Artificial Intelligence 257 (2018) 61–71

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

The complexity landscape of decompositional parameters for ILP

Robert Ganian *, Sebastian Ordyniak

Algorithms and Complexity Group, TU Wien, Favoritenstrasse 9-11, 1040 Wien, Austria

ARTICLE INFO

Article history:
Received 21 November 2016
Received in revised form 19 December 2017
Accepted 31 December 2017
Available online 8 January 2018

Keywords:
Integer Linear Programming
Treewidth
Treedepth
(Parameterized) complexity

ABSTRACT

Integer Linear Programming (ILP) can be seen as the archetypical problem for NP-complete optimization problems, and a wide range of problems in artificial intelligence are solved in practice via a translation to ILP. Despite its huge range of applications, only few tractable fragments of ILP are known, probably the most prominent of which is based on the notion of total unimodularity. Using entirely different techniques, we identify new tractable fragments of ILP by studying structural parameterizations of the constraint matrix within the framework of parameterized complexity.

In particular, we show that ILP is fixed-parameter tractable when parameterized by the treedepth of the constraint matrix and the maximum absolute value of any coefficient occurring in the ILP instance. Together with matching hardness results for the more general parameter treewidth, we give an overview of the complexity of ILP w.r.t. decompositional parameters defined on the constraint matrix.

1. Introduction

Integer Linear Programming (ILP) is among the most successful and general paradigms for solving computationally intractable optimization problems in computer science. In particular, a wide variety of problems in artificial intelligence are efficiently solved in practice via a translation into an Integer Linear Program, including problems from areas such as process scheduling [10], planning [31,32], vehicle routing [30], packing [23], and network hub location [1]. In its most general form ILP can be formalized as follows:

Integer Linear Program	
Input:	A matrix $A \in \mathbb{Z}^{m\times n}$ and two vectors $b \in \mathbb{Z}^m$ and $s \in \mathbb{Z}^n$.
Question:	Maximize $s^T x$ for every $x \in \mathbb{Z}^n$ with $Ax \le b$.

Closely related to ILP is the ILP-feasibility problem, where given A and b as above, the problem is to decide whether there is an $x \in \mathbb{Z}^n$ such that $Ax \le b$. The decision version of ILP, ILP-feasibility and various other highly restricted variants are well-known to be NP-complete [28].

* Corresponding author.
E-mail addresses: rganian@gmail.com (R. Ganian), sordyniak@gmail.com (S. Ordyniak).

https://doi.org/10.1016/j.artint.2017.12.006
0004-3702/© 2018 Published by Elsevier B.V.

Table 1
The complexity landscape of ILP obtained in this paper. The table shows the parameterized complexity of ILP parameterized by the treedepth (TD), treewidth (TW), or cliquewidth (CW) of the primal graph with (second column "ℓ") and without (third column "without ℓ") the additional parameterization by the maximum absolute value ℓ of any coefficient in A or b.

	ℓ	without ℓ
TD	FPT (Theorem 6)	paraNP-h (Theorem 12)
TW/CW	paraNP-h (Theorem 13)	paraNP-h (Theorem 13)
None	paraNP-h (Observation 1)	n.a.

Despite the importance of the problem, an understanding of the influence of structural restrictions on the complexity of ILP is still in its infancy. This is in stark contrast to another well-known and general paradigm for the solution of problems in Computer Science, the Satisfiability problem (SAT). There, the parameterized complexity framework [7] has yielded deep results capturing the tractability and intractability of SAT with respect to a plethora of structural restrictions. In the context of SAT, one often considers structural restrictions on a graphical representation of the formula (such as the primal graph), and the aim is to design efficient fixed-parameter algorithms for SAT, i.e., algorithms running in time $\mathcal{O}(f(k)n^{\mathcal{O}(1)})$ where k is the value of the considered structural parameter for the given SAT instance and n is its input size. It is known that SAT is fixed-parameter tractable w.r.t. a variety of structural parameters, including the prominent parameters treewidth [29] but also more specialized parameters [9,13,14].

Our contribution In this work, we carry out a similar line of research for ILP by studying the parameterized complexity of ILP w.r.t. various structural parameterizations. In particular, we consider parameterizations of the *primal graph* of the ILP instance, i.e., the undirected graph whose vertex set is the set of variables of the ILP instance and whose edges represent the occurrence of two variables in a common expression. We obtain a complete picture of the parameterized complexity of ILP w.r.t. well-known decompositional parameters of the primal graph, specifically treedepth, treewidth, and cliquewidth; our results are summarized in Table 1.

Our main algorithmic result (Theorem 6) shows that ILP is fixed-parameter tractable parameterized by the *treedepth* of the primal graph and the maximum absolute value ℓ of any coefficient occurring in A or b. Together with the classical results for *totally unimodular* matrices [27, Section 13.2.] and fixed number of variables [22], which use entirely different techniques, our result is one of the few tractability results for ILP without additional restrictions. We note that the presented algorithm is primarily of theoretical interest; the intent here is to classify the complexity of ILP by providing runtime guarantees, not to compete with state-of-the-art ILP solvers.

We complement our algorithmic results with matching lower bounds, provided in terms of paraNP-hardness results (see the Preliminaries); an overview of the obtained results is provided in Table 1. Namely, we show that already ILP-FEASIBILITY is unlikely to be fixed-parameter tractable when parameterized by treedepth (whereas the case of parameterizing by only ℓ is known to be hard); in fact, our results also exclude algorithms running in time $(n+m)^{f(k)}$, where k is the parameter. Moreover, the hardness results provided here also hold in the strong sense, i.e., even for ILP instances whose size is bounded by a polynomial of n and m; it is worth noting that this requires a more careful approach than what would suffice for weak paraNP-hardness.

One might be tempted to think that, as is the case for SAT and numerous other problems, the fixed-parameter tractability result for treedepth carries over to the more general structural parameter treewidth. We show that this is not the case for ILP. Along with recent results for the Mixed Chinese Postman Problem [18], this is only the second known case of a natural problem where using treedepth instead of treewidth actually "helps" in terms of fixed parameter tractability. In fact, we show that already ILP-FEASIBILITY remains NP-hard for ILP instances of treewidth at most two and whose maximum coefficient is at most one. Observe that this also implies the same intractability results for the more general parameter clique-width [2].

Related work We are not the first to consider decompositional parameterizations of the primal graph for ILP. However, previous results in this area required either implicit or explicit bounds on the domain values of variables together with further restrictions on the coefficients. In particular, for the case of non-negative ILP instances, i.e., ILP instances where all coefficients as well as all variable domains are assumed to be non-negative, ILP is known to be fixed-parameter tractable parameterized by the branchwidth, a decompositional parameter closely related to treewidth, of the primal graph and the maximum value B of any coefficient in the constraint vector b [3]. Note that B also bounds the maximum domain value of any variable in the case of non-negative ILP instances. A more recent result by Jansen and Kratsch [20] showed that ILP is fixed-parameter tractable parameterized by the treewidth of the primal graph and the maximum absolute domain value of any variable. Hence in both cases the maximum absolute domain value of any variable is bounded by the considered parameters, whereas the results presented in this paper do not require any bound on the domain values of variables.

Furthermore, a series of tractability results for ILP based on restrictions on the constraint matrix A, instead of restrictions on the primal graph, have been obtained [5,19,26]. These results apply whenever the constraint matrix A can be written as an arbitrary large product of matrices of bounded size and are usually referred to as n-fold ILP, two-stage stochastic ILP, and 4-block n-fold ILP.

2. Preliminaries

We will use standard graph terminology, see for instance [6]. A graph G is a tuple (V, E), where V or $V(G)$ is the vertex set and E or $E(G)$ is the edge set. A graph H is a subgraph of a graph G, denoted $H \subseteq G$, if H can be obtained by deleting vertices and edges from G. All our graphs are simple and loopless.

A *path* from vertex v_1 to vertex v_j in G is a sequence of distinct vertices $v_1, \dots, v_j$ such that for each $1 \leq i < j$, $\{v_i, v_{i+1}\} \in E(G)$. A *tree* is a graph in which, for any two vertices $v, w \in G$, there is precisely one unique path from v to w; a tree is *rooted* if it contains a specially designated vertex r, the *root*. Given a vertex v in a tree G with root r, the *parent* of v is the unique vertex w with the property that $\{v, w\}$ is the first edge on the path from v to r.

2.1. Integer Linear Programming

For our purposes, it will be useful to view an ILP instance as a set of linear inequalities rather than using the constraint matrix. Formally, let an ILP instance I be a tuple $(\mathcal{F}, \eta)$ where $\mathcal{F}$ is a set of linear inequalities over variables $X = \{x_1, \dots, x_n\}$ and η is a linear function over X of the form $\eta(X) = s_1x_1 + \dots + s_nx_n$. Each inequality $A \in \mathcal{F}$ ranges over variables $\text{var}(A)$ is said to have arity $|\text{var}(A)| = l$ and is assumed to be of the form $c_{A,1}x_{A,1} + c_{A,2}x_{A,2} + \dots + c_{A,l}x_{A,l} \leq b_A$; we also define $\text{var}(I) = X$. We say that two constraints are equal if they range over the same variables with the same coefficients and have the same right-hand side.

For a set of variables Y, let $\mathcal{F}(Y)$ denote the subset of $\mathcal{F}$ containing all inequalities $A \in \mathcal{F}$ such that $Y \cap \text{var}(A) \neq \emptyset$. We will generally use the term *coefficients* to refer to numbers that occur in the inequalities in $\mathcal{F}$. In some cases, we will be dealing with certain selected "named" variables which will not be marked with subscripts to improve readability (e.g., a); there, we may use s_a to denote the coefficient of a in η, i.e., s_a is shorthand for s_j where $a = x_j$.

An assignment α is a mapping from X to $\mathbb{Z}$. For an assignment α and an inequality A of arity l, we denote by $A(\alpha)$ the left-side value of A obtained by applying α, i.e., $A(\alpha) = c_{A,1}\alpha(x_{A,1}) + c_{A,2}\alpha(x_{A,2}) + \dots + c_{A,l}\alpha(x_{A,l})$. Similarly, we let $\eta(\alpha)$ denote the value of the linear function η after applying α.

An assignment α is called *feasible* if it satisfies every $A \in \mathcal{F}$, i.e., if $A(\alpha) \leq b_A$ for each $A \in \mathcal{F}$. Furthermore, α is called a *solution* if the value of $\eta(\alpha)$ is maximized over all feasible assignments; observe that the existence of a feasible assignment does not guarantee the existence of a solution (there may exist an infinite sequence of feasible assignments α with increasing values of $\eta(\alpha)$). Given an instance I, the task in the ILP problem is to compute a solution for I if one exists, and otherwise to decide whether there exists a feasible assignment. On the other hand, the ILP-FEASIBILITY problem asks whether a given instance I admits a feasible assignment (here, we may assume without loss of generality that all coefficients in η are equal to 0).

Given an ILP instance $I = (\mathcal{F}, \eta)$, the primal graph G_I of I is the graph whose vertex set is the set X of variables in I, and two vertices a, b are adjacent iff either there exists some $A \in \mathcal{F}$ containing both a and b or a, b both occur in η with non-zero coefficients.

2.2. Parameterized complexity

In parameterized algorithmics [4,11,25,7] the runtime of an algorithm is studied with respect to a parameter $k \in \mathbb{N}$ and input size n. The basic idea is to find a parameter that describes the structure of the instance such that the combinatorial explosion can be confined to this parameter. In this respect, the most favorable complexity class is FPT (*fixed-parameter tractable*) which contains all problems that can be decided by an algorithm running in time $f(k) \cdot n^{\mathcal{O}(1)}$, where f is a computable function. Algorithms with this running time are called *fpt-algorithms*.

To obtain our lower bounds, we will need the notion of a *parameterized reduction* and the complexity class paraNP [7]. Since we obtain all our lower bounds already for ILP-FEASIBILITY, we only need to consider these notions for decision problems; formally, a *parameterized decision problem* is a subset of $\Sigma^* \times \mathbb{N}$, where Σ is the input alphabet.

Let L_1 and L_2 be parameterized decision problems, with $L_1 \subseteq \Sigma_1^* \times \mathbb{N}$ and $L_2 \subseteq \Sigma_2^* \times \mathbb{N}$. A *parameterized reduction* (or fpt-reduction) from L_1 to L_2 is a mapping $P : \Sigma_1^* \times \mathbb{N} \to \Sigma_2^* \times \mathbb{N}$ such that:

1. $(x, k) \in L_1$ if and only if $P(x, k) \in L_2$;
2. the mapping can be computed by an fpt-algorithm with respect to parameter k;
3. there is a computable function g such that $k' \leq g(k)$, where $(x', k') = P(x, k)$.

There is a variety of classes capturing *parameterized intractability*. For our results, we require only the class paraNP, which is defined as the class of problems that are solvable by a nondeterministic Turing-machine in fpt-time. We will make use of the characterization of paraNP-hardness given by Flum and Grohe [11], Theorem 2.14: any parameterized (decision) problem that remains NP-hard when the parameter is set to some constant is paraNP-hard. Showing paraNP-hardness for a problem rules out the existence of an fpt-algorithm under the assumption that $\mathsf{P} \neq \mathsf{NP}$. In fact, it even allows us to rule out algorithms running in time $n^{f(k)}$ for any function f (these are sometimes called *XP* algorithms).

For our algorithms, we will use the following result as a subroutine. Note that this is a streamlined version of the original statement of the theorem, as used in the area of parameterized algorithms [8,15].

Theorem 1 *([22,21,12]). An ILP instance $I = (\mathcal{F}, \eta)$ can be solved in time $\mathcal{O}(p^{2.5p+o(p)} \cdot |I|)$, where $p = |\mathrm{var}(I)|$.*

2.3. Treewidth and treedepth

Treewidth is the most prominent structural parameter and has been extensively studied in a number of fields. In order to define treewidth, we begin with the definition of its associated decomposition. A *tree-decomposition* $\mathcal{T}$ of a graph $G = (V, E)$ is a pair (T, χ), where T is a tree and χ is a function that assigns each tree node t a set $\chi(t) \subseteq V$ of vertices such that the following conditions hold:

(P1) For every vertex $u \in V$, there is a tree node t such that $u \in \chi(t)$.
(P2) For every edge $\{u, v\} \in E(G)$ there is a tree node t such that $u, v \in \chi(t)$.
(P3) For every vertex $v \in V(G)$, the set of tree nodes t with $v \in \chi(t)$ forms a subtree of T.

The sets $\chi(t)$ are called *bags* of the decomposition $\mathcal{T}$ and $\chi(t)$ is the bag associated with the tree node t. The *width* of a tree-decomposition (T, χ) is the size of a largest bag minus 1. A tree-decomposition of minimum width is called *optimal*. The *treewidth* of a graph G, denoted by $\mathrm{tw}(G)$, is the width of an optimal tree decomposition of G.

Another important notion that we make use of extensively is that of treedepth. Treedepth is a structural parameter closely related to treewidth, and the structure of graphs of bounded treedepth is well understood [24]. A useful way of thinking about graphs of bounded treedepth is that they are (sparse) graphs with no long paths.

We formalize a few notions needed to define treedepth. A *rooted forest* is a disjoint union of rooted trees. For a vertex x in a tree T of a rooted forest, the *height* (or *depth*) of x in the forest is the number of vertices in the path from the root of T to x. The *height of a rooted forest* is the maximum height of a vertex of the forest.

Definition 2 *(Treedepth).* Let the *closure* of a rooted forest $\mathcal{F}$ be the graph $\mathrm{clos}(\mathcal{F}) = (V_c, E_c)$ with the vertex set $V_c = \bigcup_{T \in \mathcal{F}} V(T)$ and the edge set $E_c = \{xy\colon x \text{ is an ancestor of } y \text{ in some } T \in \mathcal{F}\}$. A *treedepth decomposition* of a graph G is a rooted forest $\mathcal{F}$ such that $G \subseteq \mathrm{clos}(\mathcal{F})$. The *treedepth* $td(G)$ of a graph G is the minimum height of any treedepth decomposition of G.

We will later use T_x to denote the vertex set of the subtree of T rooted at a vertex x of T. Similarly to treewidth, it is possible to determine the treedepth of a graph in FPT time.

Proposition 3 *([24]). Given a graph G with n nodes and a constant w, it is possible to decide whether G has treedepth at most w, and if so, to compute an optimal treedepth decomposition of G in time $\mathcal{O}(n)$.*

The following alternative (equivalent) characterization of treedepth will be useful later for ascertaining the exact treedepth in our reduction (specifically in Theorem 12).

Proposition 4 *([24]). Let G_i be the connected components of G. Then*

$$td(G) = \begin{cases} 1, & \text{if } |V(G)| = 1; \\ 1 + \min_{v \in V(G)} td(G - v), & \text{if } G \text{ is connected and } |V(G)| > 1; \\ \max_i td(G_i), & \text{otherwise.} \end{cases}$$

We conclude with a few useful facts about treedepth.

Proposition 5 *([24]).*

1. *If a graph G has no path of length d, then $td(G) \leq d$.*
2. *If $td(G) \leq d$, then G has no path of length 2^d.*
3. *$tw(G) \leq td(G)$.*
4. *If $td(G) \leq d$, then $td(G') \leq d + 1$ for any graph G' obtained by adding one vertex into G.*

Within this manuscript, for an ILP instance I we will use treewidth (treedepth) of I *as shorthand for the treewidth (treedepth) of the primal graph G_I of I.*

3. Exploiting treedepth to solve ILP

Our goal in this section is to show that ILP is fixed parameter tractable when parameterized by the treedepth of the primal graph and the maximum coefficient in any constraint. We begin by formalizing our parameters. Given an ILP instance I, let $td(I)$ be the treedepth of G_I and let $\ell(I)$ be the maximum absolute coefficient which occurs in any inequality in I; to be more precise, $\ell(I) = \max\{\, |c_{A,j}|, |b_A| : A \in \mathcal{F}, j \in \mathbb{N}\}$. When the instance I is clear from the context, we will simply write ℓ and $k = td(I)$ for brevity. We will now state our main algorithmic result of this section.

Theorem 6. *ILP is fixed-parameter tractable parameterized by ℓ and k.*

The main idea behind our fixed-parameter algorithm for ILP is to show that we can reduce the instance into an "equivalent instance" such that the number of variables of the reduced instance can be bounded by our parameters ℓ and k. We then apply Theorem 1 to solve the reduced instance.

For the following considerations, we fix an ILP instance $I = (\mathcal{F}, \eta)$ of size n along with a treedepth decomposition T of G_I with depth k. Given a variable set Y, the operation of *omitting* consists of deleting all inequalities containing at least one variable in Y and all variables in Y; formally, omitting Y from I results in the instance $I' = (\mathcal{F}', \eta')$ where $\mathcal{F}' = \mathcal{F} \setminus \mathcal{F}(Y)$ and η' is obtained by removing all variables in Y from η.

The following notion of equivalence will be crucial for the proof of Theorem 6. Let x, y be two variables that share a common parent in T, and recall that T_x (T_y) denotes the vertex set of the subtree of T rooted at x (y). We say that x are y are *equivalent*, denoted $x \sim y$, if there exists a bijective function $\delta_{x,y} : T_x \to T_y$ (called the *renaming function*) such that $\delta_{x,y}(\mathcal{F}(T_x)) = \mathcal{F}(T_y)$; here $\delta_{x,y}(\mathcal{F}(T_x))$ denotes the set of inequalities in $\mathcal{F}(T_x)$ after the application of $\delta_{x,y}$ on each variable in T_x. In other words, $x \sim y$ means that there exists a way of "renaming" the variables in T_y so that $\mathcal{F}(T_y)$ becomes $\mathcal{F}(T_x)$.

It is easy to verify that $\sim$ is indeed an equivalence relation. Intuitively, the following lemma shows that if $x \sim y$ for two variables x and y of I, then (up to renaming) the set of all feasible assignments of the variables in T_x is equal to the set of all feasible assignments of the variables in T_y; it will be useful to recall the meaning of s_a from Subsection 2.1.

Lemma 7. *Let x, y be two variables of I such that $x \sim y$ and $s_a = 0$ for each $a \in T_x \cup T_y$. Let $I' = (\mathcal{F}', \eta')$ be the instance obtained from I by omitting T_y. Then there exists a solution α of $\mathrm{var}(I)$ of value $w = \eta(\alpha)$ if and only if there exists a solution α' of $\mathrm{var}(I')$ of value $w = \eta'(\alpha')$. Moreover, a solution α can be computed from any solution α' in linear time if the renaming function $\delta_{x,y}$ is known.*

Proof. Let α be a solution of $\mathrm{var}(I)$ of value $w = \eta(\alpha)$. Since $\mathcal{F}' \subseteq \mathcal{F}$, it follows that setting α' to be a restriction of α to $\mathrm{var}(I) \setminus T_y$ satisfies every inequality in $\mathcal{F}'$. Since variables in T_y do not contribute to η, it also follows that $\eta(\alpha) = \eta(\alpha')$.

On the other hand, let α' be a solution of $\mathrm{var}(I')$ of value $w = \eta'(\alpha')$. Consider the assignment α obtained by extending α' to T_y by reusing the assignments of T_x on T_y. Formally, for each $z \in T_y$ we set $\alpha(z) = \alpha'(\delta^{-1}_{x,y}(z))$ and for all other variables $w \in \mathrm{var}(I')$ we set $\alpha(w) = \alpha'(w)$. By assumption, α and α' must assign the same values to any variable w such that $s_w \neq 0$, and hence $\eta(\alpha) = \eta(\alpha')$. To argue feasibility, first observe that any $A \in \mathcal{F}'$ must be satisfied by α since α and α' only differ on variables which do not occur in I'. Moreover, by definition of $\sim$ for each $A \in \mathcal{F} \setminus \mathcal{F}' = \mathcal{F}(T_y)$ there exists an inequality $A' \in \mathcal{F}'$ such that $\delta_{x,y}(A') = A$. In particular, this implies that $A(\alpha) = A'(\alpha) = A'(\alpha')$, and since $A'(\alpha') \le b_{A'} = b_A$ we conclude that $A(\alpha) \le b_A$. Consequently, α satisfies A.

The final claim of the lemma follows from the construction of α described above. □

In the following let z be a variable of I at depth $k - i$ in T for every i with $1 \le i < k$ and let Z be the set of all children of z in T. Moreover, let m be the maximum size of any subtree rooted at a child of z in T, i.e., $m := \max_{z' \in Z} |T_{z'}|$. We will show next that the number of equivalence classes among the children of z can be bounded by the function $\#\mathrm{C}(\ell, k, i, m) := 2^{(2\ell+1)^{k+1} \cdot m^i}$. Observe that this bound depends only on ℓ, k, m, and i and not on the size of I.

Lemma 8. *The equivalence relation $\sim$ has at most $\#\mathrm{C}(\ell, k, i, m)$ equivalence classes over Z.*

Proof. Consider an element $a \in Z$. By construction of G_I, each inequality $A \in \mathcal{F}(T_a)$ only contains at most $k - i$ variables outside of T_a (specifically, the ancestors of a) and at most i variables in T_a. Furthermore, b_A and each coefficient of a variable in A is an integer whose absolute value does not exceed ℓ. From this it follows that there exists a finite number of inequalities which can occur in $\mathcal{F}(T_a)$. Specifically, the number of distinct combinations of coefficients for all the variables in A and for b_A is $(2\ell + 1)^{k+1}$, and the number of distinct choices of variables in $\mathrm{var}(A) \cap T_a$ is upper-bounded by $\binom{m}{i}$, and so we arrive at $|\mathcal{F}(T_a)| \le (2\ell + 1)^{k+1} \cdot \binom{m}{i} \le (2\ell + 1)^{k+1} \cdot m^i$.

Consequently, the set of inequalities for each child $y \in Z$ of z has bounded cardinality. We will use this to bound the number of equivalence classes in $\#\mathrm{C}(\ell, k, i, m)$ by observing that two elements are equivalent if and only if they occur in precisely the same sets of inequalities (up to renaming). To formalize this intuition, we need a formal way of canonically renaming all variables in the individual subtrees rooted in Z; without renaming, each $\mathcal{F}(T_y)$ would span a distinct set of variables and hence it would not be possible to bound the set of all such inequalities. So, for each y let δ_{y,x_0} be a bijective renaming function which renames all of the variables in T_y to the variable set $\{x_0^1, x_0^2, \dots, x_0^{|T_y|}\}$ (in an arbitrary way). Now we can formally define $\Gamma_z = \{\mathcal{F}(T_{x_0}) : \delta_{y,x_0}(\mathcal{F}(T_y)), y \in Z\}$, and observe that Γ_z has cardinality at most $2^{(2\ell+1)^{k+1} \cdot m^i} = \#\mathrm{C}(\ell, k, i, m)$. To conclude the proof, recall that if two variables a, b satisfy $\mathcal{F}(T_a) = \delta_{b,a}(\mathcal{F}(T_b))$ for a bijective renaming function $\delta_{b,a}$, then $b \sim a$. Hence, the absolute bound on the cardinality of Γ_z implies that $\sim$ has at most $\#\mathrm{C}(\ell, k, i, m)$ equivalence classes over Z. □

It follows from the above lemma that if z has more than $\#\mathrm{C}(\ell, k, i, m)$ children, then two of those must be equivalent. The next lemma shows that it is also possible to find such a pair of equivalent children efficiently.

Lemma 9. *Given a subset Z' of Z with $|Z'| = \#\mathrm{C}(\ell, k, i, m) + 1$, then in time $\mathcal{O}(\#\mathrm{C}(\ell, k, i, m)^2 \cdot m!m)$ one can find two children x and y of Z such that $x \sim y$ together with a renaming function $\delta_{x,y}$ which certifies this.*

Proof. Consider the following algorithm $\mathbb{A}$. First, $\mathbb{A}$ computes a subset Z' consisting of exactly (arbitrarily chosen) $\#\mathrm{C}(\ell, k, i, m) + 1$ children of Z. Then $\mathbb{A}$ branches over all distinct pairs $x, y \in Z'$ in time at most $\mathcal{O}(\#\mathrm{C}(\ell, k, i, m)^2)$. Second, $\mathbb{A}$ branches over all of the at most $m!$ bijective renaming functions $\delta_{x,y}$. Third, $\mathbb{A}$ computes $\delta_{x,y}(\mathcal{F}(T_x))$ and tests whether it is equal to $\mathcal{F}(T_y)$ (which takes at most $\mathcal{O}(m)$ time); if this is the case, then $\mathbb{A}$ terminates and outputs x, y and $\delta_{x,y}$.

We argue correctness. By Lemma 8 and due to the cardinality of Z', there must exist $x, y \in Z'$ such that $x \sim y$. In particular, there must exist a renaming function $\delta_{x,y}$ such that $\delta_{x,y}(\mathcal{F}(T_x)) = \mathcal{F}(T_y)$. But then $\mathbb{A}$ is guaranteed to find such x, y, $\delta_{x,y}$ since it performs an exhaustive search. □

Combining Lemma 7 and Lemma 9, we arrive at the following corollary.

Corollary 10. *If $|Z| > \#\mathrm{C}(\ell, k, i, m) + 1$, then in time $\mathcal{O}(\#\mathrm{C}(\ell, k, i, m)^2 \cdot m!m)$ one can compute a subinstance $I' = (\mathcal{F}', \eta)$ of I with strictly less variables and the following property: there exists a solution α of I of value $w = \eta(\alpha)$ if and only if there exists a solution α' of I' of value w. Moreover, a solution α can be computed from any solution α' in linear time.*

Proof. In order to avoid having to consider all children of z, the algorithm first computes (an arbitrary) subset Z' of Z such that $|Z'| = \#\mathrm{C}(\ell, k, i, m) + 2$. Then to be able to apply Lemma 9 without changing the set of solutions of I, the algorithm computes a subset Z'' of Z' such that $|Z''| = \#\mathrm{C}(\ell, k, i, m) + 1$ and for every $z' \in Z''$ it holds that $s_{z''} = 0$ for every $z'' \in T_{z'}$. Note that since there are at most k variables of I with non-zero coefficients in η and these variables form a clique in G_I, all of them occur only in a single branch of T_z. It follows that Z'' as specified above exists and it can be obtained from Z' by removing the (at most one) element z' in Z' with $s_{z''} \neq 0$ for some $z'' \in T_{z'}$. Observe that this step of the algorithm takes time at most $\mathcal{O}(m \cdot (\#\mathrm{C}(\ell, k, i, m) + 1))$.

The algorithm then proceeds as follows. It uses Lemma 9 to find two variables $x, y \in Z''$ such that $x \sim y$ and computes I' from I by omitting T_y from I. The running time of the algorithm follows from Lemma 9 since the running times of the other steps of the algorithm are dominated by the application of Lemma 9. The corollary now follows from Lemma 7 and Lemma 9, which certify that:

- there exists a solution α of I of value $w = \eta(\alpha)$ if and only if there exists a solution α' of I' of value w, and
- a solution α can be computed from any solution α' in linear time. □

Let e_i and d_i for every i with $1 \le i \le k$ be defined inductively by setting $e_k = 1$, $d_k = 0$, $d_i = \#\mathrm{C}(\ell, k, i, s_{i+1}) + 1$, and $e_i = d_i e_{i+1} + 1$. The following lemma shows that in time $\mathcal{O}(|I|d_1^2 \cdot e_1!e_1)$ one can compute an "equivalent" subinstance I' of I containing at most e_1 variables. Informally, e_i is an upper bound on the number of nodes in a subtree rooted at depth i and d_i is an upper bound on the number of children of a node at level i in I'.

Lemma 11. *There exists an algorithm that takes as input I and T, runs in time $\mathcal{O}(|I|d_1^2 \cdot e_1!e_1)$ and outputs an ILP instance I' containing at most e_1 variables with the following property: there exists a solution α of I of value $w = \eta(\alpha)$ if and only if there exists a solution α' of I' of value $w = \eta'(\alpha')$. Moreover, a solution α can be computed from any solution α' in linear time.*

Proof. The algorithm exhaustively applies Corollary 10 to every variable of T in a bottom-up manner, i.e., it starts by applying the corollary exhaustively to all variables at depth $k - 1$ and then proceeds up the levels of T until it reaches depth 1. Let T' be the subtree of T obtained after the exhaustive application of Corollary 10 to T.

We will first show that if x is a variable at depth i of T', then x has at most d_i children and $|T'_x| \le e_i$. We will show the claim by induction on the depth i starting from depth k. Because all variables x of T at level k are leaves, it holds that x has $0 = d_k$ children in T' and $|T'_x| = 1 \le e_k$, showing the start of the induction. Now let x be a variable at depth i of T' and let y be a child of x in T'. It follows from the induction hypothesis that $|T'_y| \le e_{i+1}$. Moreover, using Corollary 10, we obtain that x has at most $\tilde{\#}\mathrm{C}(\ell, k, i, e_{i+1}) + 1 = d_i$ children in T' and thus $|T'_x| \le d_i e_{i+1} + 1 = e_i$, as required.

The running time of the algorithm now follows from the observation that (because every application of Corollary 10 removes at least one variable of I) Corollary 10 is applied at most $|I|$ times and moreover the maximum running time of any call to Corollary 10 is at most $\mathcal{O}(d_1^2 \cdot e_1!e_1)$. Correctness and the fact that α can be computed from α' follow from Corollary 10; more specifically, we extend α' into α by assigning pruned variables in the same way as their equivalent counterparts. □

Proof of Theorem 6. The algorithm proceeds in three steps. First, it applies Lemma 11 to reduce the instance I into an "equivalent" instance I' containing at most e_1 variables in time $\mathcal{O}(|I|d_1^2 \cdot e_1!e_1)$; in particular, a solution α of I can be computed in linear time from a solution α' of I'. Second, it uses Theorem 1 to compute a solution α' of I' in time at most $\mathcal{O}(e_1^{2.5e_1+o(e_1)} \cdot |I'|)$; because e_1 and d_1 are bounded by our parameters, the whole algorithm runs in FPT time. Third, it transforms the solution α' into a solution α of I. Correctness follows from Lemma 11 and Theorem 1. □

4. Lower bounds and hardness

In this section we will complement our algorithmic results by providing matching hardness results. Namely, we will show that already the ILP-FEASIBILITY problem is NP-hard on graphs of bounded treedepth and also NP-hard on graphs of bounded treewidth and bounded maximum coefficient.[1]

We begin by noting that ILP-FEASIBILITY remains NP-hard even if the maximum absolute value of any coefficient is at most one. This follows, e.g., by enhancing the standard reduction from the decision version of VERTEX COVER (given a graph G and a bound ν, does G admit a vertex cover of size at most ν?) to ILP-FEASIBILITY as follows:

- add variables $x_1, \ldots, x_\nu$ and force each of them to be 1,
- set $x = \sum_{i\in[\nu]} x_i$,
- add a constraint requiring that the sum of all variables which represent vertices of G is at most x.

Observation 1. ILP-feasibility is NP-hard even on instances with a maximum absolute value of every coefficient of 1.

To simplify the constructions in the hardness proofs, we will often talk about constraints as equalities instead of inequalities. Clearly, every equality can be written in terms of two inequalities.

Theorem 12. ILP-FEASIBILITY *is* NP-*hard even on instances of bounded treedepth.*

Proof. We will show the theorem by a polynomial-time reduction from the well-known NP-hard 3-COLORABILITY problem [16]: given a graph, decide whether the vertices of G can be colored with three colors such that no two adjacent vertices of G share the same color.

The main idea behind the reduction is to represent a 3-partition of the vertex set of G (which in turn represents a 3-coloring of G) by the domain values of three "global" variables. The value of each of these global variables will represent a subset of vertices of G that will be colored using the same color. To represent a subset of the vertices of G in terms of domain values of the global variables, we will represent every vertex of G with a unique prime number and a subset by the value obtained from the multiplication of all prime numbers of vertices contained in the subset. To ensure that the subsets represented by the global variables correspond to a valid 3-partition of G we will introduce constraints which ensure that:

C1 For every prime number representing some vertex of G exactly one of the global variables is divisible by that prime number. This ensures that every vertex of G is assigned to exactly one color class.

C2 For every edge $\{u, v\}$ of G it holds that no global variable is divisible by the prime numbers representing u and v at the same time. This ensures that no two adjacent vertices of G are assigned to the same color class.

Thus let G be the given instance of 3-COLORING and assume that the vertices of G are uniquely identified as elements of $\{1, \ldots, |V(G)|\}$. In the following we denote by $p(i)$ the i-th prime number for any positive integer i, where $p(1) = 2$. We construct an instance I of ILP-FEASIBILITY in polynomial-time with treedepth at most 8 and coefficients bounded by a polynomial in $V(G)$ such that G has a 3-coloring if and only if I has a feasible assignment. This instance I has the following variables:

- The *global variables* g_1, g_2, and g_3 with an arbitrary positive domain, whose values will represent a valid 3-Partitioning of $V(G)$.
- For every i and j with $1 \le i \le |V(G)|$ and $1 \le j \le 3$, the variables $m_{i,j}$ (with an arbitrary non-negative domain), $r_{i,j}$ (with domain between 0 and $p(i) - 1$), and $u_{i,j}$ (with binary domain). These variables are used to secure condition C1.

[1] Unless explicitly mentioned otherwise, all the presented NP-hardness results hold in the strong sense, i.e., when the input is encoded in unary.

- For every $e \in E(G)$, $v \in e$, and j with $1 \le j \le 3$, the variables $m_{e,v,j}$ (with an arbitrary non-negative domain), $r_{e,v,j}$ (with domain between 0 and $p(v) - 1$), and $u_{e,v,j}$ (with binary domain). These variables are used to secure condition C2.

I has the following constraints (in the following let α be any feasible assignment of I):

- Constraints that restrict the domains of all variables as specified above, i.e.:
  - for every i and j with $1 \le i \le |V(G)|$ and $1 \le j \le 3$, the constraints $g_j \ge 0$, $m_{i,j} \ge 0$, $0 \le r_{i,j} \le p(i) - 1$, and $0 \le u_{i,j} \le 1$;
  - for every $e \in E(G)$, $v \in e$, and j with $1 \le j \le 3$, the constraints $m_{e,v,j} \ge 0$, $0 \le r_{e,v,j} \le p(v) - 1$, and $0 \le u_{e,v,j} \le 1$.
- The following constraints, introduced for each $1 \le i \le |V(G)|$ and $1 \le j \le 3$, together guarantee that condition C1 holds:
  - Constraints that ensure that $\alpha(r_{i,j})$ is equal to the remainder of $\alpha(g_j)$ divided by $p(i)$, i.e., the constraint $g_j = p(i)m_{i,j} + r_{i,j}$.
  - Constraints that ensure that $\alpha(u_{i,j}) = 0$ if and only if $\alpha(r_{i,j}) = 0$, i.e., the constraints $u_{i,j} \le r_{i,j}$ and $r_{i,j} \le (p(i)-1)u_{i,j}$. Note that together the above constraints now ensure that $\alpha(u_{i,j}) = 0$ if and only if g_j is divisible by $p(i)$.
  - Constraints that ensure that exactly one of $\alpha(u_{i,1})$, $\alpha(u_{i,2})$, and $\alpha(u_{i,3})$ is equal to 0, i.e., the constraints $2 \le u_{i,1} + u_{i,2} + u_{i,3} \le 2$. Note that together all the above constraints now ensure condition C1 holds.
- The following constraints, introduced for each $1 \le j \le 3$, together guarantee that condition C2 holds:
  - Constraints that ensure that for every $e \in E(G)$ and $v \in e$, it holds that $\alpha(r_{e,v,j})$ is equal to the remainder of g_j divided by $p(v)$, i.e., the constraint $g_j = p(i)m_{e,v,j} + r_{e,v,j}$.
  - Constraints that ensure that for every $e \in E(G)$, $v \in e$, and j with $1 \le j \le 3$ it holds that $\alpha(u_{e,v,j}) = 0$ if and only if $\alpha(r_{e,v,j}) = 0$, i.e., the constraints $u_{e,v,j} \le r_{e,v,j}$ and $r_{e,v,j} \le p(v)u_{e,v,j}$. Note that together the above constraints now ensure that $\alpha(u_{e,v,j}) = 0$ if and only if g_j is divisible by $p(v)$.
  - Constraints that ensure that for every $e = \{v, w\} \in E(G)$ and j with $1 \le j \le 3$ it holds that at least one of $\alpha(u_{e,w,j})$ and $\alpha(u_{e,v,j})$ is non-zero, i.e., the constraint $u_{e,u,j} + u_{e,v,j} \ge 1$. Note that together with all of the above constraints this now ensures condition C2.

This completes the construction of I and the largest coefficient used in I is $p(|V(G)|)$. It is well-known that $p(i)$ is upper-bounded by $\mathcal{O}(i \log i)$ due to the Prime Number Theorem, and so this in particular implies that the numbers which occur in I are bounded by a polynomial in $|V(G)|$. Hence I can be constructed in polynomial time.

Following the construction and explanations provided above, it is not difficult to see that I has a feasible assignment if and only if G has a 3-coloring. Indeed, for any 3-coloring of G, one can construct a feasible assignment of I by computing the prime-number encoding for the vertex sets that receive colors 1, 2, 3 and assign these three numbers to g_1, g_2, g_3, respectively. Such an assignment allows us to straightforwardly satisfy the constraints ensuring C1 holds (since each prime occurs in exactly one global constraint), the constraints ensuring C2 holds (since each edge is incident to at most one of each color) while maintaining the domain bounds.

On the other hand, for any feasible assignment α, clearly each of $\alpha(g_1), \alpha(g_2), \alpha(g_3)$ will be divisible by some subset of prime numbers between 2 and $p(|V(G)|)$. In particular, since α is feasible it follows from the construction of our first group of constraints that each prime between 2 and $p(|V(G)|)$ divides precisely one of $\alpha(g_1), \alpha(g_2), \alpha(g_3)$, and so this uniquely encodes a corresponding candidate 3-coloring for the vertices of the graph. Finally, since α also satisfies the second group of constraints, this candidate 3-coloring must have the property that each edge is incident to exactly 2 colors, and so it is in fact a valid 3-coloring.

It remains to show that the treedepth of I is at most 8. We will show this by using the characterization of treedepth given in Proposition 4. We first observe that the graph $G_I \setminus \{g_1, g_2, g_3\}$ consists of the following components:

- for every i with $1 \le i \le |V(G)|$, one component on the vertices $m_{i,1}, \dots, m_{i,3}$, $r_{i,1}, r_{i,2}, r_{i,3}$, $u_{i,1}, u_{i,2}, u_{i,3}$. Note that all of these components are isomorphic to each other and we will therefore in the following refer to these components as *vertex-type components*;
- for every $e = \{w, v\} \in E(G)$ and j with $1 \le j \le 3$, one component on the vertices $m_{e,w,j}$, $m_{e,v,j}$, $r_{e,w,j}$, $r_{e,v,j}$, $u_{e,w,j}$, and $u_{e,v,j}$. Note that all of these components are isomorphic to each other and we will therefore in the following refer to these components as *edge-type components*.

The two types of components are illustrated in Fig. 1. We will show next that any vertex-type component has treedepth at most 5 and every edge-type component has treedepth at most 4. This would then imply that G_I has treedepth at most 8 (since it suffices to remove the vertices $\{g_1, g_2, g_3\}$ in order to decompose the graph into these components). Hence let i with $1 \le i \le |V(G)|$ and consider the vertex-type component C_i on the vertices $m_{i,1}, m_{i,2}, m_{i,3}$, $r_{i,1}, r_{i,2}, r_{i,3}$, $u_{i,1}, u_{i,2}, u_{i,3}$. Note that $C_i \setminus \{u_{i,1}, u_{i,2}, u_{i,3}\}$ consists of one component for every j with $1 \le j \le 3$ that contains the vertices $m_{i,j}$ and $r_{i,j}$. Clearly each of these three components has treedepth at most 2 and hence the treedepth of C_i is at most $2 + 3 = 5$, as required.

In order to show that every edge-type component has treedepth at most 4, consider an edge $e = \{w, v\} \in E(G)$ and some j satisfying $1 \le j \le 3$. Let $C_{e,j}$ be the edge-type component consisting of the vertices $m_{e,w,j}$, $m_{e,v,j}$, $r_{e,w,j}$, $r_{e,v,j}$, $u_{e,w,j}$, and $u_{e,v,j}$. Note that $C_{e,j} \setminus \{u_{e,w,j}, u_{e,v,j}\}$ consists of two components, one containing the vertices $m_{e,w,j}$ and $r_{e,u,j}$

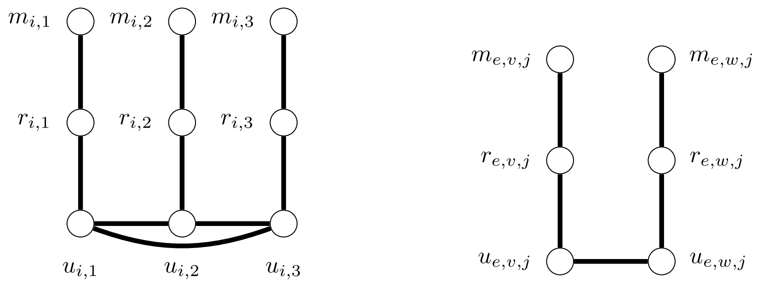

Fig. 1. Illustration of a vertex-type component (left) and an edge-type component (right) in the graph $G_I \setminus \{g_1, g_2, g_3\}$.

and one containing the vertices $m_{e,v,j}$ and $r_{e,v,j}$. Clearly, each of these two components has treedepth at most 2 and hence the treedepth of $C_{e,j}$ is at most $2 + 2 = 4$, as required. $\square$

The next theorem shows that ILP-FEASIBILITY is paraNP-hard parameterized by both treewidth and the maximum absolute value of any number in the instance; observe that since we are bounding all numbers in the instance, the theorem in particular implies NP-hardness. We note that the idea to reduce from SUBSET SUM was inspired by previous work of Jansen and Kratsch [20].

Theorem 13. ILP-FEASIBILITY *is* NP-*hard even on instances with treewidth at most two and where the maximum absolute value of any coefficient is at most one.*

Proof. We show the result by a polynomial reduction from the SUBSET SUM problem, which is well-known to be weakly NP-complete.

SUBSET SUM	
Input:	A set $Q := \{q_1, \ldots, q_n\}$ of integers and an integer r.
Question:	Is there a subset $Q' \subseteq Q$ such that $\sum_{q' \in Q'} q' = r$?

Let $I := (Q, r)$ with $Q := \{q_1, \ldots, q_n\}$ be an instance of SUBSET SUM, which we assume to be given in binary encoding. We will construct an instance I' of ILP-FEASIBILITY equivalent to I in polynomial-time (with respect to the input size of I) with treewidth at most 2 that uses only -1, 0, and 1 as coefficients. Crucial to our construction are the following auxiliary ILP instances.

Claim 1. For every $q \in \mathbb{N}$ and any two variables x and y there is an ILP instance $I(q, x, y)$ satisfying the following conditions:

(P1) $I(q, x, y)$ has at most $\mathcal{O}(\log q)$ variables and constraints,
(P2) the maximum absolute value of any coefficient in $I(q, x, y)$ is at most one,
(P3) the treewidth of $I(q, x, y)$ is at most two and
(P4) for every feasible assignment α of $I(q, x, y)$, it holds that $\alpha(y) \in \{\alpha(x), \alpha(x) + q\}$.

Moreover, there are ILP instances $I(q, y)$ and $I_C(q, y)$ satisfying (P1)–(P3) and additionally:

- $\alpha(y) \in \{0, q\}$ for $I(q, y)$, and
- $\alpha(y) = q$ for $I_C(q, y)$.

Proof. For an integer q, let $B(q)$ be the set of indices of all bits that are equal to one in the binary representation of q, i.e., we have $q = \sum_{j \in B(q)} 2^j$. Moreover, let $m = b_{\max}(q)$ be the largest index in $B(q)$.

We construct the ILP instance $I(q, x, y)$ as follows. We first introduce $m+1$ variables $h_0, \ldots, h_m$ together with m variables $h'_0, \ldots, h'_{m-1}$ and add the following constraints: $0 \le h_0 \le 1$, and for every i with $0 \le i < m$ we set $h'_i = h_i$ and $h_{i+1} = h_i + h'_i$. Observe that the above constraints ensure that $\alpha(h_i)$ is equal to $2^i \alpha(h_0)$ for every i with $0 \le i \le m$ and every feasible assignment α. We also introduce the new auxiliary variables $z_0, \ldots, z_m$ together with the following constraints:

- If $0 \in B(q)$ then we add the constraint $z_0 = h_0 + x$, and otherwise we add the constraint $z_0 = x$.
- For every i with $0 \le i < m$, if $i + 1 \in B(q)$ then we add the constraint $z_{i+1} = h_{i+1} + z_i$ and otherwise the constraint $z_{i+1} = z_i$.

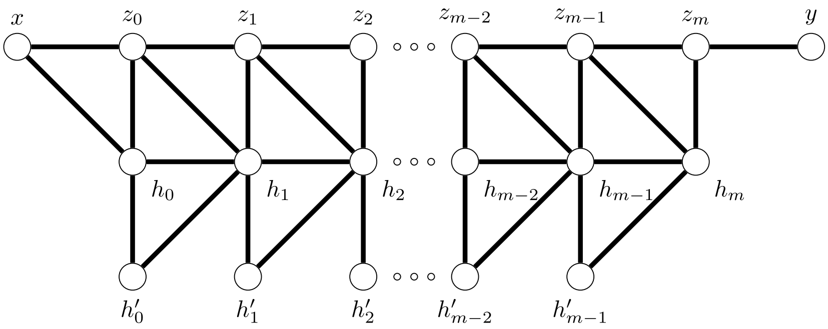

Fig. 2. Illustration of the primal graph of the instance $I(q, x, y)$.

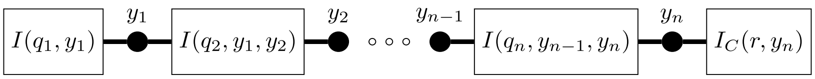

Fig. 3. Illustration of the ILP instance I'.

Observe that these constraints ensure that $\alpha(z_i)$ is equal to $\alpha(x) + \sum_{j\in B(q)\wedge j\leq i} \alpha(h_j)$ for every i with $0 \leq i \leq m$ and any feasible assignment α. Finally we introduce the constraint $y = z_m$. This concludes the construction of $I(q, x, y)$. By construction $I(q, x, y)$ satisfies (P1) and (P2). Moreover, because $\alpha(y) = \alpha(z_m)$ is equal to $q\alpha(h_0) + \alpha(x)$ for any feasible assignment α and since $\alpha(h_0) \in \{0, 1\}$, we obtain that $\alpha(y) \in \{\alpha(x), \alpha(x) + q\}$ showing that $I(q, x, y)$ satisfies (P4). Finally, with the help of Fig. 2, it is straightforward to verify that $I(q, x, y)$ has treewidth at most two.

The ILP instance $I(q, y)$ can now be obtained from $I(q, x, y)$ by removing the variable x. Moreover, the ILP instance $I_C(q, y)$ can now be obtained from $I(q, y)$ by replacing the constraints $0 \leq h_0 \leq 1$ with the constraint $h_0 = 1$. □

We now obtain I' as the (non-disjoint) union of the instances $I(q_1, y_1)$, $I(q_i, y_{i-1}, y_i)$ for every i with $1 < i \leq n$, and the instance $I_C(r, y_n)$ (see Fig. 3 for an illustration of I'). The size of each of these $n + 1$ instances is bounded by $\mathcal{O}(\log m)$, where m is the maximum of $\{q_1, \dots, q_n, r\}$, and it can be verified that each of these instances can be constructed in time $\mathcal{O}(\log m)$. Hence the construction of I' from I can be completed in polynomial time (with respect to the size of the binary encoding of I). We also observe that the maximum absolute value of any coefficient in I' is at most 1. Finally, because I' is a simple concatenation of ILP instances with treewidth at most 2, it is straightforward to verify that I' has treewidth at most 2. □

5. Concluding notes

We presented new results that add to the complexity landscape for ILP w.r.t. structural parameterizations of the constraint matrix. Our main algorithmic result pushes the frontiers of tractability for ILP instances and will hopefully serve as a precursor for the study of further structural parameterizations for ILP. We note that the running time of the presented algorithm has a highly nontrivial dependence on the treedepth of the ILP instance, and hence the algorithm is unlikely to outperform dedicated solvers in practical settings.

The provided results draw an initial complexity landscape for ILP w.r.t. the most prominent decompositional width parameters. However, other approaches exploiting the structural properties of ILP instances still remain unexplored and represent interesting directions for future research. For instance, an adaptation of *backdoors* [17] to the ILP setting could lead to highly relevant algorithmic results.

Acknowledgements

The authors acknowledge support by the Austrian Science Fund (FWF, project P26696). Robert Ganian is also affiliated with FI MU, Brno, Czech Republic. The authors also thank the anonymous reviewers for many insightful suggestions and comments—and in particular for helping improve Theorem 13.

References

[1] S.A. Alumur, B.Y. Kara, Network hub location problems: the state of the art, Eur. J. Oper. Res. 190 (1) (2008) 1–21.
[2] B. Courcelle, J.A. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width, Theory Comput. Syst. 33 (2) (2000) 125–150.
[3] W.H. Cunningham, J. Geelen, On integer programming and the branch-width of the constraint matrix, in: M. Fischetti, D.P. Williamson (Eds.), Integer Programming and Combinatorial Optimization, Proceedings, 12th International IPCO Conference, Ithaca, NY, USA, June 25–27, 2007, in: Lecture Notes in Computer Science, vol. 4513, Springer, 2007, pp. 158–166.

[4] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, Parameterized Algorithms, Springer, 2015.
[5] J.A. De Loera, R. Hemmecke, M. Köppe, Algebraic and Geometric Ideas in the Theory of Discrete Optimization, MOS-SIAM Series on Optimization, vol. 14, SIAM, 2013.
[6] R. Diestel, Graph Theory, 4th edition, Graduate Texts in Mathematics, vol. 173, Springer, 2012.
[7] R.G. Downey, M.R. Fellows, Fundamentals of Parameterized Complexity, Texts in Computer Science, Springer, 2013.
[8] M.R. Fellows, D. Lokshtanov, N. Misra, F.A. Rosamond, S. Saurabh, Graph layout problems parameterized by vertex cover, in: ISAAC, in: Lecture Notes in Computer Science, Springer, 2008, pp. 294–305.
[9] E. Fischer, J.A. Makowsky, E.R. Ravve, Counting truth assignments of formulas of bounded tree-width or clique-width, Discrete Appl. Math. 156 (4) (2008) 511–529.
[10] C. Floudas, X. Lin, Mixed integer linear programming in process scheduling: modeling, algorithms, and applications, Ann. Oper. Res. 139 (1) (2005) 131–162.
[11] J. Flum, M. Grohe, Parameterized Complexity Theory, Texts in Theoretical Computer Science. An EATCS Series, vol. XIV, Springer Verlag, Berlin, 2006.
[12] A. Frank, É. Tardos, An application of simultaneous diophantine approximation in combinatorial optimization, Combinatorica 7 (1) (1987) 49–65.
[13] R. Ganian, S. Szeider, Community structure inspired algorithms for SAT and #SAT, in: Theory and Applications of Satisfiability Testing, Proceedings, 18th International Conference, SAT 2015, Austin, Texas, USA, September 24–27, 2015, in: Lecture Notes in Computer Science, vol. 9340, Springer, 2015, pp. 294–305.
[14] R. Ganian, P. Hliněný, J. Obdržálek, Better algorithms for satisfiability problems for formulas of bounded rank-width, Fundam. Inform. 123 (1) (2013) 59–76.
[15] R. Ganian, E.J. Kim, S. Szeider, Algorithmic applications of tree-cut width, in: Mathematical Foundations of Computer Science 2015, Proceedings, Part II, 40th International Symposium, MFCS 2015, Milan, Italy, August 24–28, 2015, 2015, pp. 348–360.
[16] M.R. Garey, D.R. Johnson, Computers and Intractability, W.H. Freeman and Company, San Francisco, New York, 1979.
[17] S. Gaspers, S. Szeider, Backdoors to satisfaction, in: The Multivariate Algorithmic Revolution and Beyond – Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday, in: Lecture Notes in Computer Science, vol. 7370, Springer Verlag, 2012, pp. 287–317.
[18] G. Gutin, M. Jones, M. Wahlström, Structural parameterizations of the Mixed Chinese Postman Problem, in: Algorithms, Proceedings, 23rd Annual European Symposium, ESA 2015, Patras, Greece, September 14–16, 2015, 2015, pp. 668–679.
[19] R. Hemmecke, S. Onn, L. Romanchuk, N-fold integer programming in cubic time, Math. Program. 137 (1–2) (2013) 325–341.
[20] B.M.P. Jansen, S. Kratsch, A structural approach to kernels for ILPs: treewidth and total unimodularity, in: Algorithms, Proceedings, 23rd Annual European Symposium, ESA 2015, Patras, Greece, September 14–16, 2015, in: Lecture Notes in Computer Science, vol. 9294, Springer, 2015, pp. 779–791.
[21] R. Kannan, Minkowski's convex body theorem and integer programming, Math. Oper. Res. 12 (3) (1987) 415–440.
[22] H.W. Lenstra Jr., Integer programming with a fixed number of variables, Math. Oper. Res. 8 (4) (1983) 538–548.
[23] A. Lodi, S. Martello, M. Monaci, Two-dimensional packing problems: a survey, Eur. J. Oper. Res. 141 (2) (2002) 241–252.
[24] J. Nešetřil, P. Ossona de Mendez, Sparsity: Graphs, Structures, and Algorithms, Algorithms and Combinatorics, vol. 28, Springer, 2012.
[25] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford University Press, 2006.
[26] S. Onn, Nonlinear Discrete Optimization, Zurich Lectures in Advanced Mathematics, European Mathematical Society, 2010.
[27] C.H. Papadimitriou, K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Prentice–Hall, 1982.
[28] C.H. Papadimitriou, On the complexity of integer programming, J. ACM 28 (4) (1981) 765–768.
[29] S. Szeider, Finding paths in graphs avoiding forbidden transitions, Discrete Appl. Math. 126 (2–3) (2003) 239–251.
[30] P. Toth, D. Vigo (Eds.), The Vehicle Routing Problem, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.
[31] M. van den Briel, T. Vossen, S. Kambhampati, Reviving integer programming approaches for AI planning: a branch-and-cut framework, in: Proceedings of the Fifteenth International Conference on Automated Planning and Scheduling, ICAPS 2005, Monterey, California, USA, June 5–10, 2005, AAAI, 2005, pp. 310–319.
[32] T. Vossen, M.O. Ball, A. Lotem, D.S. Nau, On the use of integer programming models in AI planning, in: Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, IJCAI 99, Stockholm, Sweden, July 31–August 6, 1999, Morgan Kaufmann, 1999, pp. 304–309 (2 volumes, 1450 pages).