On Integer Programs That Look Like Paths*

Marcin Briański
Jagiellonian University
marcin.brianski@doctoral.uj.edu.pl

Alexandra Lassota
Eindhoven University of Technology
a.a.lassota@tue.nl

Kristýna Pekárková
AGH University of Krakow
pekarkova@agh.edu.pl

Michał Pilipczuk
University of Warsaw
michal.pilipczuk@mimuw.edu.pl

Janina Reuter
Kiel University
janina.reuter@informatik.uni-kiel.de

Abstract

Solving integer programs of the form $\min\{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n\}$ is, in general, NP-hard. Hence, great effort has been put into identifying subclasses of integer programs that are solvable in polynomial or FPT time. A common scheme for many of these integer programs is a star-like structure of the constraint matrix. The arguably simplest form that is not a star is a path. We study integer programs where the constraint matrix A has such a path-like structure: every non-zero coefficient appears in at most two consecutive constraints. We prove that even if all coefficients of A are bounded by 8, deciding the feasibility of such integer programs is NP-hard via a reduction from 3-SAT. Given the existence of efficient algorithms for integer programs with star-like structures and a closely related pattern where the sum of absolute values is column-wise bounded by 2 (hence, there are at most two non-zero entries per column of size at most 2), this hardness result is surprising.

*M.P. was supported by the project BOBR that is funded from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme with grant agreement No. 948057. K.P. was supported by the project GA24-11098S of the Czech Science Foundation at the beginning of the project, and by the funding received from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101002854) towards the end of the project.

1 Introduction

We study integer programs (to which we refer to as *IP*s) of the form

$$\min\{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}, \mathbf{x} \in \mathbb{Z}^n\}, \tag{G}$$

with constraint matrix $A \in \mathbb{Z}^{m \times n}$, right-hand side $\mathbf{b} \in \mathbb{Z}^m$, lower and upper bounds $\mathbf{l} \in (\mathbb{Z} \cup \{-\infty\})^n$ and $\mathbf{u} \in (\mathbb{Z} \cup \{\infty\})^n$. Solving such IPs means to either decide that the system is infeasible, there exists an optimal feasible solution (and give the value of it), or it is unbounded and provides a solution with an unbounded direction of improvement. Note that restricting to equality constraints does not restrict the problem, as every inequality constraint can be turned to equality by introducing slack variables (which also preserve the special structures we discuss in this paper).

Integer programming is a powerful tool that has been applied to many combinatorial problems, such as scheduling and bin packing [13, 18], graph problems [8, 9], multichoice optimization [7] and computational social choice [1, 21], among others. Unfortunately, solving integer programs is, in general, NP-hard.

This motivated a flourishing line of research that aims to understand the structural properties of matrices that allow the corresponding IPs to be solved efficiently, see, e.g. [8, 13, 14, 18, 21]. Arguably, the most famous is Lenstra's 1983 algorithm, which introduced the first FPT time algorithm for constraint matrices with few columns [19]. The body of literature for the over three decades-long research and the many structures, parameters, and applications is too vast to cover here; we thus refer to [12] for an overview.

One striking property for many of these IPs is the star-likeness of the constraint matrix, which is probably best exemplified by n-*fold* and 2-*stage stochastic* integer programs. Here, an integer program is of n-fold form if its constraint matrix, after removing at most k rows (*aka* constraints), consists of n independent blocks with at most k rows each. The transpose of this matrix is called 2-stage stochastic integer programs, i.e., after removal of at most k columns (*aka* variables), they are decomposed into independent blocks with at most k columns each. As proven in [16, 15], the optimization problem for n-fold and 2-stage stochastic integer programs can be solved in time $f(k, \Delta) \cdot |I|^{1+o(1)}$, where f is a computable function, Δ is the largest absolute coefficient in the constraint matrix, and $|I|$ is the total bitsize of the instance. This running time is FPT with parameters k and Δ.

The structure of n-fold and 2-stage stochastic integer programs can be generalized by allowing multiple stages of deleting rows or columns and decomposing into blocks. This gives rise to so-called *tree-fold* and *multistage stochastic* integer programs, which retain efficient solvability, see, e.g., [6, 3]. The structure of tree-fold and multistage stochastic IPs can be perhaps best understood through decompositions of the associated graphs. For a matrix A, we define the *constraint graph* of A as the graph on the rows (constraints) of A in which two rows are adjacent if they simultaneously have non-zero entries in the same column (in other words, there is a variable appearing in both the corresponding constraints). The *variable graph* of A is defined symmetrically for the columns (variables) of A. Then, the *primal treedepth* of A is the treedepth of its variable graph, and the *dual treedepth* of A is the treedepth of its constraint graph. Here, the treedepth of a graph is a parameter that measures star-likeness through the maximum depth of a procedure that alternately deletes a vertex and decomposes the graph into connected components, see for example [23, Chapter 6] for a broad introduction. As proven in [17, 10, 11], the optimization problem for integer programs can be solved in time $f(d, \Delta) \cdot |I|^{1+o(1)}$ for a computable function f, where d is either the primal or the dual treedepth of the constraint matrix. We refer the reader to the article of Eisenbrand, Hunkenschröder, Klein, Koutecký, Levin and Onn [6] for a comprehensive introduction to the theory of parameterized algorithms for block-structured integer programming.

It is, therefore, natural to ask to what extent the (primal or dual) treedepth of the constraint matrix captures the efficient solvability of integer programs. Several results in the literature suggest that integer programming quickly becomes NP-hard once one relaxes the assumption of bounded treedepth. For instance, Ganian and Ordyniak [10] showed that the feasibility problem for integer programs of the form

$\{\mathbf{x} \mid A\mathbf{x} \leqslant \mathbf{b}\}$ is NP-hard even when all the entries of A and $\mathbf{b}$ are in $\{-2,-1,0,1,2\}$ and the treewidth of the constraint graph of A is at most 4. Their reduction is based on an earlier work of Jansen and Kratsch [17], who achieved a similar result, but without obtaining the bound on the entries. We note that Ganian and Ordyniak stated their result in terms of the treewidth of the *incidence graph* of A: the bipartite graph on rows and columns of A where a row is adjacent to a column if the entry at their intersection is non-zero. As shown in [10], the treewidth of the incidence graph of the constructed matrix is at most 3, but one can also easily verify that the treewidth of the constraint graph is at most 4 (bounding the treewidth/treedepth of the constraint graph implies a bound on the treewidth of the incidence graph, but not vice versa.) Similar lower bounds were also reported in the work of Ganian, Ordyniak, and Ramanujan [11]. Later, Eiben, Ganian, Knop, Ordyniak, Pilipczuk, and Wrochna [5] showed that when the incidence graph is considered, even bounded treedepth is not enough: deciding feasibility of integer programs of the form $\{\mathbf{x} \mid A\mathbf{x} = 0, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}\}$, is NP-hard even when the entries of $A, \mathbf{l}, \mathbf{u}$ are in $\{-1,0,1\}$ and the incidence graph of A has treedepth 5. Klein, Polak, and Rohwedder [20] show NP-hardness for constraint matrices, where a small 3×5 block matrix is repeated in a (lower-left) triangle, i.e., the same matrix is repeated over the diagonal and below, and entries in $\{-2,-1,0,1\}$. Finally. Eiben, Ganian, Knop, and Ordyniak [4] showed a number of lower bounds showing that allowing A to have arbitrarily large entries quickly leads to NP-hardness, even if A has bounded treedepth.

The goal of this work is to contribute to the understanding of the complexity of integer programming by considering matrices that have the simplest possible form while still having unbounded treedepth. Since length of the longest path (contained as a subgraph) is functionally equivalent to treedepth, the simplest possible graph with arbitrarily large treedepth is a path. Thus, our aim is to understand the complexity of integer programs whose constraint graph[1] is a path. Equivalently, these are integer programs of the form $\{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \ell \leqslant \mathbf{x} \leqslant \mathbf{u}\}$ where every variable can appear only in at most two constraints, which moreover must be consecutive. Up to permuting the columns, the constraint matrix A must have the path-like structure as depicted in Fig. 1: the non-zero entries in each row may only appear in an interval, these intervals are naturally ordered from left to right, and only the intervals in consecutive rows may overlap.

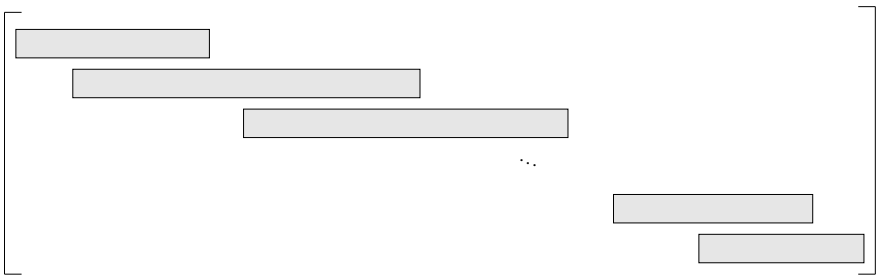

Figure 1: A schematic depiction of a path-like constraint matrix A.

It is known that if the coefficients of the constraint matrix are in $\{-2,-1,0,1,2\}$ and the sum of absolute values is at most 2 for each column, then the problem is polynomial time solvable [24]. Indeed, the corresponding problem called (generalized) matching problem is well-studied. It captures a variety of well-known problems in P such as minimum cost flow, minimum cost (b-)matching and certain graph factor problems [24]. These structures even remain FPT time solvable parameterized by the number p of additional columns [22].

[1] One could also consider the setting when the variable graph is a path, but a quick examination of this setting reveals that it makes little sense, and in fact reduces to a subcase of the setting where the constraint graph is a path.

The case of *two* non-zero entries per column seems to be indeed the turning point where analyzing the computational complexity becomes intriguing. Indeed, it is easy to see that the problem is NP-hard when *three* non-zero entries per column are allowed and entries are in $\{-1, 0, 1, 2\}$. Consider the following reduction from subset sum where n non-negative numbers $a_1, \ldots, a_n$ and a target $t \in \mathbb{Z}_{\geqslant 0}$ are given and the goal is to decide whether a subset of $\{a_1, \ldots, a_n\}$ sums up to t. Let $enc(x)$ be the binary encoding of a non-negative number x and consider the matrices

$$E = \begin{pmatrix} -1 & 2 & & \\ & -1 & 2 & \\ & & \ddots & \ddots \\ & & & -1 \quad 2 \end{pmatrix} \quad \text{and} \quad A = \begin{pmatrix} enc(a_1) & \ldots & enc(a_n) \\ E & & \\ & \ddots & \\ & & E \end{pmatrix}.$$

The first matrix, E, is called the *encoding matrix* and is often used in lower bounds for IPs with a special structure of non-zero entries in the constraint matrix. The IP defined by the constraint matrix A, right-hand side $\mathbf{b} = (t, 0, \ldots, 0)$, and suitable upper and lower bounds $\mathbf{l}$ and $\mathbf{u}$ is feasible if and only if the corresponding subset sum instance is feasible. Clearly, the constraint matrix A has at most three non-zero entries per column.

In stark contrast to the above-mentioned results for an even more restricted form in terms of the placement of non-zero coefficients, we show that the feasibility problem for integer programs with path-like constraint matrices is NP-hard already when all entries are bounded by a constant.

Theorem 1. *The problem of deciding whether a given integer program* $\{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}\}$ *has a solution is* NP*-hard even when* A *is a path-like matrix with all entries of absolute value at most* 8*. Here, we assume that* $\mathbf{b}$ *is provided on input in binary.*

We stress that the assumption that $\mathbf{b}$ is encoded in binary is essential. Indeed, since the incidence graphs of path-like matrices have treewidth at most 2, from the result of Ganian, Ordyniak, and Ramanujan [11, Theorem 13] it follows that the feasibility of integer programs of the form discussed in Theorem 1 can be decided in time polynomial in $\|A\|_\infty + \|\mathbf{b}\|_\infty$ and the total bitsize of the input.

Theorem 1 therefore provides a strong lower bound for integer programming and intuitively shows that any deviation from constraint matrices with bounded treedepth renders the problem intractable. The proof of the theorem is significantly more elaborate than the proofs of the lower bounds from [17, 10, 11, 5, 4] mentioned above, which to a large extent relied on a direct encoding of Subset Sum as an integer program with a simple structure. Our reduction is from 3-SAT. Before we prove this result in Section 2, we give an intuitive argument and sketch of the reduction at the beginning of the section.

2 Hardness

We now prove that path-like integer programs are NP-hard even when all coefficients of the constraint matrix are bounded by a constant. We prove this by reducing the 3-SAT problem to path-like IPs.

The main idea is to design constraints such that each constraint C_i in the matrix restricts the solution space of feasible assignments to be feasible for the first i constraints. This way, starting with a constraint that encodes all possible variable assignments, we propagate and constrict the feasible assignments according to the clauses. We construct the matrix in such a way that every variable occurs at most twice and in consecutive constraints. We say that the first constraint for a variable *introduces* the variable, and that in a (potential) second constraint, the solution space of the variable is *forwarded* or *output*. We use a limited number of different *constraint types* to transform the solution space of a forwarded variable to a solution

Operation	Constraint
Forward	$x - y = 0$
Check $x \geqslant a$	$x - y = a$
Multiplication by $a \in \mathbb{Z}$	$ax - y = 0$
Division with remainder by $a \in \mathbb{Z}$	$x - ay - r = 0$ and $0 \leqslant r < a$
Combination of two constraints with independent variables	$\mathbf{c}_1^\intercal \mathbf{x}_1 = \mathbf{b}_1$ and $\mathbf{c}_2^\intercal \mathbf{x}_2 = \mathbf{b}_2$ with $\mathbf{0} \leqslant \mathbf{x}_1 \leqslant \mathbf{u}$ combined to $\mathbf{c}_1^\intercal \mathbf{x}_1 + a \cdot \mathbf{c}_2^\intercal \mathbf{x}_2 = \mathbf{b}_1 + a\mathbf{b}_2$ with $a > \mathbf{c}_1^\intercal \mathbf{u}$

Table 1: Types of constraints and the operation that is thereby performed on the solution space from an output variable x and an introduced variable y that are used in the proof of Theorem 2.

space for an introduced variable. Here, we consider the solution space as the set of feasible solutions for that variable considering the current and all previous constraints but ignoring all following constraints. The constraint types of the reduction are summarized in Table 1.

A cornerstone for the reduction is a number-theoretical identity that allows us to perform operations that check satisfaction of a clause while preserving the information stored in that variable. In particular, we need to look up the assignment of specific variables, check whether they yield a true literal in the clause, and then restore the original representation of the assignment. Considering only one clause, it would be relatively easy to repeatedly divide by 2 with remainder and check whether the remainder implies a true literal. However, this procedure is in fact limited to one clause, as the information of the satisfying assignment are lost in the process.

Theorem 2. *The problem of deciding the existence of a vector* $\mathbf{x} \in \mathbb{Z}^n$ *satisfying* $A\mathbf{x} = \mathbf{b}, \mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}$ *is* NP*-hard even when* A *is a hollow path-like matrix with* $\|A\|_\infty \leqslant 8$.

Proof. We reduce from 3-SAT, which is known to be NP-complete [2], i.e., we define a hollow path-like matrix M, lower and upper bounds $\mathbf{l}$ and $\mathbf{u}$, and a right-hand side $\mathbf{b}$ such that $M\mathbf{x} = \mathbf{b}$ has a solution $\mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}$ if and only if the 3-SAT formula is satisfiable.

Let $\varphi = \bigwedge_{i=1}^{m} C_i$ be a 3-SAT formula over the set of variables $V = \{v_1, v_2, \ldots, v_n\}$. We begin by checking whether the all-zero assignment $\nu \colon V \ni v_j \mapsto 0 \in \{0, 1\}$ is a satisfying assignment. If it does satisfy φ, then the construction outputs a trivial instance where A is the zero matrix, and $\mathbf{b}, \mathbf{l}, \mathbf{u}$ are zero vectors, i.e., $\begin{bmatrix}0\end{bmatrix} \mathbf{v} = \begin{bmatrix}0\end{bmatrix}, \mathbf{l} = \mathbf{u} = \begin{bmatrix}0\end{bmatrix}$. Henceforth, we assume that the constant 0 assignment is not a satisfying assignment of φ.

We first construct a constraint where the set of solutions contains all the $2^n - 1$ possible variable assignments for the 3-SAT formula, except for the assignment mapping all variables to 0, as follows:

$$x_{0,0} - \alpha = 0,$$

where the lower and upper bounds are $\alpha \in \{1, \ldots, 2^n - 1\}$. The constraint is feasible for variable $x_{0,0} \in \mathcal{A}_0 = \{1, \ldots, 2^n - 1\} = \{[b_{n-1}b_{n-2}\ldots b_0]_2 \neq 0 \mid b_i \in \{0, 1\}\}$, where $[b_{n-1}b_{n-2}\ldots b_0]_2$ represents the number $\sum_{i=0} b_i 2^i$.

The main part is to delete all representations of non-satisfying assignments from the set. Each element of $\mathcal{A}_i$ represents a variable assignment that satisfies all clauses C_j with $j \leqslant i$. Therefore, in order to generate the next set $\mathcal{A}_{i+1}$, we describe constraints that delete assignments that do not satisfy clause C_{i+1}.

Although it might seem natural to do the checking of the clause directly on the set $\mathcal{A}_i$, we need a trick in order to preserve the information about the assignment for checking further clauses.

Claim 3. *The following equations are true.*

1. $x = \lfloor \lfloor x \cdot 5^n / 2^n \rfloor 2^n / 5^n \rfloor + 1$ *for all* $x \in \{1, \dots 2^n - 1\}$
2. $\{x \cdot 5^n \mod 2^n \mid x \in \{1, \dots 2^n - 1\}\} = \{1, \dots 2^n - 1\}$

Proof. Regarding the first equation, we get that the first floor rounds down by $0 < (5^n x/2^n - \lfloor 5^n x/2^n \rfloor) \cdot 2^n/5^n < 1$ and the second floor rounds down by $2^n y/5^n - \lfloor 2^n y/5^n \rfloor < 1$ for $y = \lfloor 5^n x/2^n \rfloor$. In total, the rounding scrapes off more that 0 but less than 2, starts with an integer and ends with an integer. Thus, adding 1 after the second rounding restores x. The second equation follows directly as 2 and 5 are coprime. □

Roughly speaking, we will use the second equality in Claim 3 to extract clause satisfiability from the remainder during the division by 2^n and then restore the original set of candidate assignments using the first identity of Claim 3. For any $x \in \mathcal{A}_i$ the represented assignment are the bits of $[b_{n-1} \dots b_0]_2 = 5^n x \bmod 2^n$. The following steps recreate the above identity while checking one clause for each application.

Preparation phase. We first construct the set $\mathcal{B}_i = \{5^n \cdot a \mid a \in \mathcal{A}_i\}$. The following constraints transform variable $x_{i,0}$ with feasible assignments $\mathcal{A}_i$ to variable $y_{i,0}$ with feasible assignment $\mathcal{B}_i$

$$\begin{aligned} 5x_{i,j} - x_{i,j+1} &= 0 \quad \text{for all } 0 \leqslant j < n \\ x_{i,n} - y_{i,0} &= 0. \end{aligned}$$

Here, every constraint of the first type is a multiplication-by-5-type of constraint (see Table 1) and hence ensures that any feasible value for $x_{i,j+1}$ is five times any feasible value for $x_{i,j}$. The second constraint only adds syntactical sugar with $x_{i,n} = y_{i,0}$ to simplify variable names and readability in the next step.

Checking clause. Following Claim 3, we use division by 2^n while checking the remainder for satisfaction of clause C_i. We use an additional variable $f_{i,j}$ to remember whether the clause is satisfied. This variable is increased if a literal of the clause is true for the assignment. If $f_{i,n} \neq 0$ after the division with remainder, then the clause is satisfied. The goal is to achieve a set of feasible values for $(f_{i,n}, y_{i,n})$ of

$$\mathcal{C}_i = \{(f, \lfloor b/2^n \rfloor) \mid b \in \mathcal{B}_i \text{ and } f = t_i(b)\},$$

where $t_i(b)$ is the number of true literals in clause C_i for assignment $[b_{n-1} \dots b_0]_2 = b \bmod 2^n$. To be precise, in one case the constructed set contains additional elements where $f \leqslant t_i(b)$. However, this does not effect correctness as the following step will only check for values of $f > 0$, hence for assignemnts, where there was at least one true literal found.

For each SAT variable v_j, where $1 \leqslant j \leqslant n$, we use the following types of constraints to transform any feasible value for $y_{i,0}$ from $\mathcal{B}_i$ to a corresponding pair $(f_{i,n}, y_{i,n})$ from $\mathcal{C}_i$:

$$\begin{aligned} f_{i,j} - f_{i,j+1} + 4y_{i,j} - 8y_{i,j+1} - 4\alpha_{i,j} &= 0 \quad \text{if } v_{j+1} \text{ is not in clause } C_i \\ f_{i,j} - f_{i,j+1} + 4y_{i,j} - 8y_{i,j+1} - 3\alpha_{i,j} &= 0 \quad \text{if } v_{j+1} \text{ is in clause } C_i \\ f_{i,j} - f_{i,j+1} + 4y_{i,j} - 8y_{i,j+1} - 5\alpha_{i,j} + \alpha'_{i,j} &= 0 \quad \text{if } \neg v_{j+1} \text{ is in clause } C_i. \end{aligned}$$

We define variable bounds $\alpha_{i,j}, \alpha'_{i,j} \in \{0, 1\}$. Further we define $f_{i,j} \in \{0, 1, 2, 3\}$ if all literals in clause C_i are positive literals and $f_{i,j} \in \{0, 1, 2\}$ if there is at least one negative literal.

The first constraint forwards the set of feasible values for $f_{i,j} = f_{i,j+1}$ and for every feasible value of $y_{i,j+1} = \lfloor y_{i,j}/2 \rfloor$. This is a combination of a division with remainder constraint type (for y variables) and a forward constraint type (for f variables), compare with Table 1. As the variable v_j is not in the clause, the remainder is irrelevant and gets absorbed by $\alpha_{i,j}$. The division by two, if independent from the context, could be achieved by

$$y_{i,j} - 2y_{i,j+1} - \alpha_{i,j} = 0 \quad \text{and } \alpha_{i,j} \in \{0, 1\}.$$

However, we need to ensure that the feasible values for $f_{i,j}$ and $y_{i,j}$ do not interfere and create new feasible values. Hence, the multiplication of the above simple form of the division by $4 > f_{i,j+1} \in \{0, 1, 2, 3\}$ makes sure that the feasible values transform independently as any value of $f_{i,j+1}$ can not compensate for an unwanted change in $y_{i,j+1}$.

In the second and third type of constraint, the feasible values for $f_{i,j}$ are incremented in $f_{i;j+1}$ depending on the remainder from the division. As before, this is similar to a division with remainder constraint type and a forward constraint type. However, the variables should not evolve independently and instead the solution space for $f_{i,j+1}$ depends on the remainder of the division performed on $y_{i,j}$. In the second constraint it is incremented if the remainder is 1 and in the third constraint it *could or could not* be incremented if the remainder is 0.

In the case of the second constraint type, this is achieved from a coefficient of 3 for $\alpha_{i,j}$ instead of 4 (compared to the first type of constraint). Then, in case of a remainder of 1 (representing a true literal), the variable $f_{i,j+1}$ has to make up the difference of 1.

In the third constraint type, if the remainder is 1 and hence the literal is false, then the feasible values either set $\alpha_{i,j} = \alpha'_{i,j} = 1$ and forwards the value of $f_{i,j+1} = f_{i,j}$ or set $\alpha_{i,j} = 1$, $\alpha'_{i,j} = 0$ and forwards the value of $f_{i,j+1} = f_{i,j} - 1$, which is only possible if $f_{i,j} > 0$. Hence, this does not transform an invalid $f_{i,j}$ to a false positive $f_{i,j+1} > 0$ and similarly, any positive $f_{i,j}$ is forwarded. In the case that the remainder is 0 and hence the literal is true, there are again two possible transformations. Either we set $\alpha_{i,j} = \alpha'_{i,j} = 0$ and $f_{i,j+1} = f_{i,j}$ or we set $\alpha_{i,j} = 0$, $\alpha'_{i,j} = 1$, and $f_{i,j+1} = f_{i,j} + 1$. In other words, if this literal is true and hence the clause is satisfied, the set of feasible solutions for $f_{i,j+1}$ contains also a value of $f_{i,j} + 1 > 0$. The additional possibility of an unchanged $f_{i,j+1} = f_{i,j}$ does not matter as we eventually only require one feasible version of this assignment, when testing in the next step. Hence, this derivation from the defined set $\mathcal{C}_i$ have no implications on the reduction. Moreover, note that if this constraint is present $f_{i,j}, f_{i,j+1} \in \{0, 1, 2\}$ and thus $-f_{i,j+1} - 5\alpha_{i,j} > -8$ to ensure correct division by 2. This does not alter the set of feasible assignments with $f_{i,j+1} > 0$, since $f_{i,j} = f_{i,j+1}$ is feasible with $\alpha'_{i,j} = 0$.

Delete not satisfied clauses. Next, we delete any assignment that does not satisfy the clause C_i. We add feasible values $(f_{i,n}, y_{i,n})$ from $\mathcal{C}_i$ to the next set $\mathcal{D}_i$ iff $f_{i,n} > 0$. We want to achieve a set of feasible solutions for $z_{i,0}$ of $\mathcal{D}_i = \{c \mid (f, c) \in \mathcal{C}_i \text{ with } f > 0\}$. We can achieve this with the constraint

$$f_{i,n} + 4y_{i,n} - 4z_{i,0} - \beta_i = 1,$$

where $\beta_i \in \{0, 1, 2\}$. Since $\beta_i < 3$, this constraint is feasible iff $f_{i,n} > 0$ and then $y_{i,n} = z_{i,0}$. This is again a combination of constraint types with a checking type for $f_{i,n}$ and a forward type for $y_{i,n}$ to $z_{i,0}$. Also note that this transformation eliminates any derivation from $\mathcal{C}_i$ in the case of negated variables present in the clause as explained in the previous step.

Restore representation. For the current clause C_i, it remains to restore the original representation using the identity from Claim 3. Hence, we transform any feasible value for $z_{i,0}$ from $\mathcal{D}_i$ to a value for $x_{i+1,0}$ from

$$\mathcal{A}_{i+1} = \{\lfloor d \cdot 2^n/5^n \rfloor + 1 \mid d \in \mathcal{D}_i\}.$$

The transformation uses similar multiplication and division constraint types as before, i.e.

$$\begin{aligned} 2z_{i,j} - z_{i,j+1} &= 0 \quad \text{for all } 0 \leqslant j < n \\ z_{i,n+j} - 5z_{i,n+j+1} - \gamma_{i,j} &= 0 \quad \text{for all } 0 \leqslant j < n \\ z_{i,2n} - x_{i+1,0} &= -1, \end{aligned}$$

with variable bounds $\gamma_{i,j} \in \{0, 1, 2, 3, 4\}$.

Adding the constraints for *preparations phase*, *checking clause*, *delete not satisfied*, and *restore representation* for all clauses $C_1, \ldots, C_m$ results in a constraint matrix M with a hollow path-like structure such that the IP is feasible if and only if the 3-SAT formula is satisfiable. To be precise, *restore representation* is not required for the last clause C_m as it does not impact feasibility of the IP.

The set of feasible solutions for the IP at position $x_{0,0} = x_{1,0} = \ldots = x_{m,0}$ is exactly the set $\mathcal{A}_m$. Solutions of the 3-SAT formula are given by $5^n \cdot x_{m,0} \mod 2^n$ for any $x_{m,0} \in \mathcal{A}_m$. In other words, the set

$$\mathcal{S} = \{5^n \cdot a \mod 2^n \mid a \in \mathcal{A}_m\}.$$

consists of all satisfying assignments $\nu \colon V \ni v_j \mapsto b_j \in \{0, 1\}$ of φ for $[b_{n-1} \ldots b_0]_2 \in \mathcal{S}$ and any feasible solution to the hollow path-like IP sets variable $x_{m,0}$ to an element of $\mathcal{A}_m$.

Hence, this problem is hard. □

We use the construction from the above proof to additionally derive a fine-grained lower on the running time to solve path-like IPs assuming the Exponential Time Hypothesis (ETH).

Corollary 4. *Assuming the ETH, the problem of deciding the existence of a vector $\mathbf{x} \in \mathbb{Z}^n$ satisfying $A\mathbf{x} = \mathbf{b}$, $\mathbf{l} \leqslant \mathbf{x} \leqslant \mathbf{u}$ cannot be solved in time $2^{\delta\sqrt{n}} \cdot \mathrm{poly}(|I|)$ for some $\delta > 0$, where $|I|$ denotes the encoding length, even when A is a hollow path-like matrix with $\|A\|_\infty \leqslant 8$.*

Proof. Consider a 3-SAT formula φ over $\tilde{n}$ variables. Using sparsification, assume that φ has $O(\tilde{n})$ clauses. The ETH states that there exists $\tilde{\delta} > 0$ such that 3-SAT can not be solved in time $2^{\tilde{\delta}\tilde{n}}$.

The transformation in the proof of Theorem 2 gives us an equivalent path-like IP with the largest entry in the coefficient matrix of $\|A\|_\infty \leqslant 8$ and $\|\mathbf{l}\|_\infty < \|\mathbf{u}\|_\infty < 2^n$. The complexity of the transformation is dominated by the variables $x_{i,j}$, $y_{i,j}$, $z_{i,j}$, $z_{i,\tilde{n}+j}$, $f_{i,j}$, $\alpha_{i,j}$, $\alpha'_{i,j}$, $\gamma_{i,j}$, where i iterates over the clauses and j over the variables (and possibly one more in both cases) of φ. Hence, the number of variables in the path-like IP is bounded by $n \in O(\tilde{n}^2)$. As every variable of the path-like IP occurs in at most two constraints, this bound carries over to the number of constraints.

Hence, hollow path-like IPs can not be solved in time

$$2^{\delta\sqrt{n}} \cdot \mathrm{poly}(|I|) = 2^{\tilde{\delta}\tilde{n}}$$

for some $\delta > 0$ even when the coefficient matrix has entries bounded by 8. □

3 Summary and Open Problems

We showed that the feasibility problem for integer programs with path-like constraint matrices is NP-hard already when all entries of the constraint matrix are bounded by 8. We prove this result by a reduction from 3-SAT. The reduction builds a path of constraints to repeatedly filter the set of all 3-SAT assignments for those satisfying a clause from the formula until only assignments satisfying all clauses are left. The coefficient 8 comes as a result of handling two types of information simultaneously in one constraint. This requires coefficients for one flow of information to be multiplied by the largest value for the other

information in order prevent interaction between the variables of the two and their respective information held by the variables. Hence, the largest coefficient of our reduction is the largest value for one information flow 4 multiplied by the largest coefficient 2 in the other information flow. Thus, solving such IPs is harder than than solving IPs where the coefficients are bounded by 1, which is known to be polynomial-time solvable (or more precisely, any IP with constraint matrix where the sum ob absolute values per column is bounded by 2).

This leaves two questions: First, what is the complexity of integer programs with path-like constraint matrices with coefficients smaller than 8. Second, are integer programs with path-like constraint matrices but without upper bounds on the variables, that is, ILPs of the form $\{\mathbf{x} \mid A\mathbf{x} = \mathbf{b}, \mathbf{x} \geqslant 0\}$, NP-hard as well.

References

[1] John Bartholdi, Craig A Tovey, and Michael A Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[2] Stephen A. Cook. The complexity of theorem-proving procedures. In *ACM Symposium on Theory of Computing*, pages 151–158. ACM, 1971.

[3] Jana Cslovjecsek, Friedrich Eisenbrand, Michał Pilipczuk, Moritz Venzin, and Robert Weismantel. Efficient sequential and parallel algorithms for multistage stochastic integer programming using proximity. In *Proceedings of the 29th Annual European Symposium on Algorithms, ESA*, volume 204, pages 33:1–33:14, 2021.

[4] Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. Unary integer linear programming with structural restrictions. In *27th International Joint Conference on Artificial Intelligence, IJCAI 2018*, pages 1284–1290, 2018.

[5] Eduard Eiben, Robert Ganian, Dušan Knop, Sebastian Ordyniak, Michał Pilipczuk, and Marcin Wrochna. Integer programming and incidence treedepth. In *20th International Conference on Integer Programming and Combinatorial Optimization, IPCO 2019*, volume 11480 of *Lecture Notes in Computer Science*, pages 194–204. Springer, 2019.

[6] Friedrich Eisenbrand, Christoph Hunkenschröder, Kim-Manuel Klein, Martin Koutecký, Asaf Levin, and Shmuel Onn. An algorithmic theory of integer programming.

[7] T. Ermolieva, Y Ermoliev, P. Havlik, A. Lessa-Derci-Augustynczik, N. Komendantova, T. Kahil, J. Balkovic, R. Skalsky, C. Folberth, P. S. Knopov, and G. Wang. Connections between robust statistical estimation, robust decision-making with two-stage stochastic optimization, and robust machine learning problems. *Cybernetics and Systems Analysis*, 59(3):385–397, 2023.

[8] Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *Proceedings of the 19th International Symposium on Algorithms and Computation, ISAAC*, pages 294–305, 2008.

[9] Jiří Fiala, Tomáš Gavenčiak, Dušan Knop, Martin Koutecký, and Jan Kratochvíl. Parameterized complexity of distance labeling and uniform channel assignment problems. *Discrete Applied Mathematics*, 248:46–55, 2018.

[10] Robert Ganian and Sebastian Ordyniak. The complexity landscape of decompositional parameters for ILP. *Artif. Intell.*, 257:61–71, 2018.

[11] Robert Ganian, Sebastian Ordyniak, and M. S. Ramanujan. Going beyond primal treewidth for (M)ILP. In *31st AAAI Conference on Artificial Intelligence*, pages 815–821. AAAI Press, 2017.

[12] Tomáš Gavenčiak, Martin Koutecký, and Dušan Knop. Integer programming in parameterized complexity: Five miniatures. *Discrete Optimization*, 44(Part 1):100596, 2022.

[13] Michel X. Goemans and Thomas Rothvoss. Polynomiality for bin packing with a constant number of item types. *Journal of the ACM*, 67(6):38:1–38:21, 2020.

[14] Jens Gramm, Rolf Niedermeier, and Peter Rossmanith. Fixed-parameter algorithms for CLOSEST STRING and related problems. *Algorithmica*, 37(1):25–42, 2003.

[15] Raymond Hemmecke, Shmuel Onn, and Lyubov Romanchuk. n-fold integer programming in cubic time. *Math. Program.*, 137(1-2):325–341, 2013.

[16] Raymond Hemmecke and Rüdiger Schultz. Decomposition of test sets in stochastic integer programming. *Math. Program.*, 94(2-3):323–341, 2003.

[17] Bart M. P. Jansen and Stefan Kratsch. A structural approach to kernels for ILPs: Treewidth and total unimodularity. In *23rd Annual European Symposium on Algorithms, ESA 2015*, volume 9294 of *Lecture Notes in Computer Science*, pages 779–791. Springer, 2015.

[18] Klaus Jansen, Kim-Manuel Klein, Marten Maack, and Malin Rau. Empowering the configuration-ip: new PTAS results for scheduling with setup times. *Math. Program.*, 195(1):367–401, 2022.

[19] Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[20] Kim-Manuel Klein, Adam Polak, and Lars Rohwedder. On minimizing tardy processing time, max-min skewed convolution, and triangular structured ilps. In *Proceedings of the 2023 ACM-SIAM Symposium on Discrete Algorithms, SODA 2023, Florence, Italy, January 22-25, 2023*, pages 2947–2960. SIAM, 2023.

[21] Dusan Knop, Martin Koutecký, and Matthias Mnich. Voting and bribing in single-exponential time. *ACM Transactions on Economics and Computation*, 8(3):12:1–12:28, 2020.

[22] Alexandra Lassota and Koen Ligthart. Parameterized algorithms for matching integer programs with additional rows and columns. In *ICALP*, volume 334 of *LIPIcs*, pages 112:1–112:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

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

[24] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24 of *Algorithms and Combinatorics*. Springer, 2003.