Optimal Length Cutting Plane Refutations of Integer Programs

K. Subramani(✉) and P. Wojciechowski

LDCSEE, West Virginia University, Morgantown, WV, USA
{k.subramani,pwojciec}@mail.wvu.edu

Abstract. In this paper, we discuss the computational complexities of determining optimal length refutations of infeasible integer programs (IPs). We focus on three different types of refutations, namely read-once refutations, tree-like refutations, and dag-like refutations. For each refutation type, we are interested in finding the length of the shortest possible refutation of that type. For our purposes, the length of a refutation is equal to the number of inferences in that refutation. The refutations in this paper are also defined by the types of inferences that can be used to derive new constraints. We are interested in refutations with two inference rules. The first rule corresponds to the summation of two constraints and is called the ADD rule. The second rule is the DIV rule which divides a constraint by a positive integer. For integer programs, we study the complexity of approximating the length of the shortest refutation of each type (read-once, tree-like, and dag-like). In this paper, we show that the problem of finding the shortest read-once refutation is **NPO PB-complete**. Additionally, we show that the problem of finding the shortest tree-like refutation is **NPO-hard** for IPs. We also show that the problem of finding the shortest dag-like refutation is **NPO-hard** for IPs. Finally, we show that the problems of finding the shortest tree-like and dag-like refutations are in **FPSPACE**.

1 Introduction

This paper examines the problems of finding optimal length refutations of infeasible integer programs (IPs). We study three different types of refutations, each of which is characterized by how the reuse of constraints is permitted.

The first type of refutation examined in this paper is read-once refutation. In a read-once refutation, for the most part, constraints cannot be reused. However, if a constraint can be rederived without reusing constraints from the original system, then it can be used as many times as it can be derived.

The second type of refutation examined is tree-like refutation. In a tree-like refutation, constraints from the original system can be reused, and derived

This research was supported in part by the Air-Force Office of Scientific Research through Grant FA9550-19-1-0177 and in part by the Air-Force Research Laboratory, Rome through Contract FA8750-17-S-7007.

A. Bagchi and R. Muthu (Eds.): CALDAM 2023, LNCS 13947, pp. 15–27, 2023.
https://doi.org/10.1007/978-3-031-25211-2_2

constraints cannot. As with read-once refutations, constraints can be rederived. However, for tree-like refutations, these rederivations *can* reuse constraints from the original system.

The third type of refutation examined is dag-like refutation. In a dag-like refutation, both constraints in the original system and derived constraints can be reused. This means that constraints do not need to be rederived.

For each refutation type, we are interested in finding the length of the shortest possible refutation of that type. For our purposes, the length of a refutation is equal to the number of inferences in that refutation. For both read-once refutations and tree-like refutations, the rederivation of a constraint increases the length of the refutation. This does not matter in the case of dag-like refutations, since rederivation is unnecessary. Each refutation system is associated with a set of inference rules that can be used to produce refutations of a given type of constraint system. For integer programs, we examine refutations that allow for two types of inferences.

The first rule corresponds to the summation of two constraints and is called the ADD rule. The second rule is the DIV rule which divides a constraint by a positive integer. For IPs, we study the complexity of approximating the length of the shortest refutation of each type (read-once, tree-like, and dag-like). In this paper, we show that the problem of finding the shortest read-once refutation is **NPO PB-complete**. Additionally, we show that the problem of finding the shortest tree-like refutation and the problem of finding the shortest dag-like refutation are both **NPO-hard**. Finally, we show that the problems of finding the shortest tree-like and dag-like refutations are in **FPSPACE**.

2 Statement of Problems

In this section, we introduce the concepts examined in this paper and define the problems under consideration.

Definition 1. *A* **polyhedral constraint system** *is a conjunction of constraints in which each constraint in* $\mathbf{C}$ *is an inequality of the form* $\mathbf{a}_j \cdot \mathbf{x} \leq b_j$ *where* $\mathbf{a}_j \in \mathbb{Q}^n$ *and* $b_j \in \mathbb{Q}$.

Note that $\mathbf{C}$ can be represented in matrix form as $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$. In the constraint $\mathbf{a}_j \cdot \mathbf{x} \leq b_j$, b_j is referred to as the defining constant.

Definition 2. *An* **integer** *polyhedral constraint system is a polyhedral constraint system in which for each variable* x_i*, the corresponding domain* $D_i = \mathbb{Z}$.

Such a constraint system is known as an integer program (IP).

Example 1. System (1) is an integer program.

$$\begin{gathered} 3 \cdot x_1 + 5 \cdot x_2 - 4 \cdot x_3 \leq -2 \quad -2 \cdot x_2 + 7 \cdot x_3 \leq 4 \\ x_1 \in \{0,1\} \quad x_2 \in \{-1,0,1\} \quad x_3 \in \{0,1,2\} \end{gathered} \tag{1}$$

Refutations are defined by the inference rules that can be used to deduce a contradiction. Refutations of integer programs can use two inference rules.

The first rule corresponds to the summation of two constraints and is defined as follows:

$$\textbf{ADD}: \frac{\sum_{i=1}^{n} a_i \cdot x_i \leq b_1 \quad \sum_{i=1}^{n} a'_i \cdot x_i \leq b_2}{\sum_{i=1}^{n} (a_i + a'_i) \cdot x_i \leq b_1 + b_2} \tag{2}$$

We refer to Rule (2) as the ADD rule.

Example 2. Consider the constraints $3 \cdot x_1 + 5 \cdot x_2 - 4 \cdot x_3 \leq -2$ and $-2 \cdot x_2 + 7 \cdot x_3 \leq 4$. Applying the ADD rule to these constraints results in the constraint $3 \cdot x_1 + 3 \cdot x_2 + 3 \cdot x_3 \leq 2$.

It is easy to see that Rule (2) is sound in that any assignment satisfying the hypotheses **must** satisfy the consequent.

Refutations of integer programs also use an additional rule. This is referred to as the DIV rule and is defined as follows:

$$\textbf{DIV}: \frac{\sum_{i=1}^{n} a_i \cdot x_i \leq b \quad k \in \mathbb{Z}^+ : \frac{a_i}{k} \in \mathbb{Z}}{\sum_{i=1}^{n} \frac{a_i}{k} \cdot x_i \leq \lfloor \frac{b}{k} \rfloor} \tag{3}$$

Example 3. Consider the constraint $3 \cdot x_1 + 3 \cdot x_2 + 3 \cdot x_3 \leq 2$. Applying the DIV rule to this constraint with $k = 3$ results in the constraint $x_1 + x_2 + x_3 \leq 0$.

Rule (3) corresponds to dividing a constraint by a common divisor of the left-hand coefficients and then rounding the right-hand side. Since each $\frac{a_i}{k}$ is an integer. This inference preserves integer solutions but does necessarily preserve linear solutions. A constraint derived using the DIV rule is also known as a Chvátal-Gomory cut [10].

Definition 3. *An* **integer refutation** *is a sequence of applications of the ADD and DIV rules that results in a contradiction of the form* $0 \leq b$, $b < 0$.

In this paper, we study several types of refutations. These are read-once refutations, tree-like refutations, and dag-like refutations. Our focus is on determining the optimal number of inferences in a refutation. Note that both the ADD rule and the DIV rule contribute to the length of the refutation.

Definition 4. *A* **read-once** *refutation is a refutation in which each constraint* C *can be used in only one inference. This applies to constraints present in the original formula and those derived as a result of previous inferences.*

Note that in a read-once refutation, a constraint can be reused if it can be rederived. However, it must be rederived from a different set of input constraints.

Definition 5. *A* **tree-like** *refutation is a refutation in which each derived constraint can be used at most once.*

Note that in tree-like refutations, the input constraints can be used multiple times. Thus any derived constraint can be derived multiple times as long as it is rederived each time it is used. This rederivation can reuse derived constraints. However, those constraints also need to be rederived. Tree-like refutation is a **complete** refutation system [5].

Definition 6. *A* **dag-like** *refutation is a refutation in which each constraint can be used multiple times.*

It follows that dag-like refutations procedures are **complete** as well.

We now define the notion of length of a refutation.

Definition 7. *The* **length** *of a refutation R of a constraint system is the number of inferences (both ADD and DIV) made in R.*

For each type of refutation, there are two associated problems. These are the decision problem, asking if a system has the desired type of refutation, and the optimization problem, asking for the length of the shortest refutation of the desired type. Note that every infeasible constraint system has a tree-like refutation and a dag-like refutation. Thus, the decision problems for these two refutation types are trivial. Furthermore, we have shown that the problem of determining if an IP has a read-once refutation is **NP-hard** even when the constraints are UTVPI (Unit Two Variable Per Inequality) constraints [26,27].

In this paper, we examine the following optimization problems:

1. The **Integer Programming Optimal Length Read-once Refutation (IP-OLRR)** problem: Given an infeasible IP **I**, what is the length of the shortest read-once refutation of **I**?
2. The **Integer Programming Optimal Length Tree-like Refutation (IP-OLTR)** problem: Given an infeasible IP **I**, what is the length of the shortest tree-like refutation of **I**?
3. The **Integer Programming Optimal Length Dag-like Refutation (IP-OLDR)** problem: Given an infeasible IP **I**, what is the length of the shortest dag-like refutation of **I**?

Note that the problem of determining if an IP has a refutation is only interesting if the IP is infeasible. Thus, the problems studied in this paper are promise problems [17]. That is, the problems are only defined on a subset of possible inputs. Observe that the IP-OLRR, IP-OLTR, and IP-OLDR problems are only defined on infeasible IPs. Additionally, the problem of determining if an integer program is infeasible is **coNP-complete**. The reductions used in this paper are guaranteed to generate infeasible IPs. Thus, the complexity results we obtain only apply to the set of infeasible IPs. Note that a feasible IP trivially lacks any refutation. Since the set of infeasible IPs is a subset of all IPs, our results can easily be generalized to all IPs. Thus, we can consider the non-promise versions of each problem.

3 Motivation and Related Work

In this section, we motivate our work and describe existing work for related problems.

Constraint systems are heavily used in the field of software verification [11,12]. Corresponding to a piece of software, a constraint system can be derived and then combined with constraints corresponding to the negation of the specifications. If the resultant system of constraints is infeasible, then the software is consistent with its specifications. Although this approach is intuitive and straightforward, it may become impractical because of the large number of constraints that are generated. A constraint-based approach to program verification has also been attempted for rule-based programming [6]. Rule-based programming has gained interest in the software industry over the past years, because of the growing use of Business Rules Management Systems. Hence, a demand for the verification of rule programs has emerged. Also, in [19] it is shown how the constraint-based approach can be used to model a wide spectrum of program analysis using disjunctions and conjunctions of linear inequalities. Linear programs have also been used as a finer grained abstraction for sequential programs offering an effective model checking procedure [2].

For integer programs, we are interested in cutting plane based refutations. Cutting planes are often used to refute integer programs constructed from CNF formulas [14]. When applied to such systems, cutting plane based refutations can be exponentially more compact than resolution based refutations [7,15,20,25]. Several restricted versions of cutting planes have been examined [28]. These restrictions included limiting addition to cases where a variable is canceled, and replacing the division rule with a saturation rule. It was shown that these restricted versions of cutting planes can be simulated by resolution when the coefficients are small [28]. Every infeasible integer program with m constraints over n variables has a cutting plane refutation of length $O(n^{3 \cdot n})$ that can be computed using polynomial workspace [13]. Workspace is defined as the amount of space used to store the intermediate constraints. Once an intermediate constraint is no longer necessary, it is removed from the workspace [13]. Recently, Cheung et al. discussed the verification of integer programming results using cutting plane refutations, but from an empirical perspective [9]. To the best of our knowledge, our paper is the first of its kind to focus on approximation complexity for the problem of determining optimal length refutations.

Closely related to the problem of finding the length of the shortest refutation of a system S under a proof system P, is the problem of automatizability [1]. A proof system P is automatizable if there exists a deterministic algorithm that, when given an infeasible system S, generates a refutation of S in time polynomial in the length of shortest refutation of S [3]. It was shown that resolution as a proof system is not automatizable, unless $\mathbf{P} = \mathbf{NP}$ [4]. In the case of integer programming, cutting planes are not automatizable, unless $\mathbf{P} = \mathbf{NP}$ [18]. In [18] it is also shown that it is **NP-hard** to approximate the minimum length of a dag-like cutting plane proof length to within $2^{n^{\epsilon}}$. In this paper, we show that this problem is **NPO-hard** for a different proof system.

4 Optimal Length Read-Once Refutations

In this section, we show that the problem of finding the shortest read-once refutation of an IP is **NPO PB-complete**.

Theorem 1. *The IP-OLRR problem is* **NPO PB-complete**.

Proof. A read-once refutation R of an integer program $\mathbf{I}$ is polynomially sized in terms of the size of $\mathbf{I}$. Additionally, the length of a read-once refutation can be computed in polynomial time. Finally, since each constraint in $\mathbf{I}$ is used at most once, the length of the read-once refutation is linear in terms of the size of $\mathbf{I}$. Thus, the IP-OLRR problem is in **NPO PB**. Now we need to show **NPO PB-hardness**.

This is accomplished by a reduction from the Minimum 0-1 Programming problem. This problem is formulated as follows:

Given an integer program $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$, $\mathbf{x} \in \{0,1\}^n$, find the minimum value of $\mathbf{c} \cdot \mathbf{x}$ for some integer valued vector $\mathbf{c} \geq \mathbf{0}$.

In the general case, this problem is known to be **NPO-complete** [24]. However, for this reduction, we are only interested in the case where $\mathbf{c} = \mathbf{1}$. This specific form of Minimum 0-1 Programming is known to be **NPO PB-complete** [22].

Consider the following instance of the Minimum 0-1 programming problem: $\min \sum_{i=1}^{n} x_i \quad \mathbf{A} \cdot \mathbf{x} = \mathbf{b} \quad \mathbf{x} \in \{0,1\}^n$. Even in this form, the Minimum 0-1 programming problem is **NPO PB-complete** [22].

Corresponding to this system, we can construct the following linear program $\mathbf{L}$:

$\mathbf{y} \cdot \mathbf{A} \leq \mathbf{0} \quad -\mathbf{y} \cdot \mathbf{b} \leq -1$.

From $\mathbf{L}$, we can construct the IP $\mathbf{I}$ as follows:

1. For each variable y_i in $\mathbf{L}$, add the variable y_i to $\mathbf{I}$. Additionally, add the new variable x_0 to $\mathbf{I}$. 2. Add all of the constraints in $\mathbf{L}$ to $\mathbf{I}$. 3. Let p, be the first prime larger than $\max_{i=1\ldots n}(\sum_{i=1}^{m} |a_{ij}|)$ where a_{ij} is an element of the matrix $\mathbf{A}$. 4. Add the term $p \cdot x_0$ to the constraint in $\mathbf{I}$ with defining constant -1. By construction, there is exactly one such constraint. We will refer to this new constraint as I_1. 5. Add the constraint $-x_0 \leq 0$ to $\mathbf{I}$.

We will now show that $\mathbf{I}$ has a read-once integer refutation of length $(k+2)$ if and only if $\mathbf{L}$ has a read-once linear refutation of length k.

First, assume that $\mathbf{L}$ has a read-once linear refutation of length k. By construction, any read-once refutation of $\mathbf{L}$ corresponds to a read-once derivation of the constraint $p \cdot x_0 \leq -1$. We can then divide this constraint by p and sum the result with the constraint $-x_0 \leq 0$ to obtain a contradiction. Since the original refutation had length k, the new refutation has length $(k+2)$.

Now assume that $\mathbf{I}$ has a read-once integer refutation of length $(k+2)$. As mentioned above, we can assume without loss of generality that the constraint I_1 is the only constraint in $\mathbf{I}$ with negative defining constant. Thus, it must be used in any refutation of $\mathbf{I}$.

The only other constraint in $\mathbf{I}$ with the variable x_0 is $-x_0 \leq 0$. Thus, this constraint must be used to cancel the variable x_0. Since the refutation is read-once, we must first apply the DIV rule to the constraint derived from I_1. By construction, the DIV rule must be applied to this constraint with coefficient p. However, by construction, p is larger than any coefficient in any constraint derived from I_1. Thus, the DIV rule cannot be applied until all other variables are eliminated from I_1. Thus, we must derive the constraint $p \cdot x_0 \leq -1$. This derivation takes k steps and corresponds to a read-once linear refutation of $\mathbf{I}$. This means that $\mathbf{I}$ has a read-once linear refutation of length k. Thus, the IP-OLRR problem is **NPO PB-complete**. □

Since the IP-OLRR problem is **NPO PB-complete**, there exists an $\epsilon > 0$ such that the IP-OLRR cannot be approximated to within a factor of $O(n^{\epsilon})$, unless $\mathbf{P} = \mathbf{NP}$ [21]. Thus, the IP-OLRR cannot be approximated to within a polylogarithmic factor, unless $\mathbf{P} = \mathbf{NP}$.

5 Optimal Length Tree-Like and Dag-Like Refutations

In this section, we show that the IP-OLTR and IP-OLDR are **NPO-hard**. Note that for these problems, we are not guaranteed polynomial length refutations. Thus we do not have **NPO-completeness**.

Theorem 2. *The IP-OLTR problem is* **NPO-hard**.

Proof. This will be accomplished by a reduction from the Traveling Salesman Path Problem. This problem is **NPO-complete** [24].

Let $\mathbf{G}$ be a complete undirected graph with n vertices. From $\mathbf{G}$ we create an IP $\mathbf{I}$ as follows: 1. For each vertex v_i in $\mathbf{G}$, create the variable x_i. 2. Create the constraint $x_1 + 2 \cdot x_2 + 2 \cdot x_3 + \ldots + 2 \cdot x_{n-1} + 2 \cdot x_n + p \cdot x_0 \leq -1$. Where p is the fist prime such that $p > 2 \cdot n \cdot \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} w(e_{i,j})$. Additionally, create the constraint $-x_0 \leq 0$. 3. For each edge $e_{i,j}$ in $\mathbf{G}$, create the variables $y_{i,j}$ and $z_{i,j,l}$ for $l = 1, \ldots, n-1$. Additionally, create the constraint $-y_{i,j} \leq 0$. 4. For each edge $e_{i,j}$ such that $i, j \in \{2, \ldots, n\}$, and each $l = 2, \ldots, n-2$, create the constraint $-x_i - x_j + 2 \cdot n \cdot w(e_{i,j}) \cdot y_{i,j} + 2 \cdot z_{i,j,l} \leq 0$. 5. For each edge $e_{i,n}$, create the constraint $-x_i - x_n + 2 \cdot n \cdot w(e_{i,n}) \cdot y_{i,n} + z_{i,n,n-1} \leq 0$. 6. For each edge $e_{1,j}$, create the constraint $-x_1 - x_j + 2 \cdot n \cdot w(e_{1,j}) \cdot y_{1,j} + z_{1,j,1} \leq 0$. 7. For each pair of edges $e_{i,j}$ and $e_{j,k}$ that share an endpoint, and each $l = 1, \ldots, n-2$, create the constraint $-z_{i,j,l} - z_{j,k,l+1} \leq 0$. This construction forms the function f for our PTAS reduction.

First, assume that $\mathbf{G}$ has a Traveling Salesman Path P of length W from x_1 to x_n. Let P traverse the vertices in the order $v_{P(1)}$ through $v_{P(n)}$. We can construct a tree-like refutation R of length $2 \cdot n \cdot (W+1)$ for $\mathbf{I}$ as follows: 1. Start with the constraint $x_1 + 2 \cdot x_2 + 3 \cdot x_2 + \ldots + 2 \cdot x_{n-1} + 2 \cdot x_n + p \cdot x_0 \leq -1$. 2. Add the constraint $-x_1 - x_{P(2)} + 2 \cdot n \cdot w(e_{1,P(2)}) \cdot y_{1,P(2)} + z_{1,P(2),1} \leq 0$ to R. 3. For $i = 2 \ldots n-2$, add the constraint $-x_{P(i)} - x_{P(i+1)} + 2 \cdot n \cdot w(e_{P(i),P(i+1)}) \cdot y_{P(i),P(i+1)} + 2 \cdot z_{P(i),P(i+1),i} \leq 0$ to R. 4. Add the constraint $-x_{P(n-1)} - x_n +$

$2 \cdot n \cdot w(e_{P(n-1),n}) \cdot y_{P(n-1),n} + z_{P(n-1),n,n-1} \leq 0$ to R. 5. For $i = 2 \ldots n-2$, add $2 \cdot n \cdot w(e_{P(i),P(i+1)})$ copies of the constraint $-y_{P(i),P(i+1)} \leq 0$ to R. 6. For $i = 1 \ldots n-2$, the constraint $-z_{P(i),P(i+1),i} - z_{P(i+1),P(i+2),i+1} \leq 0$ to R. Observe that summing the constraints in R results in the constraint $p \cdot x_0 \leq -1$. Applying the DIV rule with $d = p$ to this constraint results in the constraint $x_0 \leq -1$. Adding the constraint $x_0 \leq 0$ results in the contradiction $0 \leq -1$. Note that, R contains a total of $2 \cdot n \cdot (W+1)$ inferences. Thus R is a tree-like refutation of length $2 \cdot n \cdot W$ for $\mathbf{I}$.

Now assume that $\mathbf{I}$ has a tree-like refutation R of length $2 \cdot n \cdot (W+1)$. We can construct a set of edges P as follows: For each edge $e_{i,j}$ if R contains the constraint $-y_{i,j} \leq 0$, add $e_{i,j}$ to P. This forms the function g for our PTAS reduction. Observe the following:

1. The constraint $x_1 + 2 \cdot x_2 + 3 \cdot x_2 + \ldots + 2 \cdot x_{n-1} + 2 \cdot x_n + p \cdot x_0 \leq -1$, is the only constraint in the system with a negative defining constant. Thus, it must be part of R. We will refer to this constraint as C. By construction of $\mathbf{I}$, $p > 2 \cdot n \cdot (W+1)$. Thus, if the DIV rule is not applied to C, then the constraint $x_0 \leq 0$ will need to be used at least $p > 2 \cdot n \cdot (W+1)$ times. In this case, the length of R is more than $2 \cdot n \cdot (W+1)$. Thus, the DIV rule must be applied to C. Due to the value chosen for p, this can only happen after everything else is canceled from C.
2. To cancel x_1 from C, R must include a constraint of the form $-x_1 - x_j + 2 \cdot n \cdot w(e_{1,j}) \cdot y_{1,j} + z_{1,j,1} \leq 0$. Let $P(2) = j$. Note that this constraint also cancels a copy of $x_{P(2)}$ from C.
3. To cancel the other copy of $x_{P(2)}$ from C, R must include a constraint of the form $-x_{P(2)} - x_j + 2 \cdot n \cdot w(e_{P(2),j}) \cdot y_{P(2),j} + 2 \cdot z_{P(2),j,1} \leq 0$. Let $P(3) = j$. Note that this constraint also cancels a copy of $x_{P(3)}$ from C.
4. We can continue this process until $P(h) = n$ for some $h \leq n$. Due to the structure of C, the vertices v_1, $v_{P(2)}$, $v_{P(3)}$, …, $v_{P(h)}$ are all distinct.
5. Consider the constraint $-x_{P(h-1)} - x_{P(h)} + 2 \cdot n \cdot w(e_{P(h-1),P(h)}) \cdot y_{P(h-1),P(h)} + z_{P(h-1),P(h),1} \leq 0$ in R. Since $P(h) = n$, by construction of $\mathbf{I}$, this constraint must be $-x_{P(h-1)} - x_{P(h)} + 2 \cdot n \cdot w(e_{P(h-1),P(h)}) \cdot y_{P(h-1),P(h)} + z_{P(h-1),P(h),n-1} \leq 0$. Note that this constraint introduces the variable $z_{P(h-1),P(h),n-1}$ to R
6. Consider the constraint $-x_{P(h-2)} - x_{P(h-1)} + 2 \cdot n \cdot w(e_{P(h-2),P(h-1)}) \cdot y_{P(h-2),P(h-1)} + 2 \cdot z_{P(h-2),P(h-1),1} \leq 0$ in R. Recall that R contains the variable $z_{P(h-1),P(h),n-1}$. To cancel this variable, R must contain a constraint of the form $-z_{j,P(h-1),n-2} - z_{P(h-1),P(h),n-1} \leq 0$. By construction, $z_{P(h-2),P(h-1),1} = z_{j,P(h-1),n-2}$. Thus, $l = n-2$.
7. Continuing this process, we see that $1 = n - (h-1)$. Thus, $h = n$. As shown previously, the vertices v_1, $v_{P(2)}$, $v_{P(3)}$, …, $v_{P(n)}$ are all distinct. Thus P is a Traveling Salesman Path in $\mathbf{G}$. For each edge $e_{P(i),P(i+1)}$ in P, R contains $2 \cdot nw(e_{P(i),P(i+1)})$ copies of the constraint $-y_{P(i),P(i+1)} \leq 0$. From the observations above, R contains an additional $2 \cdot n$ constraints. Thus, R contains a total of $2 \cdot n \cdot (W'+1)$ constraints where W' is the total length of P. Since R has length $2 \cdot n \cdot (W+1)$, P has length W.

All that remains is to establish that a **PTAS** reduction exists from the Minimum 0-1 Programming problem to the IP-OLTR problem. This will be done by establishing the existence of the functions f, g, and α.

1. The function f: We provided a method for constructing an integer program **I** from a graph **G**. This forms the function f required for the **PTAS** reduction.
2. The function g: We provided a method to take a tree-like refutation of **I** and construct a Traveling Salesman Path in **G**. This forms the function g required for the **PTAS** reduction.
3. The function α: Let W^* be the shortest Traveling Salesman Path in **G**. **I** has a tree-like refutation of length $2 \cdot n \cdot (W^* + 1)$. Additionally, if **I** had a shorter tree-like refutation, then **G** would have a shorter path. Thus, the IP-OLTR of **I** has length $2 \cdot n \cdot (W^* + 1)$. Let $\alpha(\epsilon) = \frac{\epsilon - 1}{2}$.
   Let R be a tree-like refutation of **I** of length $2 \cdot n \cdot (W + 1)$. The function g produces a Traveling Salesman Path of length W. If $\frac{2 \cdot n \cdot (W+1)}{2 \cdot n \cdot (W^*+1)} \leq 1 + \alpha(\epsilon) = \frac{\epsilon+1}{2}$, then

$$\frac{W}{W^*} \leq \frac{2 \cdot W}{2 \cdot W^*} \leq \frac{2 \cdot (W+1)}{W^*+1} \leq \frac{2 \cdot (\epsilon+1)}{2} = 1 + \epsilon.$$

   Thus, the IP-OLTR problem for linear programs is **NPO-hard**. □

Since the IP-OLTR problem is **NPO-hard**, there exists an $\epsilon > 0$ such that the IP-OLTR cannot be approximated to within a factor of $O(2^{n^\epsilon})$, unless $\mathbf{P} = \mathbf{NP}$ [21]. Thus, the IP-OLTR cannot be approximated to within a polynomial factor, unless $\mathbf{P} = \mathbf{NP}$.

Theorem 3. *The IP-OLDR problem is* **NPO-hard**.

Proof. This will be accomplished by a reduction from the Minimum Integer Programming problem.

Consider the following instance of the Minimum 0-1 programming problem:

$$\min \sum_{i=1}^{n} (2 \cdot \log c_i + 1) \cdot x_i \quad \mathbf{A} \cdot \mathbf{x} = \mathbf{b} \quad \mathbf{x} \in \{0,1\}^n.$$

Assume without loss of generality that $\mathbf{c} \geq \mathbf{1}$.

While in general Minimum 0-1 programming is **NPO-complete**, the values of the coefficients in the optimization function are polynomial in the size of the input. Thus, the final value of the objective function is polynomial in the size of the input. Consequently, this problem is **NPO PB-complete** [22,24].

Let **D** be the $n \times n$ matrix such that $d_{i,i} = c_i - 1$ and $d_{i,j} = 0$ for $i \neq j$. Corresponding to the Minimum 0-1 programming instance, we can construct the following linear program **L**:

$$\mathbf{y} \cdot \mathbf{A} + \mathbf{z} \cdot \mathbf{D} \leq \mathbf{0} \quad -\mathbf{z} \leq 0 \quad -\mathbf{y} \cdot \mathbf{b} \leq -1$$

From **L**, we can construct the IP **I** as follows:

1. For each variable y_i in $\mathbf{L}$, add the variable y_i to $\mathbf{I}$. Additionally, add the variable x_0 to $\mathbf{I}$. 2. Add all of the constraints in $\mathbf{L}$ to $\mathbf{I}$. 3. Let p, be the first prime larger than $\max_{i=1\ldots n}(\sum_{i=1}^{m} |a_{ij}|)$. 4. Add the term $p \cdot x_0$ to the constraint L_1 in $\mathbf{L}$ with defining constant -1. By construction, there is exactly one such constraint. We will refer to this new constraint as I_1. 5. Add the constraint $-x_0 \leq 0$ to $\mathbf{I}$.

We will now show that $\mathbf{I}$ has a dag-like integer refutation of length $(k+2)$ if and only if $\mathbf{L}$ has a dag-like linear refutation of length k.

First, assume that $\mathbf{L}$ has a dag-like linear refutation of length k. By construction, any dag-like refutation of $\mathbf{L}$ corresponds to a dag-like derivation of the constraint $c_L \cdot p \cdot x_0 \leq -c_L$ where c_L is the number of times constraint L_1 was used in the dag-like refutation. We can then divide this constraint by $c_L \cdot p$ and sum the result with the constraint $-x_0 \leq 0$ to obtain a contradiction. Since the original refutation had length k, the new refutation has length $(k+2)$.

Now assume that $\mathbf{I}$ has a dag-like integer refutation of length $(k+2)$. As mentioned previously, we can assume without loss of generality that the constraint I_1 is the only constraint in $\mathbf{I}$ with negative defining constant. Thus, it must be used in any refutation of $\mathbf{I}$.

The only other constraint in $\mathbf{I}$ with the variable x_0 is $-x_0 \leq 0$. Thus, this constraint must be used to cancel the variable x_0. We want to avoid using the constraint $-x_0 \leq 0$ p times. Thus, we must first apply the DIV rule to the constraint derived from I_1. By construction, the DIV rule must be applied to this constraint with a coefficient divisible by p. However, by construction, p is larger than any coefficient in any constraint derived from I_1 by a dag-like derivation of length k. Thus, the DIV rule cannot be applied until all other variables are eliminated from I_1. Thus, we must derive the constraint $c_L \cdot p \cdot x_0 \leq -c_L$. This derivation takes k steps and corresponds to a dag-like linear refutation of $\mathbf{I}$. This means that $\mathbf{I}$ has a dag-like linear refutation of length k. Thus, the IP-OLDR problem is **NPO-hard**. □

Since the IP-OLDR problem is **NPO-hard**, there exists an $\epsilon > 0$ such that the IP-OLDR cannot be approximated to within a factor of $O(2^{n^\epsilon})$, unless $\mathbf{P} = \mathbf{NP}$ [21]. Thus, the IP-OLDR cannot be approximated to within a polynomial factor, unless $\mathbf{P} = \mathbf{NP}$.

Note that in general, tree-like and dag-like cutting plane based refutations can be exponentially long [7]. Thus, these problems do not belong to the class **NPO**. However, we can show that both the IP-OLDR problem and the IP-OLTR problem belong to the class **FPSPACE**.

Theorem 4. *The IP-OLTR problem is in* **FPSPACE**.

Proof. Let $\mathbf{I}$ be an infeasible integer program with m constraints over n variables. We will show that a tree-like integer refutation of $\mathbf{I}$ can be constructed by a nondeterministic Turing Machine using working space polynomial in the size of $\mathbf{I}$. We know that any integer program has a tree-like integer refutation of length at most $O(n^{3 \cdot n})$ [13]. Additionally, this refutation only needs to store polynomially many constraints at a time. Thus, each inference in a tree-like integer refutation

can be identified using a number of bits polynomial in the size of the input integer program.

For each inference in a possible refutation R, we can non-deterministically guess the following:

1. The constraints used by the inference – Note that we can assume without loss of generality that the coefficients in these constraints have an absolute value of at most $(n \cdot C_1 + C_2)$ where C_1 is the largest coefficient of any constraint in $\mathbf{I}$ and C_2 is the largest defining constant [16]. Thus, each constraint can be represented using space polynomial in the size of $\mathbf{I}$.
2. The constraint produced by the inference.
3. The source of each constraint used by the inference – This is either the inference used to derive the constraint or the original integer program $\mathbf{I}$.
4. The inference that will use the derived constraint – Note that since the refutation is tree-like, there is at most one such inference.

Thus, each inference can be generated in polynomial space. Once each inference is generated, the space can then be reused to generate the next inference. Thus, the entire refutation can be generated using at most polynomial space.

The correctness of R can similarly be verified in polynomial space as follows:

1. Non-deterministically guess an inference in the refutation.
2. Verify that the derived constraint is correct for the given input constraints.
3. Verify that the input constraints come from the specified sources.
4. Verify that each source is either $\mathbf{I}$ or a previous inference.
5. For each constraint derived by a previous inference, verify that inference lists the current inference as using its derived constraint. Note that this ensures that derived constraints are not repeated.

This can be easily done in space polynomial in the size of $\mathbf{I}$. Once every inference in the refutation is verified, we know that the constraint derived by the last inference is derivable from the constraints in $\mathbf{I}$. If the last inference of R derives a contradiction, then R is a tree-like integer refutation of $\mathbf{I}$. By performing both the construction and verification procedures for each possible refutation length, the first tree-like integer refutation generated in this way is IP-OLTR of $\mathbf{I}$. □

Note that the refutations generated in the proof of Theorem 4 are tree-like because we ensure that each derived constraint is used by at most one future inference. If we remove this restriction, then the procedure instead generates dag-like integer refutations. This gives us the following corollary.

Corollary 1. *The IP-OLDR problem is in* **FPSPACE**.

6 Conclusion

In this paper, we studied the problems of finding optimal length refutations of infeasible integer programs (IPs). We looked at three different types of refutations, namely read-once refutations, tree-like refutations, and dag-like refutations.

Constraint systems are heavily used in the field of software verification [8, 12]. Refutations of these constraint systems provide evidence that the system is infeasible. Thus, refutations, especially short refutations, are also very useful in this field. As a result, the contributions in this paper will provide insights useful in software verification.

Specifically, we showed that the IP-OLRR problem is **NPO PB-complete** while the IP-OLTR and IP-OLDR problems are **NPO-hard** and in **FPSPACE**.

This paper only examined general forms of integer programs. However, restricting the form of the program can change the complexity of the problems examined. For example, in systems of difference constraints, the OLRR, OLTR, and OLDR problems for integer feasibility can be solved in polynomial time. Thus, future work can examine the complexity of these problems for other restricted IPs.

References

1. Alekhnovich, M., Razborov, A.: Resolution is not automatizable unless W[P] is tractable. In: Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, FOCS 2001, pp. 210, USA. IEEE Computer Society (2001)
2. Armando, A., Castellini, C., Mantovani, J.: Software model checking using linear constraints. In: Davies, J., Schulte, W., Barnett, M. (eds.) ICFEM 2004. LNCS, vol. 3308, pp. 209–223. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30482-1_22
3. Atserias, A., Luisa Bonet, M.: On the automatizability of resolution and related propositional proof systems. Inf. Comput. **189**(2), 182–201 (2004)
4. Atserias, A., Müller, M.: Automating resolution is NP-hard. CoRR, abs/1904.02991 (2019)
5. Beame, P., Pitassi, T.: Simplified and improved resolution lower bounds. In: 37th Annual Symposium on Foundations of Computer Science, pages 274–282, Burlington, Vermont, 14–16 October 1996. IEEE (1996)
6. Berste, B., Leconte, M.: Using constraints to verify properties of rule programs. In: Proceedings of the 2010 International Conference on Software Testing, Verification, and Validation Workshops, pp. 349–354 (2008)
7. Bonet, M.L., Pitassi, T., Raz, R.: Lower bounds for cutting planes proofs with small coefficients. J. Symb. Log., **62**(3), 708–728 (1997)
8. Ceberio, M., Acosta, C., Servin, C.: A constraint-based approach to verification of programs with floating-point numbers. In: Proceedings of the 2008 International Conference on Software Engineering Research and Practice, pp. 225–230 (2008)
9. Cheung, K.K.H., Gleixner, A.M., Steffy. D.E.: Verifying integer programming results. In: Integer Programming and Combinatorial Optimization - 19th International Conference, IPCO 2017, vol. 10328 of Lecture Notes in Computer Science, pp. 148–160 (2017)

10. Chvatal, V.: Edmonds polytopes and a hierarchy of combinatorial problems. Discret. Math. **4**(10–11), 886–904 (1973)
11. Collavizza, H., Reuher, N.: Exploration of the capabilities of constraint programming for software verification. In: Proceedings of the 2006 International Conference on Tools and Algorithms for the Construction and Analysis of Systems (2006)
12. Collavizza, H., Rueher, M., Van Hentenryck, P.: CPBPV: a constraint-programming framework for bounded program verification. In: Stuckey, P.J. (ed.) CP 2008. LNCS, vol. 5202, pp. 327–341. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-85958-1_22
13. Cook, W.: Cutting-plane proofs in polynomial space. Math. Program. **47**(1), 11–18 (1990)
14. Cook, W., Coullard, C.R., Turan, Gy. On the complexity of cutting-plane proofs. Discrete Appl. Math. **18**, 25–38 (1987)
15. Fleming, N., Pankratov, D., Pitassi, T., Robere, R.: Random Θ(log n)-CNFs are hard for cutting planes. In: 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, pp. 109–120 (2017)
16. Galesi, N., Pudlák, P., Thapen, N.: The space complexity of cutting planes refutations. In: Proceedings of the 30th Conference on Computational Complexity, CCC 2015, pp. 433–447 (2005)
17. Goldreich, O.: On promise problems (a survey in memory of Shimon even [1935-2004]). Electron. Colloquium Comput. Complex **61**(018) (2005)
18. Sajin Koroth, M., Mertz, I., Pitassi, T.: Automating cutting planes is NP-hard. In: Proceedings of the 52nd Annual ACM SIGACT Symposium on Theory of Computing, STOC 2020, pp. 68–77 (2020)
19. Gulwani, S., Srivastava, S., Venkatesan, R.: Program analysis as constraint solving. In: Proceedings of the 2008 ACM SIGPLAN Conference on Programming language design and implementation, New York, NY. ACM (2008)
20. Hrubes, P., Pudlák, P.: Random formulas, monotone circuits, and interpolation. In: 58th IEEE Annual Symposium on Foundations of Computer Science, FOCS 2017, pp. 121–131 (2017)
21. Kann, V.: On the approximability of np-complete optimization problems. Ph.D. thesis, Royal Institute of Technology Stockholm (1992)
22. Kann, V.: Polynomially bounded minimization problems that are hard to approximate. Nordic J. of Compu. **1**(3), 317–331 (1994)
23. Krentel, M.W.: The complexity of optimization problems. J. Comput. Syst. Sci. **36**(3), 490–509 (1988)
24. Orponen, P., Mannila, H.: On approximation preserving reductions: complete problems and robust measures. Technical report, Department of Computer Science, University of Helsinki (1987)
25. Pudlák, P.: Lower bounds for resolution and cutting plane proofs and monotone computations. J. Symbol Logic **62**(3), 981–998 (1997)
26. Subramani, K., Wojciechowki, P.: Integer feasibility and refutations in UTVPI constraints using bit-scaling. Algorithmica (Accepted In Press 2022)
27. Subramani, K., Wojciechowski, P.J.: A bit-scaling algorithm for integer feasibility in UTVPI constraints. In: Combinatorial Algorithms - 27th International Workshop, IWOCA 2016, vol. 9843, pp. 321–333 (2016)
28. Vinyals, M., Elffers, J., Giráldez-Cru, J., Gocht, S., Nordström, J.: In between resolution and cutting planes: a study of proof systems for pseudo-Boolean SAT solving. In: Beyersdorff, O., Wintersteiger, C.M. (eds.) SAT 2018. LNCS, vol. 10929, pp. 292–310. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94144-8_18