinterpreted in terms of row removals and partitioning into blocks as follows. A matrix A is *block-decomposable* if after permuting its rows and columns it can be presented in block-diagonal form, i.e., rows and columns can be partitioned into intervals $R_1, \ldots, R_p$ and $C_1, \ldots, C_p$, for some $p \geqslant 2$, such that non-zero entries appear only in blocks $B_1, \ldots, B_p$, where B_i is the block of entries at intersections of rows from R_i with columns from C_i. It is easy to see that A is block-decomposable if and only if $G_D(A)$ is disconnected, and the finest block decomposition of A corresponds to the partition of $G_D(A)$ into connected components. The blocks $B_1, \ldots, B_p$ in this finest partition are called the *block components* of A – they are not block-decomposable. Then the recursive definition of treedepth provided in (1) translates to the following definition of the dual treedepth of a matrix A:

$$\mathrm{td}_D(A) = \begin{cases} 1 & \text{if } A \text{ has one row;} \\ \max(\mathrm{td}_D(B_1), \ldots, \mathrm{td}_D(B_p)) & \text{if } A \text{ is block-decomposable and} \\ & B_1, \ldots, B_p \text{ are its block components;} \\ 1 + \min\limits_{\mathbf{a}^\intercal \,:\, \text{rows of } A} \mathrm{td}_D(A \backslash \mathbf{a}^\intercal) & \text{if } A \text{ has more than one row and} \\ & \text{is not block decomposable.} \end{cases} \tag{2}$$

Here $A\backslash\mathbf{a}^\intercal$ is the matrix obtained from A by removing the row $\mathbf{a}^\intercal$. Intuitively, dual treedepth formalizes the idea that a block-decomposable matrix is as hard as the hardest of its block components, and that adding a single row makes it a bit harder, but not uncontrollably so.

Graver bases. Two integer vectors $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^n$ are *sign-compatible* if $a_i \cdot b_i \geqslant 0$ for all $i = 1, \ldots, n$. For $\mathbf{a}, \mathbf{b} \in \mathbb{Z}^n$ we write $\mathbf{a} \sqsubseteq \mathbf{b}$ if $\mathbf{a}$ and $\mathbf{b}$ are sign-compatible and $|a_i| \leqslant |b_i|$ for all $i = 1, \ldots, n$. Then $\sqsubseteq$ is a partial order on $\mathbb{Z}^n$; we call it the *conformal order*. Note that $\sqsubseteq$ has a unique minimum element, which is the zero vector $\mathbf{0}$.

For a matrix A, the *Graver basis* of A, denoted $\mathcal{G}(A)$ is the set of conformally minimal vectors in $(\ker A \cap \mathbb{Z}^n) - \{\mathbf{0}\}$. It is easy to see by Dickson's lemma that $(\mathbb{Z}^n, \sqsubseteq)$ is a well quasi-ordering, hence there are no infinite antichains with respect to the conformal order. It follows that the Graver basis of every matrix is finite, though it can be quite large. For a matrix A and $p \in [1, \infty]$, we denote $g_p(A) = \max_{\mathbf{u} \in \mathcal{G}(A)} \|\mathbf{u}\|_p$.

3.2 Upper bound

We start with the upper bound for the dual treedepth parameterization, that is, Theorem 2. As explained in the introduction, this result easily follows from the work of Koutecký et al. [24] and the following lemma bounding $g_1(A)$ in terms of $\mathrm{td}_D(A)$ and $\|A\|_\infty$, for any integer matrix A.

▶ **Lemma 10.** *For any matrix A with integer entries, it holds that*

$$g_1(A) \leqslant (2\|A\|_\infty + 1)^{2^{\mathrm{td}_D(A)}-1}.$$

Before we prove Lemma 10, let us sketch how using the reasoning from Koutecký et al. [24] one can derive Theorem 2. Using the bound on the ℓ_1-norm of vectors in the Graver basis of A, we can construct a Λ-*Graver-best oracle* for the considered ILP OPTIMIZATION instance. This is an oracle that given any feasible solution $\mathbf{x}$, returns another feasible solution $\mathbf{x}'$ that differs from $\mathbf{x}$ only by an integer multiple not larger than Λ of a vector from the Graver basis of A, and among such solution achieves the best goal value of $\mathbf{w}^\intercal\mathbf{x}'$. Such a Λ-Graver-best oracle runs in time $(\|A\|_\infty \cdot g_1(A))^{\mathcal{O}(\mathrm{tw}_D(A))} \cdot |I|^{\mathcal{O}(1)}$, where $\mathrm{tw}_D(A)$ is the treewidth of the constraint graph $G_D(A)$, which is always upper bounded by $\mathrm{td}_D(A)+1$. See the proof of Lemma 25 and the beginning of the proof of Theorem 3 in [24]; the reasoning there is explained in the context of tree-fold ILPs, but it uses only boundedness of the dual treedepth of A. Once a Λ-Graver-best oracle is implemented, we can use it to implement a *Graver-best oracle* (Lemma 14 in [24]) within the same asymptotic running time, and finally use the main theorem – Theorem 1 in [24] – to obtain the algorithm promised in Theorem 2 above.

We now proceed to the proof of Lemma 10.

Proof of Lemma 10. We proceed by induction on the number of rows of A using the recursive definition (2). For the base case – when A has one row – we may use the following well-known bound.

▷ Claim 11 (Lemma 3.5.7 in [27]). If A is an integer matrix with one row, then

$$g_1(A) \leqslant 2\|A\|_\infty + 1.$$

We note that the original bound of $2\|A\|_\infty - 1$, stated in [27], works only for non-zero A.

We now move to the induction step, so suppose the considered matrix A has more than one row. We consider two cases: either A is block-decomposable, or it is not.

First suppose that A is block-decomposable. Let $B_1, \ldots, B_p$ be the block components of A, and let $R_1, \ldots, R_p$ and $C_1, \ldots, C_p$ be the corresponding partitions of rows and columns of A into segments, respectively. Observe that integer vectors $\mathbf{u}$ from $\ker A$ are exactly vectors of the form $(\,\mathbf{v}^{(1)} \mid \mathbf{v}^{(2)} \mid \ldots \mid \mathbf{v}^{(p)}\,)$, where each $\mathbf{v}^{(i)}$ is an integer vector of length $|C_i|$ that belongs to $\ker B_i$. It follows that $\mathcal{G}(A)$ consists of vectors of the following form: for some $i \in \{1, \ldots, p\}$ put a vector from $\mathcal{G}(B_i)$ on coordinates corresponding to the columns of C_i, and fill all the other entries with zeroes. Consequently, we have

$$g_1(A) \leqslant \max_{i=1,\ldots,p} g_1(B_i). \tag{3}$$

On the other hand, by (2) we have

$$\mathrm{td}_D(A) = \max_{i=1,\ldots,p} \mathrm{td}_D(B_i). \tag{4}$$

Since each matrix B_i has fewer rows than A, we may apply the induction assumption to matrices $B_1, \ldots, B_p$, thus inferring by (3) and (4) that

$$g_1(A) \leqslant \max_{i=1,\ldots,p} g_1(B_i) \leqslant \max_{i=1,\ldots,p} (2\|B_i\|_\infty + 1)^{2^{\mathrm{td}_D(B_i)}-1} \leqslant (2\|A\|_\infty + 1)^{2^{\mathrm{td}_D(A)}-1}.$$

We are left with the case when A is not block-decomposable. For this, we use the following claim, which is essentially Lemma 3.7.6 and Corollary 3.7.7 in [27]. The statement there is slightly different, but the same proof in fact proves the following bound; for convenience, we repeat the argument in the full version.

▷ Claim 12 (♠). Let A be an integer matrix and let $\mathbf{a}^\intercal$ be a row of A. Then

$$g_1(A) \leqslant (2\|\mathbf{a}^\intercal\|_\infty + 1) \cdot g_1(A \backslash \mathbf{a}^\intercal) \cdot g_\infty(A \backslash \mathbf{a}^\intercal).$$

Suppose then that A is not block-decomposable. By (2), there exists a row $\mathbf{a}^\intercal$ of A such that $\mathrm{td}_D(A\backslash\mathbf{a}^\intercal) = \mathrm{td}_D(A) - 1$. Then, by Claim 12 and the inductive assumption, we have

$$\begin{aligned} g_1(A) &\leqslant (2\|\mathbf{a}^\intercal\|_\infty + 1) \cdot g_1(A\backslash\mathbf{a}^\intercal) \cdot g_\infty(A\backslash\mathbf{a}^\intercal) \leqslant (2\|A\|_\infty + 1) \cdot (g_1(A\backslash\mathbf{a}^\intercal))^2 \\ &\leqslant (2\|A\|_\infty + 1)^{1+2\cdot(2^{\mathrm{td}_D(A)-1}-1)} = (2\|A\|_\infty + 1)^{2^{\mathrm{td}_D(A)}-1}. \end{aligned}$$

This concludes the proof. ◀

3.3 Lower bound

We now move to the proof of the lower bound, Theorem 3. We will reduce from the Subset Sum problem: given non-negative integers $s_1, \ldots, s_k, t$, encoded in binary, decide whether there is a subset of numbers $s_1, \ldots, s_k$ that sums up to t. The standard NP-hardness reduction from 3SAT to Subset Sum takes an instance of 3SAT with n variables and m clauses, and produces an instance $(s_1, \ldots, s_k, t)$ of Subset Sum with a linear number of numbers and each of them of linear bit-length, that is, $k \leqslant \mathcal{O}(n + m)$ and $0 \leqslant s_1, \ldots, s_k, t < 2^\delta$, for some $\delta \leqslant \mathcal{O}(n+m)$. See e.g. [1] for an even finer reduction, yielding lower bounds for Subset Sum under Strong ETH. By Theorem 8, this immediately implies an ETH-based lower bound for Subset Sum.

▶ **Lemma 13.** *Unless ETH fails, there is no algorithm for* Subset Sum *that would solve any input instance $(s_1, \ldots, s_k, t)$ in time $2^{o(k+\delta)}$, where δ is the smallest integer such that $s_1, \ldots, s_k, t < 2^\delta$.*

The idea for our reduction from Subset Sum to ILP Feasibility is as follows. Given an instance $(s_1, \ldots, s_k, t)$, we first *construct* numbers $s_1, \ldots, s_k$ using ILPs $P_1, \ldots, P_k$, where each P_i uses only constant-size coefficients and has dual treedepth $\mathcal{O}(\log \delta)$. The ILP P_i will have a designated variable z_i and two feasible solutions: one that sets z_i to 0 and one that sets it to s_i. Similarly we can construct an ILP Q that forces a designated variable w to be set to t. Having that, the whole input instance can be encoded using one additional constraint: $z_1 + \ldots + z_k - w = 0$. To construct each P_i, we first create δ variables $y_0, y_1, \ldots, y_{\delta-1}$ that are either all evaluated to 0 or all evaluated to $2^0, 2^1, \ldots, 2^{\delta-1}$, respectively; this involves constraints of the form $y_{j+1} = 2y_j$. Then the number s_i (or 0) can be obtained on a new variable z_i using a single constraint that assembles the binary encoding of s_i. The crucial observations is that the constraint graph $G_D(P_i)$ consists of a path on δ vertices and one additional vertex, and thus has treedepth $\mathcal{O}(\log \delta)$.

We start implementing this plan formally by giving the construction for a single number s.

▶ **Lemma 14.** *For all positive integers δ and s satisfying $0 \leqslant s < 2^\delta$, there exists an instance $P = \{A\mathbf{x} = \mathbf{b},\ \mathbf{x} \geqslant 0\}$ of* ILP Feasibility *with the following properties:*

- *A has all entries in $\{-1, 0, 1, 2\}$ and $\mathrm{td}_D(A) \leqslant \mathcal{O}(\log \delta)$;*
- *$\mathbf{b}$ is a vector with all entries in $\{0, 1\}$; and*
- *P has exactly two solutions $\mathbf{x}^{(1)}$ and $\mathbf{x}^{(2)}$, where $x_1^{(1)} = 0$ and $x_1^{(2)} = s$.*

Moreover, the instance P can be constructed in time polynomial in $\delta + \log s$.

Proof. We shall use $n + 2$ variables, denoted for convenience by $y_0, y_1, \ldots, y_{\delta-1}, z, u$; these are arranged into the variable vector $\mathbf{x}$ of length $\delta + 2$ so that $x_1 = z$. Letting $b_0, b_1, \ldots, b_{\delta-1}$ be the consecutive digits of the number s in the binary encoding, the instance P then looks as follows:

$$\begin{array}{ccccccccccccccc} u & + & y_0 & & & & & & & & & & & = & 1 \\ & & 2y_0 & - & y_1 & & & & & & & & & = & 0 \\ & & & & 2y_1 & - & y_2 & & & & & & & = & 0 \\ & & & & & & \ddots & & \ddots & & & & & & \vdots \\ & & & & & & & & 2y_{\delta-2} & - & y_{\delta-1} & & & = & 0 \\ & & b_0y_0 & + & b_1y_1 & + & \dots & + & b_{\delta-2}y_{\delta-2} & + & b_{\delta-1}y_{\delta-1} & - & z & = & 0 \end{array}$$

Since $0 \leqslant u \leqslant 1$, it is easy to see that P has exactly two solutions in nonnegative integers:

- If one sets $u = 1$, then all the other variables need to be set to 0.
- If one sets $u = 0$, then y_i needs to be set to 2^i for all $i = 0, 1, \dots, \delta - 1$, and then z needs to be set to s by the last equation.

It remains to analyze the dual treedepth of A. Observe that the constraint graph $G_D(A)$ consists of a path of length δ, plus one vertex corresponding to the last equation that may have an arbitrary neighborhood within the path. Since the path on δ vertices has treedepth $\lceil \log(\delta + 1)\rceil$, it follows that $G_D(A)$ has treedepth at most $1 + \lceil \log(\delta + 1)\rceil \leqslant \mathcal{O}(\log \delta)$. ◀

We note that in the above construction one may remove the variable u and replace the constraint $u + y_0 = 1$ with $y_0 = 1$, thus forcing only one solution: the one that sets the first variable to s. This will be used later.

We are ready to show the core part of the reduction.

▶ **Lemma 15.** *An instance $(s_1, \dots, s_k, t)$ of* Subset Sum *with $0 \leqslant s_i, t < 2^\delta$ for $i = 1, \dots, k$, can be reduced in polynomial time to an equivalent instance $\{A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$ of* ILP Feasibility *where the entries of A are in $\{-1, 0, 1, 2\}$, entries of $\mathbf{b}$ are in $\{0, 1\}$, and $\mathrm{td}_D(A) \leqslant \mathcal{O}(\log \delta)$.*

Proof. For each $i \in \{1, \dots, k\}$, apply Lemma 14 to construct a suitable instance $P_i = \{A_i\mathbf{x} = \mathbf{b}_i, \mathbf{x} \geqslant 0\}$ of ILP Feasibility for $s = s_i$. Also, apply Lemma 14 to construct a suitable instance $Q = \{C\mathbf{x} = \mathbf{d}, \mathbf{x} \geqslant 0\}$ of ILP Feasibility for $s = t$, and modify it as explained after the lemma's proof so that there is only one solution, setting the first variable to t. Let

$$A = \begin{pmatrix} & & \mathbf{c}^\intercal & & \\ \hline A_1 & & & & \\ & A_2 & & & \\ & & \ddots & & \\ & & & A_k & \\ & & & & C \end{pmatrix}$$

where

$$\mathbf{c}^\intercal = (\,1\,0\,\dots\,0 \mid 1\,0\,\dots\,0 \mid \,\dots\, \mid 1\,0\,\dots\,0 \mid (-1)\,0\,\dots\,0\,)$$

with consecutive blocks of lengths equal to the numbers of columns of $A_1, \dots, A_k$, and C, respectively. Observe that

$$\mathrm{td}_D(A) \leqslant 1 + \max(\mathrm{td}_D(A_1), \dots, \mathrm{td}_D(A_k), \mathrm{td}_D(C)) = \mathcal{O}(\log \delta).$$

Further, let

$$\mathbf{b}^\intercal = (\,0 \mid \mathbf{b}_1^\intercal \mid \,\dots\, \mid \mathbf{b}_k^\intercal \mid \mathbf{d}^\intercal\,).$$

We now claim that the ILP $\{A\mathbf{x} = \mathbf{b},\, \mathbf{x} \geqslant 0\}$ is feasible if and only if the input instance of Subset Sum has a solution. Indeed, if we denote by $z_1, \ldots, z_k, w$ the variables corresponding to the first columns of blocks $A_1, \ldots, A_k, C$, respectively, then by Lemma 14 within each block A_i there are two ways of evaluating variables corresponding to columns of A_i: one setting $z_i = 0$ and second setting $z_i = s_i$. However, there is only one way of evaluating the variables corresponding to columns of C, which sets $w = t$. The first row of A then constitutes the constraint $z_1 + \ldots + z_k - w = 0$, which can be satisfied by setting z_i-s and w as above if and only if some subset of the numbers $s_1, \ldots, s_k$ sums up to t. ◀

It remains to reduce the entries in A equal to 2, simply by duplicating variables.

▶ **Lemma 16** (♠)**.** *An instance $\{A\mathbf{x} = \mathbf{b},\, \mathbf{x} \geqslant 0\}$ of* ILP Feasibility *where the entries of A are in $\{-1, 0, 1, 2\}$ and the entries of $\mathbf{b}$ are in $\{0, 1\}$ can be reduced in polynomial time to an equivalent instance $\{A'\mathbf{x} = \mathbf{b}',\, \mathbf{x} \geqslant 0\}$ of* ILP Feasibility *with all entries in $\{-1, 0, 1\}$ and $\mathrm{td}_D(A') \leqslant \mathrm{td}_D(A) + 1$.*

Theorem 3 now follows by observing that combining the reductions of Lemma 15 and Lemma 16 with a hypothetical algorithm for ILP Feasibility on $\{-1, 0, 1\}$-input with running time $2^{2^{o(\mathrm{td}_D(A))}} \cdot |I|^{\mathcal{O}(1)}$ would yield an algorithm for Subset Sum with running time $2^{o(k+\delta)}$, contradicting ETH by Lemma 13.

4 Conclusions

We conclude this work by stating two concrete open problems in the topic.

First, apart from considering the standard form $\{A\mathbf{x} = \mathbf{b},\, \mathbf{x} \geqslant 0\}$, Eisenbrand and Weismantel [9] also studied the more general setting of ILPs of the form $\{A\mathbf{x} = \mathbf{b},\, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}\}$, where $\mathbf{l}$ and $\mathbf{u}$ are integer vectors. That is, instead of only requiring that every variable is nonnegative, we put an arbitrary lower and upper bound on the values it can take. Note that such lower and upper bounds can be easily emulated in the standard formulation using slack variables, but this would require adding more constraints to the matrix A; the key here is that we *do not* count these lower and upper bounds in the total number of constraints k. For this more general setting, Eisenbrand and Weismantel [9] gave an algorithm with running time $k^{\mathcal{O}(k^2)} \cdot \|A\|_\infty^{\mathcal{O}(k^2)} \cdot |I|^{\mathcal{O}(1)}$, which boils down to $2^{\mathcal{O}(k^2 \log k)} \cdot |I|^{\mathcal{O}(1)}$ when $\|A\|_\infty = \mathcal{O}(1)$. (A typo leading to a $2^{\mathcal{O}(k^2)} \cdot |I|^{\mathcal{O}(1)}$ bound has been fixed in later versions of the paper). Is this running time optimal or could the $2^{\mathcal{O}(k^2 \log k)}$ factor be improved?

Second, in this work we studied the parameter dual treedepth of the constraint matrix A, but of course one can also consider the *primal treedepth*. It can be defined as the treedepth of the graph over the columns (variables) of A, where two columns are adjacent if they have a non-zero entry in same row (the variables appear simultaneously in some constraint). It is known that ILP Feasibility and ILP Optimization are fixed-parameter tractable when parameterized by $\|A\|_\infty$ and $\mathrm{td}_P(A)$, that this, there is an algorithm with running time $f(\|A\|_\infty, \mathrm{td}_P(A)) \cdot |I|^{\mathcal{O}(1)}$, for some function f [24]. Again, the key ingredient here is an inequality on ℓ_∞-norms of the elements of the Graver basis of any integer matrix A: $g_\infty(A) \leqslant h(\|A\|_\infty, \mathrm{td}_P(A))$ for some function h. Unfortunately, the known proofs of this fact[1], see [2][2], use the theory of well quasi-orderings (in a highly non-trivial way) and consequently

[1] Very recently, Klein [19] provided an alternative constructive proof for 2-stage and multistage IPs.
[2] The work of Aschenbrenner and Hemmecke [2] considers the setting of *multi-stage stochastic programming*, which is related to primal treedepth in the same way as tree-fold ILPs are related to dual treedepth. The translation between MSSP and primal treedepth was formulated by Koutecký et al. [24].

give no direct bounds on the function h. A good upper bound on h would directly lead to a correspondingly efficient FPT algorithm for ILP OPTIMIZATION parameterized by $\|A\|_\infty$ and $\mathrm{td}_P(A)$. However, we conjecture that the function h has to be non-elementary in $\mathrm{td}_P(A)$. If this was the case, an example could likely be used to prove a non-elementary lower bound under ETH for ILP FEASIBILITY under that $\mathrm{td}_P(A)$ parameterization (with $\|A\| = \mathcal{O}(1)$).

References

1. Amir Abboud, Karl Bringmann, Danny Hermelin, and Dvir Shabtay. SETH-based lower bounds for Subset Sum and Bicriteria Path. In *SODA 2019*, pages 41–57. SIAM, 2019.
2. Matthias Aschenbrenner and Raymond Hemmecke. Finiteness Theorems in Stochastic Integer Programming. *Foundations of Computational Mathematics*, 7(2):183–227, 2007.
3. Marthe Bonamy, Łukasz Kowalik, Michał Pilipczuk, Arkadiusz Socała, and Marcin Wrochna. Tight Lower Bounds for the Complexity of Multicoloring. In *ESA 2017*, volume 87 of *LIPIcs*, pages 18:1–18:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017.
4. Nader H. Bshouty. Optimal Algorithms for the Coin Weighing Problem with a Spring Scale. In *COLT 2009*, 2009.
5. David G Cantor and WH Mills. Determination of a subset from certain combinatorial properties. *Canad. J. Math*, 18:42–48, 1966.
6. Lin Chen and Dániel Marx. Covering a tree with rooted subtrees — parameterized and approximation algorithms. In *SODA 2018*, pages 2801–2820. SIAM, 2018.
7. Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.
8. Friedrich Eisenbrand, Christoph Hunkenschröder, and Kim-Manuel Klein. Faster Algorithms for Integer Programs with Block Structure. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 49:1–49:13. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018.
9. Friedrich Eisenbrand and Robert Weismantel. Proximity results and faster algorithms for Integer Programming using the Steinitz Lemma. In *SODA 2018*, pages 808–816. SIAM, 2018.
10. Fedor V. Fomin, Fahad Panolan, M. S. Ramanujan, and Saket Saurabh. On the Optimality of Pseudo-polynomial Algorithms for Integer Programming. In *ESA 2018*, volume 112 of *LIPIcs*, pages 31:1–31:13. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018.
11. András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7(1):49–65, 1987.
12. Vladimir Grebinski and Gregory Kucherov. Optimal Reconstruction of Graphs under the Additive Model. *Algorithmica*, 28(1):104–124, 2000.
13. Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n-Fold integer programming in cubic time. *Math. Program.*, 137(1-2):325–341, 2013.
14. Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *J. Comput. Syst. Sci.*, 62(2):367–375, 2001.
15. Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.
16. Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the Configuration-IP — New PTAS Results for Scheduling with Setups Times. In *ITCS 2019*, volume 124 of *LIPIcs*, pages 44:1–44:19. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2019.
17. Klaus Jansen and Lars Rohwedder. On Integer Programming and Convolution. In *ITCS 2019*, volume 124 of *LIPIcs*, pages 43:1–43:17. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2019.
18. Ravi Kannan. Minkowski's Convex Body Theorem and Integer Programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.
19. Kim-Manuel Klein. About the Complexity of Two-Stage Stochastic IPs. *CoRR*, abs/1901.01135, 2019.

20 Dušan Knop and Martin Koutecký. Scheduling meets n-fold integer programming. *J. Scheduling*, 21(5):493–503, 2018.

21 Dušan Knop, Martin Koutecký, and Matthias Mnich. Combinatorial n-fold Integer Programming and Applications. In *ESA 2017*, volume 87 of *LIPIcs*, pages 54:1–54:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017.

22 Dušan Knop, Martin Koutecký, and Matthias Mnich. Voting and Bribing in Single-Exponential Time. In *STACS 2017*, volume 66 of *LIPIcs*, pages 46:1–46:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2017.

23 Dušan Knop, Michał Pilipczuk, and Marcin Wrochna. Tight complexity lower bounds for integer linear programming with few constraints. *CoRR*, abs/1811.01296, 2018. URL: `http://arxiv.org/abs/1811.01296`.

24 Martin Koutecký, Asaf Levin, and Shmuel Onn. A Parameterized Strongly Polynomial Algorithm for Block Structured Integer Programs. In *ICALP 2018*, volume 107 of *LIPIcs*, pages 85:1–85:14. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2018.

25 Hendrik W. Lenstra. Integer Programming with a Fixed Number of Variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

26 Bernt Lindström. On a combinatorial problem in number theory. *Canad. Math. Bull*, 8(4):477–490, 1965.

27 Jesús A. De Loera, Raymond Hemmecke, and Matthias Köppe. *Algebraic and Geometric Ideas in the Theory of Discrete Optimization*, volume 14 of *MOS-SIAM Series on Optimization*. SIAM, 2013.

28 Jaroslav Nešetřil and Patrice Ossona de Mendez. *Sparsity — Graphs, Structures, and Algorithms*, volume 28 of *Algorithms and combinatorics*. Springer, 2012.

29 Christos H. Papadimitriou. On the complexity of integer programming. *J. ACM*, 28(4):765–768, 1981.

30 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.