Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Going Beyond Primal Treewidth for (M)ILP

Robert Ganian, Sebastian Ordyniak, M. S. Ramanujan
Algorithms and Complexity group,
TU Wien, Vienna, Austria
({ganian,ordyniak, ramanujan}@ac.tuwien.ac.at)

Abstract

Integer Linear Programming (ILP) and its mixed variant (MILP) are archetypical examples of NP-complete optimization problems which have a wide range of applications in various areas of artificial intelligence. However, we still lack a thorough understanding of which structural restrictions make these problems tractable. Here we focus on structure captured via so-called decompositional parameters, which have been highly successful in fields such as boolean satisfiability and constraint satisfaction but have not yet reached their full potential in the ILP setting. In particular, primal treewidth (an established decompositional parameter) can only be algorithmically exploited to solve ILP under restricted circumstances.

Our main contribution is the introduction and algorithmic exploitation of two new decompositional parameters for ILP and MILP. The first, *torso-width*, is specifically tailored to the linear programming setting and is the first decompositional parameter which can also be used for MILP. The latter, *incidence treewidth*, is a concept which originates from boolean satisfiability but has not yet been used in the ILP setting; here we obtain a full complexity landscape mapping the precise conditions under which incidence treewidth can be used to obtain efficient algorithms. Both of these parameters overcome previous shortcomings of primal treewidth for ILP in unique ways, and consequently push the frontiers of tractability for these important problems.

Introduction

Integer Linear Programming (ILP) is a widely used and highly successful framework for solving difficult computational problems. 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 planning (van den Briel, Vossen, and Kambhampati 2005; Vossen et al. 1999), vehicle routing (Toth and Vigo 2001), process scheduling (Floudas and Lin 2005), packing (Lodi, Martello, and Monaci 2002), and network hub location (Alumur and Kara 2008). In its most general form, the ILP optimization problem (ILP) can be formalized as follows:

INTEGER LINEAR PROGRAMMING	
Input:	A matrix $A \in \mathbb{Z}^{m \times n}$ and two vectors $b \in \mathbb{Z}^m$ and $c \in \mathbb{R}^n$.
Question:	Maximize cx for every $x \in \mathbb{Z}^n$ with $Ax \leq b$.

Closely related to ILP is the mixed ILP (MILP) problem, which additionally partitions the variable set into integer and real variables. Given the significance of both problems, it is surprising that fairly little is known about which structural restrictions make them tractable. After all, problem inputs naturally tend to contain some form of structure, and such structure can often be exploited to efficiently compute exact solutions. Over the past decade, this line of research has been thoroughly investigated in related areas such as boolean satisfiability (Szeider 2003; Ganian and Szeider 2015; Ganian, Hliněný, and Obdržálek 2013) and constraint satisfaction (Samer and Szeider 2010; Ganian, Ramanujan, and Szeider 2016; Marx 2010), but has yet to reach its full potential in the ILP setting.

The parameterized complexity paradigm (Downey and Fellows 2013; Cygan et al. 2015) offers the perfect tools to measure, quantify and exploit various forms of structure in order to design efficient algorithms. There, one associates each instance I with a structural parameter k, and the goal is to obtain algorithms which run in time $f(k) \cdot |I|^{\mathcal{O}(1)}$ for some computable function f. Such algorithms are called *fixed-parameter algorithms*, and the underlying idea is that well-structured instances (i.e., those where k is small) will run in uniformly polynomial time. Whether such algorithms can be obtained or not naturally depends on our choice of parameter—corresponding to the choice of structure that we wish to algorithmically exploit.

So-called *decompositional parameters* are by far the most studied type of structural parameters: these seek to construct and exploit natural decompositions of graph representations of instances, and have been highly successful in boolean satisfiability (Szeider 2003), constraint satisfaction and optimization (Marx 2010; Gottlob et al. 2005; Marinescu and Dechter 2010) and other areas. However, the situation in the area of ILP is not as positive: already ILP-FEASIBILITY remains NP-hard (Ganian and Ordyniak 2016) even when the most popular decompositional parameters (in particular, *treewidth* (Kloks 1994) and *clique-width* (Cour-

celle, Makowsky, and Rotics 2000)) are bounded, and furthermore none of the known fixed-parameter algorithms using decompositional parameters generalize to MILP.

In this paper we overcome the shortcomings of classical decompositional parameters on ILP by taking a careful look at the graph representation used for ILP instances. All the results on the use of decompositional parameters for integer linear programming have so far used the simplest graph representation, notably so-called *primal graphs*. There, decompositional parameters in combination with additional restrictions have led to some positive results for ILP (Jansen and Kratsch 2015; Cunningham and Geelen 2007; Ganian and Ordyniak 2016). We explore two ways to go beyond simple primal graphs, and develop new algorithms which allow us to push the frontiers of tractability for ILP and MILP.

Our first contribution is the introduction of a new structural parameter called *torso-width*; unlike generic parameters such as treewidth, torso-width is specifically tailored to (mixed) integer linear programming. The core idea behind the parameter relies on a combination of pruning the original primal graph representation via Lenstra's celebrated algorithm (Lenstra and Jr. 1983) and dynamic programming. Torso-width is the first structural parameter that can be used not only for ILP but also for MILP. On ILP, torso-width is a strict generalization of primal treewidth: it allows the efficient solution of all instances where treewidth was known to help (Marx 2010), but can also solve instances which were beyond the reach of state-of-the-art algorithms.

Our second contribution is a complete complexity landscape for the parameterized complexity of ILP with respect to the treewidth of the *incidence graph representation*. Similarly to primal graphs, incidence graphs are a natural graph representation that has been extensively studied in the boolean satisfiability (Szeider 2003) and constraint satisfaction (Samer and Szeider 2010) settings. An incidence graph captures more information about the original instance than a primal graph, and as such it offers a more truthful representation. One particular advantage of using treewidth on incidence instead of primal graphs is that the treewidth of the former can be small even when constraints spanning many variables are present (on the other hand, even a single large constraint prevents the efficient use of treewidth on primal graphs). We obtain exact algorithms for ILP parameterized by the treewidth of the incidence graph under certain additional conditions on the instance, and show that these conditions are unavoidable by obtaining matching lower bounds.

The paper is structured as follows. After the necessary preliminaries, we introduce and develop our results for torso-width. The last technical section is then dedicated to the full complexity map for incidence treewidth, containing both the necessary algorithmic and lower-bound results. We conclude with a summary and discussion of our results.

Preliminaries

We will use standard graph terminology, see for instance the handbook by 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. For a subset $V' \subseteq V(G)$, we denote by $G[V']$, the *induced subgraph* of G induced by the vertices in V', i.e., $G[V']$ has vertices V' and edges $\{\{u, v\} \in E(G) : u, v \in V'\}$. A vertex v is a neighbor of a vertex set X if $v \notin X$ and there exists an edge between a vertex in X and v. The operation of *collapsing* a vertex set X, denoted $G \circ X$, deletes X from the graph and adds an edge between each pair of neighbors of X. All our graphs are simple and loopless.

(Mixed) Integer Linear Programming

For our purposes, 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 $(\mathcal{F}, \eta)$ where $\mathcal{F}$ is a set of linear inequalities over variables $X = x_1, \ldots, x_n$ and η is a linear function over X of the form $\eta(X) = s_{x_1}x_1 + \cdots + s_{x_n n}x_n$. Each inequality $A \in \mathcal{F}$ is assumed to be of the form $c_{A,1}x + c_{A,2}x_{A,2} + \cdots \leq b_A$ 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 $\text{var}(I) = X$. The *arity* of A is $|\text{var}(A)|$. 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 $Y \cap \text{var}(A) \neq \emptyset$. An integer variable x_i has *bounded domain* if there exist constants c, d such that $\mathcal{F}$ contains the inequalities $x_i \leq c$ and $x_i \geq d$; furthermore, we say that it has q-bounded domain if $c - d \leq q$.

A (partial) assignment α is a (partial) mapping from X to $\mathbb{Z}$. 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), i.e., $A(\alpha) = c_{A,1}\alpha(x_{A,1}) + c_{A,2}\alpha(x_{A,2}) + \ldots$. Note that if α is a partial assignment, it may happen that $\text{var}(A)$ has an empty intersection with the domain of α, in which case we set $A(\alpha) = 0$.

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. As in the case of constraints, for partial assignments α we set $\eta(\alpha)$ to be the value of η restricted to the domain of α, and in particular we set $\eta(\alpha) = 0$ if η has an empty intersection with the domain of α.

Analogously to the above, we will view a Mixed Integer Linear Programming (MILP) instance also as a tuple $(\mathcal{F}, \eta)$, where $\mathcal{F}$ is a set of linear inequalities over two disjoint variable-sets $X \cup Y$ and η is a linear function over $X \cup Y$. We call X the integer variables and Y the real variables, and set $X = \text{var}_{\mathbb{Z}}(I)$ and $Y = \text{var}_{\mathbb{R}}(I)$. Note that in the MILP setting, only integer variables are considered to have bounded domain.

There are several ways of naturally representing (M)ILP instances as graphs. Given an ILP instance $I = (\mathcal{F}, \eta)$, the *simplified primal graph* 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. On the other hand, the *extended primal graph* of I is the graph whose vertex set

is the set $\text{var}(I)$, 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 (Ganian and Ordyniak 2016). We will not deal with extended primal graphs (which have greater or equal treewidth than the simplified variant) in this paper, and so we will consistently use *primal graph* as shorthand for the simplified primal graph. Finally, the *incidence graph* 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 $a \in \text{var}(b)$. We denote by G_I and H_I the primal and incidence graphs of I, respectively.

Parameterized Complexity

In parameterized algorithmics (Flum and Grohe 2006; Niedermeier 2006; Downey and Fellows 2013) 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 *fixed-parameter algorithms*. Aside from the complexity class FPT, we will also make use of the complexity classes XP (containing problems that can be solved in polynomial time whenever the parameter is bounded) and paraNP (problems complete for paraNP remain NP-complete even when the parameter is bounded).

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 (Fellows et al. 2008).

Proposition 1 (Lenstra and Jr.; Kannan; Frank and Tardos (1983; 1987; 1987))**.** *A MILP instance $I = (\mathcal{F}, \eta)$ can be solved in time $\mathcal{O}(p^{2.5p+o(p)} \cdot |I| \cdot q^4)$, where $p = |\text{var}_{\mathbb{Z}}(I)|$ and $q = |\text{var}_{\mathbb{R}}(I)|$.*

Treewidth

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 $\text{tw}(G)$, is the width of an optimal tree decomposition of G.

For presenting our dynamic programming algorithms, it is convinient to consider tree decompositions in the following normal form (Kloks 1994): A tuple (T, χ) is a *nice tree decomposition* of a graph G if (T, χ) is a tree decomposition of G, the tree T is rooted at node r, and each node of T is of one of the following four types: 1) a *leaf node* is a node t having no children and $|\chi(t)| = 1$, 2) a *join node* is a node t having exactly two children t_1, t_2, and $\chi(t) = \chi(t_1) = \chi(t_2)$, 3) an *introduce node* is a node t having exactly one child t' and $\chi(t) = \chi(t') \cup \{v\}$ for a node v of G, and 4) a *forget node* is a node t having exactly one child t', and $\chi(t) = \chi(t') \setminus \{v\}$ for a node v of G.

For convenience we will also assume that $\chi(r) = \emptyset$ for the root r of T. For $t \in V(T)$ we denote by T_t the subtree of T rooted at t and we write $\chi(T_t)$ for the set $\bigcup_{t' \in V(T_t)} \chi(t')$. We conclude the preliminaries with an important proposition for treewidth.

Proposition 2 (Kloks; Bodlaender; Bodlaender et al. (1994; 1996; 2016))**.** *It is possible to compute an optimal (nice) tree-decomposition of an n-vertex graph G with treewidth k in time $k^{\mathcal{O}(k^3)}n$, and to compute a 5-approximate one in time $2^{\mathcal{O}(k)}n$. Moreover, the number of nodes in the obtained tree decompositions is at most $\mathcal{O}(kn)$.*

Torso-width

In this section we present the first of our considered parameters, torso-width. The idea underlying torso-width is to prune the used graph representation by only focusing on the parts which we can and intend to handle with a dynamic programming algorithm. We proceed with an initial batch of definitions before giving more details.

Let us fix an arbitrary constant q. For a MILP instance I, let $B_q(I)$ be the set of all q-bounded domain variables in I and let $U_q(I) = \text{var}(I) \setminus B_q(I)$; observe that $B_q(I) \subseteq \text{var}_{\mathbb{Z}}(I)$. A *$q$-torso* is a graph obtained by collapsing at least all the vertices in $U_q(I)$; formally, a graph G is a q-torso of I iff there exists $P_G \supseteq U_q(I)$ such that $G = G_I \circ P_G$. We also introduce one bit of additional notation: given a variable subset X of I, we let $I[X] = (\mathcal{F}', \eta')$ be the subinstance of I restricted to X; formally, $I[X]$ contains all constraints that do not contain any variable outside of X, and η' is the restriction of η to X.

Informally, a torso allows us to distinguish two "parts" of the MILP instance: the part that is removed from the torso by collapsing, and the part that remains in the torso. Outside the torso, we make no assumptions about the structure of the instance, and so the best known option for dealing with this part is the use of Lenstra's algorithm (Proposition 1). However, the torso itself only contains bounded-domain variables and so a treewidth-based algorithm can be used instead (Jansen and Kratsch 2015). The torso itself is then designed in a way which ensures that we can seamlessly combine these two approaches on different parts of the MILP instance, and the rest of this section is devoted to formalizing these claims through the definition of torso-width.

Before we define the torso-width formally, we first need to introduce the notion of "fitness". Let $G = G_I \circ P_G$ be a q-torso of I and let H be the vertex set of a connected component in $G_I[P_G]$ (i.e., in the collapsed part of the primal graph). Then the *fitness* of H, denoted $\tau(H)$, is the number of integer variables in H. We then let the fitness of G,

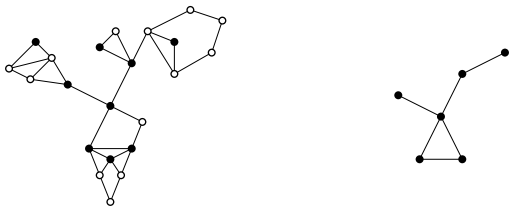

Figure 1: *Left*: The primal graph of an ILP instance, with black vertices representing q-bounded variables.
Right: One possible q-torso.

denoted $\tau(G)$, be the maximim fitness over all connected components in $G_I[P_G]$.

Now we can define the torso-width of the q-torso G as $torw(G) = \max(tw(G), \tau(G))$. Since determining whether a graph has treewidth k takes time at most $k^{\mathcal{O}(k^3)}|V(G)|$ due to Proposition 2, and since checking $\tau(G)$ takes linear time, we observe that it is possible to compute the torso-width k of a q-torso of I in time at most $k^{\mathcal{O}(k^3)}|V(G)|$. Finally, the q-torso-width of a MILP instance I, denoted $torw_q(I)$, is the minimum $torw(G)$ over all q-torsos G of I.

Before proceeding to our algorithms for torso-width, we make one useful observation. Let (T, χ) be a nice tree decomposition of G and let $\mathcal{C}(G)$ be the set of connected components of $G_I - V(G)$. Then for each $C \in \mathcal{C}(G)$, there is a bag of (T, χ) which contains all the neighbors of C.

Algorithms for Torso-width

Our next order of business is to show how torso-width can be used to solve a MILP instance, assuming that a suitable torso is provided as part of the input. We introduce some extra notation which will be useful in our proofs. Let $\zeta_T : \mathcal{C}(G) \to V(T)$ be a mapping which assigns a unique bag of (T, χ) to each connected component in $\mathcal{C}(G)$. For each $C \in \mathcal{C}(G)$, we set $\zeta_T(C)$ to be an arbitrary node $t \in V(T)$ such that (i) t is an introduce node, (ii) $\chi(t)$ contains the neighborhood of C and (iii) there is no other node t' which is a descendant of t and satisfies the first two properties. For $t \in V(T)$, we use $\omega_T(t)$ to denote the set of all connected components which ζ maps to t **extended by** $\chi(t)$; formally, $\omega_T(t) = \{\, \chi(t) \cup C : \zeta_T(C) \mapsto t \,\}$. The reference to T is dropped in the above definition when it is clear from the context.

Theorem 3. *Let q be a fixed integer and I be an input MILP instance with b real variables. Given a q-torso G of I with width k, it is possible to solve I in time $(q^{k+1} \cdot k^{\mathcal{O}(k)} + k^{\mathcal{O}(k^3)}) \cdot |I| b^4$.*

Sketch of Proof. Let (T, χ) be a nice tree decomposition of G of minimum width. Recall that by Proposition 2, (T, χ) can be computed in time $k^{\mathcal{O}(k^3)}n$. For each $t \in V(T)$, we denote by I_t the subinstance of I induced on the set $\chi(T_t) \cup \bigcup_{t' \in T_t}(\bigcup_{C \in \omega(t')} C)$.

We now proceed to describe our dynamic programming algorithm. For each $t \in V(T)$, we denote by $\mathcal{P}_t$ the set of all partial assignments of the variables in $\chi(t)$; observe that $|\mathcal{P}_t| \leq q^{k+1}$. The core idea behind the algorithm relies on computing the function $opt_t : \mathcal{P}_t \to \mathbb{R} \cup \perp$ for each $t \in V(T)$. Intuitively, opt_t maps a partial assignment α_0 of $\chi(t)$ to the optimal value of $\eta(\alpha)$ over all partial assignments α of I_t that comply with α_0. More formally, opt_t is defined as $opt_t(\tau) = \perp$ if I_t has no feasible solution extending τ and $opt_t(\tau) = \delta$ otherwise, where δ is the maximum value of η for I_t *among* all feasible assignments which extend τ. Clearly, the solution for I is $\max\{opt_t(\tau) | \tau \in \mathcal{P}_t\}$ where t is the root node of T. The remainder of the proof then shows how the function opt_t can be computed for each node $t \in V(T)$. □

Now, we turn our attention to finding a suitable torso in order to apply Theorem 3.

Lemma 4. *Let q be a fixed integer and I be an input MILP instance. It is possible to find a q-torso G of I such that $torw(G) \leq 2 \cdot torw_q(I)$ in time $\mathcal{O}(|I|)$.*

Sketch of Proof. Let G' be the trivial q-torso, which is obtained by collapsing all variables that are not q-bounded. It can be shown that G' is in fact a 2-approximation of an optimal q-torso. Obviously, G' can be computed in linear time. □

With these results in hand, we can proceed to the main result of this section, Theorem 5.

Theorem 5. *Let q be a fixed integer and I be an input MILP instance. Then I is fixed-parameter tractable parameterized by q-torso-width.*

Proof. By Lemma 4, we can compute a q-torso G of I such that $torw(G) \leq 2 \cdot torw_q(I)$. The proof then follows from Theorem 3. □

Incidence Treewidth

In this section we provide our complexity landscape for ILP w.r.t. incidence treewidth. We start by presenting our main algorithm for ILP using incidence treewidth in a very general setting, and the next section will then discuss specific tractability results that can be obtained from the algorithm as well as matching hardness results.

The Algorithm

Here we will present a bottom-up dynamic programming algorithm solving ILP along a tree decomposition of the incidence graph H_I of an instance I. In the following, we assume that we want to solve an ILP instance $I = (\mathcal{F}, \eta)$ and we denote by Γ the maximum absolute value of $A(\alpha)$ over every constraint $A \in \mathcal{F}$ and every partial assignment α of the variables of I that can be extended to a feasible assignment for I. Note that Γ could be undefined (specifically, when variable domains are unbounded); we remark that the algorithm presented below only applies for instances where Γ is defined. We will show that this actually a necessary condition to algorithmically exploit incidence treewidth.

Observe that under the natural assumption that every variable of I occurs in at least one constraint of I, Γ is also

an upper bound on the maximum domain value of any variable in a feasible assignment of I. Moreover, we denote the treewidth of H_I by tw_I, the number of variables in I by n, and the number of constraints in I by m. The remainder of this subsection is devoted to a proof of our main tractability result, formalized below.

Theorem 6. ILP *can be solved in time* $\mathcal{O}(\Gamma^{2\mathrm{tw}_I+2}\mathrm{tw}_I(n+m))$ *given that an optimal nice tree decomposition of* H_I *is provided in the input.*

Informally, the algorithm behind the above theorem works as follows. Let $\mathcal{T} = (T, \chi)$ be an optimal nice tree decomposition of H_I. The algorithm uses a bottom-up dynamic programming approach on the nodes of T to compute a compact representation, in the following represented by a set of valid records, of all feasible assignments of I restricted to variables and constraints occuring in T_t for every node $t \in V(T)$. In the following, we denote by $\chi_{\text{var}}(t)$ and $\chi_A(t)$ the function $\chi(t)$ restricted to $\text{var}(I)$ and $\mathcal{F}$, respectively, i.e., $\chi_{\text{var}}(t) = \chi(t) \cap \text{var}(I)$ and $\chi_A(t) = \chi(t) \cap \mathcal{F}$.

A *record* for a node $t \in V(T)$ is a triple (τ, γ, v), where:

- $\tau : \chi_{\text{var}}(t) \to [-\Gamma, \Gamma]$,
- $\gamma : \chi_A(t) \to [-\Gamma, \Gamma]$, and
- v is a non-negative integer.

The semantics of a record are as follows. We say that a record (τ, γ, v) for a node $t \in V(T)$ is *valid* if v is the maximum value of $\eta(\alpha)$ for any assignment $\alpha : \chi_{\text{var}}(T_t) \to [-\Gamma, \Gamma]$ satisfying:

R1 $\alpha(x) = \tau(x)$ for every variable $x \in \chi_{\text{var}}(t)$,

R2 $A(\alpha) = \gamma(A)$ for every constraint $A \in \chi_A(t)$.

For a node $t \in V(T)$ we denote by $\mathcal{R}(t)$ the set of all valid records for t. Observe that I has a feasible assignment if and only if $\mathcal{R}(r) \neq \emptyset$, and in this case the solution of I achieves the value of v in the unique record $(\emptyset, \emptyset, v) \in \mathcal{R}(r)$ for the root r of T. We will show next that $\mathcal{R}(t)$ can be computed via a dynamic programming algorithm on $\mathcal{T}$ in a bottom-up manner. The algorithm starts by computing the set of all valid records for the leaves of T and then proceeds by computing the set of all valid records for the other three types of nodes of a nice tree decomposition (whereas it selects nodes whose children have already been processed).

The following four lemmas show how this can be achieved for all of the four types of nodes in a nice tree decomposition. Here we only provide a proof for the introduce nodes.

Lemma 7. *Let* $t \in V(T)$ *be an introduce node with child* t'. *Then* $\mathcal{R}(t)$ *can be computed from* $\mathcal{R}(t')$ *in time* $O(|\mathcal{R}(t')|\Gamma)$.

Sketch of Proof. We distinguish two cases depending on whether $\Delta := \chi(t) \setminus \chi(t')$ consists of a variable or a constraint. If $\Delta = \{x\}$ for a variable $x \in \text{var}(I)$, then we obtain the set $\mathcal{R}(t)$ from the set $\mathcal{R}(t')$ as follows. For every record (τ', γ', v') in $\mathcal{R}(t')$ and every assignment $\tau_x : \{x\} \to [-\Gamma, \Gamma]$ of x, let (τ, γ, v) be the triple defined as follows:

- $\tau : \chi_{\text{var}}(t) \to [-\Gamma, \Gamma]$ to be the assignment with $\tau(x') = \tau'(x')$ for every $x' \in \chi_{\text{var}}(t')$ and $\tau(x) = \tau_x(x)$,
- $\gamma : \chi_A(t) \to [-\Gamma, \Gamma]$ to be assignment with $\gamma(A) = \gamma'(A) + A(\tau_x)$, and
- $v = v' + \eta(\tau_x)$.

If $|\gamma(A)| \leq \Gamma$ for every $A \in \chi_A(t)$, we add the record (τ, γ, v) to the set $\mathcal{R}(t)$, otherwise we skip the record. This completes the computation of $\mathcal{R}(t)$.

If, on the other hand $\Delta = \{A\}$, for $A \in \mathcal{F}$, then we obtain the set $\mathcal{R}(t)$ from the set $\mathcal{R}(t')$ as follows. For every record (τ', γ', v') in $\mathcal{R}(t')$, we check whether $|A(\tau')| \leq \Gamma$. If so we add the record (τ', γ, v') to $\mathcal{R}(t)$, where γ is defined by setting $\gamma(A') = \gamma'(A')$ for every $A' \in \chi_A(t')$ and $\gamma(A) = A(\tau')$. Otherwise we skip the record. □

Lemma 8. *Let* $l \in V(T)$ *be a leaf node. Then* $\mathcal{R}(l)$ *can be computed in time* $O(\Gamma)$.

Lemma 9. *Let* $t \in V(T)$ *be a forget node with child* t'. *Then* $\mathcal{R}(t)$ *can be computed from* $\mathcal{R}(t')$ *in time* $O(|\mathcal{R}(t')| \log |\mathcal{R}(t')|)$.

Lemma 10. *Let* $t \in V(T)$ *be a join node with children* t_1 *and* t_2. *Then* $\mathcal{R}(t)$ *can be computed from* $\mathcal{R}(t_1)$ *and* $\mathcal{R}(t_1)$ *in time* $O(|\mathcal{R}(t_1)||\mathcal{R}(t_2)|)$.

We are now ready to prove our main theorem.

Proof of Theorem 6. The algorithm computes the set of all valid records $\mathcal{R}(t)$ for every node t of T using a bottom-up dynamic programming algorithm starting in the leaves of T. It then solves I by checking whether $\mathcal{R}(r) \neq \emptyset$ and if so outputs v as the solution of I, where $(\emptyset, \emptyset, v)$ is the unique record in $\mathcal{R}(r)$. If, on the other hand, $\mathcal{R}(r) = \emptyset$, then the algorithm returns correctly that I has no feasible solution. Note that the correctness of the algorithm follows from the correctness of Lemmas 8, 7, 9, and 10. The running time of the algorithm is at most the number of nodes of T, i.e., at most $\mathrm{tw}_I(|\text{var}(I)| + |\mathcal{F}|)$, times the maximum time required to compute $\mathcal{R}(t)$ for any of the four node types of a nice tree-decomposition, which because of lemmas 8, 7, 9, and 10 is at most $O(|\mathcal{R}(t)|^2 + |\mathcal{R}(t)|\Gamma)$. Because $|\mathcal{R}(t)| \leq \Gamma^{\mathrm{tw}_I+1}$, we obtain $O((\Gamma^{\mathrm{tw}_I+1})^2\mathrm{tw}_I(|\text{var}(I)| + |\mathcal{F}|))$ as the total running time of the algorithm. □

Discussion and Hardness Results

The algorithm presented in the previous subsection can only be applied to ILP instances for which Γ is defined. Here we show that this is indeed the case for several natural classes of ILP instances, and we also show that a bound on Γ is necessary for using incidence treewidth.

Let $A_{\max}$, $B_{\max}$, and $X_{\max}$ denote the maximum absolute value of the coefficients of the matrix, the coefficients of the vector b, and the domain values of all variables, respectively. The following theorem follows immediately from Theorem 6 and the fact that Γ is at most $A_{\max} \cdot X_{\max} \cdot n$.

Theorem 11. ILP *can be solved in time* $\mathcal{O}((A_{max} \cdot X_{max} \cdot n)^{\mathrm{tw}_I+1})(n+m))$ *given that an optimal nice tree decomposition of* H_I *is provided in the input.*

Together with Proposition 2, the above theorem shows that ILP is solvable in polynomial time for ILP instances of bounded incidence treewidth as long as both $A_{\max}$ and $X_{\max}$

can be bounded by a polynomial in the input size. It is natural to ask whether the same result still holds if either $A_{\max}$ or $X_{\max}$ is not polynomially bounded. We answer this in the following theorem. In fact, the part of the theorem also holds for so-called *non-negative* instances, where all coefficients in the matrix A as well as the domain values of all variables of I are non-negative. This fact will be useful later.

Theorem 12. ILP-FEASIBILITY *is* NP*-hard even for instances with incidence treewidth at most three that additionally satisfy either:*

(1) all variables have a binary domain, i.e., $\{0, 1\}$ *and the ILP instance is non-negative, or*

(2) all coefficents of the matrix and the vector b *are between* -2 *and* 2*.*

Sketch of Proof. We show the result by a polynomial reduction from the SUBSET SUM problem, which is well-known to be weakly NP-complete. Given a set $S := \{s_1, \ldots, s_n\}$ of integers and an integer s, the SUBSET SUM problem asks whether there is a subset $S' \subseteq S$ such that $\sum_{s' \in S'} s' = s$. Let $I := (S, s)$ with $S := \{s_1, \ldots, s_n\}$ be an instance of SUBSET SUM. We show the two statements of the theorem in two steps: First we construct an ILP instance I^1 from I that verifies (1) and then we show how to alter I^1 into an ILP instance I^2 that proves (2). The ILP instance I^1 is obtained as follows. I^1 constains one binary variable x_i (with domain $\{0, 1\}$) for every i with $1 \leq i \leq n$, which indicates whether or not the element s_i is chosen to be part of the subset. Apart from the constraints $0 \leq x_i \leq 1$, which ensure that x_i is a binary variable, I^1 merely contains one additional constraint, i.e., the constraint $\sum_{1 \leq i \leq n} s_i x_i = s$. Clearly, I^1 is a YES-instance if and only if so is I, I^1 is non-negative, uses only binary variabels, and $\mathrm{tw}_I \leq 1$, which concludes the proof of (1). It now suffices to show how to alter I^1 into the desired ILP instance I^2 by replacing the large coefficients in I^1 with variables of large domain without increasing the incidence treewidth. □

For general ILP instances, the aforementioned Theorem 12 settles the question of whether we can get rid of the polynomial bounds on $A_{\max}$ and $X_{\max}$. We now focus on the aforementioned non-negative ILP instances. It is worth noting that such instances have been studied in the literature (Fomin et al. 2016). We show that Theorem 6 allows us to solve such instances more efficiently; the proof follows from Theorem 6 using the fact that Γ is at most $B_{\max}$ for non-negative ILP instances.

Theorem 13. *On non-negative instances,* ILP *can be solved in time* $\mathcal{O}((B_{max})^{\mathrm{tw}_I+1}(n + m)$ *given that an optimal nice tree decomposition of* H_I *is provided in the input.*

Together with Proposition 2, the above theorem shows that ILP is fixed-parameter tractable parameterized by tw_I and $B_{\max}$. Furthermore, for those instances where $B_{\max}$ is bounded polynomially in the number of variables n, the above theorem gives an algorithm with running time $n^{\mathcal{O}(tw)}$. We note that (unless P=NP) it is not possible to obtain fixed-parameter (or even XP) algorithms when parameterizing by only one of these parameters. On one hand, ILP remains NP-hard when parameterized only by $B_{\max}$, since it can still express problems such as VERTEX COVER. The other case is ruled out by Theorem 12 (1).

Table 2 and Table 1 summarize our results for general and non-negative ILP instances, respectively. Observe that Γ is always defined in the case of non-negative ILP instances, while general instances with undefined Γ fall into the paraNP-complete case.

parameter	complexity
tw_I	paraNP-c (Thm 12 (1))
tw_I $\left(\text{if } B_{\max} \leq \lvert I\rvert^{\mathcal{O}(1)}\right)$	XP (Thm 13)
$B_{\max}$	paraNP-c (Folklore)
$\mathrm{tw}_I + B_{\max}$	FPT (Thm 13)

Table 1: The table shows the parameterized complexity of non-negative ILP parameterized by the single parameters tw_I and $B_{\max}$ as well as the combined parameter $\mathrm{tw}_I + B_{\max}$.

polynomial bound on	complexity par. by tw_I
$A_{\max}$	paraNP-c (Thm 12 (2))
$X_{\max}$	paraNP-c (Thm 12 (1))
$A_{\max}$ and $X_{\max}$	XP (Thm 11)

Table 2: The table shows the parameterized complexity of ILP parameterized by tw_I dependening on whether $A_{\max}$, $X_{\max}$, or both are polynomially bounded in the input size.

Finally, we discuss the optimality of the algorithm of Theorem 13 in the case when $B_{\max}$ is polynomially bounded in the input size. To this end, we will use the following recent result of Fomin et al. to rule out FPT or even more efficient XP algorithms. Since incidence treewidth is upper-bounded by m, we restate their theorem in terms of incidence treewidth.

Theorem 14 (Fomin et al. (2016))**.** ILP-FEASIBILITY *cannot be solved in time* $B_{max}^{o(\mathrm{tw}_I)} n^{o(\frac{\mathrm{tw}_I}{\log \mathrm{tw}_I})}$ *even for non-negative ILP instances with bounded* B_{max} *(unless ETH fails).*

Concluding Notes

Our results form a valuable contribution to the relatively unexplored area of exploiting the structure of integer and mixed linear programs. In particular, torso-width is the first parameter which allows the solution of MILP instances using decompositional techniques by a non-trivial combination of Lenstra's algorithm and dynamic programming. Moreover, incidence treewidth allows us to lift dynamic promming techniques to ILP instances which contain large constraints, in contrast to the previously considered primal treewidth.

One promising area for future research is to measure and compare the values of these parameters on ILP and MILP instances which occur in practice. Since instances come from many diverse settings and processes, it would not be surprising to see that some practically relevant ILP instances naturally exhibit the forms of structure studied herein.

Acknowledgments The authors wish to thank the anonymous reviewers for their helpful comments. The authors acknowledge support by the Austrian Science Fund (FWF, project P26696). Robert Ganian is also affiliated with FI MU, Brno, Czech Republic.

References

Alumur, S. A., and Kara, B. Y. 2008. Network hub location problems: The state of the art. *European Journal of Operational Research* 190(1):1–21.

Bodlaender, H. L.; Drange, P. G.; Dregi, M. S.; Fomin, F. V.; Lokshtanov, D.; and Pilipczuk, M. 2016. A c^k n 5-approximation algorithm for treewidth. *SIAM J. Comput.* 45(2):317–378.

Bodlaender, H. L. 1996. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.* 25(6):1305–1317.

Courcelle, B.; Makowsky, J. A.; and Rotics, U. 2000. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.* 33(2):125–150.

Cunningham, W. H., and Geelen, J. 2007. On integer programming and the branch-width of the constraint matrix. In *Integer Programming and Combinatorial Optimization, IPCO 2007. Proceedings*, volume 4513 of *Lecture Notes in Computer Science*, 158–166. Springer.

Cygan, M.; Fomin, F. V.; Kowalik, L.; Lokshtanov, D.; Marx, D.; Pilipczuk, M.; Pilipczuk, M.; and Saurabh, S. 2015. *Parameterized Algorithms*. Springer.

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

Downey, R. G., and Fellows, M. R. 2013. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer.

Fellows, M. R.; Lokshtanov, D.; Misra, N.; Rosamond, F. A.; and Saurabh, S. 2008. Graph layout problems parameterized by vertex cover. In *Algorithms and Computation, ISAAC 2008. Proceedings*, volume 5369 of *Lecture Notes in Computer Science*, 294–305. Springer.

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

Flum, J., and Grohe, M. 2006. *Parameterized Complexity Theory*. Springer.

Fomin, F. V.; Panolan, F.; Ramanujan, M. S.; and Saurabh, S. 2016. Fine-grained complexity of integer programming: The case of bounded branch-width and rank. *CoRR* abs/1607.05342.

Frank, A., and Tardos, É. 1987. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica* 7(1):49–65.

Ganian, R., and Ordyniak, S. 2016. The complexity landscape of decompositional parameters for ILP. In *AAAI Conference on Artificial Intelligence, AAAI 2016. Proceedings*, 710–716.

Ganian, R., and Szeider, S. 2015. Community structure inspired algorithms for SAT and #SAT. In *Theory and Applications of Satisfiability Testing, SAT 2015. Proceedings*. Springer.

Ganian, R.; Hliněný, P.; and Obdržálek, J. 2013. Better algorithms for satisfiability problems for formulas of bounded rank-width. *Fundam. Inform.* 123(1):59–76.

Ganian, R.; Ramanujan, M. S.; and Szeider, S. 2016. Discovering archipelagos of tractability for constraint satisfaction and counting. In *ACM-SIAM Symposium on Discrete Algorithms, SODA 2016. Proceedings*, 1670–1681.

Gottlob, G.; Grohe, M.; Musliu, N.; Samer, M.; and Scarcello, F. 2005. Hypertree decompositions: Structure, algorithms, and applications. In Kratsch, D., ed., *Graph-Theoretic Concepts in Computer Science, WG 2005. Revised Selected Papers*, volume 3787 of *Lecture Notes in Computer Science*, 1–15. Springer.

Jansen, B. M. P., and Kratsch, S. 2015. A Structural Approach to Kernels for ILPs: Treewidth and Total Unimodularity. In *Algorithms, ESA 2015. Proceedings*, volume 9294 of *Lecture Notes in Computer Science*, 779–791. Springer Verlag.

Kannan, R. 1987. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.* 12(3):415–440.

Kloks, T. 1994. *Treewidth: Computations and Approximations*. Berlin: Springer Verlag.

Lenstra, H. W., and Jr. 1983. Integer programming with a fixed number of variables. *Math. Oper. Res.* 8(4):538–548.

Lodi, A.; Martello, S.; and Monaci, M. 2002. Two-dimensional packing problems: A survey. *European Journal of Operational Research* 141(2):241–252.

Marinescu, R., and Dechter, R. 2010. Evaluating the impact of AND/OR search on 0-1 integer linear programming. *Constraints* 15(1):29–63.

Marx, D. 2010. Can you beat treewidth? *Theory of Computing* 6(1):85–112.

Niedermeier, R. 2006. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press.

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

Szeider, S. 2003. Finding paths in graphs avoiding forbidden transitions. *Discr. Appl. Math.* 126(2-3):239–251.

Toth, P., and Vigo, D., eds. 2001. *The Vehicle Routing Problem*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

van den Briel, M.; Vossen, T.; and Kambhampati, S. 2005. Reviving integer programming approaches for AI planning: A branch-and-cut framework. In *Automated Planning and Scheduling, ICAPS 2005. Proceedings*, 310–319. AAAI.

Vossen, T.; Ball, M. O.; Lotem, A.; and Nau, D. S. 1999. On the use of integer programming models in AI planning. In *International Joint Conference on Artificial Intelligence, IJCAI 99. Proceedings*, 304–309. Morgan Kaufmann.