On the computational complexities of finding selected refutations of linear programs

K. Subramani[*], **Piotr Wojciechowski**
LDCSEE, West Virginia University, Morgantown, West Virginia, USA
k.subramani@mail.wvu.edu, pwojciec@mail.wvu.edu

Abstract

In this paper, we establish the computational complexities of selected forms of refutations of linear programs. Linear programming is in the complexity class **P** and hence, it must have short affirmative and disqualifying certificates. One of the more celebrated lemmata in linear programming is Farkas' lemma, which establishes that both "yes" and "no" certificates can be thought of as solutions to complementary linear programs. Since then, it has been established that, if a linear program is feasible, then it must have a solution which is bounded by a polynomial function of the input size. The latter observation, coupled with Farkas' lemma, immediately establishes that linear programming is in **NP** $\cap$ **coNP**. Our goal is to study the computational complexities of finding various constrained refutations of arbitrary linear programs. This paper focuses on three distinct refutation forms, viz., **read-once**, **tree-like** and **dag-like**. We establish that checking if a linear program has a read-once refutation is **NP-complete**, even when it is defined by Binary Two Variable Per Inequality (BTVPI) constraints. Furthermore, the problems of finding the **shortest** tree-like and dag-like refutations are **NPO-complete** and **NPO PB-complete**, respectively.

1 Introduction

In this paper, we focus on the problems of checking if an unsatisfiable linear program has specific types of refutations under the ADD refutation system (see Section 2). A refutation is a "no"-certificate that attests to the infeasibility of the linear programming instance. Certificates are important from the perspective of enhancing trust in the underlying software system. Arbitrary certificates are difficult to verify, and hence there is significant interest in providing certificates of certain constrained forms. This paper focuses on three types of refutations, viz., **read-once**, **tree-like** and **dag-like**. Recall that in a read-once refutation, constraints (input or derived) 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. In a tree-like refutation, derived constraints cannot be reused. However, constraints can be rederived and these rederivations can reuse input constraints. In a dag-like refutation, any constraint can be reused without needing to be rederived.

[*]This research was supported in part by the Defense Advanced Research Projects Agency through grant HR001123S0001-FP-004.

An orthogonal problem is the problem of providing succinct certificates, i.e., certificates satisfying a length criterion under an appropriately defined notion of length. In the case of this paper, the length of a refutation is equal to the number of inference steps in that refutation. Read-once refutations are succinct by definition; they are also **incomplete** in that an unsatisfiable linear program may not have a read-once refutation. Incomplete refutations have their uses in program verification and proof complexity (Iwama and Miyano 1995). Both tree-like and dag-like refutations are **complete**. However, finding refutations of these types which satisfy a length requirement is **NP-hard**.

The notion of refutations is associated with a set of inference rules that can be used to produce contradictions. As mentioned before, the only inference rule in our refutation system is the ADD rule, discussed in Section 2.

Inasmuch as the computational complexity of finding optimal (minimum length) refutations in all three constrained refutation types is **NP-hard**, we analyze the problems of approximating the optimal length. As we will establish later, the problem of checking if a linear program has a read-once refutation is **NP-complete**, even for very restricted forms. Furthermore, the problem of finding the length of a shortest tree-like refutation is **NPO-complete**, and the problem of finding the length of a shortest dag-like refutation is **NPO PB-complete**.

2 Statement of Problems

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

In this paper, we examine proofs of infeasibility for linear programs.

Definition 2.1 *A* **linear program** *(LP) is a conjunction of constraints in which each constraint is an inequality of the form $\mathbf{a}_j \cdot \mathbf{x} \leq b_j$ where $\mathbf{a}_j \in \mathbb{Z}^n$, $b_j \in \mathbb{Z}$, and each variable x_i can take any real value.*

Example (1): System (1) is a linear program.

$$\begin{aligned} 3 \cdot x_1 + 5 \cdot x_2 - 4 \cdot x_3 &\leq -2 \\ -2 \cdot x_2 + 7 \cdot x_3 &\leq 4 \end{aligned} \tag{1}$$

Restricted versions of LPs have also been studied. In some of these restrictions, each constraint has at most two non-zero coefficients.

Definition 2.2 *A* **Two Variable per Inequality (TVPI)** *constraint is a constraint with at most two non-zero coefficients.*

Other restrictions limit the values which the non-zero coefficients can take.

Definition 2.3 *A* **Unit Two Variable per Inequality (UTVPI)** *constraint is a TVPI constraint such that each non-zero coefficient belongs to the set* $\{\pm 1\}$.

An LP in which every constraint is a UTVPI constraint is known as a UTVPI Constraint System (UCS).

Definition 2.4 *A* **Binary Two Variable per Inequality (BTVPI)** *constraint is a TVPI constraint such that each non-zero coefficient belongs to the set* $\{\pm 1, \pm 2\}$.

Note that every UTVPI constraint is a BTVPI constraint and that every BTVPI constraint is a TVPI constraint. An LP in which every constraint is a BTVPI constraint is known as a BTVPI Constraint System (BCS).

Refutations are defined by the inference rules that can be used. Refutations of linear programs can use one inference rule. This rule corresponds to the summation of two constraints and is defined as follows:

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

We refer to Rule (2) as the ADD rule. The above rule is to be interpreted as follows: From the constraints $\sum_{i=1}^{n} a_i \cdot x_i \leq b_1$ and $\sum_{i=1}^{n} a'_i \cdot x_i \leq b_2$, we can derive the constraint $\sum_{i=1}^{n} (a_i + a'_i) \cdot x_i \leq b_1 + b_2$.

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. Furthermore, the rule is **complete** in that, if the original system is linear infeasible, then repeated application of Rule (2) will result in a contradiction of the form: $0 \leq b$, $b < 0$. The completeness of the ADD rule was established by Farkas (Farkas 1902), in a lemma that is famously known as Farkas' Lemma for systems of linear inequalities (Schrijver 1987).

Farkas' lemma and the fact that linear programs must have basic feasible solutions establish that the linear programming problem is in the complexity class **NP** $\cap$ **coNP**.

Definition 2.5 *A* **linear refutation** *is a sequence of applications of the ADD rule that results in a contradiction of the form* $0 \leq b$, $b < 0$.

Example (3): Consider the LP represented by System (3).

$$\begin{aligned} 3 \cdot x_1 + 5 \cdot x_2 - 4 \cdot x_3 &\leq -2 \\ -2 \cdot x_2 + 7 \cdot x_3 \leq 4 \quad - x_1 - x_2 - x_3 &\leq -1 \end{aligned} \tag{3}$$

This system has the following linear refutation:

1. Apply the ADD rule to $3 \cdot x_1 + 5 \cdot x_2 - 4 \cdot x_3 \leq -2$ and $-2 \cdot x_2 + 7 \cdot x_3 \leq 4$ to get $3 \cdot x_1 + 3 \cdot x_2 + 3 \cdot x_3 \leq 2$.
2. Apply the ADD rule to $3 \cdot x_1 + 3 \cdot x_2 + 3 \cdot x_3 \leq 2$ and $-x_1 - x_2 - x_3 \leq -1$ to get $2 \cdot x_1 + 2 \cdot x_2 + 2 \cdot x_3 \leq 1$.
3. Apply the ADD rule to $2 \cdot x_1 + 2 \cdot x_2 + 2 \cdot x_3 \leq 1$ and $-x_1 - x_2 - x_3 \leq -1$ to get $x_1 + x_2 + x_3 \leq 0$.
4. Apply the ADD rule to $x_1 + x_2 + x_3 \leq 0$ and $-x_1 - x_2 - x_3 \leq -1$ to get $0 \leq -1$.

In this paper, we study read-once refutations, tree-like refutations, and dag-like refutations.

Definition 2.6 *A* **read-once** *refutation is a refutation in which each constraint 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.

Example (4): Consider the refutation of System (3) in Example 3. The constraint $-x_1 - x_2 - x_3 \leq -1$ is used multiple times by the refutation. Thus, the refutation in Example 3 is not read-once.

Definition 2.7 *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. Tree-like refutation is a **complete** refutation procedure (Beame and Pitassi 1996).

Definition 2.8 *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.

For any refutation, we can define the length of that refutation.

Definition 2.9 *The* **length** *of a refutation* R *of a constraint system is the number of inferences made in* R.

Example (5): The refutation of System (3) in Example 3 consists of 4 inferences. Thus, the refutation has length 4.

Recall that the feasibility of linear programs can also be determined by using Farkas' Lemma. Note that, by using the Farkas variables, refutations of a linear program can be represented in polynomial space.

In this paper, we study the following problems:

1. The **Read-once Refutation (ROR)** problem: Given an infeasible linear program **L**, does **L** have a read-once linear refutation?
2. The **Optimal Length Read-once Refutation (OLRR)** problem: Given an infeasible linear program **L**, what is the length of the shortest read-once refutation of **L**?
3. The **Optimal Length Tree-like Refutation (OLTR)** problem: Given an infeasible linear program **L**, what is the length of the shortest tree-like refutation of **L**?
4. The **Optimal Length Dag-like Refutation (OLDR)** problem: Given an infeasible linear program **L**, what is the length of the shortest dag-like refutation of **L**?

Lemma 2.1 *Let $\mathbf{L}$ be the infeasible linear program $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ with m constraints over n variables and let $\mathbf{y} \in \mathbb{Z}^m$ be a vector such that $\mathbf{y} \geq \mathbf{0}$, $\mathbf{y} \cdot \mathbf{A} = \mathbf{0}$, and $\mathbf{y} \cdot \mathbf{b} < 0$. $\mathbf{L}$ has a tree-like refutation of length $(\sum_{j=1}^{m} y_j - 1)$ and a dag-like refutation of length at most $(\sum_{j=1}^{m} 2 \cdot \log(y_j + 1) + m - 1)$.*

Proof: Since $\mathbf{L}$ is an infeasible LP, the vector $\mathbf{y}$ is guaranteed to exist by Farkas' Lemma (Farkas 1902). We will now use $\mathbf{y}$ to generate both a tree-like and a dag-like refutation of $\mathbf{L}$.

For $j = 1 \dots m$, let l_j be the constraint $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$. Note that this is the constraint associated with the Farkas variable y_j.

For any constant $c \in \mathbb{Z}^+$, the constraint $c \cdot \mathbf{a_j} \cdot \mathbf{x} \leq c \cdot b_j$ can be derived from $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$ by the following tree-like derivation:

1. Apply the ADD rule to $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$ and $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$ to get $2 \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2 \cdot b_j$.
2. Apply the ADD rule to $2 \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2 \cdot b_j$ and $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$ to get $3 \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 3 \cdot b_j$.
3. $\vdots$
4. Apply the ADD rule to $(c-1) \cdot \mathbf{a_j} \cdot \mathbf{x} \leq (c-1) \cdot b_j$ and $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$ to get $c \cdot \mathbf{a_j} \cdot \mathbf{x} \leq c \cdot b_j$.

Note that this derivation uses the ADD rule $(c-1)$ times.

Thus, $\mathbf{L}$ has the following tree-like refutation:

1. For each constraint l_j, the constraint $y_j \cdot \mathbf{a_j} \cdot \mathbf{x} \leq y_j \cdot b_j$ can be generated by applying the ADD rule $(y_j - 1)$ times.
2. For each $i = 2 \dots m$, apply the ADD rule to $(\sum_{i=1}^{j-1} y_i \cdot \mathbf{a_i}) \cdot \mathbf{x} \leq \sum_{i=1}^{j-1} y_i \cdot b_i$ and $y_j \cdot \mathbf{a_j} \cdot \mathbf{x} \leq y_j \cdot b_j$ to get $(\sum_{i=1}^{j} y_i \cdot \mathbf{a_i}) \cdot \mathbf{x} \leq \sum_{i=1}^{j} y_i \cdot b_i$.
3. After $(m-1)$ applications of the ADD rule this derives the constraint
$(\mathbf{y} \cdot \mathbf{A}) \cdot \mathbf{x} \leq (\mathbf{y} \cdot \mathbf{b})$. Recall that $\mathbf{y} \cdot \mathbf{A} = \mathbf{0}$ and $\mathbf{y} \cdot \mathbf{b} < 0$. Thus, this is a contradiction.

Note that this refutation uses a total of $(\sum_{j=1}^{m} y_j - 1)$ inferences. Thus, $\mathbf{L}$ has a tree-like refutation of length $(\sum_{j=1}^{m} y_j - 1)$ as desired.

For any constant $c \in \mathbb{Z}^+$, the constraint $c \cdot \mathbf{a_j} \cdot \mathbf{x} \leq c \cdot b_j$ can be derived from $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$ by the following dag-like derivation:

1. Apply the ADD rule to $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$ and $\mathbf{a_j} \cdot \mathbf{x} \leq b_j$ to get $2 \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2 \cdot b_j$.
2. Apply the ADD rule to $2 \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2 \cdot b_j$ and $2 \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2 \cdot b_j$ to get $4 \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 4 \cdot b_j$.
3. $\vdots$
4. Apply the ADD rule to $2^{\lfloor \log c \rfloor - 1} \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2^{\lfloor \log c \rfloor - 1} \cdot b_j$ and $2^{\lfloor \log c \rfloor - 1} \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2^{\lfloor \log c \rfloor - 1} \cdot b_j$ to get $2^{\lfloor \log c \rfloor} \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2^{\lfloor \log c \rfloor} \cdot b_j$.
5. Let $S \subseteq \{0, \dots, \lfloor \log c \rfloor\}$ be such that $\sum_{i \in S} 2^i = c$. Note that we have already derived the constraint $2^i \cdot \mathbf{a_j} \cdot \mathbf{x} \leq 2^i \cdot b_j$ for each $i \in S$. Thus, applying the ADD rule $(|S| - 1)$ times lets us derive the constraint $(\sum_{i \in S} 2^i) \cdot \mathbf{a_j} \cdot \mathbf{x} \leq (\sum_{i \in S} 2^i) \cdot b_j$. This is precisely the constraint $c \cdot \mathbf{a_j} \cdot \mathbf{x} \leq c \cdot b_j$.

Note that this derivation uses the ADD rule at most $(2 \cdot \log(c+1))$ times. The remainder of the refutation is the same as the tree-like refutation of $\mathbf{L}$. This refutation uses at most $(\sum_{j=1}^{m} 2 \cdot \log(y_j + 1) + m - 1)$ inferences. Thus, $\mathbf{L}$ has a dag-like refutation of length at most $(\sum_{j=1}^{m} 2 \cdot \log(y_j + 1) + m - 1)$ as desired.

Note that the size of $\mathbf{y}$ is polynomial in the size of $\mathbf{L}$. Thus, the length of this dag-like refutation is also polynomial in the size of $\mathbf{L}$.

Observe that, if $\mathbf{y} \in \{0, 1\}^m$, then both of these refutations are read-once. $\square$

We now define the complexity classes used in this paper.

2.1 Complexity classes

We now define the complexity classes **NPO** and **NPO PB** used in this paper.

We begin by defining the complexity class **NPO** (Orponen and Mannila 1987).

Definition 2.10 *The complexity class* **NPO** *is the set of optimization problems such that:*

1. *The set of instances can be recognized in polynomial time.*
2. *Solutions are polynomially sized and can be verified in polynomial time.*
3. *The objective function can be computed in polynomial time.*

We next define the complexity class **NPO PB** (Kann 1994).

Definition 2.11 **NPO PB** *is the set of* **NPO** *problems for which the value of the objective function is polynomial in the size of the input.*

Finally, we introduce the notion of **PTAS** reductions (Orponen and Mannila 1987).

Definition 2.12 *A* **PTAS** *reduction from problem A to problem B, is a trio of functions f, g, and α computable in polynomial time, such that:*

1. *f maps instances of problem A to instances of problem B.*
2. *g takes an instance x of problem A, an approximate solution to the corresponding problem $f(x)$ in B, and an error parameter ϵ and produces an approximate solution to x.*
3. *α maps error parameters for solutions to instances of problem A to error parameters for solutions to problem B.*
4. *If the solution y to $f(x)$ (an instance of problem B) is at most $(1 + \alpha(\epsilon))$ times worse than the optimal solution, then the corresponding solution $g(x, y, \epsilon)$ to x (an instance of problem A) is at most $(1 + \epsilon)$ times worse than the optimal solution.*

Definition 2.13 *A problem P is* **NPO PB-hard** *under* **PTAS** *reductions, if every problem in* **NPO PB** *can be reduced to P by a* **PTAS** *reduction.*

Unless otherwise stated, we assume that **NPO PB-hardness** is specified with respect to **PTAS** reductions.

The set of problems which are in the class **NPO PB** and are **NPO PB-hard** are called **NPO PB-complete**. Additionally, for every **NPO PB-complete** problem P there exists an $\epsilon > 0$ such that P cannot be approximated to within a factor of $O(n^{\epsilon})$ unless $\mathbf{P} = \mathbf{NP}$ (Berman and Schnitger 1992). Thus, if any **NPO PB-complete** problem can be approximated to within a polylogarithmic factor, then $\mathbf{P} = \mathbf{NP}$.

An example of an **NPO PB-complete** problem is Bounded 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 $\mathbf{1} \cdot \mathbf{x}$. This specific form of Minimum 0-1 Programming is known to be **NPO PB-complete** (Kann 1994).

The principal contributions of this paper are as follows:

1. Establishing that the ROR problem is **NP-complete** for BCSs (see Section 3).
2. Establishing that the OLTR problem for LPs is **NPO-complete** (see Section 4).
3. Establishing that the OLDR problem for LPs is **NPO PB-complete** (see Section 5).

3 Read-once Refutations

In this section, we examine the ROR and OLRR problems for linear programs.

First, we consider the problem of checking whether a linear program has a read-once refutation. Specifically, we show that the ROR problem for BCSs is **NP-complete**.

Theorem 3.1 *The ROR problem for BCSs is* **NP-complete***.*

Proof: The ROR problem is clearly in **NP**, since we can guess a vector $\mathbf{y} \in \{0,1\}^m$ and check that $\mathbf{y} \cdot \mathbf{A} = \mathbf{0}$ and $\mathbf{y} \cdot \mathbf{b} < 0$. Note that this vector corresponds to a set of constraints that, when summed together, produces a contradiction. This summation can be represented as a sequence of applications of the ADD rule where each constraint is used at most once. This is precisely a read-once refutation.

We establish **NP-hardness** through a reduction from the Exact Cover by 3-Sets (X3C) problem. We recall the definition of the X3C problem.

X3C is one of the core six problems proved to be **NP-complete** in (Garey and Johnson 1979). An instance of the X3C problem consists of a set X with $3 \cdot n$ elements and a collection $\mathbf{C}$ of 3-element subsets of X.

The query is: Can we find a $\mathbf{C}' \subseteq \mathbf{C}$, such that every element of X occurs in exactly one member of $\mathbf{C}'$? Note that the query is asking whether there is a subset of $\mathbf{C}'$ that *exactly* covers all the elements of X.

Assume that we are given the following instance of X3C:

1. Set $X = \{x_1, x_2, \ldots, x_{3 \cdot n}\}$.
2. Sets $C_1, C_2, \ldots C_m \subseteq X$, with $|C_i| = 3$, for $i = 1, 2, \ldots, m$.

From this instance, we construct the following instance of ROR.

1. Corresponding to each set $C_j = (x_{j_1}, x_{j_2}, x_{j_3})$, create the constraints $l_{j,1} :\ x_{j_1} - 2 \cdot w_j \leq 0$, $l_{j,2} :\ w_j + x_{j_2} \leq 0$, and $l_{j,3} :\ w_j + x_{j_3} \leq 0$. Note that these constraints are equivalent to the constraint $x_{j_1} + x_{j_2} + x_{j_3} \leq 0$.
2. Create the constraints $l_1 :\ -x_1 + 2 \cdot y_1 \leq -1$, $l_2 :\ -y_1 - x_2 \leq 0$, $l_3 :\ 2 \cdot y_2 - y_1 \leq 0$, $l_4 :\ -y_2 - x_3 \leq 0$, $l_5 :\ 2 \cdot y_3 - y_2 \leq 0, \ldots$, $l_{6 \cdot n-5} :\ 2 \cdot y_{3 \cdot n-2} - y_{3 \cdot n-3} \leq 0$, $l_{6 \cdot n-4} :\ -y_{3 \cdot n-2} - x_{3 \cdot n-1} \leq 0$, and $l_{6 \cdot n-3} :\ -y_{3 \cdot n-2} - x_{3 \cdot n} \leq 0$. Note that these constraints are equivalent to the constraint $-x_1 - \ldots - x_{3 \cdot n} \leq -1$.

The resulting linear program is:

$$\mathbf{L} :\ \bigwedge_{j=1}^{m} (l_{j,1} \wedge l_{j,2} \wedge l_{j,3}) \wedge \bigwedge_{i=1}^{6 \cdot n-3} l_i \qquad (4)$$

We now argue that there is a one-to-one correspondence between Exact Covers of X by 3-Sets and RORs of the constructed linear program.

Consider any exact cover $\mathbf{C}'$ of X by 3-Sets from the given subsets $C_1, \ldots, C_m$. It is clear that $|\mathbf{C}'| = n$. Without loss of generality, we can assume that $C_1, \ldots C_n$ are the subsets picked.

First, let us consider the constraints l_1 through $l_{6 \cdot n-3}$. By construction, each x_i occurs in exactly one constraint with coefficient -1. Additionally, each y_i occurs in one constraint with coefficient 2 and in two constraints with coefficient -1. Thus, summing these constraints results in the constraint $d_0 :\ -x_1 - x_2 - \ldots - x_{3 \cdot n} \leq -1$.

Now focus on the corresponding linear constraints. Corresponding to each C_j, we have the constraints $l_{j,1}$, $l_{j,2}$, and $l_{j,3}$. Summing these constraints results in the constraint $d_j \quad x_{j_1} + x_{j_2} + x_{j_3} \leq 0$. Since the C_js form an exact cover, each variable x_i will occur precisely once across all the d_js, $j = 1, 2, \ldots, n$. When we sum the constraints d_j, $j = 1, 2, \ldots, n$ with the constraint d_0, every variable is canceled and we get the contradiction $0 \leq -1$. Since each constraint was used at most once, this is a read-once refutation.

Now assume that the linear program represented by System (4) has a read-once refutation. This means that some subset of the constraints in $\mathbf{L}$, when added together, produces a contradiction.

By construction, l_1 is the only constraint with a negative defining constant. Thus, l_1 must be part of this refutation; indeed, it is part of *every* refutation (read-once or otherwise).

Any read-once refutation of $\mathbf{L}$ must eliminate the $2 \cdot y_1$ term from l_1. By construction, the only constraints with $-y_1$ are l_2 and l_3. Thus, both of these constraints must be used in the refutation. Summing all three constraints together results in the constraint $-x_1 - x_2 + 2 \cdot y_2 \geq -1$.

Any read-once refutation of $\mathbf{L}$ must eliminate the $2 \cdot y_2$ term from this constraint. By construction, the only constraints with $-y_2$ are l_4 and l_5. Thus, both of these constraints must be used in the refutation. Summing all three constraints together results in the constraint $-x_1 - x_2 - x_3 + 2 \cdot y_3 \geq -1$. This continues until all y_is are eliminated in this fashion. This results in the constraint $d_0 :\ -x_1 - x_2 - \ldots - x_{3 \cdot n} \leq -1$.

Let us consider the set of constraints corresponding to C_j. By construction, these are the only constraints with the variable w_j. Observe that in $l_{j,1}$, w_j has coefficient -2, while in $l_{j,2}$ and $l_{j,3}$ it has coefficient 1. Thus, any read-once refutation of L must use either all three of these constraints, or none of these constraints. Otherwise, a w_j term will be left in the final summation. This means that, from the perspective of read-once refutation, the constraints $l_{j,1}$, $l_{j,2}$ and $l_{j,3}$ are equivalent to the constraint $d_j\ :\ x_{j_1} + x_{j_2} + x_{j_3} \leq 0$. Let $\mathbf{D} = \bigwedge_{j=1}^{m} d_j$.

Since all the variables in d_0 must be canceled by the refutation, the remaining constraints in the refutation correspond to a subset of $\mathbf{D}$ such that each x_i occurs in exactly one constraint. Let $\mathbf{D}'$ denote this set of constraints. When we look at the subsets C_j corresponding to the constraints $d_j \in D'$, it is clear that they form an exact cover by 3-Sets of the set S. $\Box$

4 Tree-like refutations

In this section, we determine the complexity of finding short tree-like refutations in general linear programs.

Theorem 4.1 *The OLTR problem is* **NPO-complete** *under* **PTAS** *reductions.*

Proof: Note that a tree-like refutation of a linear program can be represented by the coefficients generated from Farkas' Lemma (Farkas 1902). Thus, a tree-like refutation R of a linear program $\mathbf{L}$ is polynomially sized in terms of the size of $\mathbf{L}$. Additionally, from Lemma 2.1, the length of a tree-like refutation can be computed in polynomial time. Thus, the OLTR problem is in **NPO**. Now we need to show **NPO-hardness**.

This will be accomplished by a reduction from the Traveling Salesman Path Problem. This problem is **NPO-complete** (Orponen and Mannila 1987).

Let $\mathbf{G}$ be a complete undirected graph with n vertices. From $\mathbf{G}$ we create an LP $\mathbf{L}$ as follows:

1. For each vertex v_i in $\mathbf{G}$, create the variable x_i.
2. Create the constraint $x_1 + 2 \cdot x_2 + 2 \cdot x_3 + \ldots + 2 \cdot x_{n-1} + x_n \leq -1$.
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 $-x_i - x_j + 2 \cdot (n-1) \cdot w(e_{i,j}) \cdot y_{i,j} + 2 \cdot z_{i,j,l} \leq 0$.
5. For each edge $e_{i,n}$, create the constraint $-x_i - x_n + 2 \cdot (n-1) \cdot w(e_{i,n}) \cdot y_{i,n} + z_{i,n,n-1} \leq 0$.
6. For each edge $e_{1,j}$, create the constraint $-x_1 - x_j + 2 \cdot (n-1) \cdot w(e_{1,j}) \cdot y_{1,j} + z_{1,j,1} \leq 0$.
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 $2 \cdot (n-1) \cdot (W+1)$ for $\mathbf{L}$ as follows:

1. Start with the constraint $x_1 + 2 \cdot x_2 + 3 \cdot x_2 + \ldots + 2 \cdot x_{n-1} + x_n \leq -1$.
2. Add the constraint $-x_1 - x_{P(2)} + 2 \cdot (n-1) \cdot w(e_{1,P(2)}) \cdot y_{1,P(2)} + z_{1,P(2),1} \leq 0$ to R.
3. For $i = 2 \ldots n-2$, add the constraint $-x_{P(i)} - x_{P(i+1)} + 2 \cdot (n-1) \cdot w(e_{P(i),P(i+1)}) \cdot y_{P(i),P(i+1)} + 2 \cdot z_{P(i),P(i+1),i} \leq 0$ to R.
4. Add the constraint $-x_{P(n-1)} - x_n + 2 \cdot (n-1) \cdot w(e_{P(n-1),n}) \cdot y_{P(n-1),n} + z_{P(n-1),n,n-1} \leq 0$ to R.
5. For $i = 1 \ldots n-1$, add $2 \cdot (n-1) \cdot w(e_{P(i),P(i+1)})$ 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 contradiction $0 \leq -1$. Additionally, R contains a total of $2 \cdot (n-1) \cdot (W+1)$ constraints. Thus R is a tree-like refutation of length $2 \cdot (n-1) \cdot W$ for $\mathbf{L}$.

Now assume that $\mathbf{L}$ has a tree-like refutation R of length $2 \cdot (n-1) \cdot (W+1)$ for $\mathbf{L}$. 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 $x_1 + 2 \cdot x_2 + 3 \cdot x_2 + \ldots + 2 \cdot x_{n-1} + x_n \leq -1$, 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.
2. To cancel x_1 from C, R must include a constraint of the form $-x_1 - x_j + 2 \cdot (n-1) \cdot w(e_{1,j}) \cdot y_{1,j} + z_{1,j,1} \leq 0$. 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 $-x_{P(2)} - x_j + 2 \cdot (n-1) \cdot w(e_{P(2),j}) \cdot y_{P(2),j} + 2 \cdot z_{P(2),j,1} \leq 0$. 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 $-x_{P(h-1)} - x_{P(h)} + 2 \cdot (n-1) \cdot w(e_{P(h-1),P(h)}) \cdot y_{P(h-1),P(h)} + z_{P(h-1),P(h),1} \leq 0$ in R. Since $P(h) = n$, by construction of $\mathbf{L}$, this constraint must be $-x_{P(h-1)} - x_{P(h)} + 2 \cdot (n-1) \cdot w(e_{P(h-1),P(h)}) \cdot y_{P(h-1),P(h)} + z_{P(h-1),P(h),n-1} \leq 0$. Note that this constraint introduces the variable $z_{P(h-1),P(h),n-1}$ to R
6. Consider the constraint $-x_{P(h-2)} - x_{P(h-1)} + 2 \cdot (n-1) \cdot w(e_{P(h-2),P(h-1)}) \cdot y_{P(h-2),P(h-1)} + 2 \cdot z_{P(h-2),P(h-1),1} \leq 0$ 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, $v_{P(2)}$,

$v_{P(3)}, \ldots, v_{P(n)}$ 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 $2 \cdot (n-1)w(e_{P(i),P(i+1)})$ copies of the constraint $-y_{P(i),P(i+1)} \leq 0$. From the observations above, R contains an additional $2 \cdot (n-1)$ constraints. Thus, R contains a total of $2 \cdot (n-1) \cdot (W'+1)$ constraints where W' is the total length of P. Since R has length $2 \cdot (n-1) \cdot (W+1)$, P has length W.

All that remains is show that this is a **PTAS** reduction. 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 a linear program $\mathbf{L}$ from a graph $\mathbf{G}$. 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 $\mathbf{L}$ and construct a Traveling Salesman Path in $\mathbf{G}$. This forms the function g required for the **PTAS** reduction.
3. The function α: Let W^* be the shortest Traveling Salesman Path in $\mathbf{G}$. $\mathbf{L}$ has a tree-like refutation of length $2 \cdot (n-1) \cdot (W^*+1)$. Additionally, if $\mathbf{L}$ had a shorter tree-like refutation, then $\mathbf{G}$ would have a shorter path. Thus, the OLTR of $\mathbf{L}$ has length $2 \cdot (n-1) \cdot (W^*+1)$. Let $\alpha(\epsilon) = \frac{\epsilon - 1}{2}$.

   Let R be a tree-like refutation of $\mathbf{L}$ of length $2 \cdot (n-1) \cdot (W+1)$. The function g produces a Traveling Salesman Path of length W. If $\frac{2 \cdot (n-1) \cdot (W+1)}{2 \cdot (n-1) \cdot (W^*+1)} \leq 1 + \alpha(\epsilon) = \frac{\epsilon+1}{2}$, 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 OLTR problem for linear programs is **NPO-complete**. □

5 Dag-like refutations

In this section, we determine the complexity of finding short dag-like refutations in general linear programs.

Theorem 5.1 *The OLDR problem is* **NPO PB-complete** *under* **PTAS** *reductions.*

Proof: From Farkas' Lemma, a dag-like refutation R of a linear program $\mathbf{L}$ is polynomially sized in terms of the size of $\mathbf{L}$. Additionally, from Lemma 2.1, the length of a dag-like refutation can be computed in polynomial time. Finally, from Lemma 2.1, the length of the dag-like refutation is polynomial in terms of the size of $\mathbf{L}$. Thus, the OLDR problem is in **NPO PB**. Now we need to show **NPO PB-hardness**.

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

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

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

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** (Kann 1994; Orponen and Mannila 1987).

Let $\mathbf{D}$ 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:

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

We make the following observations about any dag-like linear refutation R of this LP:

1. The constraint $-\mathbf{y} \cdot \mathbf{b} \leq -1$, is the only constraint in the system with a negative defining constant. Thus, it must be part of the refutation.
2. Let $\mathbf{y} \cdot \mathbf{a}_i + (c_i - 1) \cdot z_i \leq 0$ be the i^{th} constraint of $\mathbf{y} \cdot \mathbf{A} + \mathbf{z} \cdot \mathbf{D} \leq \mathbf{0}$, and let x_i be 1, if this constraint is used in the refutation and 0 otherwise.
3. For each constraint $\mathbf{y} \cdot \mathbf{a}_i + (c_i - 1) \cdot z_i \leq 0$, an additional $(c_i - 1)$ copies of the constraint $-z_i \leq 0$ must be used to cancel the z_i term introduced in the refutation. As per the proof of Lemma 2.1, a dag-like refutation requires $(2 \cdot \log c_i + 1)$ uses of the ADD rule to derive the constraint $-(c_i - 1) \cdot z_i$. Since the length of the dag-like refutation is at most $\sum_{i=1}^{n} (2 \cdot \log c_i + 1) \cdot x_i + 1$, each constraint of the form $\mathbf{y} \cdot \mathbf{a}_i + (c_i - 1) \cdot z_i \leq 0$ can be used at most once.
4. Summing all constraints in the refutation results in the constraint $0 \leq -1$. Thus, after canceling the $\mathbf{z}$ variables, we must have that each term of $-\mathbf{y} \cdot \mathbf{b}$ is canceled by a term of $(\mathbf{y} \cdot \mathbf{A}) \cdot \mathbf{x}$. Thus, we must have that $\mathbf{A} \cdot \mathbf{x} = \mathbf{b}$. Thus, $\mathbf{x}$ is a valid solution to the original 0-1 program.

This means that, from a solution $\mathbf{x}$ of the 0-1 programming problem, we can construct a dag-like refutation to this LP. This is done by adding the constraint $\mathbf{y} \cdot \mathbf{a}_i + (c_i - 1) \cdot z_i \leq 0$ and $(c_i - 1)$ copies of the constraint $-z_i \leq 0$ for each $x_i = 1$ in $\mathbf{x}$. From the observations above, this is a dag-like refutation of the constructed LP.

All that remains is show that this is a **PTAS** reduction. 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 a linear program $\mathbf{L}$ from an instance of the Minimum 0-1 Programming problem $\mathbf{M}$. This forms the function f required for the **PTAS** reduction.
2. The function g: We provided a method to take a dag-like refutation of $\mathbf{L}$ and construct a feasible solution to $\mathbf{M}$. This forms the function g required for the **PTAS** reduction.
3. The function α: Let k^* be the optimal solution to $\mathbf{M}$. $\mathbf{L}$ has a dag-like refutation of length (k^*+1). Additionally, if $\mathbf{L}$ had a shorter dag-like refutation, then $\mathbf{M}$ would have a solution for which the optimization function has a lower value. Thus, the OLDR of $\mathbf{L}$ has length (k^*+1). Let $\alpha(\epsilon) = \frac{\epsilon - 1}{2}$.

Let R be a dag-like refutation of $\mathbf{L}$ of length $(k+1)$. The function g produces a solution to $\mathbf{M}$ with value k. Since the feasibility of the zero vector can be tested in polynomial time, we can assume without loss of generality that $k^* \geq 1$. If $\frac{k+1}{k^*+1} \leq 1 + \alpha(\epsilon) = \frac{\epsilon+1}{2}$, then

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

Thus, the OLDR problem for linear programs is **NPO PB-complete**. □

6 Conclusion

This paper was concerned with checking if an unsatisfiable linear program has specific types of refutations. As has been discussed extensively in the literature, general refutations may be difficult to verify and hence the search for constrained certificates is of significant interest. We studied three types of refutations under the ADD refutation rule. Our investigations established that the problem of checking if a restricted linear program (BTVPI constraint system) has a read-once refutation is **NP-complete**. Furthermore, the problems of finding the shortest tree-like and shortest dag-like refutations are **NPO-complete** and **NPO PB-complete** respectively. Our results essentially rule out the existence of efficient approximation schemes with "good" error bounds, unless **P**=**NP**. From our perspective, an interesting line of work is investigating **fixed-parameter** (**FPT**) approaches for these problems.

From our perspective, the following problems are worth pursuing:

1. Closing the gap - We established that the read-once refutation problem for linear programs is **NP-hard**, using an extremely restricted class of inputs. However, for establishing approximation complexity, we used the class of arbitrary linear programs. The question of interest is whether the **NPO PB-completeness** holds for the restricted class of linear programs or whether this class belongs to a smaller approximation complexity class such as **APX**.
2. Algorithmic approaches - Our results do not rule out the existence of fixed parameter tractable (**FPT**) algorithms for the problems discussed in this paper. Indeed, we have had some success with designing **FPT** algorithms for similar problems (Subramani and Wojciechowski 2020).

References

Armando, A.; Castellini, C.; and Mantovani, J. 2004. Software model checking using linear constraints. In *Lecture Notes in Computer Sciente*, volume 3308, 209–223. Springer.

Beame, P.; and Pitassi, T. 1996. Simplified and Improved Resolution Lower Bounds. In *37th Annual Symposium on Foundations of Computer Science*, 274–282. Burlington, Vermont: IEEE.

Berman, P.; and Schnitger, G. 1992. On the complexity of approximating the independent set problem. *Information and Computation*, 96(1): 77 – 94.

Berstel, B.; and Leconte, M. 2008. Using constraints to verify properties of rule programs. In *Proceedings of the 2010 Int. Conf. on Software Testing, Verification, and Validation Workshops*, 349–354.

Ceberio, M.; Acosta, C.; and Servin, C. 2008. A Constraint-based approach to Verification of Programs with Floating-point Numbers. In *Proceedings of the 2008 Int. Conf. of Software Engineering Research and Practice*, 225–230.

Collavizza, H.; and Reuher, M. 2006. Exploration of the capabilities of constraint programming for software verification. In *Proceedings of the 2006 Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*.

Collavizza, H.; Reuher, M.; and Van Hentenryck, P. 2004. CPBPV:A constraint-programming framework for bounded program verification. In *Proceedings of the 2008 Int. Conf. on Principles and Practices of Constraint Programming*, volume 5202 of *Lecture Notes in Computer Science*. Springer.

Cormen, T. H.; Leiserson, C. E.; Rivest, R. L.; and Stein, C. 2001. *Introduction to Algorithms.* MIT Press.

Cousot, P.; and Halbwachs, N. 1978. Automatic Discovery of Linear Restraints Among Variables of a Program. In *POPL*, 84–96.

Dill, D. L. 1989. Timing assumptions and verification of finite-state concurrent systems. In *International Conference on Computer Aided Verification*, 197–212. Springer.

Farkas, G. 1902. Über die Theorie der Einfachen Ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124(124): 1–27.

Garey, M. R.; and Johnson, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman Company, San Francisco.

Gulwani, S.; Srivastava, S.; and Venkatesan, R. 2008. Program analysis as constraint solving. In *Proceedings of the 2008 ACM SIGPLAN Conf. on Programming language design and implementation*. New York, NY: ACM.

Iwama, K.; and Miyano, E. 1995. Intractability of Read-Once Resolution. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory (SCTC '95)*, 29–36. Los Alamitos, CA, USA: IEEE Computer Society Press. ISBN 0-8186-7052-5.

Kann, V. 1994. Polynomially Bounded Minimization Problems That Are Hard to Approximate. *Nordic J. of Computing*, 1(3): 317–331.

Kleine Büning, H.; Wojciechowski, P. J.; Chandrasekaran, R.; and Subramani, K. 2019. Restricted Cutting Plane Proofs in Horn Constraint Systems. In Herzig, A.; and Popescu, A., eds., *Frontiers of Combining Systems - 12th International Symposium, FroCoS 2019, London, UK, September 4-6, 2019, Proceedings*, volume 11715 of *Lecture Notes in Computer Science*, 149–164. Springer.

Kleine Büning, H.; Wojciechowski, P. J.; and Subramani, K. 2018. Finding read-once resolution refutations in systems of 2CNF clauses. *Theor. Comput. Sci.*, 729: 42–56.

Laviron, V.; and Logozzo, F. 2009. SubPolyhedra: A (More) Scalable Approach to Infer Linear Inequalities. In Jones, N. D.; and Müller-Olm, M., eds., *Verification, Model Checking, and Abstract Interpretation: 10th International Conference, VMCAI 2009, Savannah, GA, USA, January 18-20, 2009. Proceedings*, 229–244. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-93900-9.

Miné, A. 2006. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1): 31–100.

Nemhauser, G. L.; and Wolsey, L. A. 1999. *Integer and Combinatorial Optimization*. New York: John Wiley & Sons.

Orponen, P.; and Mannila, H. 1987. On approximation preserving reductions: Complete problems and robust measures. Technical report, Department of Computer Science, University of Helsinki.

Schrijver, A. 1987. *Theory of Linear and Integer Programming*. New York: John Wiley and Sons.

Subramani, K. 2009. Optimal Length Resolution Refutations of Difference Constraint Systems. *Journal of Automated Reasoning (JAR)*, 43(2): 121–137.

Subramani, K.; and Wojciechowki, P. 2019. A Polynomial Time Algorithm for Read-Once Certification of Linear Infeasibility in UTVPI Constraints. *Algorithmica*, 81(7): 2765–2794.

Subramani, K.; and Wojciechowski, P. 2020. Finding read-once refutations in 2CNF formulas and variants - a parameterized perspective. In *The* 16^{th} *International Symposium on Artificial Intelligence and Mathematics*.

Wojciechowski, P.; and Subramani, K. 2022. On the lengths of tree-like and Dag-like cutting plane refutations of Horn constraint systemsi. *Annals of Mathematics and Artificial Intelligence*, 90: 979–995.

Wojciechowski, P.; and Subramani, K. 2023. A Faster Algorithm for Determining the Linear Feasibility of Systems of BTVPI Constraints. In Gasieniec, L., ed., *SOFSEM 2023: Theory and Practice of Computer Science - 48th International Conference on Current Trends in Theory and Practice of Computer Science, SOFSEM 2023, Nový Smokovec, Slovakia, January 15-18, 2023, Proceedings*, volume 13878 of *Lecture Notes in Computer Science*, 313–327. Springer.

Wojciechowski, P.; Subramani, K.; and Chandrasekaran, R. 2022. Analyzing Read-Once Cutting Plane Proofs in Horn Systems. *Journal of Automated Reasoning (JAR)*, 66: 239–274.

Wojciechowski, P. J.; Subramani, K.; Velasquez, A.; and Williamson, M. D. 2022. On the Approximability of Path and Cycle Problems in Arc-Dependent Networks. In Balachandran, N.; and Inkulu, R., eds., *Algorithms and Discrete Applied Mathematics - 8th International Conference, CALDAM 2022, Puducherry, India, February 10-12, 2022, Proceedings*, volume 13179 of *Lecture Notes in Computer Science*, 292–304. Springer.

Yannakakis, M. 1991. Expressing Combinatorial Optimization Problems by Linear Programs. *Journal of Computer and System Sciences*, 43(3): 441–466.