Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence (IJCAI-18)

Unary Integer Linear Programming with Structural Restrictions

Eduard Eiben[1], **Robert Ganian**[2], **Dušan Knop**[1], **Sebastian Ordyniak**[3]
[1] Department of Informatics, University of Bergen, Norway
[2] Algorithms and Complexity group, TU Wien, Austria
[3] Algorithms group, University of Sheffield, UK
eduard.eiben@uib.no, rganian@gmail.com, dusan.knop@uib.no, sordyniak@gmail.com

Abstract

Recently a number of algorithmic results have appeared which show the tractability of Integer Linear Programming (ILP) instances under strong restrictions on variable domains and/or coefficients (AAAI 2016, AAAI 2017, IJCAI 2017). In this paper, we target ILPs where neither the variable domains nor the coefficients are restricted by a fixed constant or parameter; instead, we only require that our instances can be encoded in unary.

We provide new algorithms and lower bounds for such ILPs by exploiting the structure of their variable interactions, represented as a graph. Our first set of results focuses on solving ILP instances through the use of a graph parameter called clique-width, which can be seen as an extension of treewidth which also captures well-structured dense graphs. In particular, we obtain a polynomial-time algorithm for instances of bounded clique-width whose domain and coefficients are polynomially bounded by the input size, and we complement this positive result by a number of algorithmic lower bounds. Afterwards, we turn our attention to ILPs with acyclic variable interactions. In this setting, we obtain a complexity map for the problem with respect to the graph representation used and restrictions on the encoding.

1 Introduction

Integer Linear Programming (ILP) is the archetypical NP-complete optimization problem and is used in a multitude of applications in diverse areas of artificial intelligence. In particular, a wide range of problems in artificial intelligence can be efficiently solved in practice via a translation into an ILP instance, including problems from areas such as process scheduling [Floudas and Lin, 2005], planning [Vossen *et al.*, 1999; van den Briel *et al.*, 2005], vehicle routing [Toth and Vigo, 2001], packing [Lodi *et al.*, 2002], and network hub location [Alumur and Kara, 2008].

Recently, a number of papers have explored the complexity of ILP instances where the interactions between variables have specific forms of structure, captured in terms of structural parameters such as *primal treewidth* [Jansen and Kratsch, 2015], *primal treedepth* [Ganian and Ordyniak, 2018], *torso-width* [Ganian *et al.*, 2017], and *fracture backdoors* [Dvořák *et al.*, 2017]. All of these parameters give rise to polynomial (and also fixed-parameter [Downey and Fellows, 2013; Cygan *et al.*, 2015]) algorithms for solving ILP, under the condition that either the domain of variables (primal treewidth, torso-width) or the size of the coefficients (fracture backdoors, primal treedepth) in the ILP are bounded.

In this paper, we turn our attention to the significantly less explored case of using the structure of variable interactions for solving ILPs where neither the domain nor the coefficients are bounded by a fixed constant or a parameter (as per the parameterized complexity setting). On the other hand, due to the trivial structure of ILP instances which encode weakly NP-hard problems such as SUBSET SUM, one cannot hope to obtain polynomial algorithms even for very simple ILPs encoded in binary. Hence in this paper we consider variants of ILP where both the domain and the coefficients are not bounded to only a fixed number of options, but where either one or both of these components are encoded in unary. We note that such ILPs arise naturally, e.g., when encoding combinatorial problems as ILPs (consider for instance VERTEX COVER or vehicle routing problems). To this end, we distinguish UNARY ILP (coefficients and domains are encoded in unary and bounded by the input size), UNARY-COEFFICIENT ILP (coefficients are encoded in unary), and UNARY-DOMAIN ILP (domains are encoded in unary and bounded by the input size).

The first series of results focus on solving ILP using the structural parameter *clique-width*. Unlike parameters such as treewidth and treedepth, clique-width can remain bounded even on well-structured instances that are very dense, i.e., have a high number of variable-constraint interactions. Clique-width (and in particular its "signed" variant) has been successfully applied in areas such as Boolean Satisfiability [Fischer *et al.*, 2008], Answer Set Programming [Bliem *et al.*, 2016], and Graph Algorithms [Courcelle *et al.*, 2000]. Since clique-width originates from the graph setting, one needs to specify the graph representation of ILPs where clique-width will be measured: two graph representations of ILP instances are used in this context, namely the *primal graph* (which captures variable-variable interactions) and the *incidence graph* (which captures variable-constraint

	Clique-width	Primal Ac.	Incidence Ac.	
			FEAS	OPT
UNARY	P*	P	P	P
UNARY-C	NP-hard	NP-hard*	P*	NP-hard*
UNARY-D	NP-hard	P	P*	NP-hard*

Table 1: The complexity map for UNARY, UNARY-COEFFICIENT, UNARY-DOMAIN ILP (in the three rows). The first column contains results for bounded signed incidence clique-width (Theorem 1, Lemma 2). The second column contains results for instances whose primal graph is acyclic (Theorem 5). The third (Theorem 7) and fourth (Lemmas 8 and 9) column both present results for instances with an acyclic incidence graph: here we distinguish between ILP feasibility (i.e., whether there exists a feasible assignment) and ILP optimization (i.e., finding an optimal feasible assignment). Entirely new results covered in this work are marked with *.

interactions) [Ganian *et al.*, 2017; Samer and Szeider, 2010].

Our main result involving clique-width is a polynomial-time algorithm for UNARY ILP on instances of bounded signed clique-width of the incidence graph representation. The algorithm is based on dynamic programming, but requires careful introduction of data records tailored to ILP which are fundamentally different from those used, e.g., for Boolean Satisfiability [Fischer *et al.*, 2008]. We complement the algorithm with lower bounds which show that the result is essentially tight. In particular, we show that UNARY ILP remains NP-hard when the clique-width of the primal graph is used (Lemma 4), UNARY ILP remains NP-hard when the unsigned clique-width[1] of the incidence graph is used (Lemma 4), and finally both UNARY-COEFFICIENT ILP and UNARY-DOMAIN ILP are NP-hard when the signed incidence clique-width is bounded (Lemma 2).

In the second part of the paper, we turn our attention to acyclic ILP instances, i.e., ILP instances whose primal and/or incidence graph is a forest. In this context, we obtain new algorithms and hardness results in order to complete the complexity landscape for UNARY ILP, UNARY-COEFFICIENT ILP and UNARY-DOMAIN ILP on acyclic instances. We present a summary of our results in Table 1.

We note that the NP-hardness proof for UNARY-COEFFICIENT ILP presented in Theorem 5 is of particular interest—surprisingly, it shows that already deciding feasibility on instances whose primal graph is a star is NP-hard. The proof uses a nontrivial reduction from 3-SAT which uses unary-encoded coefficients to force very high domain values in a way which encodes variable assignments of the SAT instance. Another interesting observation is that in some cases, having an acyclic primal graph allows for a polynomial-time algorithm even though an acyclic incidence graph does not, while in other cases the situation is reversed. Finally, we note that Lemma 7 shows an even stronger result: deciding whether an ILP is feasible is polynomial-time solvable whenever the incidence graph is acyclic, even if the instance is encoded in binary.

Related Work and Technical Remarks. Two of the structural parameters studied in previous works give rise to polynomial-time algorithms even when the original restrictions to bounded domain and/or coefficients are relaxed to weaker restrictions in terms of the encoding. In particular, UNARY-DOMAIN ILP instances of bounded primal treewidth can be solved in polynomial-time [Jansen and Kratsch, 2015] and UNARY ILP is polynomial-time tractable on instances of bounded incidence treewidth [Ganian *et al.*, 2017]. On the other hand, UNARY-COEFFICIENT ILP remains NP-hard even on instances of bounded primal/incidence treewidth [Ganian *et al.*, 2017; Ganian and Ordyniak, 2018] and UNARY-DOMAIN ILP is NP-hard on instances of bounded incidence treewidth [Ganian *et al.*, 2017].

Clique-width and the associated *k-expressions* used for dynamic programming are NP-hard to compute [Fellows *et al.*, 2009], which might be considered as an obstacle towards practical applications. Furthermore, while it is possible to check in polynomial time whether the clique-width of a graph is bounded by a fixed k, these algorithms are infeasible in practice and involve an approximation error that is exponential in k [Oum and Seymour, 2006]. Recently, SAT solvers have been used to compute the clique-width of graphs [Heule and Szeider, 2015]. In some cases, a suitable k-expression witnessing a bound on the clique-width might already be part of the input (this is the case, e.g., for certain applications in the area of verification [Fischer *et al.*, 2008, Section1.4]). In line with previous work on clique-width [Fischer *et al.*, 2008; Bliem *et al.*, 2016; Courcelle *et al.*, 2000], we will generally assume in our theorems that a suitable k-expression is provided as part of the input.

We note that the paper considers ILP instances in inequality form, i.e., each instance consists of a set of inequalities (see Subsection 2.1). Other definitions of the ILP problem exist, and the transformations necessary to convert an instance from one definition to another can affect the structural properties of the instance; this is particularly true for determining whether an ILP instance is acyclic. For example, an instance of ILP in equality form (i.e., all constraints are equalities) containing a single equality has an acyclic incidence graph representation, but would no longer be acyclic if we convert it to an inequality form. On the other hand, the property of having bounded (signed) clique-width is preserved regardless of the used formulation.

2 Preliminaries

We will use standard graph terminology, see for instance the handbook by Diestel [Diestel, 2012]. An undirected 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. All our graphs are simple and loopless. For natural numbers $i < j$, we set $[i, j] = \{i, \ldots, j\}$, $[i] = [1, i]$ and $\pm[i] = [-i, i]$.

2.1 Integer Linear Programming

It will be useful to view an ILP instance as a set of linear inequalities (constraints) rather than using the constraint matrix. Formally, let an ILP instance I be a tuple $(X, \mathrm{dom}, \mathcal{F}, \epsilon)$ where X is a set of variables, $\mathrm{dom} : X \to [\mathbb{Z} \cup \{-\infty\}, \mathbb{Z} \cup \{\infty\}]$ is a function which maps each variable to an interval, $\mathcal{F}$ is a set of linear inequalities over $X = x_1, \ldots, x_n$ and ϵ is a

[1] Unlike signed clique-width, unsigned clique-width does not capture information about how variables interact with constraints.

linear function over X of the form $\epsilon(X) = s_1x_1 + \cdots + s_nx_n$. Each inequality $A \in \mathcal{F}$ is assumed to be of the form $c_{A,1}x_{A,1} + \cdots + c_{A,j}x_{A,j} \leq b_A$ where the coefficients $c_{A,i}$ are non-zero; the set of variables which occur in A is denoted $\text{var}(A)$, and we let $\text{var}(I) = X$. We call $\text{dom}(x)$ the *domain* of x, elements of $\mathcal{F}$ are the *constraints*, and the numbers s_i, $c_{A,i}$ and b_A are called the *coefficients*.

A *(partial) assignment* $\alpha\colon X \to \mathbb{Z}$ is a (partial) mapping such that $\alpha(x) \in \text{dom}(x)$ for all x to which α assigns a value. For a (partial) assignment α and an inequality A, we denote by $A(\alpha)$ the left-side value of A obtained by applying α (specifically, the part of α which intersects with A); for instance, if α is a partial assignment which only maps a single variable x to the value 3 and A is the inequality $2x - y - z \leq 7$, then $A(\alpha) = 6$. Formally, $A(\alpha) = c_{A,1}\alpha(x_{A,1}) + \cdots + c_{A,n}\alpha(x_{A,n})$, where undefined values of α are treated as 0; this means that if $\text{var}(A)$ has an empty intersection with the domain of α, then $A(\alpha) = 0$. Similarly, we set $\epsilon(\alpha)$ to be the value of ϵ obtained by applying α, i.e., $\epsilon(\alpha) = s_1\alpha(x_1) + \cdots + s_n\alpha(x_n)$.

An assignment α is called a *feasible assignment* or a *solution* 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 an *optimal solution* if the value of $\epsilon(\alpha)$ is maximized over all solutions. Given an instance I, the task in the ILP problem is to compute an optimal solution for I if one exists, and otherwise to decide whether there exists a feasible assignment.

In this paper, we focus on ILP instances which are either completely or partially encoded in unary. We distinguish:

- UNARY ILP, where both the domain and the coefficients are bounded by the input size and encoded in unary.
- UNARY-COEFFICIENT ILP, where the coefficients are encoded in unary; the domain is encoded in binary and can be unbounded.
- UNARY-DOMAIN ILP, where the domain is bounded by the input size and encoded in unary; coefficients are encoded in binary.

For example, consider an instance I of UNARY ILP and let the unary encoding size $|A|$ of an inequality A of the form $c_{A,1}x_{A,1} + \cdots + c_{A,q}x_{A,q} \leq b_A$ be defined as $|A| = |b_A| + \sum_{i\in[q]}(|c_{A,i}| + 1)$. Then the input size of I is $|I| = (\sum_{i\in|\text{var}(I)|} |s_i| + (\max_{d\in\text{dom}(i)} |d|) + 1) + (\sum_{A\in\mathcal{F}} |A|)$. For each of the above problems, the FEASIBILITY variant is the restriction of the chosen problem to inputs where ϵ is empty.

There are several ways of naturally representing (the variable interactions in) ILP instances as graphs. Given an ILP instance $I = (X, \text{dom}, \mathcal{F}, \epsilon)$, the (simplified) *primal graph* [Ganian *et al.*, 2017] of I is the graph whose vertex set is the set $\text{var}(I)$, and two vertices a, b are adjacent iff there exists some $A \in \mathcal{F}$ containing both a and b. The *incidence graph* [Ganian *et al.*, 2017] of I is the graph whose vertex set is $\text{var}(I) \cup \mathcal{F}$ and two vertices a, b are adjacent iff $a \in \text{var}(I)$, $b \in \mathcal{F}$ and $a \in \text{var}(b)$. Finally, the *signed incidence graph* of I is the incidence graph of I where an edge between constraint A and variable $x_{A,j}$ carries the sign $c_{A,j}$; observe that each edge e in the signed incidence graph carries precisely one sign. We call graphs where all edges carry a sign *signed graphs*. Given an ILP instance I, let G_I, H_I denote the primal graph and the signed incidence graph of I, respectively.

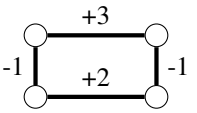

Figure 1: An example of a graph with clique-width 2 and signed clique-width 4.

2.2 Clique-width

Let k be a positive integer. A k-graph is a graph whose vertices are labeled by $[k]$; formally, the graph is equipped with a labeling function $\gamma\colon V(G) \to [k]$, and we also use $\gamma^{-1}(i)$ to denote the set of vertices labeled i for $i \in [k]$. We consider an arbitrary graph as a k-graph with all vertices labeled by 1. We call the k-graph consisting of exactly one vertex v (say, labeled by i) an initial k-graph and denote it by $i(v)$. The clique-width of a graph G is the smallest integer k such that G can be constructed from initial k-graphs by means of repeated application of the following three operations:

1. Disjoint union (denoted by $\oplus$);
2. Relabeling: changing all labels i to j (denoted by $p_{i\to j}$);
3. Edge insertion: adding an edge between each vertex labeled by i and each vertex labeled by j, where $i \neq j$ (denoted by $\eta_{i,j}$ or $\eta_{j,i}$).

A construction of a k-graph G using the above operations can be represented by an algebraic term composed of $\oplus$, $p_{i\to j}$ and $\eta_{i,j}$ (where $i \neq j$ and $i, j \in [k]$). Such a term is called a k-expression defining G, and the *clique-width* of G is the smallest integer k such that G can be defined by a k-expression. Many graph classes are known to have bounded clique-width; examples include all graph classes of bounded treewidth [Courcelle and Olariu, 2000] and co-graphs [Courcelle and Olariu, 2000].

A *k-expression tree* (also called parse trees in the literature [Courcelle *et al.*, 2000]) is a rooted tree representation of a k-expression; specifically, the k-expression tree can be built from a k-expression in a leaves-to-root fashion by using a leaf to represent each $i(v)$, each $\oplus$ operator is represented by an $\oplus$ node with two children, and each $p_{i\to j}$ and $\eta_{j,i}$ operator is represented by a corresponding node with a single child.

If the edges of G have signs, then one can define two different variants of clique-width for G. The *unsigned clique-width* of G is simply the clique-width of the graph G' obtained by removing all signs on the edges of G. On the other hand, the *signed clique-width* of G is the minimum k such that G can be defined by a *signed k-expression*, which is analogous to a k-expression with the sole distinction that the operation $\eta_{i,j}$ is replaced by $\eta^{\ell}_{i,j}$ which adds an edge with sign ℓ between all vertices labeled i and j.

We list some known facts about clique-width below:

- The difference between the signed clique-width (scw) and unsigned clique-width (cw) of a signed graph G can be arbitrarily large [Bliem *et al.*, 2016].

- A signed k-expression of a bipartite signed graph G with bi-partition V_1, V_2 can be converted to a signed $(k+1)$-expression of G such that the labels used for V_1 are completely disjoint from those used for V_2 (this is because any label that was originally used for V_1 and V_2 cannot be used to create new edges).

As per the above observation, from now on we will assume that, at each node of a signed k-expression tree, a label is either a variable-label (i.e., used exclusively by variables) or a constraint-label (i.e., used exclusively by constraints).

3 ILP on Graphs of Bounded Signed Clique-width

In this section, we obtain our polynomial-time algorithm for solving UNARY ILP using signed incidence clique-width. We then turn our attention to providing lower bounds which complement the algorithmic result.

Theorem 1. *There exists an algorithm which takes as input an instance I of* UNARY ILP *and a signed k-expression tree T of H_I, runs in time $\mathcal{O}(|I|^{4k} \cdot |T|)$, and solves* UNARY ILP.

At its core, the algorithm performs dynamic programming along T. Let t be a node of T, and recall that t could be one of the following four types of nodes: $i(v)$, $\oplus$, $\eta_{i,j}$ or $p_{i \to j}$. Let T_t be the subtree of T rooted at t, and let $G(t)$ be signed graph defined by the signed k-expression tree T_t; furthermore, let X_t and $\mathcal{F}_t$ denote the variable set and set of constraints of $G(t)$, respectively, and let $\omega(X_t)$ and $\omega(\mathcal{F}_t)$ denote the set of variable-labels and the set of constraint-labels. For instance, if r is the root of T then $G(r) = H_I$, and for each leaf t in T it holds that $G(t)$ is a graph with a single labeled vertex (which may be either a variable or a constraint).

3.1 Towards a Proof of Theorem 1

The high-level idea of the algorithm is to compute certain information (a *data table*) about our graph $G(t)$ in a leaf-to-root fashion, and once we reach the root this information allows us to output an optimal solution to the ILP instance. Let us first provide an informal description of the data table before giving the formal definition. Consider a graph $G(t)$ defined by the k-expression tree T_t. Our data table will remember a set of *signatures*, where each signature will contain all the information we need to store about one possible set of partial assignments of X_t.

Let $\varrho = |I|$. We can now formally define the notion of *signature*: a signature ϕ is a tuple (P, Q) where $P : \omega(X_t) \to \pm[\varrho]$ maps each variable-label to an integer, and $Q : \omega(\mathcal{F}_t) \to \left[-\infty, \pm[\varrho^2]\right]$ maps each constraint-label to an interval whose left endpoint is $-\infty$ and right endpoint is an integer. Observe that since the number of labels is upper-bounded by k and each label is either a variable-label or constraint-label, the total number of possible signatures is upper-bounded by ϱ^{2k}. Let Φ_t be the set of all possible signatures at node t.

Our next step is to link signatures with partial assignments; this will also define the semantics behind the definition of a signature. Informally, an assignment α of X_t matches the signature $\phi = (P, Q)$ ($\alpha \models \phi$) if P captures the sum of all variables for each individual variable-label and Q specifies

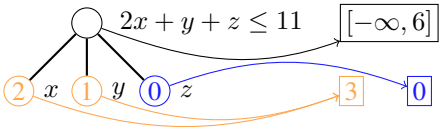

Figure 2: A small ILP instance with variables x, y, z, with labels on the vertices of the incidence graph represented by orange, blue and black. The signature (P, Q) assigns $P(\text{orange}) = 3$, $P(\text{blue}) = 0$, and $Q(\text{black}) = [-\infty, 6]$. An assignment α that is compatible with (P, Q) is represented by the numbers in the variable vertices.

the "offset" under which each constraint of a given constraint-label is satisfied. Formally, an assignment α of X_t matches the signature $\phi = (P, Q)$ if the following holds:

1. $\forall i \in \omega(X_t) : P(i) = \sum_{x \in \gamma^{-1}(i)} \alpha(x)$,
2. $\forall j \in \omega(\mathcal{F}_t)\, \forall A \in \gamma^{-1}(j)\, \forall h \in Q(j) : A(\alpha) + h \leq b_A$, and
3. $\forall j \in \omega(\mathcal{F}_t)\, \exists A \in \gamma^{-1}(j)\, \forall h \notin Q(j) : A(\alpha) + h > b_A$.

The first condition ensures that $P(i)$ captures the sum of the variables with variable-label i, the second condition ensures that $Q(j)$ captures how much each constraint with constraint-label j can change while being satisfied, and the third condition ensures that $Q(j)$ is the maximum interval with the desired property. If a variable-label i is empty then we let $P(i) = 0$, and if a constraint-label j is empty then we let $Q(j) = [-\infty, \varrho^2]$. Note that the restriction to $[-\infty, \varrho^2]$ is justified by the fact that $|A(\alpha)|$ is upper-bounded by ϱ^2.

It follows from the definition that an assignment of X_t cannot have more than one signature. However, it is not difficult to show that each assignment of X_t has precisely one signature, i.e., that the bounds on P and Q can never be exceeded.

Observation 1. Let α be an assignment of X_t. Then there exists a signature ϕ such that $\alpha \models \phi$.

Having established the notion of a signature, we can now define the data table used in the dynamic programming algorithm. For a node t of T, we let DT_t be a mapping from each signature ϕ in Φ_t to a partial assignment which matches ϕ and maximizes ϵ; formally, $\text{DT}_t(\phi \in \Phi_t) = \alpha : (\alpha \models \phi) \wedge (\forall \alpha' \models \phi : \epsilon(\alpha') \leq \epsilon(\alpha))$. If there is no partial assignment α which matches ϕ, then we set $\text{DT}_t(\phi) = \emptyset$.

Proof of Theorem 1. We compute the data table DT_t in a leaf-to-root fashion along the signed k-expression tree T of H_I. Once we reach the root r of T, then we can output an optimal solution to I as follows. Let a signature $\phi = (P, Q)$ be *good* if for all constraint-labels i, $0 \in Q(i)$. If $\text{DT}_r(\phi) = \emptyset$ for every good signature ϕ, then there exists no feasible assignment for I. Otherwise, we loop over all good signatures ϕ, compute the value of $\epsilon(\text{DT}_r(\phi))$, and output an assignment $\alpha_{opt} = \text{DT}_r(\phi_{opt})$ with maximum $\epsilon(\text{DT}_r(\phi_{opt}))$.

For completeness, we remark that such an assignment can be obtained from DT_r in time ϱ^{2k} and that correctness follows from Observation 1. Indeed, we are outputting an assignment which maximizes ϵ over all assignments which match a signature in Φ_t, and Observation 1 establishes that each assignment has a signature in Φ_t. So, all that remains is to show how to compute DT_t for each node t in T. We note that computing

DT_t for $i(v)$ nodes and $p_{i\to j}$ nodes is straightforward, and so here we only focus on the remaining two cases.

If t is a $\oplus$ node: Let u and w be the two children of t. For each pair of signatures $\phi_u = (P_u, Q_u) \in \Phi_u$ and $\phi_w = (P_w, Q_w) \in \Phi_w$ we construct a signature $\phi_{uw} = (P, Q) \in \Phi_t$ as follows:

- for each variable-label i, $P(i) = P_u(i) + P_w(i)$, and
- for each constraint-label j, $Q(j) = Q_u(j) \cap Q_w(j)$.

Intuitively, the signature ϕ_{uw} is the signature of assignments of X_t which can be obtained by taking the disjoint union of partial assignments matching ϕ_u and ϕ_w. To avoid confusion, we remark that a signature $\phi \in \Phi_t$ can be constructed in the above manner from more than a single tuple of signature ϕ_u and ϕ_w; in other words, there may exist $u' \neq u$ and/or $w' \neq w$ such that $\phi = \phi_{uw} = \phi_{u'w'}$. In this case, we will proceed by keeping the assignment compatible with ϕ which optimizes ϵ, regardless of how ϕ was constructed.

We start with an empty data table DT_t. Then, for each ϕ_{uw} constructed as above, we compare the assignment that is currently stored in $\mathrm{DT}_t(\phi_{uw})$ with the one obtained by merging ϕ_u and ϕ_w and store the better one in DT_t. More precisely, if $\epsilon(\mathrm{DT}_t(\phi_{uw})) \geq \epsilon(\mathrm{DT}_u(\phi_u)) + \epsilon(\mathrm{DT}_w(\phi_w))$ then $\mathrm{DT}_t(\phi_{uw})$ remains unchanged, and otherwise (or if $\mathrm{DT}_t(\phi_{uw}) = \emptyset$) we set $\mathrm{DT}_t(\phi_{uw}) = \mathrm{DT}_u(\phi_u) \cup \mathrm{DT}_w(\phi_w)$.

If t is a $\eta^{\ell}_{i,j}$ node: Let u be the child of t, and assume w.l.o.g. that i is a variable-label and j is a constraint-label. For each $\phi' = (P', Q') \in \Phi_u$, we construct a signature $\phi = (P', Q)$ by retaining all entries except for $Q'(j)$; to obtain the value of $Q(j)$, we alter the interval $Q'(j)$ by subtracting the value $\ell \cdot P'(i)$ (from the right side of the interval), i.e., $Q(j) = (Q'(j) - \ell \cdot P'(i))$. To populate the data table DT_t, we proceed similarly as in the case of $\oplus$ nodes. We begin with an empty DT_t, and for each $\phi' \in \Phi_u$ we compute ϕ as above and compare $\epsilon(\mathrm{DT}_u(\phi'))$ with $\epsilon(\mathrm{DT}_t(\phi))$: if $\epsilon(\mathrm{DT}_u(\phi')) \geq \epsilon(\mathrm{DT}_t(\phi))$ or if $\mathrm{DT}_t(\phi)$ is undefined then we set $\mathrm{DT}_t(\phi) = \mathrm{DT}_u(\phi')$, and otherwise we leave $\mathrm{DT}_t(\phi)$ unchanged.

Running time: Each of the operations carried out at the nodes of T can be completed in time ϱ^{2k} with the exception of the operations for $\oplus$ nodes, which take time at most ϱ^{4k}. The running time of the algorithm follows. □

3.2 Lower Bounds

We begin by noting that the algorithm cannot be extended to work on either UNARY-COEFFICIENT ILP or UNARY-DOMAIN ILP—in other words, signed incidence clique-width does not give rise to a polynomial-time algorithm if either the domain or the constraints are encoded in binary.

Lemma 2. UNARY-DOMAIN ILP FEASIBILITY *and* UNARY-COEFFICIENT ILP FEASIBILITY *are both* NP*-hard when restricted to instances of signed incidence clique-width at most* 5*, even if a signed* 5*-expression is given in the input.*

Proof Sketch. For the first claim, we give a simple polynomial reduction from the SUBSET SUM problem, which is well known to be weakly NP-complete. The resulting instance I of UNARY-DOMAIN ILP FEASIBILITY has two constraints which model the desired equality, and it is easy to verify that H_I has signed clique-width at most 4. In order to establish the second claim, it suffices to observe that the instances produced by the NP-hardness proof presented in the recent work by Ganian and Ordyniak (2018, Theorem 13) have signed clique-width at most 5. □

Next, we rule out the existence of algorithms with a uniformly polynomial running time for UNARY ILP on instances of bounded signed clique-width; in the language of *parameterized complexity theory* [Downey and Fellows, 2013; Cygan *et al.*, 2015], we show that the problem is W[1]-hard.

Lemma 3. UNARY ILP FEASIBILITY *is* W[1]*-hard parameterized by signed incidence clique-width, even if an optimal signed k-expression is given as part of the input.*

Proof Sketch. One can observe that the ILP instances obtained in the W[1]-hardness proof by Ganian and Ordyniak (2016) have bounded signed incidence clique-width. □

Our final result rules out the use of unsigned clique-width.

Lemma 4. UNARY ILP *is* NP*-hard when restricted to instances of unsigned incidence clique-width at most* 2*, even if a* 2*-expression is given as part of the input. Furthermore,* UNARY ILP *is* NP*-hard when restricted to instances of primal clique-width at most* 2*, even if a* 2*-expression is given as part of the input.*

Proof. We reduce from the well-known VERTEX COVER problem. Given an m-vertex graph G and an upper bound ℓ on the size of the vertex cover, we construct an ILP instance I as follows. For each vertex v_i we create a variable x_i where $\mathrm{dom}(x_i) = [0, 1]$. For each edge v_iv_j, we create the constraint $m \cdot x_i + m \cdot x_j + \sum_{x_p \in \mathrm{var}(I) \setminus \{x_i, x_j\}} x_p \geq m$. Finally, we set $\epsilon = \sum_{x_p \in X} -x_p$. Since each constraint can only be satisfied if at least one of x_i, x_j is set to 1, it follows that (G, ℓ) is a YES-instance if and only if there exists a solution α for I such that $\epsilon(\alpha) \geq -\ell$. To conclude the proof, observe that H_I is a complete bipartite graph, G_I is a complete graph, and the size of the unary encoding of I is upper-bounded by a polynomial of $|V(G)| + \ell$. □

4 Results for Acyclic ILPs

This section is dedicated to obtaining a complexity classification for ILPs with an acyclic graph representation.

4.1 ILPs with an Acyclic Primal Graph

First, observe that UNARY-DOMAIN ILP (and hence also UNARY ILP) can be solved in polynomial time if the primal graph is a tree by applying the dynamic programming algorithm of Jansen and Kratsch (2015). We now proceed to our main lower-bound result for this section, Theorem 5.

Theorem 5. UNARY-COEFFICIENT ILP FEASIBILITY *is* NP*-complete even on instances whose primal graph is a star.*

Proof Sketch. Clearly, UNARY-COEFFICIENT ILP FEASIBILITY is in NP, since ILP FEASIBILITY is in NP. To prove NP-hardness, we will give a polynomial time reduction from the well-known NP-complete problem 3-SAT.

Let F be an instance of 3-SAT with n variables $u_1, \ldots, u_n$ and m clauses $C_1, \ldots, C_m$. We will define an instance I of

UNARY ILP with $1+n+m$ variables and $2(n+m)$ inequalities, whose primal graph is the star S_{n+m} such that there exists a feasible assignment for I if and only if F is a satisfiable instance of 3-SAT. Note that we can assume that none of the clauses C_i contains a variable along with its negation.

The reduced instance $I = (X, \mathrm{dom}, \mathcal{F}, \emptyset)$ is defined as follows. We set $X = \{y\} \cup \{x_1, \ldots, x_n\} \cup \{z_1, \ldots, z_m\}$ and $\mathrm{dom}(x) = [-\infty, \infty]$ for every variable $x \in X$. In what follows let p_i be the i-th prime number. Then, $\mathcal{F}$ contains the following inequalities (variables marked in **bold**):

- $\forall i \in [n]$, $\mathcal{F}$ contains $\mathbf{y} - p_i\mathbf{x_i} \leq 1$ and $p_i\mathbf{x_i} - \mathbf{y} \leq 0$.

Note that since these are the only inequalities containing x_i, there exists a feasible assignment for x_i if and only if y is assigned a number that has a remainder of 0 or 1 modulo p_i.

- $\forall i \in [m]$, $\mathcal{F}$ contains $\mathbf{y} - p_r p_s p_t \mathbf{z_i} \leq d_i + p_r p_s p_t - 1$ and $p_r p_s p_t \mathbf{z_i} - \mathbf{y} \leq -d_i - 1$, where
  - r, s, t are such that C_m is a clause over the variables u_r, u_s, u_t, and
  - let $u_r \mapsto b_r$, $u_s \mapsto b_s$, $u_t \mapsto b_t$ be the assignment which falsifies C_m. Then d_i is the unique integer in $[0, p_r p_s p_t - 1]$ such that $d_i \equiv b_r \mod p_r$, $d_i \equiv b_s \mod p_s$, $d_i \equiv b_t \mod p_t$.

We note that the existence and uniqueness of d_i follows directly from the Chinese Remainder Theorem. Observe that for any partial assignment α such that $\alpha(y) \not\equiv d_i \mod p_r p_s p_t$, it is easy to compute a value of $\alpha(z_i)$ such that the two inequalities where z_i appears are satisfied.

It follows from the Prime Number Theorem that $p_i = \mathcal{O}(i \log(i))$. Hence, the absolute value of the largest coefficient in I is upper-bounded by $2p_n^3 = \mathcal{O}(n^4)$, and in particular I is an instance of UNARY-COEFFICIENT ILP FEASIBILITY whose size is polynomial in $|F|$. The primal graph of I is a star with center y and leaves $x_1, \ldots, x_n, z_1, \ldots, z_m$. Now, it suffices to show that there is a satisfying assignment for F iff there is a feasible assignment for I. □

The reduction provided in Theorem 5 immediately yields the following lower bound based on the Exponential Time Hypothesis (ETH) [Impagliazzo *et al.*, 2001].

Corollary 6. *Unless the Exponential Time Hypothesis fails, there is no $2^{o(n+m)}$ algorithm solving* UNARY-COEFFICIENT ILP FEASIBILITY *with n variables and m inequalities, even on instances whose primal graph is a star.*

4.2 ILP on Tree-like Incidence Graph

Theorem 7. ILP FEASIBILITY *restricted to instances with an acyclic incidence graph is polynomial-time solvable.*

Proof Sketch. Let $I = (X, \mathrm{dom}, \mathcal{F}, \emptyset)$ be the input instance, and recall that H_I is the incidence graph of I, i.e., a tree. Let us set an arbitrary variable as the root r of H_I. The algorithm works as follows: It takes an inequality A that is furthest from r, computes the domain restrictions imposed by A on the unique variable $x \in \mathrm{var}(A)$ that lies on the path between A and r (i.e., x is the parent of A), and then it removes A and all variables in $\mathrm{var}(A) \setminus \{x\}$ from H_I. The algorithm proceeds in this manner until only r is left. If the domain of r is not empty, the algorithm outputs yes, otherwise it outputs no. The proof of correctness and the step of simplifying an instance by removing an inequality can be handled by induction on the number of inequalities. □

Lemma 8. UNARY-COEFFICIENT ILP *is* NP*-complete on instances whose incidence graph is a tree of depth* 2*.*

Proof Sketch. The proof follows a similar strategy as the proof of Theorem 5. As before, we will reduce from a 3-SAT instance F with n variables $u_1, \ldots, u_n$ and m clauses $C_1, \ldots, C_m$ and we will have a central variable y in the ILP instance I such that $u_i \mapsto y \mod p_i$ should satisfy F.

However, here we are not allowed to specify both lower and upper bounds for the same pair of variables of ILP, as this would introduce cycles in the incidence graph. So instead of specifying both bounds, we introduce new variables $x_1^0, \ldots, x_n^0, z_1^0, \ldots, z_m^0$, which represent the remainder of y after dividing, respectively, by p_i's for variables $(x_1, \ldots, x_n)$ and by $p_r \cdot p_s \cdot p_t$ for clauses $(z_1, \ldots, z_m)$ with variables u_r, u_s, u_t. Furthermore, to achieve equality, we will set ϵ so that it maximizes the sum of all left sides of the inequalities in I. Observe that while a feasible assignment of I can satisfy an inequality without having the left side equal to its right side, the optimization function only achieves the value $\sum_{A \in \mathcal{F}} b_A$ if and only if all inequalities are tight, i.e., their left side equals the right side; in all other cases, the optimization function will achieve a value lower than $\sum_{A \in \mathcal{F}} b_A$. In the end, we get an ILP instance I that has a solution α such that $\epsilon(\alpha) \geq \sum_{A \in \mathcal{F}} b_A$ if and only if F is a YES-instance. □

Lemma 9. UNARY-DOMAIN ILP *is* NP*-complete, even on instances with incidence graph isomorphic to a star.*

Proof Sketch. We employ a reduction from SUBSET SUM similar to the one used in Lemma 2; however, here we use a single inequality constraint together with ϵ applied on the left side of the inequality to model the desired. □

5 Concluding Remarks

We have presented new algorithms and lower bounds for ILP in the case where rather mild restrictions on the formulation of the instance (unary encoding of the coefficients or domain) are combined with structural restrictions on variable-constraint interactions. The results naturally complement recent works on ILPs [Jansen and Kratsch, 2015; Ganian and Ordyniak, 2018; Ganian *et al.*, 2017; Dvořák *et al.*, 2017].

We believe that the algorithm using signed incidence clique-width (Theorem 1) is of particular interest; since, unlike previously considered parameters, it can deal with ILP instances which are "dense". We also consider the nontrivial reduction presented in Theorem 5 to be surprising.

Acknowledgments

Eduard Eiben was supported by Pareto-Optimal Parameterized Algorithms (ERC Starting Grant 715744). Robert Ganian is also affiliated with FI MU, Brno, Czech Republic. Dušan Knop is also affiliated with MFF UK, Praha, Czech Republic and was supported by projects NFR MULTIVAL and P202/12/G061 of GAČR.

References

[Alumur and Kara, 2008] Sibel A. Alumur and Bahar Yetis Kara. Network hub location problems: The state of the art. *European Journal of Operational Research*, 190(1):1–21, 2008.

[Bliem *et al.*, 2016] Bernhard Bliem, Sebastian Ordyniak, and Stefan Woltran. Clique-width and directed width measures for answer-set programming. In Gal A. Kaminka, Maria Fox, Paolo Bouquet, Eyke Hüllermeier, Virginia Dignum, Frank Dignum, and Frank van Harmelen, editors, *Proc. ECAI 2016*, volume 285, pages 1105–1113. IOS Press, 2016.

[Courcelle and Olariu, 2000] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[Courcelle *et al.*, 2000] Bruno Courcelle, Johann A. Makowsky, and Udi. Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[Cygan *et al.*, 2015] Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

[Diestel, 2012] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[Downey and Fellows, 2013] Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

[Dvořák *et al.*, 2017] Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In Carles Sierra, editor, *Proc. IJCAI 2017*, pages 607–613. ijcai.org, 2017.

[Fellows *et al.*, 2009] Michael R. Fellows, Frances A. Rosamond, Udi Rotics, and Stefan Szeider. Clique-width is np-complete. *SIAM J. Discrete Math.*, 23(2):909–939, 2009.

[Fischer *et al.*, 2008] Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Applied Mathematics*, 156(4):511–529, 2008.

[Floudas and Lin, 2005] ChristodoulosA. Floudas and Xiaoxia Lin. Mixed integer linear programming in process scheduling: Modeling, algorithms, and applications. *Annals of Operations Research*, 139(1):131–162, 2005.

[Ganian and Ordyniak, 2016] Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. In *Proc. AAAI 2016*, pages 710–716, 2016.

[Ganian and Ordyniak, 2018] Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ilp. *Artificial Intelligence*, 257:61 – 71, 2018.

[Ganian *et al.*, 2017] Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In Satinder P. Singh and Shaul Markovitch, editors, *Proc. AAAI 2017*, pages 815–821. AAAI Press, 2017.

[Heule and Szeider, 2015] Marijn Heule and Stefan Szeider. A SAT approach to clique-width. *ACM Trans. Comput. Log.*, 16(3):24:1–24:27, 2015.

[Impagliazzo *et al.*, 2001] Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *J. Comput. Syst. Sci.*, 63(4):512–530, 2001.

[Jansen and Kratsch, 2015] Bart M. P. Jansen and Stefan Kratsch. A structural approach to kernels for ilps: Treewidth and total unimodularity. In Nikhil Bansal and Irene Finocchi, editors, *Proc. ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 779–791. Springer, 2015.

[Lodi *et al.*, 2002] Andrea Lodi, Silvano Martello, and Michele Monaci. Two-dimensional packing problems: A survey. *European Journal of Operational Research*, 141(2):241–252, 2002.

[Oum and Seymour, 2006] Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.

[Samer and Szeider, 2010] Marko Samer and Stefan Szeider. Constraint satisfaction with bounded treewidth revisited. *J. Comput. Syst. Sci.*, 76(2):103–114, 2010.

[Toth and Vigo, 2001] Paolo Toth and Daniele Vigo, editors. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[van den Briel *et al.*, 2005] Menkes van den Briel, Thomas Vossen, and Subbarao Kambhampati. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *Proc. ICAPS 2005*, pages 310–319. AAAI, 2005.

[Vossen *et al.*, 1999] Thomas Vossen, Michael O. Ball, Amnon Lotem, and Dana S. Nau. On the use of integer programming models in AI planning. In *Proc. IJCAI 99*, pages 304–309. Morgan Kaufmann, 1999.