2019

Analyzing Satisfiability and Refutability in Selected Constraint Systems

Piotr Jerzy Wojciechowski
West Virginia University, pwojciec@mix.wvu.edu

Analyzing Satisfiability and Refutability in Selected Constraint Systems

Piotr Jerzy Wojciechowski

Dissertation submitted to the
College of Engineering and Mineral Resources
at West Virginia University
in partial fulfillment of the requirements
for the degree of

Doctor of Philosophy
in
Computer Science

K. Subramani, Ph.D., Chair
Elaine Eschen, Ph.D.
Frances VanScoy, Ph.D.
Hong-Jian Lai, Ph.D.
John Goldwasser, Ph.D.

Lane Department of Computer Science and Electrical Engineering

Morgantown, West Virginia
2019

Keywords: Constraint Systems, CNF clauses, UTVPI Constraints, Horn Constraints, Satisfiability, Refutability

Abstract

Analyzing Satisfiability and Refutability in Selected Constraint Systems

Piotr Jerzy Wojciechowski

This dissertation is concerned with the satisfiability and refutability problems for several constraint systems. We examine both boolean constraint systems, in which each variable is limited to the values **true** and **false**, and polyhedral constraint systems, in which each variable is limited to the set of real numbers $\mathbb{R}$ in the case of linear polyhedral systems or the set of integers $\mathbb{Z}$ in the case of integer polyhedral systems. An important aspect of our research is that we focus on providing certificates. That is, we provide satisfying assignments or easily checkable proofs of infeasibility depending on whether the instance is feasible or not. Providing easily checkable certificates has become a much sought after feature in algorithms, especially in light of spectacular failures in the implementations of some well-known algorithms. There exist a number of problems in the constraint-solving domain for which efficient algorithms have been proposed, but which lack a certifying counterpart. When examining boolean constraint systems, we specifically look at systems of 2-CNF clauses and systems of Horn clauses. When examining polyhedral constraint systems, we specifically look at systems of difference constraints, systems of UTVPI constraints, and systems of Horn constraints.

For each examined system, we determine several properties of general refutations and determine the complexity of finding restricted refutations. These restricted forms of refutation include read-once refutations, in which each constraint can be used at most once; literal-once refutations, in which for each literal at most one constraint containing that literal can be used; and unit refutations, in which each step of the refutation must use a constraint containing exactly one literal. The advantage of read-once refutations is that they are guaranteed to be short. Thus, while not every constraint system has a read-once refutation, the small size of the refutation guarantees easy checkability.

Acknowledgments

First and foremost I would like to thank my adviser Dr. K. Subramani whose insights and advice made this thesis possible. He has constantly supported me in my research efforts and has introduced me to many interesting areas of study. He introduced me to this are of study and his insights into these problems have made an invaluable contribution. Throughout my time as his student he has taught me many of the proof techniques utilized in the proofs contained within this dissertation. He has never hesitated in helping me correct my mistakes and in teaching me how to avoid those types of errors in the future. His input has made me better at both thoroughly examining the problem being researched and at writing the results of that research.

I would also like to thank my family for being there to support me in my research efforts. They have always been there for me during times of emotional hardship. They have also given me the confidence to succeed when the stress of research work got too much for me to handle by myself. This support and motivation has allowed me to keep working despite several setbacks.

Additionally, I thank the people with whom I have collaborated. In particular I thank Dr. Hans Kleine Büning, who has been a collaborator on several papers dealing with Boolean constraints. His experience with such problems has expanded my knowledge of these constraint systems and his insights and contributions to our research made many of the results in this dissertation possible. I would also like to thank Dr. Chandrasekaran with whom

we have collaborated several times. He has contributed greatly to my understanding of horn constraints and this understanding has made many of the results in this dissertation possible.

West Virginia University has provided me with the education and reseach opportunities that made this work possible. In particular the Lane Department of Computer Science and Electrical Engineering, along with its chair Dr. Brian Woerner, have given me the knowledge and funding to be able to perform the research which resulted in this dissertation.

I also thank Dr. Elaine Eschen, Dr. Frances VanScoy, Dr. Hong-Jian Lai, and Dr. John Goldwasser for agreeing to be on my dissertation committee.

This research was supported in part by the National Science Foundation through Award CCF-1305054, by NASA WV EPSCoR Grant #NNX15AK74A, and by the AFOSR through grant FA9550-19-1-017.

Contents

List of Figures

List of Tables

Part I

Preliminaries

Chapter 1

Introduction

Constraint satisfiability problems (CSPs) are a class of problem concerned with determining if there exists an assignment which satisfies a given set of constraints. CSPs come in many forms, as a result they find applications in a large and diverse number of problem domains. These include but are not limited to program verification [LM05], abstract interpretation [Min06, CC77], real-time scheduling [GPS95a] and operations research.

The research documented in this dissertation is primarily concerned with the satisfiability and refutability problems for CSPs. We focus on Boolean CSPs, in which variables can be assigned either **true** or **false**, integer polyhedral CSPs, in which variables can be assigned any integer value, and linear polyhedral CSPs, in which variables can be assigned any real value.

1.1 Proofs and Refutations

This dissertation focuses on certificates. This refers to both certificates of feasibility provided by proof systems and refutations provided by refutation systems. For the CSPs examined, a satisfying assignment constitutes a certificate of feasibility since it is easy to check if the assignment satisfies all of the constraints in the CSP.

Creating certifying algorithms is important because it validates the results of already implemented algorithms. Even if an algorithm has already been proven to give the correct result, it is still possible for the algorithm to have been implemented incorrectly. An infamous example of such an incorrect implementation is an error in the implementation of a planarity testing algorithm in the LEDA software [MN99]. Consequently, there is widespread interest in the design and development of certifying algorithms, i.e., algorithms which provide certificates that validate the answer that is provided.

Just as certificates of feasibility are used to prove the satisfiability of a CSP, refutations are used to prove unsatisfiability. The refutations examined in this dissertation are more varied than the certificates of feasibility and depend on both the CSP and the refutation system being used. In general we are interested in refutations which are both provably short and easy to verify.

The problem of finding short refutations is one of the principal problems in proof complexity [BP98]. Research proceeds along the lines of finding lower bounds on the lengths of refutations for propositional tautologies (actually, contradictions) in proof systems of increasing complexity, with a view towards separating the complexity class **NP** from the class **coNP** [Urq95].

For each CSP examined in this dissertation we determine several properties of general refutations and determine the complexity of finding restricted refutations. These restricted forms of refutation include read-once refutations, in which each constraint can be used at most once; literal-once refutations, in which for each literal at most one constraint containing that literal can be used; and unit refutations, in which each step of the refutation must use a constraint containing exactly one literal.

Fot the problem of finding read-once refutations, [IM95] showed that that checking if a boolean CSP has a read-once refutaion is **NP-complete**. This result was strenghtened in [KZ02]. This paper showed that it is **NP-complete** to check if a Boolean CSP has a read-once unit-resolution. In [Sze01], it was shown that the problem of finding literal-once resolution refutations for CNF formulas is **NP-complete**. An even stronger result was

obtained in [ABMP98]. This paper showed that is is not possible to linearly approximate the shortest resolution proof of a Horn formula unless **P** = **NP**. This result is interesting because it is easy to see that every unsatisfiable Horn formula has a resolution refutation that is at most quadratic in the number of variables.

We also look at CSPs where there are restrictions on solutions that are not expressed within the set of constraints **C**. Such constraints are known as side constraints and CSPs with these additional constraints are known as CSPs with Side Constraints (CSPSCs).

In some cases side constraints can be expressed using the language of the original CSP. In such cases, the CSPSC is equivalent to a CSP where the side constraints are incorporated into the set of regular constraints.

Once such CSPSC is a constrained form of satisfiability of Boolean Constraint Systems known as Not-All-Equal satisfiability (NAE-satisfiability). In NAE-satisfiability, the side constraint is that no clause can have all of its literals assigned to the same value. This means that each clause ϕ has at least one literal set to **true** and at least one literal set to **false**.

This is equivalent to requiring that the negation of at least one literal in ϕ is **true**. This requirement can be incorporated by adding a new clause ϕ' consisting of the negations of each of the literals in ϕ. Doing this for every clause in a CNF formula Φ generates a new formula that is satisfiable if and only if the original formula is NAE-satisfiable.

There are also forms of CSPSCs where the side constraints cannot be expressed using the language of the original CSP.

1.2 Constraint Systems

When examining Boolean CSPs, we specifically look at systems of where each clause has 2 literals (2-CNF) and systems where each clause has at most one positive literal and any number of negative literals (Horn). The satisfiability problem for both of these systems is in **P** [CLRS01]. This is in contrast to general Boolean CSPs for which the satisfiability problem is **NP-complete** [CLRS01]. The lower complexity of these problems yields inter-

esting results when examining the complexity of restricted forms of refutations as we do in this dissertation.

For Boolean CSPs, this dissertation is only concerned with resolution (See Section 3.1). However other, more powerful refutation systems exist. These include Frege Proofs, Sequent Calculus, the Davis-Putnam Procedure, and Extended Frege Proofs [Sab]. Since Boolean CSPs are **NP-complete** if any of these refutation systems is guaranteed to generate polynomially sized refutations, then **NP** = **coNP**.

Resolution is one of the weakest proof systems. However, despite the weakness of resolution as a proof system it is still difficult to show that refutation length has an exponential lower bound. The first non-trivial lower bound on the length of resolution proofs is in [Hak85]. This paper showes that any resolution based proof of the pigeonhole principle requires exponentially many steps, in the size of the input formula.

When examining polyhedral CSPs, we look at systems where the coefficient is in the set $\{0, 1, -1\}$. We then place further restrictions on how many variables with each coefficient can be in any one constraint. Specifically we look at systems where each constraints has at most one variable with coefficient 1 and at most one variable has coefficient -1 (difference constraints), systems where at most two variables have non-zero coefficients (UTVPI constraints), and systems where at most one variable has coefficient 1 but any number of variables can have coefficient -1 (Horn constraints). The satisfiability problems for these systems are in **P** for both the linear and integer cases.

The certificates of integer infeasibility that we generate for UCSs are concise. This means that the size of the refutation is polynomial in the size of the input. However, certificates of integer infeasibility for polyhedral CSPs in general are not concise. This is because integer programming is **NP-hard**. Thus, integer polyhedral CSPs cannot have concise refutations unless **NP** = **coNP**.

In polyhedral CSPs, refutations are closely related to theorems of the alternative. Typically, theorems of the alternative connect pairs of linear constraint systems and have the following form: Given two linear systems **A** and **B**, exactly one of them is feasible. System

A is called the primal system and System **B** is called the dual system. It is not hard to see that theorems of the alternative provide certificates of infeasibility. One famous theorem of the alternative is Farkas' Lemma [Sch87]. Observe that theorems of the alternative provide natural certificates of infeasibility. For instance, if we are required to prove that a difference constraint system is infeasible, then we can produce a negative cost cycle in the corresponding constraint network [CLRS01].

In particular, Farkas' lemma provides Farkas variables for a linear polyhedral CSP. These correspond to a refutation of that CSP. There are no obvious certificates for integer infeasibility in linear programs. When the constraint matrix of an integer program satisfies certain structural properties then linear feasibility implies integer feasibility. For a discussion on these specialized integer programs, see [Kan83, VD68, Cha81]. In these cases, certificates of linear infeasibility also serve as certificates of integer infeasibility. However, when linear feasibility does not imply integer feasibility, the problem of providing certificates of integer infeasibility becomes non-trivial. A few interesting structural characterizations are described in [Las04] and [Cha15].

These theorems of the alternative are closely realted to the concept of game theory. [Voh06] describes multiple applications of Farkas' lemma. These applications include game theory. [Dan51] describes the relationship between linear programming and two-person zero-sum games. [LR57] and other papers show that the concept of strong duality in linear programming is a corollary of a theorem in game theory known as the Minimax Theorem. [Adl13] discusses this relationship in detail.

A lot of the work in finding short refutations focuses on discrete domains. However, [Sub09] considers the problem of finding optimal length refutations for systems of difference constraints. In that paper it was shown that short refutations exist for difference constraints and also that the optimal length refutations for systems of difference constraints can be determined in polynomial time. The algorithm in [Sub09] is based runs in time $O(n^3 \cdot \log n)$ on a DCS with n variables and utilizes dynamic programming. In [SWG13], a time of $O(m \cdot n \cdot k)$, where m is the number of constraints and k is the length of the shortest

refutation was obtained utilizing a different dynamic programming technique. It is worth noting that in DCSs, linear and integer feasibility coincide and therefore, the departure is not strict. Furthermore, as pointed out in [Sub09], every minimal refutation (i.e., a refutation without redundant constraints) is necessarily read-once and literal-once, since every minimal refutation corresponds to a simple negative cost cycle in the corresponding constraint network [CLRS01]. In this paper though, we consider the problem of read-once refutations in UCSs. Unlike DCSs, linear feasibility does not imply integer feasibility in UCSs [SW17b]. UTVPI constraints occur in a number of problem domains. These domains include program verification [LM05], abstract interpretation [Min06, CC77], real-time scheduling [GPS95a] and operations research [HN94].

1.3 Road Map

The rest of Part I introduces the various constraint systems, refutations, and problems covered in this dissertation. In Chapter 2 we define the constraint systems being examined. Chapter 3 defines the types of refutations considered. In Chapter 4 we define the problems covered in this dissertation. Chapter 5 examines general properties of proof systems and refutation systems.

Part II covers our results for Boolean formulas. Chapter 6 covers our results for 2-CNF formulas. In Chapter 7 we give our results for 3-CNF formulas. Chapter 8 covers our results for Horn formulas.

Part III covers our results for linear polyhedral constraints. Chapter 9 covers our results for difference constraints. In Chapter 10 we describe or results for UTVPI constraints. Chapter 11 covers our results for Horn constraints.

Part IV covers our results for integer polyhedral constraints. In Chapter 12 we describe or results for UTVPI constraints. Chapter 13 covers our results for Horn constraints.

Part V covers our results for quantified systems of linear constraints. In Chapter 14 we describe or results for quantified linear programs. Chapter 15 covers our results for

quantified linear implications.

Part VI summarizes our results and discusses avenues for future research. Chapter 16 contains a summary of our results. In Chapter 17 we mention possible ways the research in this dissertation can be expanded upon in the future. Chapter 18 has a list of the publications containing the work described in this dissertation.

Chapter 2

Constraint Systems

In this chapter we will describe the constraint systems studied within this dissertation. We explore several types of Constraint Satisfaction Problems.

***Definition* 2.0.1.** *A* **Constraint Satisfaction Problem (CSP)** *is defined by the triplet* $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$*, where:*

1. $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$ *denotes a set of program variables.*
2. $\mathbf{D} = \{D_1, D_2, \ldots D_n\}$ *denotes the set of their respective domains.*
3. $\mathbf{C} = \{C_1, C_2, \ldots, C_m\}$ *denotes a set of constraints over the program variables.*

We examine three types of CSP. These are Boolean constraint systems, linear polyhedral systems, and integer polyhedral systems.

Example 1:

- Boolean Systems: Find $\mathbf{x} \in \{\mathbf{true}, \mathbf{false}\}^3$ such that $(x_1 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$.
- Linear Polyhedral Systems: Find $\mathbf{x} \in \mathbb{R}^3$ such that $x_1 - x_2 \leq 3$ and $x_2 - x_3 \leq -2$.
- Integer Polyhedral Systems: Find $\mathbf{x} \in \mathbb{Z}^2$ such that $x_1 + x_2 \leq 1$ and $-x_1 - x_2 \leq 0$.

For CSPs we are interested in two general types of problems satisfiability and refutability.

***Definition* 2.0.2.** *A CSP is* **satisfiable** *if there exists an assignment* $\mathbf{x}^*$ *such that*

1. *For each* $i = 1 \ldots n$, $x_i^* \in D_i$.

2. *Assigning each* $x_i = x_i^*$ *for* $i = 1 \ldots n$ *satisfies each constraint* C_j *for* $j = 1 \ldots m$.

A satisfying assignment to a CSP is known as a certificate of feasibility.

Example 2: The Boolean formula $(x_1 \vee \neg x_2) \wedge (\neg x_2 \vee x_3)$ is satisfiable. This can bee shown by assigning $x_1 =$ **true**, $x_2 =$ **false**, and $x_3 =$ **true**.

***Definition* 2.0.3.** *A CSP is* **refutable** *if it is possible to derive a contradiction from the constraints in* **C**.

2.1 Boolean Constraint Systems

We now explicitly define boolean constraint systems and introduce the specific boolean constraint systems examined in this dissertation.

First we need to define several terms used when discussing Boolean constraint systems.

***Definition* 2.1.1.** *A* **literal** *is a variable x or its complement* $\neg x$. *x is referred to as a positive and* $\neg x$ *is referred to as a negative literal.*

***Definition* 2.1.2.** *A* **CNF clause** *is a disjunction of literals. The empty clause, which is always false, is denoted as* $\sqcup$.

Example 3: $(x_1 \vee \neg x_2 \vee x_3)$ is a CNF clause. This clause can be satisfied by setting x_1 or x_3 to **true** or by setting x_2 to **false**.

We can now formally define Boolean constraint systems.

***Definition* 2.1.3.** *A* **Boolean constraint system** *is a CSP such that:*

1. *For each variable x_i, the corresponding domain $D_i = \{\textbf{true}, \textbf{false}\}$.*

2. *Each constraint C_j is a CNF clause.*

Such a constraint system is also called a Boolean formula.

Example 4: The Boolean formula $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$ is satisfiable. This can bee shown by assigning $x_1 = \textbf{true}$, $x_2 = \textbf{true}$, and $x_3 = \textbf{true}$. Note that each conjunction is satisfied. Thus, this assignment constitutes a certificate of feasibility.

Instead of focusing on general Boolean formulas, we restrict out examination to Boolean formulas in which the clauses have specific structure.

2.1.1 2-CNF clausal formulas

The first type of Boolean formulas we examine are 2-CNF formulas.

***Definition* 2.1.4.** *A* 2**-CNF clause** *is a CNF clause with at most* 2 *literals.*

Example 5: The clause $(x_1 \vee \neg x_2)$ is a 2-CNF clause. However, $(x_1 \vee x_2 \vee \neg x_3)$ is not since it has 3 literals.

***Definition* 2.1.5.** *A* 2**-CNF formula** *is a Boolean formula in which each clause is a* 2*-CNF clause.*

2.1.2 Horn clausal formulas

The second type of Boolean formulas we examine are Horn formulas.

***Definition* 2.1.6.** *A* **Horn** *clause is a CNF clause which contains at most one positive literal.*

Example 6: The clause $(\neg x_1 \vee x_2 \vee \neg x_3 \vee \neg x_4)$ is a Horn clause. However, $(x_1 \vee x_2 \vee \neg x_3)$ is not since it has 2 positive literals.

***Definition* 2.1.7.** *A* **Horn formula** *is a Boolean formula in which each clause is a Horn clause.*

2.2 Polyhedral Constraint Systems

We now explicitly define polyhedral constraint systems and introduce the specific polyhedral constraint systems examined in this dissertation.

***Definition* 2.2.1.** *A* **polyhedral constraint system** *is a CSP in which each constraint in* $\mathbf{C}$ *is an inequality of the form* $\mathbf{a}_j \cdot \mathbf{x} \leq b_j$.

Note that $\mathbf{C}$ can be represented in matrix form as $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$.

Depending on the domain assigned to each variable, a polyhedral constraint system can be either a linear polyhedral system or an integer polyhedral system.

***Definition* 2.2.2.** *A* **linear** *polyhedral constraint system is a polyhedral constraint system in which each variable* x_i*, the corresponding domain* $D_i = \mathbb{R}$.

Such a constraint system is known as a linear program (LP).

Example 7: Consider the LP

$$x_1 + x_2 \leq 1 \qquad x_1 - x_2 \leq 0 \qquad x_2 - x_1 \leq 0$$

is feasible. This can bee shown by assigning $x_1 = \frac{1}{2}$ and $x_2 = \frac{1}{2}$. Note that each constraint is satisfied. Thus, this assignment constitutes a certificate of feasibility.

***Definition* 2.2.3.** *An* **integer** *polyhedral constraint system is a polyhedral constraint system in which each variable* x_i*, the corresponding domain* $D_i = \mathbb{Z}$.

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

Example 8: Consider the IP

$$x_1 + x_2 \leq 2 \qquad x_1 - x_2 \leq 0 \qquad x_2 - x_1 \leq 0$$

is feasible. This can bee shown by assigning $x_1 = 1$ and $x_2 = 1$. Note that each constraint is satisfied. Thus, this assignment constitutes a certificate of feasibility.

Instead of focusing on general polyhedral constraint systems, we restrict out examination to polyhedral constraint systems in which the constraints have specific structure.

2.2.1 Difference constraint systems

The first type of polyhedral constraint systems examined are difference constraint systems. Difference constraint systems consist of absolute constraints and difference constraints.

***Definition* 2.2.4.** *A constraint of the form $a_i \cdot x_i \leq b_i$ is called an* **absolute constraint** *if $a_i \in \{1,-1\}$*

***Definition* 2.2.5.** *A constraint of the form $a_i \cdot x_i + a_j \cdot x_j \leq b_{ij}$ is called a* **difference constraint**, *if $a_i, a_j \in \{1,-1\}$ and $a_i = -a_j$.*

Example 9: The following are difference constraints:

- $x_1 - x_2 \leq 3$.
- $x_2 - x_4 \leq 5$.

***Definition* 2.2.6.** *A conjunction of difference constraints and absolute constraints is called a* **Difference Constraint System (DCS)**.

A DCS can be represented using a directed graph.

From a DCS **D**, we can construct a directed graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$ as follows:

1. For each variable x_i add the vertex x_i to **V**.
2. Add the vertex x_0 to **V**.
3. For each constraint of the form $x_i - x_j \leq b_{ij}$, add the edge $x_j \xrightarrow{b_{ij}} x_i$ to **E**.
4. For each constraint of the form $x_i \leq b_i$, add the edge $x_0 \xrightarrow{b_i} x_i$ to **E**.
5. For each constraint of the form $-x_i \leq b_i$, add the edge $x_i \xrightarrow{b_i} x_0$ to **E**.

Example 10: Consider the DCS represented by System (2.1).

$$\begin{array}{lll} x_1 - x_2 \leq 3 & x_1 - x_3 \leq 5 & x_2 - x_3 \leq 0 \\ x_3 - x_1 \leq -1 & -x_1 \leq 1 & x_3 \leq -7 \end{array} \tag{2.1}$$

The corresponding directed graph is shown in Figure 2.1.

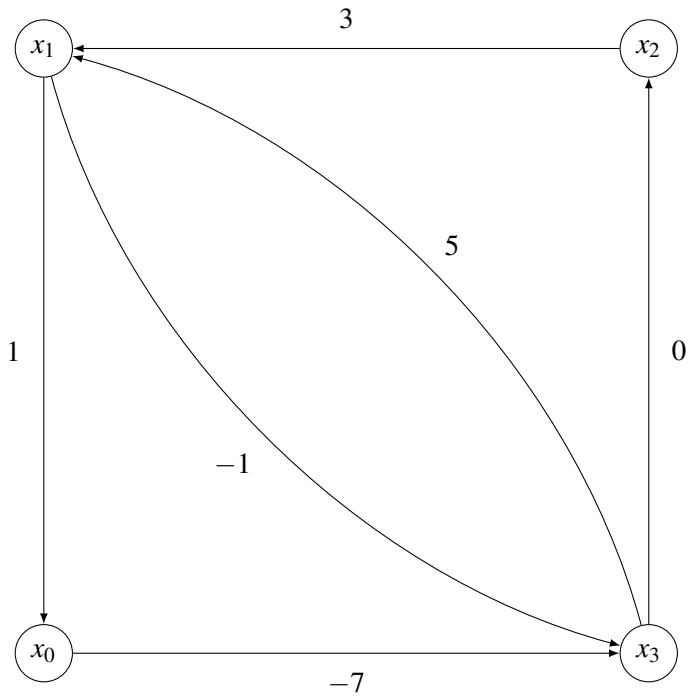

Figure 2.1: Directed graph corresponding to System (2.1).

This representation is important because the infeasibility of a DCS can be determined by identifying if the corresponding graph contains negative cycles. Every negative cycle in the graph corresponds to a subset of constraints in the DCS that can be summed together to form a contradiction of the form $0 \leq b$, where $b < 0$.

Example 11: The graph in Figure 2.1 has the negative cycle $x_0 \xrightarrow{-7} x_3 \xrightarrow{5} x_1 \xrightarrow{1} x_0$. This corresponds to the constraints $x_3 \leq -7$, $x_1 - x_3 \leq 5$, and $-x_1 \leq 1$. Summing these constraints results in the constraint $0 \leq -1$. This constraint is clearly unsatisfiable, thus the DCS is infeasible.

2.2.2 UTVPI constraint systems

The second type of polyhedral constraint systems examined are UTVPI constraint systems.

***Definition* 2.2.7.** *A constraint of the form $a_i \cdot x_i + a_j \cdot x_j \leq b_{ij}$ where $a_i, a_j \in \{-1, 0, 1\}$, is called a* **unit two variable per inequality (UTVPI) constraint**.

Note that difference constraints and absolute constraints are also UTVPI constraints.

Example 12: The following are UTVPI constraints:

- $x_1 - x_2 \leq 3$.
- $x_2 + x_4 \leq 5$.
- $-x_3 - x_4 \leq 2$.

***Definition* 2.2.8.** *A conjunction of UTVPI constraints is called a* **UTVPI Constraint System (UCS)**.

A UCS can have a linear solution but no integer solutions.

Example 13: Consider the UCS: $x_1 - x_2 \leq 0$, $x_1 + x_2 \leq 1$, $-x_1 - x_2 \leq -1$, $x_2 - x_1 \leq 0$. The only solution to this system is $x_1 = \frac{1}{2}, x_2 = \frac{1}{2}$.

Just like DCSs, UCSs also have a graphical representation. However, this representation is not a simple directed graph. Thus, the resultant structure is referred to as a constraint network.

Given a UCS $\mathbf{U}$, we construct the corresponding constraint network $\mathbf{G} = \langle V, E, \mathbf{c} \rangle$ as follows:

1. For each variable x_i, we add the vertex x_i to V.
2. For each constraint in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$, we create an edge, as per the following rules:
   (a) A constraint of the form $x_i - x_j \leq c_{ij}$, is represented by an undirected "gray" edge $(x_j \overset{c_{ij}}{\blacksquare} x_i)$. This edge can also be referred to as $(x_i \overset{c_{ij}}{\blacksquare} x_j)$.

(b) A constraint of the form $-x_i - x_j \leq c_{ij}$, is represented by an undirected "black" edge $(x_i \overset{c_{ij}}{\blacksquare} x_j)$.

(c) A constraint of the form $x_i + x_j \leq c_{ij}$, is represented by an undirected "white" edge $(x_i \overset{c_{ij}}{\square} x_j)$.

***Definition* 2.2.9.** *A* k-**path** *in* **G** *is a sequence of* $(k+1)$ *vertices,* $x_1, x_2, \ldots x_{k+1}$*, and* k *edges* $e_1, e_2, \ldots e_k$*, such that* e_i *is the edge corresponding to one of the constraints between* x_i *and* x_{i+1} *in the UCS* **U**.

***Definition* 2.2.10.** *A k-path is considered* **valid** *if it has the following property:*
For $i = 2, 3, \ldots k$*, the coefficients of* x_i *in the constraints corresponding to the edges* e_i *and* e_{i-1} *have opposite signs.*

Example 14: The path defined by the sequence of vertices x_1, x_2, x_3, x_4 and the sequence of edges $(x_1 \overset{c_{1,2}}{\blacksquare} x_2)$, $(x_2 \overset{c_{2,3}}{\blacksquare} x_3)$, $(x_3 \overset{c_{3,4}}{\square} x_4)$ is $(x_1 \overset{c_{1,2}}{\blacksquare} x_2 \overset{c_{2,3}}{\blacksquare} x_3 \overset{c_{3,4}}{\square} x_4)$. However this path is not valid because the coefficients of x_2 in the constraints corresponding to the edges $(x_1 \overset{c_{1,2}}{\blacksquare} x_2)$ and $(x_2 \overset{c_{2,3}}{\blacksquare} x_3)$ have the same sign; indeed, both of these constraints are of the form $-x_i - x_j \leq c_{ij}$.

***Definition* 2.2.11.** *The* weight *of a path is the sum of the weights of the edges along that path.*

Example 15: Consider the path $(x_1 \overset{3}{\blacksquare} x_2 \overset{1}{\blacksquare} x_3 \overset{4}{\square} x_4)$. The weight of this path is 8.

***Definition* 2.2.12.** *A* closed walk *is a valid k-path for which* $x_1 = x_{k+1}$.

In this dissertation, we refer to closed walks as cycles. Note that a cycle, as defined above can consist of edges and vertices that occur more than once. Thus, the notion of a cycle in this paper differs from the notion of a cycle in a constraint network corresponding to a difference constraint system.

The preceding concepts are illustrated in Example 2.2.2.

$$
\begin{aligned}
x_1 - x_2 &\leq -3 \\
-x_1 + x_4 &\leq 1 \\
-x_1 - x_4 &\leq 1 \\
x_1 - x_5 &\leq 1 \\
-x_1 + x_5 &\leq 0 \\
x_2 + x_3 &\leq 1 \\
x_2 - x_3 &\leq 1
\end{aligned}
$$

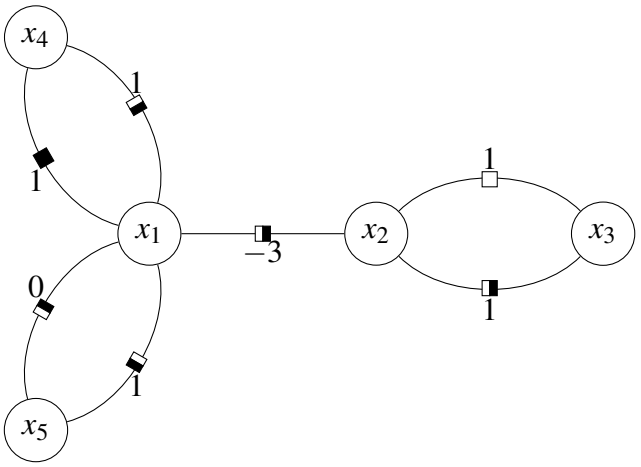

Figure 2.2: Example UCS with corresponding constraint network (without vertex x_0)

Example 16: Consider the UCS in Figure 2.2 and the corresponding constraint network.

Then, we have the following:

1. The path defined by the sequence of vertices x_4, x_1, x_5 and the sequence of edges $(x_4 \overset{1}{\blacksquare} x_1)$, $(x_1 \overset{0}{\blacksquare\!\square} x_5)$ is $(x_4 \overset{-1}{\blacksquare} x_1 \overset{0}{\blacksquare\!\square} x_5)$. However, this path is not valid because the coefficients of x_1 in the constraints corresponding to the edges $(x_4 \overset{-1}{\blacksquare} x_1)$ and $(x_1 \overset{-1}{\blacksquare\!\square} x_5)$ have the same sign.

2. The 3-path, $(x_1 \overset{-3}{\blacksquare\!\square} x_2 \overset{1}{\square} x_3 \overset{1}{\blacksquare\!\square} x_2)$, has weight -1.

3. The 8-path

$$(x_1 \overset{-3}{\blacksquare\!\square} x_2 \overset{1}{\square} x_3 \overset{1}{\blacksquare\!\square} x_2 \overset{-3}{\blacksquare\!\square} x_1 \overset{0}{\blacksquare\!\square} x_5 \overset{1}{\blacksquare\!\square} x_1 \overset{1}{\blacksquare\!\square} x_4 \overset{1}{\blacksquare} x_1)$$

   forms a cycle even though the vertices x_1 and x_2 and the edge $(x_2 \overset{-3}{\blacksquare\!\square} x_1)$ are used multiple times.

At this juncture, it is important to point out that all three types of edges, viz., "white", "black" and "gray" are directionless, i.e., it may be necessary to traverse them in either direction.

The construction of the constraint network **G** is completed as follows:

1. We create a new vertex x_0 and add it to V. The variable corresponding to this vertex will be assigned 0 in our feasibility algorithm.
2. We add the edges $(x_0 \overset{(2\cdot n+1)\cdot C}{\square} x_i)$, $(x_0 \overset{(2\cdot n+1)\cdot C}{\blacksquare} x_i)$, $(x_0 \overset{(2\cdot n+1)\cdot C}{\blacksquare\!\square} x_i)$, and $(x_i \overset{(2\cdot n+1)\cdot C}{\blacksquare\!\square} x_0)$ to E, where C is the largest absolute weight of any edge in the network.

The addition of vertex x_0 also permits the addition of absolute constraints:

1. A constraint of the form $x_i \leq c_i$ is replaced by the pair of constraints $x_i + x_0 \leq c_i$ and $x_i - x_0 \leq c_i$. The edges corresponding to the new constraints are added to E.
2. A constraint of the form: $-x_i \leq c_i$ is replaced by the pair of following constraints: $-x_i - x_0 \leq c_i$ and $x_0 - x_i \leq c_i$. The edges corresponding to the new constraints are added to E.

Example 17: Consider the UCS in Figure 2.3 and the corresponding constraint network.

$$\begin{aligned} x_1 + x_3 &\leq 0 \\ x_2 - x_3 &\leq -7 \\ x_4 - x_2 &\leq 1 \\ -x_1 - x_4 &\leq 5 \\ x_1 &\leq 6 \quad (2.2) \end{aligned}$$

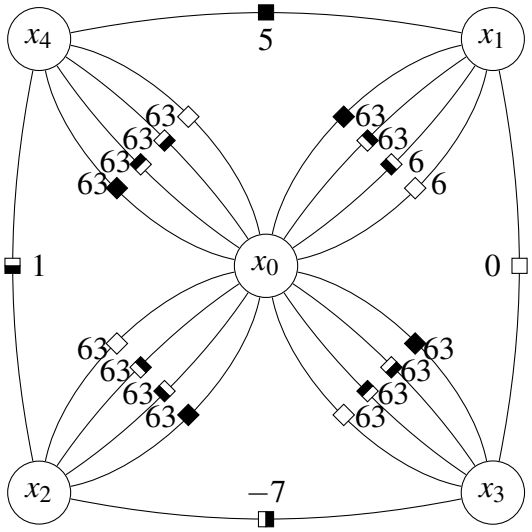

Figure 2.3: Example UCS with corresponding constraint network.

The weight of 63 on some of the edges from x_0 to the other vertices is obtained as $(2 \cdot 4+1) \cdot |-7|$. Also note that the edges $(x_0 \square x_1)$ and $(x_0 \blacksquare\!\square x_1)$ have weight 6, corresponding to the constraint $x_1 \leq 6$.

We also utilize the edge reductions from [SW17b].

***Definition* 2.2.13.** *An edge reduction is an operation which determines a single edge equivalent to a two-edge path and represents the addition of the two UTVPI constraints which correspond to the edges in question. If this addition results in a UTVPI constraint, the reduction is said to be* valid.

Table 2.1 lists all the valid edge reductions:

Constraints		Path	Reduction	Result
$x_j - x_i \leq b_{ji}$,	$x_k - x_j \leq b_{kj}$	$(x_i \overset{b_{ji}}{\blacksquarelb} x_j \overset{b_{kj}}{\blacksquarelb} x_k)$	$(x_i \overset{b_{ji}+b_{kj}}{\blacksquarelb} x_k)$	$x_k - x_i \leq b_{ji} + b_{kj}$
$x_j - x_i \leq b_{ji}$,	$-x_k - x_j \leq b_{kj}$	$(x_i \overset{b_{ji}}{\blacksquarelb} x_j \overset{b_{kj}}{\blacksquare} x_k)$	$(x_i \overset{b_{ji}+b_{kj}}{\blacksquare} x_k)$	$-x_k - x_i \leq b_{ji} + b_{kj}$
$x_j + x_i \leq b_{ji}$,	$x_k - x_j \leq b_{kj}$	$(x_i \overset{b_{ji}}{\square} x_j \overset{b_{kj}}{\blacksquarelb} x_k)$	$(x_i \overset{b_{ji}+b_{kj}}{\square} x_k)$	$x_k + x_i \leq b_{ji} + b_{kj}$
$-x_j - x_i \leq b_{ji}$,	$x_k + x_j \leq b_{kj}$	$(x_i \overset{b_{ji}}{\blacksquare} x_j \overset{b_{kj}}{\square} x_k)$	$(x_i \overset{b_{ji}+b_{kj}}{\blacksquarelb} x_k)$	$x_k - x_i \leq b_{ji} + b_{kj}$
$x_i - x_j \leq b_{ij}$,	$x_j - x_k \leq b_{jk}$	$(x_i \overset{b_{ij}}{\blacksquarerb} x_j \overset{b_{jk}}{\blacksquarerb} x_k)$	$(x_i \overset{b_{ij}+b_{jk}}{\blacksquarerb} x_k)$	$x_i - x_k \leq b_{ij} + b_{jk}$
$-x_i - x_j \leq b_{ij}$,	$x_j - x_k \leq b_{jk}$	$(x_i \overset{b_{ij}}{\blacksquare} x_j \overset{b_{jk}}{\blacksquarerb} x_k)$	$(x_i \overset{b_{ij}+b_{jk}}{\blacksquare} x_k)$	$-x_i - x_k \leq b_{ij} + b_{jk}$
$x_i - x_j \leq b_{ij}$,	$x_j + x_k \leq b_{jk}$	$(x_i \overset{b_{ij}}{\blacksquarerb} x_j \overset{b_{jk}}{\square} x_k)$	$(x_i \overset{b_{ij}+b_{jk}}{\square} x_k)$	$x_i + x_k \leq b_{ij} + b_{jk}$
$x_i + x_j \leq b_{ij}$,	$-x_j - x_k \leq b_{jk}$	$(x_i \overset{b_{ij}}{\square} x_j \overset{b_{jk}}{\blacksquare} x_k)$	$(x_i \overset{b_{ij}+b_{jk}}{\blacksquarerb} x_k)$	$x_i - x_k \leq b_{ij} + b_{jk}$

Table 2.1: Valid Edge Reductions

We use definition 2.2.13 to define paths in the constraint network.

***Definition* 2.2.14.** *We say that a path has type t, if it can be reduced to a single edge of type t, where* $t \in \{ \square, \blacksquare, \blacksquarerb, \blacksquarelb \}$ *by a series of valid edge reductions.*

We now define the concept of a shortest path of each type.

***Definition* 2.2.15.** *A shortest path of type t, for* $t \in \{ \square, \blacksquare, \blacksquarerb, \blacksquarelb \}$*, between* x_i *and* x_j *is a path of type t between* x_i *and* x_j *with minimum weight.*

In particular we are interested in paths from a vertex to itself that can be reduced to a single gray edge of negative weight. Such paths are referred to as negative weight gray cycles.

From [SW17b], we have the following result relating negative weight gray cycles in a constraint network and the feasibility of the corresponding UCS.

***Theorem* 2.2.1.** *A UCS* **U** *is linearly infeasible if and only if the corresponding constraint network has a negative weight gray cycle.*

Example 18: Consider the UCS and corresponding constraint network from Figure 2.3. The constraint network contains the negative weight gray cycle $(x_1 \overset{0}{\square} x_3 \overset{-7}{\blacksquare} x_2 \overset{1}{\blacksquare} x_4 \overset{5}{\blacksquare} x_1)$.

This cycle corresponds to the constraints $x_1 + x_3 \leq 0$, $x_2 - x_3 \leq -7$, $-x_2 + x_4 \leq 1$, and $-x_1 - x_4 \leq 5$. Summing these constraints results in the constraint $0 \leq -1$. This constraint is clearly unsatisfiable, thus the UCS is infeasible.

2.2.3 Horn constraint systems

The third type of polyhedral constraint systems examined are Horn constraint systems. Horn constraint systems consist of Horn constraints.

Definition* 2.2.16.** *A constraint of the form* $\sum_{i=1}^{n} a_i \cdot x_i \geq b_j$ *is called a* **Horn constraint, if* $a_i \in \{1, 0, -1\}$ *for* $i = 1 \ldots n$*, and at most one* $a_i = 1$.

Example 19: The following are Horn constraints:

- $x_1 - x_2 - x_3 - x_4 \geq 3$.
- $-x_1 + x_2 - x_5 \geq 5$.

***Definition* 2.2.17.** *A conjunction of Horn constraints is called a* **Horn Constraint System (HCS)**.

2.2.4 Quantified linear constraint systems

We now extend linear systems to allow for quantified variables. We specifically look at Quantified Linear Programs (QLPs) and Quantified Linear Implications (QLIs)

Quantified Linear Programming extends Linear Programming by allowing the variables to be universally or existentially quantified over an interval. More specifically, we are

interested in deciding the following query:

$$\mathbf{G} : \exists x_1 \in [a_1, b_1]\ \forall y_1 \in [l_1, u_1]\ \ \ldots \exists x_n \in [a_n, b_n]\ \forall y_n \in [l_n, u_n]$$
$$\mathbf{A} \cdot [\mathbf{x}\ \ \mathbf{y}]^{\mathbf{T}} \leq \mathbf{b},\ \ \mathbf{x} \geq \mathbf{0} \tag{2.3}$$

where

- $\mathbf{A}$ is an $m \times 2 \cdot n$ matrix called the constraint matrix,
- $\mathbf{x}$ is a $n-$vector, representing the control variables (these are existentially quantified),
- $\mathbf{y}$ is a $n-$vector, representing the variables that can assume values within a pre-specified range; i.e., component y_i has a lower bound of l_i and an upper bound of u_i (these are universally quantified),
- $\mathbf{b}$ is an $m-$vector,
- $\{a_i, b_i\},\ i = 1, 2, \ldots, n$ are rational numbers bounding variable x_i and $\{l_i, u_i\}$ are rational numbers bounding y_i.

The pair $(\mathbf{A}, \mathbf{b})$ is called the *Constraint System*. Without loss of generality, we assume that the quantifiers are strictly alternating, since we can always add dummy variables (and constraints, if necessary) without affecting the correctness or complexity of the problem [Pap94]. Let us say that an existentially quantified dummy variable x_p is added to the quantifier string. We can add the constraint $x_p = 0$ to the constraint system. Note that the value of x_p is fixed and cannot depend on the values of other y_i variables; further the variable x_p is not part of any constraint involving the original variables of the system.

The string $\exists x_1 \in [a_1, b_1]\ \forall y_1 \in [l_1, u_1]\ \exists x_2 \in [a_2, b_2]\ \forall y_2 \in [l_2, u_2]\ \ldots \exists x_n \in [a_n, b_n]\ \forall y_n \in [l_n, u_n]$ is called the quantifier string of the given QLP and is denoted by $\mathbf{Q}(\mathbf{x}, \mathbf{y})$. The length of the quantifier string, is denoted by $|\mathbf{Q}(\mathbf{x}, \mathbf{y})|$ and it is equal to the dimension of $\mathbf{A}$. Note that the range constraints on the existentially quantified variables can be included in the

constraint matrix $\mathbf{A}$ ($x_i \in [a_i, b_i]$ can be written as $a_i \leq x_i,\ x_i \leq b_i$) and thus the generic QLP can be represented as:

$$\begin{aligned} \mathbf{G}: \exists x_1\ \forall y_1 \in [l_1, u_1]\ \exists x_2\ \forall y_2 \in [l_2, u_2]\ \ldots \exists x_n\ \forall y_n \in [l_n, u_n] \\ \mathbf{A} \cdot [\mathbf{x}\ \mathbf{y}]^{\mathbf{T}} \leq \mathbf{b} \end{aligned} \tag{2.4}$$

However, the range constraints on the y_i variables cannot be moved into the constraint system.

It follows that the QLP problem can be thought of as checking whether a polyhedron described by a system of linear inequalities ($\mathbf{A} \cdot [\mathbf{x}\ \mathbf{y}]^{\mathbf{T}} \leq \mathbf{b}$) is non-empty vis-a-vis the specified quantifier string (say $\mathbf{Q}(\mathbf{x}, \mathbf{y})$). The pair $< \mathbf{Q}(\mathbf{x}, \mathbf{y}), (\mathbf{A},\ \mathbf{b}) >$ is called a *Parametric Polytope*. In other words, Quantified Linear Programming is concerned with checking the non-emptiness of Parametric Polytopes, just as traditional linear programming is concerned with checking the non-emptiness of simple polytopes. For the rest of this paper, we shall assume that the generic QLP has the form described by System (2.4), so that the analysis is simplified. Accordingly, we observe that in a QLP, the dimension of the constraint matrix $\mathbf{A}$ and hence the length of the quantifier string is always even.

An example of a *quantified linear program* (QLP) is the following:

$$\begin{aligned} \exists x_1 \in [0,1]\ \forall y_1 \in [1,4]\ \exists x_2 \in [3,9]\ x_1 + y_1 + x_2 \geq 4 \\ 3x_1 - 5y_1 + 7x_2 \leq -5. \end{aligned}$$

As with linear programs, the conjunction of inequalities is frequently written in matrix-vector form. Note that the feasibility version of a linear program is a QLP with all quantifiers existential.

***Definition* 2.2.18.** *A QLP is said to be* feasible *if it is true as a first-order sentence over the real numbers, using the standard semantics for bounded quantifiers.*

When we are interested in computational complexity, we restrict QLPs to be defined over the rational numbers.

There is an equivalent formula game definition of truth for QLPs. Player I (the $\exists$-player) chooses the values for the existentially quantified variables and Player II (the $\forall$-player) chooses the values for the universally quantified variables. The players must choose values consistent with the bounds on the variables. The $\exists$-player wins precisely when the values chosen constitute a feasible point for the matrix of the QLP. See [Sub07] for details.

In fact, we can place a further restriction on the values chosen be the $\forall$-player.

***Lemma* 2.2.1.** *[Sub07] Given a QLP ϕ, define the* modified formula game *of ϕ to be the standard formula game with the added restriction that the $\forall$-player is restricted to choosing from the endpoints of the intervals over which the universally quantified variables can range. For example, if the universally quantified variable e_i is quantified over the interval $[2,3]$, then the $\forall$-player may only choose 2 or 3 for the value of e_i. Then the $\forall$-player has a winning strategy for the modified formula game if and only if the $\forall$-player has a winning strategy for the standard formula game.*

Oftentimes we are interested in QLPs with a particular quantifier string.

***Definition* 2.2.19.** *Given a string of quantifiers $\mathscr{S}$, a QLP whose quantifiers, when grouped into blocks, are consistent with $\mathscr{S}$ is known as an $\mathscr{S}$-QLP.*

For example, an ordinary linear program (with arbitrarily many variables) is an $\exists$-QLP. Note that $\forall\exists$-QLPs are known as F-QLPs in [Sub07].

***Definition* 2.2.20.** *A generalized quantified linear program (GQLP) is a QLP where universal variables are bounded not by constants but by systems of linear constraints. i.e. $\forall u_i$ such that $u_i \leq a_1v_1 + b_1u_1 + \cdots + a_{i-1}v_{i-1} + b_{i-1}u_{i-1}$ where v_1 through v_{i-1} and u_1 though u_{i-1} appear before u_i in the quantifier string. If u_i is bounded above by multiple constraints it is bounded above by their minimum, similarly if it is bounded bellow by multiple constraints then it is bounded by their maximum. For example $\forall v'_1$ such that $v'_1 \leq 1$, $v'_1 \geq 0$, $v'_1 \leq 2v_1$, and $v'_1 \geq 2v_1 - 1$ can be expressed as $\forall v_1 \in [\max(0, 2v_1 - 1), \min(1, 2v_1)]$.*

An example of a *generalized quantified linear program* (GQLP) is the following:

$$\exists x_1 \in [0,1]\ \forall y_1 \in [x_1 - 1, 3x_1]\ \exists x_2 \in [3,9]\ x_1 + y_1 + x_2 \geq 4$$
$$3x_1 - 5y_1 + 7x_2 \leq -5.$$

Note that Quantified Linear Programming is a generalization of the *feasibility problem* of linear programming; we have not mentioned an objective function to optimize. *Multi-level games* are a generalization of Linear Programming which do incorporate objective functions. Briefly, a multi-level game involves multiple players choosing values for real-valued variables. There is a common set of inequalities in these variables; each player must ensure that her choices do not violate these inequalities. Moreover, each player has her own objective function, which she seeks to optimize. It can be shown that games with $(p+1)$ players can capture the Σ^p and Π^p levels of *PH* [Jer85].

We also examine the problem of implication in quantified linear programs.

Consider now two linear systems $P_1 : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ and $P_2 : \mathbf{C} \cdot \mathbf{x} \leq \mathbf{d}$. We say that P_1 *is included* in P_2 if every solution of P_1 is also a solution of P_2. This holds if and only if the logic formula $\forall \mathbf{x}\ [\mathbf{A}\mathbf{x} \leq \mathbf{b} \rightarrow \mathbf{C} \cdot \mathbf{x} \leq \mathbf{d}]$ is true in the domain of the reals. We extend the notion of inclusion to arbitrary quantifiers by introducing Quantified Linear Implications of two linear systems:

$$\exists \mathbf{x}_1\ \forall \mathbf{y}_1\ \ldots \exists \mathbf{x}_n\ \forall \mathbf{y}_n\ \ [\mathbf{A} \cdot \mathbf{x} + \mathbf{N} \cdot \mathbf{y} \leq \mathbf{b} \rightarrow \mathbf{C} \cdot \mathbf{x} + \mathbf{M} \cdot \mathbf{y} \leq \mathbf{d}] \tag{2.5}$$

where $\mathbf{x}_1 \ldots \mathbf{x}_n$ and $\mathbf{y}_1 \ldots \mathbf{y}_n$ are partitions of $\mathbf{x}$ and $\mathbf{y}$ respectively, and where $\mathbf{x}_1$ and/or $\mathbf{y}_n$ may be empty. We say that a QLI *holds* if it is true as a first-order formula over the domain of the reals. The *decision problem for a QLI* consists of checking whether it holds or not.

Let $\mathbf{Q}(\mathbf{x}, \mathbf{y})$ denote the quantifier string, namely $\exists \mathbf{x}_1\ \forall \mathbf{y}_1\ \ldots \exists \mathbf{x}_n\ \forall \mathbf{y}_n$ in System (2.5). We introduce a nomenclature to represent the different classes of QLIs that we will be examining. Consider a triple $\langle A, Q, R \rangle$. Let A denote the number of quantifier alternations

in the quantifier string $\mathbf{Q}(\mathbf{x},\mathbf{y})$ and Q the first quantifier of $\mathbf{Q}(\mathbf{x},\mathbf{y})$. Also, let R be an $(A+1)$-character string, specifying for each quantified set of variables in $\mathbf{Q}(\mathbf{x},\mathbf{y})$ whether they appear on the **L**eft, on the **R**ight, or on **B**oth sides of the implication. For instance, $\langle 1,\exists,\mathbf{LB}\rangle$ indicates a problem described by:

$$\exists\mathbf{x}\,\forall\mathbf{y}\;\;[\mathbf{A}\cdot\mathbf{x}+\mathbf{N}\cdot\mathbf{y}\leq\mathbf{b}\rightarrow\mathbf{M}\cdot\mathbf{y}\leq\mathbf{d}]$$

Example 20: Consider the following QLI of the class $\langle 2,\forall,\mathbf{LRB}\rangle$:

$$\forall s_1\;\exists r_1\;\forall x_1\;\forall x_2 \qquad \left.\begin{array}{l} x_1 \geq 0-3r_1 \\ x_1 \leq 2-5r_1 \\ x_2 \geq 0+2r_1 \\ x_2 \leq 1+3r_1 \end{array}\right\} \rightarrow \left\{\begin{array}{l} x_1+x_2 \geq -1+3s_1 \\ x_1+x_2 \leq 1+4s_1 \\ x_1-x_2 \geq -1+5s_1 \\ x_1-3x_2 \leq 3+7s_1 \end{array}\right.$$

Let P_1 denote the left-hand side and P_2 the right-hand side linear system of the implication. Figure 2.4 presents P_1 and P_2 for specific values of s_1 and r_1, i.e., $s_1 = r_1 = 0$. Note that for these values P_1 is included in P_2 (i.e., if both s_1 and r_1 were existentially quantified). However, in order for the above QLI to hold, for all values of s_1 there must exist a value of r_1 such that every solution x_1, x_2 of P_1 is also a solution of P_2.

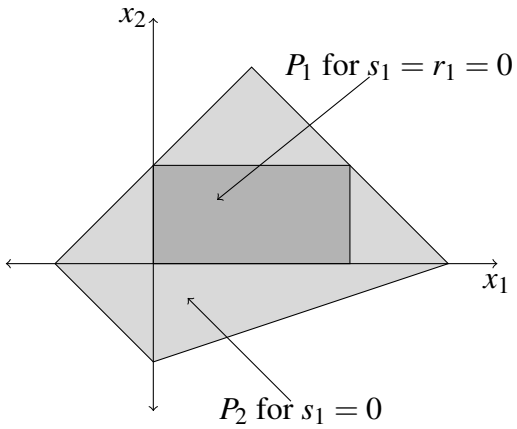

Figure 2.4: P_1 is included in P_2 for $s_1 = r_1 = 0$ (Example 2.2.4).

Chapter 3

Refutations

In this chapter, we discuss refutation systems and focus on the specific refutation systems examined in this dissertation.

3.1 Refutations in Boolean Formulas

First we examine refutations for Boolean formulas.

We focus solely on resolution refutations. A resolution refutation consists of a series of resolution steps terminating in the empty clause $\sqcup$.

***Definition* 3.1.1.** *A* **resolution step** *derives a resolvent clause from two parent clauses. A resolution step with parent clauses* $(\alpha \vee x)$ *and* $(\neg x \vee \beta)$ *with resolvent* $(\alpha \vee \beta)$*, is denoted as*

$$(\alpha \vee x), (\neg x \vee \beta) \mid\!\!\frac{1}{Res} (\alpha \vee \beta).$$

The variable x is called the matching or resolution variable.

Example 21: Consider the Boolean clauses $(x_1 \vee \neg x_2 \vee x_3)$ and $(\neg x_1 \vee x_3 \vee \neg x_4)$. Ap-

plying a resolution step with these clauses as parents results in:

$$(x_1 \vee \neg x_2 \vee x_3), (\neg x_1 \vee x_3 \vee \neg x_4) \mid\!\!\frac{1}{Res} (\neg x_2 \vee x_3 \vee \neg x_4).$$

Note that despite x_3 appearing in both parent clauses, it occurs only once in the resolvent.

***Definition* 3.1.2.** *A* **resolution refutation** *of a Boolean formula Φ is a sequence of resolution steps such that*

1. *Each parent clause is either in Φ or is the resolvent of a previous resolution step.*
2. *The final resolvent is the empty clause, $\sqcup$.*

A Boolean formula Φ is infeasible if and only if it has a resolution refutation. Such a refutation is denoted as $\Phi \mid\!\!\frac{}{Res} \sqcup$.

3.2 Refutations in Linear Programs

Next we examine refutations in linear programs.

In linear programs, we use the following rule, which plays the role that resolution does in Boolean formulas:

$$\textbf{ADD}: \quad \frac{\sum_{i=1}^{n} a_i \cdot x_i \leq b_1 \qquad \sum_{i=1}^{n} a'_i \cdot x_i \leq b_2}{\sum_{i=1}^{n} (a_i + a'_i) \cdot x_i \leq b_1 + b_2} \tag{3.1}$$

We refer to Rule (3.1) as the *ADD* rule.

Example 22: Applying the ADD rule to the constraints $x_1 - x_2 + x_3 \leq 4$ and $-x_1 + x_3 - x_4 \leq -3$ results in the constraint $2 \cdot x_3 - x_2 - x_4 \leq -1$. Note that, unlike in resolution, the extra copy of x_3 is not ignored.

It is easy to see that Rule (3.1) 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 (3.1) will result in a contradiction of the form: $0 \leq b$, $b < 0$. The completeness of the ADD rule was established by Farkas [Far02], in a lemma that is famously known as Farkas' Lemma for systems of linear inequalities [Sch87].

Farkas' lemma along with the fact that linear programs must have basic feasible solutions establishes that the linear programming problem is in the complexity class **NP** $\cap$ **coNP**. Farkas' lemma is one of several lemmata that consider pairs of linear systems in which exactly one element of the pair is feasible. These lemmas are collectively referred to as "Theorems of the Alternative" [NW99]. The **y** variables are called the Farkas' variables corresponding to the system $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ and they serve as a witness that certifies the linear infeasibility of this system. In general, the Farkas variables can assume any real value for a given constraint system.

***Definition* 3.2.1.** *A* linear refutation *is a sequence of applications of the ADD rule that results in a contradiction of the form* $0 \leq b$, $b < 0$.

In case of UTVPI constraints, Rule (3.1) can be restricted to the following rule:

$$\frac{a_i \cdot x_i + a_j \cdot x_j \leq b_{ij} \qquad\qquad -a_j \cdot x_j + a_k \cdot x_k \leq b_{jk}}{a_i \cdot x_i + a_k \cdot x_k \leq b_{ij} + b_{jk}} \tag{3.2}$$

Rule (3.2) is known as the *transitive inference rule*. Although it is a restricted version of the addition rule, it is both sound and complete for linear feasibility in UTPVI constraint systems [LM05].

3.3 Refutations in Integer Programs

Next we examine refutations in integer programs.

When studying integer feasibility, we typically use an additional rule. This is referred to as the *DIV* rule and is described as follows

$$\textbf{DIV}: \quad \frac{\sum_{i=1}^{n} a_{ij} \cdot x_i \leq b_j \qquad k \in \mathbb{Z}^+ : \frac{a_{ij}}{k} \in \mathbb{Z}, i = 1 \ldots n}{\sum_{i=1}^{n} \frac{a_{ij}}{k} \cdot x_i \leq \left\lfloor \frac{b_j}{k} \right\rfloor} \tag{3.3}$$

Rule (3.3) corresponds to dividing a constraint by a common divisor of the left-hand coefficients and then rounding the right-hand side. Since each $\frac{a_{ij}}{k}$ is an integer this inference preserves integer solutions but does necessarily preserve linear solutions. However, for systems of Horn constraints the DIV rule preserves linear feasibility, since in Horn polyhedra, linear feasibility implies integer feasibility [CS13].

Example 23: Applying the DIV rule to the constraint $3 \cdot x_1 + 6 \cdot x_2 \leq 4$ with $k = 3$ results in the constraint $x_1 + 2 \cdot x_2 \leq 1$.

***Definition* 3.3.1.** *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 case of UTVPI constraints, Rule (3.3) can be restricted to the following rule:

$$\frac{a_i \cdot x_i + a_j \cdot x_j \leq b_{ij} \qquad -a_j \cdot x_j + a_i \cdot x_i \leq b_{ji}}{a_i \cdot x_i \leq \lfloor \frac{b_{ij} + b_{ji}}{2} \rfloor} \tag{3.4}$$

Rule (3.4) is known as the *tightening inference rule*. Although it is a restricted version of the division rule, it is both sound and complete for integer feasibility in UTPVI constraint systems [LM05].

3.4 Types of Refutations

In this section we cover the types of refutations examined in this dissertation. These refutation types restrict how many times a clause or constraint can be used by a resolution step or inference rule. Note that these types of refutation will be defined in terms of clauses and resolution, however they also apply to constraints and inference rules.

3.4.1 Literal-once refutation

The first, and most restrictive, type of refutation studied is literal-once refutation.

Note that, in polyhedral CSPs, a literal consists of a variable and the sign of its coefficient. Thus, $+x_i$ and $-x_i$ are distinct literals.

***Definition* 3.4.1.** *A refutation is said to be literal-once, if each literal is used at most once in the derivation of a contradiction.*

Example 24: Consider the following UCS:

$$l_1 : \ x_1 - x_2 \ \leq \ -3 \qquad l_2 : \ x_2 - x_1 \ \leq \ 1 \tag{3.5}$$

Observe that UCS (3.5) has a refutation obtained by summing constraints l_1 and l_2. It is easy to see that this refutation is literal once.

Example 25: Consider the following UCS:

$$\begin{array}{llll} l_1 : & x_1 - x_2 \ \leq \ -4 & l_2 : & x_1 + x_2 \ \leq \ 1 \\ l_1 : & -x_1 - x_3 \ \leq \ 1 & l_2 : & x_3 - x_1 \ \leq \ 1 \end{array} \tag{3.6}$$

Observe that UCS (3.6) has a refutation obtained by summing constraints l_1, l_2, l_3, and l_4. It is easy to see that this refutation is read-once. However, the literal x_1 (and $-x_1$) is used twice in the refutation. Thus it is not a literal once refutation.

We are interested in the problem of determining if a UCS has a literal-once refutation.

We can model the LOR problem for linear polyhedral CSPs as an integer program:

Let L_i^+ be the set of constraints where the variable x_i has a positive coefficient, and let L_i^- be the set of constraints where x_i has a negative coefficient. Note that for a refutation to be literal once, at most one constraint from each set can be used. Using Farkas' Lemma, the LOR problem can be modeled as the following integer program:

$$\begin{aligned} \exists \mathbf{y}\ \mathbf{y}\cdot\mathbf{A} &= \mathbf{0} \\ \mathbf{y}\cdot\mathbf{b} &\leq -1 \\ \sum_{l_j \in L_i^+} y_j &\leq 1 \quad i = 1\ldots n \\ \sum_{l_j \in L_i^-} y_j &\leq 1 \quad i = 1\ldots n \\ \mathbf{y} &\in \{0,1\}^m \end{aligned} \tag{3.7}$$

3.4.2 Read-once refutation

The next type of refutation studied is Read-once refutation.

***Definition* 3.4.2.** *A* **Read-Once** *resolution refutation is a refutation in which each constraint, C, can be used in only one inference This applies to clauses present in the original formula and those derived as a result of previous resolution steps.*

Observe that every literal-once refutation is **guaranteed** to be read-once, but a read-once refutation need not be a literal-once refutation.

More formally, a derivation $\mathbf{C} \mid\!- C$ is read-once, if for all inferences $C_1 \wedge C_2 \mid\!\overset{1}{-} C$, we delete the constraints C_1 and C_2 from, and add the resolvent π to, the current set of Constraints. In other words, if $\mathbf{C}$ is the current set of constraints, we obtain $\mathbf{C} = (\mathbf{C} \setminus \{C_1, C_2\}) \cup \{C\}$.

Example 26: We now apply read-once resolution refutation to generate a refutation of

the 2SAT instance specified by Formula (3.8).

$$\begin{array}{ccc} (x_1, x_2) & (\neg x_1, x_3) & (\neg x_1, x_4) \\ (\neg x_2, x_3) & (\neg x_2, x_4) & (\neg x_3, x_5) \\ (\neg x_3, x_6) & (\neg x_4, \neg x_5) & (\neg x_4, \neg x_6) \end{array} \tag{3.8}$$

The application of this read-once resolution refutation to Formula (3.8) can be seen in Figure 3.1.

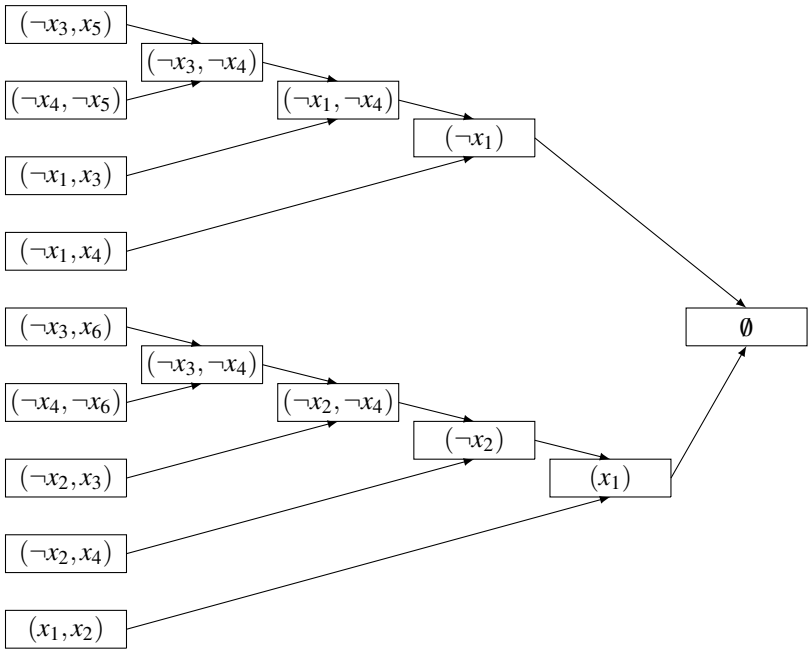

Figure 3.1: Read-once refutation of Formula (3.8)

Note that the clause $(\neg x_3, \neg x_4)$ is used twice. However, this is still a read-once refutation since each time the clause $(\neg x_3, \neg x_4)$ is derived different clauses from the original formula are used.

It is important to note that Read-Once resolution is an **incomplete** refutation procedure.

Example 27: Consider the following 2CNF formula:

$$(x_1, x_2) \quad (x_3, x_4) \quad (\neg x_1, \neg x_3)$$
$$(\neg x_1, \neg x_4) \quad (\neg x_2, \neg x_3) \quad (\neg x_2, \neg x_4)$$

We now show that this formula does not have a read-once refutation.

To derive (x_1) we need to derive $(\neg x_2)$. Similarly, to derive (x_2) we need to derive (x_1). However, the derivations of both $(\neg x_1)$ and $(\neg x_2)$ require the use of the clause (x_3, x_4).

To derive (x_3) we need to derive $(\neg x_4)$. Similarly, to derive (x_4) we need to derive $(\neg x_3)$. However, the derivations of both $(\neg x_3)$ and $(\neg x_4)$ require the use of the clause (x_1, x_2).

We now define the concept of copy complexity.

***Definition* 3.4.3.** *An unsatisfiable CSP* $\langle \mathbf{X}, \mathbf{D}, \mathbf{C} \rangle$ *has* **copy complexity** *at most k, if there exists a multi-set constraints,* $\mathbf{C}'$ *such that:*

1. *Every constraint in* $\mathbf{C}$ *appears at most k times in* $\mathbf{C}'$.
2. *Every constraint in* $\mathbf{C}'$ *appears in* $\mathbf{C}$.
3. $\langle \mathbf{X}, \mathbf{D}, \mathbf{C}' \rangle$ *has a read-once unit resolution refutation.*

3.4.3 Non-literal read-once refutation

We now define another variant of read-once refutation called non-literal read-once refutation. Here we are looking for read-once refutations which do not contain a literal-once refutation.

***Definition* 3.4.4.** *A refutation is said to be non-literal read-once, if it is a read-once refutation and does not contain a literal-once refutation.*

Example 28: Recall that UCS (3.6) has a read-once refutation obtained by summing all four constraints. Constraint l_1 is the only constraint with a negative coefficient. Thus, it

must be used in any refutation. To cancel the literal $-x_2$, we must also use the constraint l_2. However, we have now used the literal x_1 twice. Thus, the read-once refutation contains no literal once refutation.

We are interested in the problem of determining if a UCS has a non-literal read-once refutation.

3.4.4 Tree-like refutations

Next, we look at Tree-like refutations.

***Definition* 3.4.5.** *A* Tree-Like *resolution 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 and thus any derived constraint can be derived multiple times as long as it is re-derived each time it is used.

***Definition* 3.4.6.** *A derivation* $\mathbf{C} \mid\!- C$ *is tree-like, if for all inferences* $C_1, C_2 \mid\!\overset{1}{-} C$*, we have to delete the constraints* C_1, C_2 *from the current set of constraints only if they are the results of previous inferences. Afterwards we add the resolvent* C *to the current set of constraints.*

Example 29: The application of tree-like resolution refutation to Formula (3.8) can be seen in Figure 3.2. Note that the clauses $(\neg x_3, x_5)$ and $(\neg x_4, \neg x_5)$ are reused.

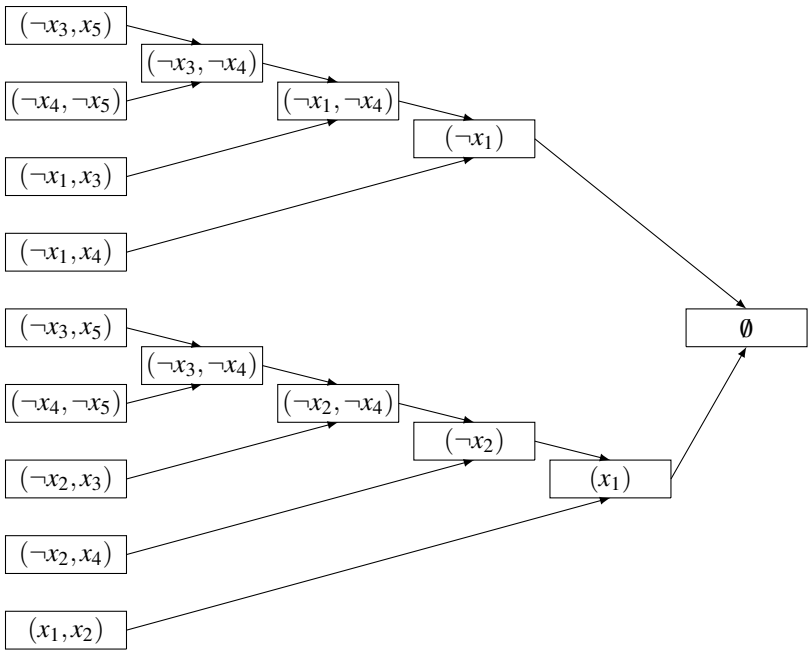

Figure 3.2: Tree-like refutation of Formula (3.8)

Tree-like refutation is a **complete** refutation procedure [BP96].

3.4.5 Dag-like refutations

Finally, we examine Dag-like refutations.

***Definition* 3.4.7.** *A* Dag-Like *resolution refutation is a refutation in which each constraint can be used multiple times.*

It follows that Dag-like refutations procedures are **complete** as well. Furthermore Dag-like refutations p-simulate tree-like refutations [CR73].

***Definition* 3.4.8.** *Let* $\mathbf{C}$ *be a set of constraints and let* C *be a constraint. A derivation* $\mathbf{C} \mid\!- C$ *is Dag-like, if for all inferences* $C_1, C_2 \mid\!\overset{1}{-} C$*, we add the resolvent* C *to the current set of constraints. In other words, if* $\mathbf{C}'$ *is the current set of clauses, we obtain* $\mathbf{C}' := \mathbf{C}' \cup \{C\}$.

Example 30: The application of dag-like resolution refutation to Formula (3.8) can be seen in Figure 3.3.

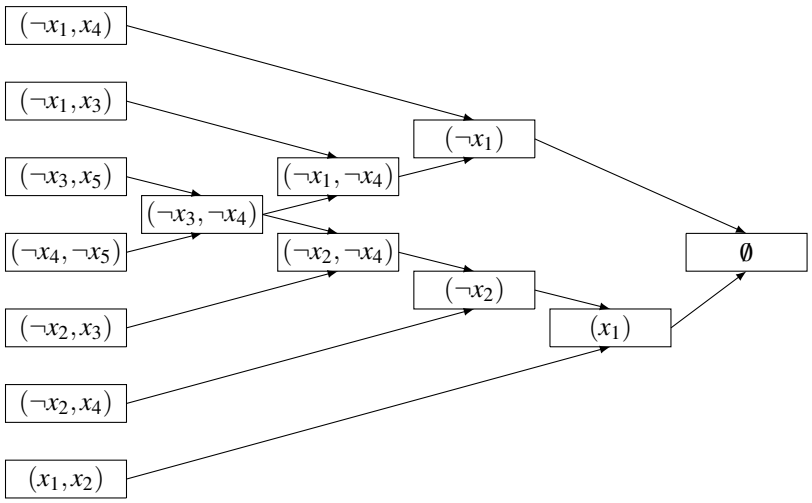

Figure 3.3: DAG-like refutation of Formula (3.8)

3.5 Theorems of the Alternative

In this section we describe another way to establish the infeasibility of constraint system. This method links the infeasibility of one constraint system to the feasibility of a second.

Typically, theorems of the alternative connect pairs of linear constraint systems and have the following form: Given two linear systems **A** and **B**, exactly one of them is feasible. System **A** is called the primal system and System **B** is called the dual system. It is not hard to see that theorems of the alternative provide certificates of infeasibility.

One famous theorem of the alternative is Farkas' Lemma [Sch87].

***Lemma* 3.5.1.** *Let* **A** *denote an* $m \times n$ *matrix and let* **c** *denote an m-vector.*

Then, either

$$\begin{aligned} &\mathbf{I}: && \exists \mathbf{x} \in \mathbb{R}^n && \mathbf{A} \cdot \mathbf{x} \leq \mathbf{c} \\ &\textbf{or} && \textit{(mutually exclusively)} \\ &\mathbf{II}: && \exists \mathbf{y} \in \mathbb{R}^m_+ && \mathbf{y^T} \cdot \mathbf{A} = \mathbf{0}, \\ & && && \mathbf{y^T} \cdot \mathbf{c} < 0. \end{aligned} \tag{3.9}$$

It is worth noting that there are several variants of Farkas' lemma in the literature. A formal proof of the above lemma along with a geometric interpretation can be found in [Sch87]. Farkas' lemma can be specialized to difference constraints through a constraint network representation, as described in [CLRS01]. Essentially, the variables become nodes and the constraints become directed edges in this setting. A consequence of Farkas' Lemma is that the difference constraint system is feasible if and only if the corresponding constraint network does not have a negative cost cycle.

Farkas' Lemma is not the only theorem of the alternative. There is Gordan's Theorem [Man69]:

***Theorem* 3.5.1.** *Let* $\mathbf{A}$ *denote an* $m \times n$ *matrix.*
Then, either

$$\begin{aligned} &\mathbf{I}: && \exists \mathbf{x} \in \mathbb{R}^n_+ && \mathbf{A} \cdot \mathbf{x} = \mathbf{0},\ \mathbf{x} \neq \mathbf{0} \\ &\textbf{or} && \textit{(mutually exclusively)} \\ &\mathbf{II}: && \exists \mathbf{y} \in \mathbb{R}^m && \mathbf{y^T} \cdot \mathbf{A} > \mathbf{0} \end{aligned}$$

There is also Stiemke's Theorem [Man69]:

***Theorem* 3.5.2.** *Let* $\mathbf{A}$ *denote an* $m \times n$ *matrix.*

Then, either

$$\begin{array}{lll} \textbf{I}: & \exists \mathbf{x} \in \mathbb{R}^n & \mathbf{A} \cdot \mathbf{x} = \mathbf{0}, \mathbf{x} > \mathbf{0} \\ \textbf{or} & \textit{(mutually exclusively)} & \\ \textbf{II}: & \exists \mathbf{y} \in \mathbb{R}^m & \mathbf{y}^{\mathbf{T}} \cdot \mathbf{A} \geq \mathbf{0},\ \mathbf{y}^{\mathbf{T}} \cdot \mathbf{A} \neq \mathbf{0} \end{array}$$

A graphical theorem of the alternative, on the other hand, relates infeasibility in a linear system to the existence of particular paths in an appropriately constructed constraint network. Such theorems of the alternative are known to exist for selected classes of linear programs. For instance, it is well-known that a system of difference constraints is infeasible if and only if the corresponding constraint network contains a negative cost cycle [Sch87].

Example 31: Consider the DCS represented by System (3.10).

$$\begin{array}{rclrclrcl} x_1 - x_2 & \leq & 3 & x_1 - x_3 & \leq & 5 & x_2 - x_3 & \leq & 0 \\ x_3 - x_1 & \leq & -1 & -x_1 & \leq & 1 & x_3 & \leq & -7 \end{array} \tag{3.10}$$

The corresponding directed graph is shown in Figure 3.4.

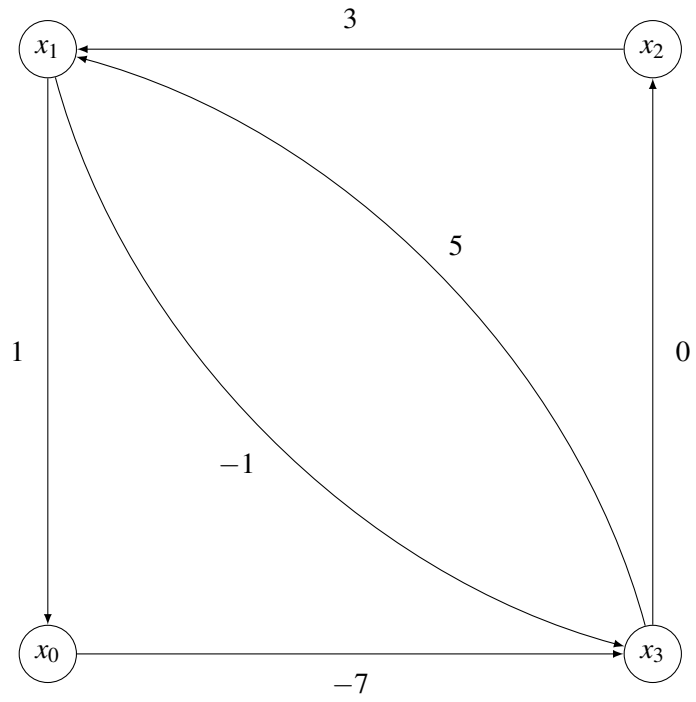

Figure 3.4: Directed graph corresponding to DCS (3.10)

The graph in Figure 3.4 has the negative cycle $x_0 \overset{-7}{\to} x_3 \overset{5}{\to} x_1 \overset{1}{\to} x_0$. This corresponds to the constraints $x_3 \leq -7$, $x_1 - x_3 \leq 5$, and $-x_1 \leq 1$. Summing these constraints results in the constraint $0 \leq -1$. This constraint is clearly unsatisfiable, thus the DCS is infeasible.

Graphical theorems of the alternative for linear feasibility UTVPI constraints were described in [LM05, SW17b]. In this dissertation, we provide a graphical theorem of the alternative for integer feasibility in UTVPI constraints.

Chapter 4

Statement of Problems

In this chapter, we define the problems examined in this dissertation. These problems are concerned with satisfiability, refutability, and closure problems for the CSPs described in Chapter 2.

4.1 Satisfiability Problems

The first type of problems we look at are satisfiability problems. Recall that a constraint system S is satisfiable if there exists an assignment to the variables in S that satisfies all domains and all constraints in S.

However, it is not enough to simply say that a given constraint system is satisfiable. We also need to provide proof that it is satisfiable. This proof is known as a certificate of feasibility. In the case of a constraint system S, the certificate of feasibility is the assignment $\mathbf{x}^*$. Thus we can define the satisfiability problem as follows:

The **Satisfiability** problem: Given a constraint system S, find an assignment $\mathbf{x}^*$ such that $\mathbf{x}^*$ satisfies S.

We can restrict the satisfiability problem by looking for assignments that have certain properties or that satisfy the constraints in a certain way. In particular we are interested in

assignments which NAE-satisfy the constraint system.

4.1.1 CSPs with side constraints

We now look at CSPs where there are restrictions on solutions that are not expressed within the set of constraints **C**. Such constraints are known as side constraints and CSPs with these additional constraints are known as CSPs with Side Constraints (CSPSCs).

In some cases side constraints can be expressed using the language of the original CSP. In such cases, the CSPSC is equivalent to a CSP where the side constraints are incorporated into the set of regular constraints.

Once such CSPSC is a constrained form of satisfiability of Boolean Constraint Systems known as Not-All-Equal satisfiability (NAE-satisfiability). In NAE-satisfiability, the side constraint is that no clause can have all of its literals assigned to the same value. This means that each clause ϕ has at least one literal set to **true** and at least one literal set to **false**.

This is equivalent to requiring that the negation of at least one literal in ϕ is **true**. This requirement can be incorporated by adding a new clause ϕ' consisting of the negations of each of the literals in ϕ. Doing this for every clause in a CNF formula Φ generates a new formula that is satisfiable if and only if the original formula is NAE-satisfiable.

There are also forms of CSPSCs where the side constraints cannot be expressed using the language of the original CSP. Consider the case of a linear program L in which no two variables can be assigned the same value. This can be accomplished by adding the side constraint $x_i \neq x_j$ for each pair of variables in L. However theses side constraints cannot be expressed using the language of linear programming. Thus, there is not necessarily an LP L' equivalent to the CSPSC L.

In terms of CSPSCs, this dissertation focuses on the NAE-satisfiability problem which we now formally define.

***Definition* 4.1.1.** *Given a CSP S, an assignment* $\mathbf{x}^*$ **NAE-satisfies** *S if*

1. $\mathbf{x}^*$ *satisfies S.*

2. *For every constraint C_j, there exist literals l_i and l_k in C_j such that, under assignment* $\mathbf{x}^*$, $l_i \neq l_k$.

Example 32: Consider the following Boolean formula $(x_1 \vee x_2) \wedge (\neg x_2 \vee x_3)$. The assignment $x_1, x_2, x_3 =$ **true** satisfies the formula. However, this assignment does not NAE-satisfy the formula since both clauses have all of their literals assigned **true**.

The assignment $x_1 =$ **true**, $x_2 =$ **false**, $x_3 =$ **true** does NAE-satisfy the formula since each clause has one literal set to **true** and one literal set to **false**.

Just like for regular satisfiability, the assignment to the variables serves as a certificate of NAE-feasibility.Thus we can define the NAE-satisfiability problem as follows:

The **NAE-satisfiability** problem: Given a constraint system S, find an assignment $\mathbf{x}^*$ such that $\mathbf{x}^*$ NAE-satisfies S.

4.2 Refutability Problems

The second type of problems we look at are refutability problems.

As with the satisfiability problem, it is insufficient to simply state that a constraint system S has a refutation. We also need to provide a refutation that proves that the constraint system is infeasible. This refutation can be read-once tree-like or dag-like. This leads to the following problems:

1. The **Read-once Refutation (ROR)** problem: Given a CSP S, find a read-once refutation for S.

2. The **Tree-like Refutation (TLR)** problem: Given a CSP S, find a tree-like refutation for S.

3. The **Dag-like Refutation (DLR)** problem: Given a CSP S, find a dag-like refutation for S.

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

***Definition* 4.2.1.** *The* **length** *of a refutation R of a CSP is the number of inferences made in R.*

Thus, for each type of refutation, we can look for the shortest refutation of that type. This results in the following problems:

1. The **Optimal Length Read-once Refutation (OLRR)** problem: Given a CSP S, find a read-once refutation for S of shortest length.
2. The **Optimal Length Tree-like Refutation (OLTR)** problem: Given a CSP S, find a tree-like refutation for S of shortest length.
3. The **Optimal Length Dag-like Refutation (OLDR)** problem: Given a CSP S, find a dag-like refutation for S of shortest length.

We can examine each of these refutation types for each constraint class being considered. However, results already exist for many of these combinations and others remain open. In this dissertation, we examine the following instances of these problems:

1. The ROR problem for resolution refutations in 2-CNF formulas.
2. The ROR problem for unit-resolution refutations in Horn formulas.
3. The OLROR problem for resolution refutations in Horn formulas.
4. The ROR problem for refutations using the ADD rule in UCSs.
5. The OTLR problem for refutations using the ADD rule in UCSs.
6. the OLRR/OTLR/ODLR problem for refutations using the ADD rule in UCSs.

4.2.1 Refutability of CSPs with side constraints

The addition of side constraints to a CSP changes the nature of the inference rules used to prove infeasibility.

Consider the Boolean CSPSC of NAE-satisfiability. Since NAE-satisfying assignments to a CSP S are also satisfying assignments to S, any refutation which prove that S is unsatisfiable also proves that S is NAE-unsatisfiable. However, since S can be satisfiable but not NAE-satisfiable it is possible for S to be NAE-unsatisfiable but have no refutation. Thus we need to include additional rules in the refutation process to be able to prove NAE-unsatisfiability.

We are interested in the NAE-satisfiability and NAE-unsatisfiablity as it pertains to Boolean formulas. Note that if a Boolean clause ϕ is NAE-satisfiable then so is the clause constructed by negating all the literals in ϕ.

Example 33: Consider the Boolean clause $(x_1 \vee \neg x_2 \vee x_3)$. The assignment $x_1, x_2, x_3 =$ **true** NAE-satisfies this clause. This assignment also NAE-satisfies the clause $(\neg x_1 \vee x_2 \vee \neg x_3)$.

This leads to the following additional resolution step used in finding NAE-refutations:

***Definition* 4.2.2.** *A* **NAE-resolution step** *derives a resolvent clause from one parent clauses. A resolution step with parent clauses* $(L_1 \vee \ldots \vee L_t)$ *and resolvent* $(\neg L_1 \vee \ldots \vee \neg L_t)$*, is denoted as*

$$(L_1 \vee \ldots \vee L_t) \mid\!\!\overline{}_{\textit{Nae-Res}} (\neg L_1 \vee \ldots \vee \neg L_t).$$

A refutation that uses both the resolution and NAE-resolution steps is called a NAE-resolution refutation. The same refutability problems that apply to resolution refutations also apply to NAE-resolution refutations.

In this dissertation, we focus on the CSPSC of NAE-satisfiability. In particular, we examine the following problems:

1. The ROR problem for NAE-refutations in Boolean formulas.

2. The ROR problem for NAE-refutations in 2-CNF formulas.

4.3 Closure Problems

In this section we described the problem of finding the closure of a constraint system. The closure of a constraint system is defined in terms of the inference rules being used. Thus for boolean formulas we can find the closure of a Boolean formula Φ with respect to resolution. Meanwhile, for HCSs we can find the closure with respect to the ADD rule.

Before defining closure, we need to define what it means for a CSP to be closed with respect to a set of inference rules.

***Definition* 4.3.1.** *A CSP S is* **closed** *with respect to a set of inference rules I if and only if it satisfies the following condition. If a constraint C_i is derivable from the constraints in S by any of the inference rules in I, then C_i is a constraint in S.*

Thus, a CSP S is closed with respect to a set of inference rule if no additional constraints can be derived from S using those inference rules.

We can now define the closure of a CSP.

***Definition* 4.3.2.** *The* **closure** *of a CSP S with respect to a set of inference rules I is the CSP S^* with the fewest constraint such that*

1. *S^* is closed with respect to I.*
2. *$S^* \supseteq S$*

Alternatively, we can define the closure of a CSP as follows:

***Definition* 4.3.3.** *A CSP S^* is the* **closure** *of a CSP S with respect to a set of inference rules I if and only if it satisfies the following condition. If a constraint C_i is derivable from the constraints in S by any of the inference rules in I, then C_i is a constraint in S^*.*

In this dissertation, we are interested in the closure of UCSs with respect to the transitive and tightening inference rules. This is known as the tightened transitive closure.

Chapter 5

Proof Systems and Refutation Systems

In this chapter we discuss proof systems and refutation systems.

For a given CSP P, let U denote the set of instances of that CSP. We can divide U into:

1. The set of satisfiable instances U_s.
2. The set of unsatisfiable instances U_u.

We can now define proof systems and refutation systems for P in terms of the sets U_s and U_u.

5.1 Proof Systems

In this section, we discuss the concept of proof systems.

Proof systems are formally defined as follows in [Sab].

***Definition* 5.1.1.** *A* **proof system** *for a language L is a polynomial time algorithm V such that for all inputs x, $x \in L$ if and only if there exists a string Π such that V accepts the input (x, Π).*

In this definition, Π is referred to as the proof, and V is referred to as the verifier. The verifier runs in time polynomial in the size of both the input x and the proof Π. Thus, the

complexity of a proof system depends not on the speed of the verifier but on the size of the proof.

Note that this definition assumes that there is a proof for every input belonging to the language and that there is no proof for inputs outside the language. Thus, under this definition, a proof system must be both sound and complete.

***Definition* 5.1.2.** *A proof system for a language L is* **sound** *if for all inputs x, $x \in L$ only if there exists a string Π such that V accepts the input (x, Π).*

This means that a sound proof system will never generate certificates for infeasible instances of a problem. Note that a proof system does not need to be sound.

Let P be a problem with a **coRP** algorithm. A **coRP** algorithm for P is a randomized algorithm that:

1. Runs in polynomial time.

2. Always accepts satisfiable instances $I_s \in U_s$.

3. Accepts unsatisfiable instances $I_u \in U_u$ with a probability of at most 50%.

Since a **coRP** algorithm has a chance of accepting unsatisfiable instances, it is an unsound proof system. We now define the concept of completeness.

***Definition* 5.1.3.** *A proof system for a language L is* **complete** *if for all inputs x, $x \in L$ if there exists a string Π such that V accepts the input (x, Π).*

This means that a complete proof system will always generate certificates for feasible instances of a problem. Note that a proof system does not need to be complete.

Let P be a problem with an **RP** algorithm. An **RP** algorithm for P is a randomized algorithm that:

1. Runs in polynomial time.

2. Always accepts satisfiable instances $I_s \in U_s$ with a probability of at least 50%.

3. Never accepts unsatisfiable instances $I_u \in U_u$.

Since an **RP** algorithm has a chance of rejecting satisfiable instances, it is an incomplete proof system.

We can adapt the defintion of a proof system to specifiacally reffer to CSPs. What we refer to a a proof system in this dissertation follows the definition given in [Sab] with the added assumption that the language L is the set of satisfiable instances U_s. This gives us the following definition:

***Definition* 5.1.4.** *A* **proof system** *for a CSP P is a polynomial time algorithm V such that for all instances I, $I \in U_s$ if and only if there exists a string Π such that V accepts the input (I, Π).*

We refer to Φ as a certificate of feasibility. For CSPs, a certificate of feasibility is an assignment to the variables that satisfies all of the constraints. Thus, a proof system for a CSP simply verifies that each constraint in **C** is satisfied by the assignment.

For a CSP, every instance $I_s \in U_s$ has a satisfying assignment, and no instance $I_u \in U_u$ has a satisfying assignment. Thus, this constitutes a sound and complete proof system.

5.2 Refutation Systems

In this section, we discuss the concept of refutation systems.

For CSPs, we defined proof systems in terms of the set of satisfiable instances U_s. A refutation system is the same concept, but applied to the set of unsatisfiable instances U_u. This gives us the following definition:

***Definition* 5.2.1.** *A* **refutation system** *for a CSP P is a polynomial time algorithm V such that for all instances I, $I \in U_u$ only if there exists a string Ψ such that V accepts the input (I, Ψ).*

Note that, unlike our definition of a proof system, our definition of a refutation system does not assume completeness. In this dissertation we explicitly deal with incomplete

refutation systems such as read-once refutation. Thus, we have loosened the definition of a refutation system to account for this possibility.

For CSPs, the form of a refutation depends on the CSP and on the refutation system used.

For Boolean CSPs, this dissertation is only concerned with resolution (See Section 3.1). However other, more powerful refutation systems exist. These include Frege Proofs, Sequent Calculus, the Davis-Putnam Procedure, and Extended Frege Proofs [Sab]. Since Boolean CSPs are **NP-complete** if any of these refutation systems is guaranteed to generate polynomially sized refutations, then **NP** = **coNP**.

For Linear Polyhedral CSPs, this dissertation is focused on refutations using the ADD inference rule (See Section 3.2). Likewise for Integer Polyhedral CSPs, we focus on refutations using the ADD and DIV inference rules (See Section 3.3). However, any theorem of the alternative corresponds to a refutation system.

5.3 Soundness and Completeness

In this section we look deeper into the soundness and completeness of proof systems and refutation systems.

Let us consider a combination proof/refutation system S for a problem P. This means that given an instance I of P, S either generates a certificate of feasibility or a refutation. We can use S to examine the relationship between soundness and completeness of proof systems and refutation systems.

***Theorem* 5.3.1.** *S is sound as a proof system for P if and only if it is complete as a refutation system for P*

Proof. First assume that S is a sound proof system for the problem P. Let $I_u \in U_u$ be an infeasible instance of P. Since S is sound as a proof system, it will not generate a certificate of feasibility for I_u. Thus, S must generate a refutation for I_u. This is true for

every infeasible instance, thus S is a complete refutation system.

Now assume that S is a complete refutation system for the problem P. Let $I_u \in U_u$ be an infeasible instance of P. Since S is complete as a proof system, it will generate a refutation for I_u. Thus, S will not generate a certificate of feasibility for I_u. This is true for every infeasible instance, thus S is a sound proof system. □

***Theorem* 5.3.2.** *S is complete as a proof system for P if and only if it is sound as a refutation system for P*

Proof. First assume that S is a complete proof system for the problem P. Let $I_s \in U_s$ be a feasible instance of P. Since S is complete as a proof system, it will generate a certificate of feasibility for I_s. Thus, S will not generate a refutation for I_s. This is true for every feasible instance, thus S is a sound refutation system.

Now assume that S is a sound refutation system for the problem P. Let $I_s \in U_s$ be a feasible instance of P. Since S is sound as a proof system, it will not generate a refutation for I_s. Thus, S must generate a certificate of feasibility for I_s. This is true for every feasible instance, thus S is a complete proof system. □

Part II

Boolean Constraints

Chapter 6

2-CNF Clausal Formulas

6.1 Motivation and Related Work

In this section, we briefly enumerate the motivation for our work and some related approaches in the literature.

1. Monotone NAE-SAT is equivalent to Hyper-graph bicolorability (set splitting). [PSSW14, GJ79, RS06]

2. Monotone NAE 2-SAT is equivalent to graph bicolorability (bipartite). The shortest (weighted) ROR NAE-resolution refutation corresponds to the shortest proof that a graph is not bipartite (the shortest (weighted) cycle with an odd number of edges) [SG11].

3. Let F be a CNF formula. The dual formula $D(F)$ is constructed by replacing all disjunctions in f with conjunctions, and all conjunctions with disjunctions. The resultant formula is clearly in disjunctive normal form.

   We have that F is NAE unsatisfiable if and only if $F \models D(F)$.

   Let F^c be the CNF formula obtained by complementing each literal in F. We see that $D(F) \approx \neg F^c$.

We have that F is NAE unsatisfiable if and only if $F \wedge F^c$ is unsatisfiable. This can happen if and only if $F \models \neg F^c$ which is equivalent to saying that $F \models D(F)$.

Thus, the question of whether a CNF formula F entails its dual formula $D(F)$ is equivalent to the problem of NAE unsatisfiablity.

Resolution is a refutation procedure that was introduced in [Rob65] to establish the unsatisfiability of clausal boolean formulas. Resolution is a sound and complete procedure, although it is not efficient in general [Hak85]. Resolution is one among many proof systems (refutation systems) that have been discussed in the literature [Urq95]; indeed it is among the weaker proof systems [BP97] in that there exist propositional formulas for which short proofs exist (in powerful proof systems) but resolution proofs of unsatisfiability are exponentially long. Resolution remains an attractive option for studying the complexity of constraint classes on account of its simplicity and wide applicability; it is important to note that resolution is the backbone of a range of automated theorem provers [BP96].

Resolution refutation techniques often arise in proof complexity. Research in proof complexity is primarily concerned with the establishment of non-trivial lower bounds on the proof lengths of propositional tautologies (alternatively refutation lengths of propositional contradictions). An essential aspect of establishing a lower bound is the proof system used to establish the bound. For instance, super-polynomial bounds for tautologies have been established for weak proof systems such as resolution [Hak85]. Establishing that there exist short refutations for all contradictions in a given proof system causes the classes **NP** and **coNP** to coincide [CR73].

There are a number of different types of resolution refutation that have been discussed in the literature [RV01]. The most important types of resolution refutation are *tree-like, dag-like and read-once*. Each type of resolution is characterized by a restriction on input clause combination. One of the simplest types of resolution is Read-once Resolution (ROR). In an ROR refutation, each input clause and each derived clause may be used at most once. There are several reasons to prefer a ROR proof over a generalized resolution proof, not the least of which is that ROR proofs must *necessarily* be of length polynomial (actually, linear) in

the size of the input. It follows that ROR cannot be a complete proof system unless **NP = coNP**. That does not preclude the possibility that we could check in polynomial time whether or not a given CNF formula has a ROR refutation. Iwama [IM95] showed that even in case of 3CNF formulas, the problem of checking ROR existence (henceforth, ROR decidability) is **NP-complete**.

It is well-known that 2CNF satisfiability is decidable in polynomial time. There are several algorithms for 2CNF satisfiability, most of which convert the clausal formula into a directed graph and then exploit the connection between the existence of labeled paths in the digraph and the satisfiability of the input formula. A natural progression of this research is to establish the ROR complexity of 2CNF formulas.

6.2 Refutability

6.2.1 The ROR problem for resolution

In this section, we show that the ROR problem for 2CNF formulas is **NP-complete**. We will be reducing from the edge-disjoint cycle problem for directed graphs.

***Definition* 6.2.1.** *Given a directed graph G and two distinct vertexes s and t, the* **edge-disjoint cycle problem** *(C-DEP) consists of finding a pair of edge-disjoint paths in G, one from s to t and the other from t to s.*

The problem is **NP-complete**. For two pairs of vertexes, the edge-disjoint path problem is **NP-complete** [EIS76]. We can reduce the edge-disjoint path problem to C-DEP the same way we reduced 2-DPP to C-DPP.

***Theorem* 6.2.1.** *The ROR problem for 2CNF formulas is* **NP-complete**.

Proof. ROR is in **NP** for arbitrary formulas in CNF [IM95]. Thus, we only need to show **NP-hardness**. That will be done by a reduction from C-DEP.

From $G = (V, E)$, s, and t we construct a formula Φ in 2CNF as follows:

1. For each vertex $v_i \in V - \{s,t\}$, create the variable x_i.

2. Create the variable x_0.

3. Let $v_i, v_j \in V - \{s,t\}$.

   (a) If $(s, v_i) \in E$ add the clause $(x_0 \rightarrow x_i)$ to Φ.

   (b) If $(t, v_i) \in E$ add the clause $(\neg x_0 \rightarrow x_i)$ to Φ.

   (c) If $(v_i, s) \in E$ add the clause $(x_i \rightarrow x_0)$ to Φ.

   (d) If $(v_i, t) \in E$ add the clause $(x_i \rightarrow \neg x_0)$ to Φ.

   (e) If $(v_i, v_j) \in E$ add the clause $(x_i \rightarrow x_j)$ to Φ.

Assume that G has two edge-disjoint paths,

$$w_1 = s, v_{i_1}, \ldots, v_{i_j}, t \text{ and } w_2 = t, v_{i_{j+1}}, \ldots, v_{i_k}, s.$$

Thus, there exist 2CNF formulas Φ_1 and Φ_2 such that:

$$\begin{aligned}
\Phi_1 &= \{(x_0 \rightarrow x_{i_1}), (x_{i_1} \rightarrow x_{i_2}), \ldots, (x_{i_j} \rightarrow \neg x_0)\} \\
\Phi_2 &= \{(\neg x_0 \rightarrow x_{i_{j+1}}), (x_{i_{j+1}} \rightarrow x_{i_{j+2}}), \ldots, (x_{i_k} \rightarrow x_0)\}.
\end{aligned}$$

Obviously, $\Phi_1 \mid\!\frac{}{\text{RO-Res}} (\neg x_0)$ and $\Phi_2 \mid\!\frac{}{\text{RO-Res}} (x_0)$. Note that x_0 has not been used as a matching variable. Since w_1 and w_2 are edge-disjoint, we have that $\Phi_1 \cap \Phi_2 = \emptyset$. Thus, $\Phi_1 \cup \Phi_2 \mid\!\frac{}{\text{RO-Res}} \sqcup$. This means that $\Phi \supseteq \Phi_1 \cup \Phi_2$ is in ROR.

Now assume that Φ is in ROR.

Let $\Phi' \subseteq \Phi$ be minimally ROR. We have that Φ' contains clauses with x_0 and $\neg x_0$. Otherwise, the formula would be satisfiable by setting each x_i to **true**.

We proceed by induction on the number of clauses in Φ'.

The shortest formula is $\Phi' = (x_0 \rightarrow \neg x_0) \wedge (\neg x_0 \rightarrow x_0)$. This Φ' is generated when $(s,t), (t,s) \in E$. These edges form the desired edge-disjoint paths.

Let $(L \to K) \wedge (K \to R) \mid\!\!\frac{1}{Res}\ (L \vee R)$ be a resolution step such that $(L \to K) \in \Phi'$ and $(K \to R) \in \Phi'$. Note that $(L \to R) \notin \Phi'$. Otherwise, Φ' would not be minimally ROR.

In a read-once refutation, we remove the parent clauses from Φ and add the resolvent $(L \to R)$. This new formula has a read-once resolution refutation and can be considered as obtained by a reduced graph without the edges $L \to K, K \to R$ but with the edge $L \to R$. By the induction hypothesis, this new graph contains the desired edge-disjoint cycle. If we replace the edge $L \to R$ in this cycle with $L \to K$ and $K \to R$, then we construct the desired edge-disjoint cycle in G. □

6.2.2 The ROR problem for NAE-resolution

In this section, we show that read-once NAE-resolution refutation is both sound and complete for formulas in 2CNF. We also show that the problem of finding the shortest read-once NAE-resolution refutation is in **P**.

***Theorem* 6.2.2.** *Let ϕ be a formula in 2CNF. We have $\phi \notin$ NAE-SAT if and only if $\phi \in$ ROR-NAE, and a refutation can be found in quadratic time.*

Proof. Let ϕ be a 2CNF formula that is not in NAE-SAT. If ϕ contains a unit clause, say (x), then $\{(x), (\neg x)\} \subseteq \phi \cup \phi^c$. We have that $(x), (\neg x) \mid\!\!\frac{1}{Res}\ \sqcup$. This is clearly a ROR-NAE-SAT refutation. Thus, we assume that ϕ contains no unit clause.

From $\phi \cup \phi^c$, we create an implication graph, G, as follows:

1. For every variable x_i, we create the verticies x_i and $\neg x_i$.

2. For every clause $(L \vee K)$, we create the edges $\neg L \to K$ and $\neg K \to L$.

G contains a strongly connected component, say G_1, with a pair of complementary literals if and only if $\phi \cup \phi^c$ is unsatisfiable. Moreover, the computation of the strongly connected components and finding a complementary pair of literals can be performed in linear time. A formula ϕ is not in NAE-SAT if and only if $\phi \cup \phi^c$ is unsatisfiable. Thus, there exists a strongly connected component C in G that contains complementary literals.

Let $\neg L_0 \to L_1 \to L_2 \ldots L_m \to L_0$ be a shortest path in C between the complementary pair of literals L_0 and $\neg L_0$. For $i \neq j$ we have $L_i \neq L_j$ and $L_i \neq \neg L_j$, otherwise there would be a shorter path in C.

Thus, there is a read-once resolution derivation

$$(L_0 \vee L_1), (\neg L_1 \vee L_2), \ldots, (\neg L_m \vee L_0) \mid\!\!\overline{\;}_{\textit{RO-Res}}\; L_0.$$

Since we are dealing with $\phi \cup \phi^c$, there are clauses $(\neg L_0 \vee \neg L_1), (L_1 \vee \neg L_2), \ldots, (L_m \vee \neg L_0)$ in $\phi \cup \phi^c$. These clauses form a read-once resolution of $\neg L_0$. Moreover, the two sets of clauses have no clause in common, because the literals L_i for $i \neq 0$ are pairwise disjoint. Finally, we can resolve L_0 and $\neg L_0$. Thus, we have a read-once resolution refutation for $\phi \cup \phi^c$. This corresponds to a ROR-NAE-SAT refutation of ϕ.

Since the computation of the strongly connected components includes deciding whether a complementary pair of literals exists costs linear time and finding a complementary pair with a shortest path costs for each variable again takes linear time, to construct a read-once resolution proof requires no more than quadratic time. □

6.2.3 The OLRR problem for NAE-resolution

Earlier in Section 2.1.1, we described an implication graph for checking the satisfiability of 2CNF formulas. We can construct a similar implication graph for checking the NAE-satisfiability of 2CNF formulas. We refer to this as the NAE-implication graph. The NAE-implication graph of a formula ϕ is equivalent to the implication graph of $\phi \cup \phi^c$.

***Theorem* 6.2.3.** *A CNF formula ϕ is* **not** *NAE-satisfiable if and only if the clause $\phi \mid\!\!\overline{\;}_{\textit{Nae-Res}}\; (x_i)$ for some variable x_i.*

Proof. Assume that $\phi \mid\!\!\overline{\;}_{\textit{Nae-Res}}\; (x_i)$ for some variable x_i. We know that any assignment, $\mathbf{x}$, that NAE-satisfies ϕ must NAE-satisfy (x_i). However, the clause (x_i) has only one literal. Thus, it cannot be NAE-satisfied. This means that ϕ is not NAE-satisfiable.

Let ϕ be a CNF formula that is not NAE-satisfiable. We can construct the unsatisfiable formula $\phi' = \phi \cup \phi^c$ of CNF clauses.

Since ϕ' is unsatisfiable we can derive the clauses (x_i) and $(\neg x_i)$ for some variable x_i.

Let $(x_{j1},\ldots,x_{jm},x_k) \wedge (\neg x_k,x_{l1},\ldots,x_{lm}) \mid\!\!\frac{1}{Res}\ (x_{j1},\ldots,x_{jm},x_{l1},\ldots,x_{lm})$ be the first step in the derivation of (x_i) from the clauses in ϕ'. We have four possibilities for the original clauses in ϕ.

1. $(x_{j1},\ldots,x_{jm},x_k),(\neg x_k,x_{l1},\ldots,x_{lm}) \in \phi$:

   From the NAE-resolution rules we get:

   $$(x_{j1},\ldots,x_{jm},x_k),(\neg x_k,x_{l1},\ldots,x_{lm}) \mid\!\!\frac{1}{Res}\ (x_{j1},\ldots,x_{jm},x_{l1},\ldots,x_{lm}).$$

2. $(x_{j1},\ldots,x_{jm},x_k),(x_k,\neg x_{l1},\ldots,\neg x_{lm}) \in \phi$:

   From the NAE-resolution rules we get:

   $$(x_k,\neg x_{l1},\ldots,\neg x_{lm}) \mid\!\!\frac{}{Nae\text{-}Res}\ (\neg x_k,x_{l1},\ldots,x_{lm}).$$

   $$(x_{j1},\ldots,x_{jm},x_k) \wedge (\neg x_k,x_{l1},\ldots,x_{lm}) \mid\!\!\frac{1}{Res}\ (x_{j1},\ldots,x_{jm},x_{l1},\ldots,x_{lm}).$$

3. $(\neg x_{j1},\ldots,\neg x_{jm},\neg x_k),(\neg x_k,x_{l1},\ldots,x_{lm}) \in \phi$:

   From the NAE-resolution rules we get:

   $$(\neg x_{j1},\ldots,\neg x_{jm},\neg x_k) \mid\!\!\frac{}{Nae\text{-}Res}\ (x_{j1},\ldots,x_{jm},x_k).$$

   $$(x_{j1},\ldots,x_{jm},x_k) \wedge (\neg x_k,x_{l1},\ldots,x_{lm}) \mid\!\!\frac{1}{Res}\ (x_{j1},\ldots,x_{jm},x_{l1},\ldots,x_{lm}).$$

4. $(\neg x_{j1},\ldots,\neg x_{jm},\neg x_k),(x_k,\neg x_{l1},\ldots,\neg x_{lm}) \in \phi$:

From the NAE-resolution rules we get:

$$(x_k, \neg x_{l1}, \ldots, \neg x_{lm}) \mid\!\!\overline{\;}_{\textit{Nae-Res}} (\neg x_k, x_{l1}, \ldots, x_{lm}).$$

$$(\neg x_{j1}, \ldots, \neg x_{jm}, \neg x_k) \mid\!\!\overline{\;}_{\textit{Nae-Res}} (x_{j1}, \ldots, x_{jm}, x_k).$$

$$(x_{j1}, \ldots, x_{jm}, x_k) \wedge (\neg x_k, x_{l1}, \ldots, x_{lm}) \mid\!\!\overline{\;}^{1}_{\textit{Res}} (x_{j1}, \ldots, x_{jm}, x_{l1}, \ldots, x_{lm}).$$

In all four cases, $\phi \mid\!\!\overline{\;}_{\textit{Nae-Res}} (x_{j1}, \ldots, x_{jm}, x_{l1}, \ldots, x_{lm})$.

This same argument can be repeated for each subsequent derivation step. Thus, $\phi \mid\!\!\overline{\;}_{\textit{Nae-Res}} (x_i)$. □

Let ϕ be a CNF formula such that there is a read-once resolution refutation $\phi \cup \phi^c \mid\!\!\overline{\;}_{\textit{RO-Res}} \sqcup$. Starting with ϕ, we apply the NAE-extension rule and generate $\phi \cup \phi^c$. Next, we apply the resolution rule according to the read-once resolution refutation for $\phi \cup \phi^c$.

Now, suppose there is a derivation $\phi \mid\!\!\overline{\;}_{\textit{Nae-Res}} \sqcup$ in which the resolution operation is read-once and the extension rule is used at most once on either ϕ or ϕ^c. We rearrange the derivation such that we first apply the extension rule and then the resolution rule.

Let $(\alpha \vee x), (\neg x \vee \beta) \mid\!\!\overline{\;}^{1}_{\textit{Res}} (\alpha \vee \beta)$ and $(\alpha \vee \beta) \mid\!\!\overline{\;}_{\textit{Nae-Res}} (\alpha^c \vee \beta^c)$ be an instance where the extension rule is used on a derived clause. We can replace these derivation steps with:

$(\alpha \vee x) \mid\!\!\overline{\;}_{\textit{Nae-Res}} (\alpha^c \vee \neg x)$, $(\neg x \vee \beta) \mid\!\!\overline{\;}_{\textit{Nae-Res}} (x \vee \beta^c)$, and $(\alpha^c \vee \neg x)(x \vee \beta^c) \mid\!\!\overline{\;}^{1}_{\textit{Res}} (\alpha^c \vee \beta^c)$.

This can be done repeatedly until the NAE-extension rule is applied to only the original clauses of the formula. Since the desired refutation starts with $\phi \cup \phi^c$, we can remove the instances of the NAE-extension rule to generate a proof of $\phi \cup \phi^c \mid\!\!\overline{\;}_{\textit{RO-Res}} \sqcup$.

To prove NAE-infeasibility we need to derive the clause (x_i).Thus, we need to find a path from $\bar{x}_i$ to x_i in the NAE-implication graph. Note that we do not need to also find a path from x_i to $\bar{x}_i$. Thus, we have the following theorem.

***Theorem* 6.2.4.** *Let ϕ be a formula in 2-CNF without unit clauses. The following state-*

ments are equivalent:

1. ϕ *is not in NAE-SAT.*

2. $\phi \cup \{(\neg L_1, \neg L_2) : (L_1, L_2) \in \phi\} \mid\!\!\overline{_{Res}}\ L$ *for some literal L, and there is a derivation in which at most one of the clauses* (L_1, L_2) *or* $(\neg L_1, \neg L_2)$ *occurs.*

A decision procedure based on the representation as a graph solves the problem in linear time.

We show that, in the case of NAE-infeasible 2CNF formulas, we always have a Read-Once NAE-resolution refutation.

***Theorem* 6.2.5.** *If a 2CNF formula,* ϕ*, has a NAE-resolution derivation of* (x_i)*, then it has a NAE-resolution derivation of* (x_i) *using only one literal more than once.*

Proof. Let G be the NAE-implication graph corresponding to ϕ. We know that $\phi \mid\!\!\overline{_{Nae\text{-}Res}}\ (x_i)$ if and only if there exists a path from $\bar{x}_i$ to x_i in G. Let p denote this path. Let x_j be the first variable such that both x_j and $\bar{x}_j$ appear on p. We are guaranteed for this x_j to exist since both x_i and $\bar{x}_i$ appear on p. We can assume without loss of generality that x_j appears before $\bar{x}_j$. Thus, we can break p up as follows:

1. a path, p_1, from $\bar{x}_i$ to x_j,

2. a path, p_2, from x_j to $\bar{x}_j$,

3. and a path, p_3, from $\bar{x}_k$ to x_i.

This can be seen in Figure 6.1.

By our choice of x_j, we know that for $k \neq j$, p_1 and p_2 together do not contain both x_k and $\bar{x}_k$. As a consequence of this no two edges in p_1 or p_2 correspond to the same constraint. Thus, p_2 corresponds to a a read-once NAE-resolution derivation of $(\neg x_j)$ in which only the literal $\neg x_j$ appears twice. We also have that p_1 is a literal once NAE-resolution derivation of (x_i, x_j) which has no literals in common with the NAE-resolution

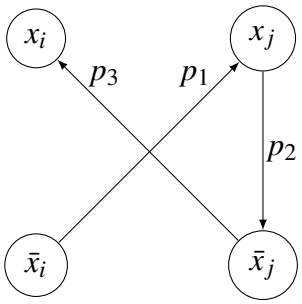

Figure 6.1: Example of path p

derivation corresponding to p_2. Combining these two yields a read-once NAE-resolution derivation of (x_i) in which only the literal $\neg x_j$ is used twice. □

Note that, in this NAE-resolution derivation the subpath p_2 from x_j to $\bar{x}_j$ is a proof of NAE-infeasibility by itself since it shows $(\neg x_j)$ which already enough to force x_j to be both **true** and **false**. Thus, we have the following corollary.

***Corollary* 6.2.1.** *If a 2CNF formula, ϕ, has a NAE-resolution derivation of (x_i), then, for some x_j, there is a NAE-resolution derivation of (x_j) (or $(\neg x_j)$) using only the literal x_j (or $\neg x_j$) more than once.*

Corollary 6.2.1 provides us with a polynomial time algorithm to find the shortest read-once NAE-refutation of a 2CNF formula.

Algorithm 6.2.1 FIND-MIN-NAE-REFUTATION

Function FIND-MIN-NAE-REFUTATION (NAE-infeasible 2CNF formula ϕ)

1: From ϕ, construct the NAE-implication graph G.
2: **for** (Each $i = 1 \ldots n$) **do**
3: Find the shortest path from $\bar{x}_i$ to x_i in G.
4: **return** (The shortest of the located paths.)

Note that, we do not need to consider the paths from x_i to $\bar{x}_i$ since the existence of such a path means that there is a path of equal length from $\bar{x}_i$ to x_i.

This algorithm can be easily modified to solve the following problem:

***Definition* 6.2.2.** *In the* minimum-weight *read-once NAE-resolution refutation problem each clause of ϕ is assigned a non-negative weight. The goal is to find a read-once NAE-resolution refutation with minimum total weight.*

To find the minimum-weight read-once NAE-resolution refutation for a 2CNF formula, we construct a weighted NAE-implication graph. In the weighted graph each edge is assigned the same weight as the corresponding 2CNF clause. We then run a modified version of Algorithm 6.2.1 on this weighted graph to find the the minimum-weight path from $\bar{x_i}$ to x_i.

We now discuss the notion of minimal NAE-read-once in CNF formulas.

***Definition* 6.2.3.** *A CNF formula ϕ is* minimal NAE-read-once*, if ϕ has a read-once NAE-resolution refutation, but no sub-formula of ϕ has a read-once NAE-resolution refutation.*

Definition 6.2.3 lets us define the following problem:

***Definition* 6.2.4.** *The* MNRR *problem is the problem of determining if a CNF formula ϕ is minimal NAE-read-once.*

The computational complexity of the MNRR problem for general CNF formulas is unknown. However, Algorithm 6.2.1 can be used to solve the MNRR problem for 2CNF formulas in polynomial time.

***Theorem* 6.2.6.** *MNRR for 2CNF formulas is in* **P**.

Proof. Let ϕ be a NAE-infeasible 2CNF formula, and let p be the path returned by running Algorithm 6.2.1 on ϕ. If ϕ is minimally NAE-read-once, then any read-once NAE-resolution refutation of ϕ must use every clause in ϕ. Thus p must use every clause in ϕ.

As described above, the path p produced by Algorithm 6.2.1 is the minimum read-once NAE-resolution refutation of ϕ. Thus, if p uses all the clauses of ϕ, then any read-once NAE-resolution refutation of ϕ must use all the clauses of ϕ. This means that ϕ is minimally NAE-read-once.

Since Algorithm 6.2.1 runs in polynomial time, the MNRR problem for 2CNF formulas is in **P**. □

6.2.4 The ROR problem for unit-resolution

In this section, we show that the Unit ROR Problem for 2-CNF is still in **P**.

Let Φ be a 2-CNF formula with m clauses over n variables. We construct a weighted undirected graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b} \rangle$ as follows:

1. For each variable x_i in Φ, add the vertices x_i^+, $x_i'^+$, x_i^-, and $x_i'^-$ to $\mathbf{V}$. Additionally, add the edges $x_i^- \overset{0}{---} x_i^+$ and $x_i'^- \overset{0}{---} x_i'^+$ to $\mathbf{E}$.

2. Add the vertices x_0^+ and x_0^- to $\mathbf{V}$.

3. For each constraint ϕ_k of Φ, add the vertices ϕ_k and ϕ_k' to $\mathbf{V}$ and the edge $\phi_k \overset{0}{---} \phi_k'$ to $\mathbf{E}$. Additionally:

   (a) If ϕ_k is $(x_i \vee x_j)$, add the edges $x_i^+ \overset{-1}{---} \phi_k$, $x_i'^+ \overset{-1}{---} \phi_k$, $x_j^+ \overset{-1}{---} \phi_k'$, and $x_j'^+ \overset{-1}{---} \phi_k'$ to $\mathbf{E}$.

   (b) If ϕ_k is $(x_i \vee \neg x_j)$, add the edges $x_i^+ \overset{-1}{---} \phi_k$, $x_i'^+ \overset{-1}{---} \phi_k$, $x_j^- \overset{-1}{---} \phi_k'$, and $x_j'^- \overset{-1}{---} \phi_k'$ to $\mathbf{E}$.

   (c) If ϕ_k is $(\neg x_i \vee x_j)$, add the edges $x_i^- \overset{-1}{---} \phi_k$, $x_i'^- \overset{-1}{---} \phi_k$, $x_j^+ \overset{-1}{---} \phi_k'$, and $x_j'^+ \overset{-1}{---} \phi_k'$ to $\mathbf{E}''$.

   (d) If ϕ_k is $(\neg x_i \vee \neg x_j)$, add the edges $x_i^- \overset{-1}{---} \phi_k$, $x_i'^- \overset{-1}{---} \phi_k$, $x_j^- \overset{-1}{---} \phi_k'$, and $x_j'^- \overset{-1}{---} \phi_k'$ to $\mathbf{E}$.

   (e) If ϕ_k is (x_i), add the edges $x_i^+ \overset{-1}{---} \phi_k$, $x_i'^+ \overset{-1}{---} \phi_k$, $x_0^+ \overset{-1}{---} \phi_k'$, and $x_0^- \overset{-1}{---} \phi_k'$ to $\mathbf{E}$.

   (f) If ϕ_k is $(\neg x_i)$, add the edges $x_i^- \overset{-1}{---} \phi_k$, $x_i'^- \overset{-1}{---} \phi_k$, $x_0^+ \overset{-1}{---} \phi_k'$, and $x_0^- \overset{-1}{---} \phi_k'$ to $\mathbf{E}$.

With this construction each variable is represented by a pair of 0-weight edges. Thus, each variable can be used twice by a refutation. However, we only have one 0-weight edge for each clause. This prevents the refutation from re-using clauses.

As per the above construction, **G** has $O(m+n)$ vertices and $O(m)$ edges. **Example 34:** Consider the following 2-CNF formula.

$$\begin{array}{lll} \phi_1: \ (x_1) & \phi_2: \ (x_2) & \phi_3: \ (\neg x_3) \\ \phi_4: \ (\neg x_4) & \phi_5: \ (\neg x_1 \vee \neg x_2) & \phi_6: \ (x_3 \vee x_4) \end{array} \tag{6.1}$$

Formula (6.1) corresponds to the undirected graph in Figure 6.2.

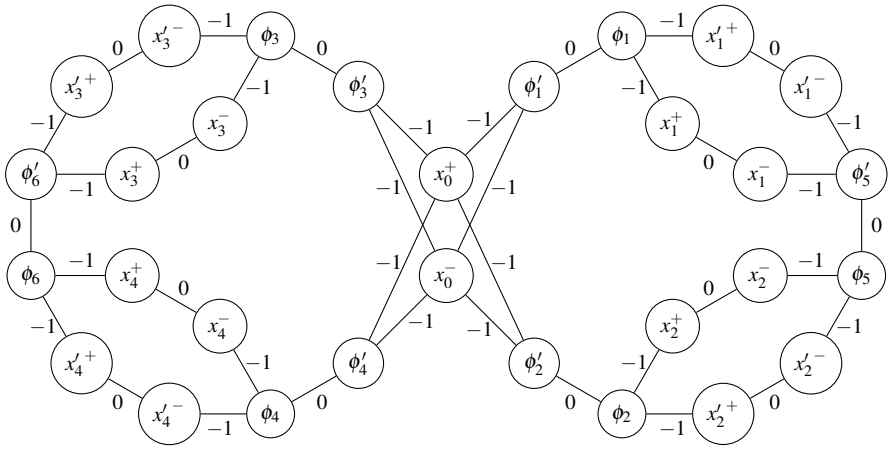

Figure 6.2: Undirected graph corresponding to Formula (6.1)

***Theorem* 6.2.7.** *Φ has a read-once unit resolution refutation if and only if* **G** *has a negative weight perfect matching.*

Proof. First assume that Φ has a read-once unit resolution refutation R. We can construct a negative weight perfect matching P of **G** as follows:

1. For each variable x_i in Φ:

   (a) If R does not use x_i, add the edges $x_i^+ \overset{0}{---} x_i^-$ and $x_i'^+ \overset{0}{---} x_i'^-$ to P.

   (b) If R uses x_i only once, add the edge $x_i'^+ \overset{0}{---} x_i'^-$ to P.

2. For each clause ϕ_k in Φ:

(a) If $\phi_k \notin R$, add the edge $\phi_k \overset{0}{---} \phi'_k$ to P.

(b) If $\phi_k \in R$ is a two variable clause:

i. If ϕ_k is the first clause to use the literal x_i, add the edge $x_i^+ \overset{-1}{---} \phi_k$ (or $x_i^+ \overset{-1}{---} \phi'_k$) to P. If it is the second, add the edge $x_i'^+ \overset{-1}{---} \phi_k$ (or $x_i'^+ \overset{-1}{---} \phi'_k$) instead.

ii. If ϕ_k is the first clause to use the literal $\neg x_i$, add the edge $x_i^- \overset{-1}{---} \phi_k$ (or $x_i^- \overset{-1}{---} \phi'_k$) to P. If it is the second, add the edge $x_i'^- \overset{-1}{---} \phi_k$ (or $x_i'^- \overset{-1}{---} \phi'_k$) instead.

(c) If $l_k \in R$ is a unit clause:

i. If l_k is the first clause to use the literal x_i, add the edge $x_i^+ \overset{-1}{---} \phi_k$ to P. If it is the second, add the edge $x_i'^+ \overset{-1}{---} \phi_k$ instead.

ii. If l_k is the first clause to use the literal $-x_i$, add the edge $x_i^- \overset{-1}{---} \phi_k$ to P. If it is the second, add the edge $x_i'^- \overset{-1}{---} \phi_k$ instead.

iii. If l_k is the first unit clause, add the edge $x_0^+ \overset{-1}{---} \phi'_k$ to P. If it is the second, add the edge $x_0^- \overset{-1}{---} \phi'_k$ instead.

Every vertex in $\mathbf{G}$ is an endpoint of exactly one edge in P. Thus, P is a perfect matching. Since $\sum_{\phi_k \in R} -1 < 0$, P has negative weight.

Now assume that $\mathbf{G}$ has a negative weight perfect matching P. We can construct a read-once unit resolution refutation R as follows:

1. Since there is no edge between x_0^+ and x_0^-, P must use the edge $x_0^+ \overset{-1}{---} \phi'_k$ for some unit clause ϕ_k. Thus, the edge $\phi_k \overset{0}{---} \phi'_k$ is not in P. Note that ϕ_k is the clause (x_i) or $(\neg x_i)$ for some x_i.

2. Without loss of generality assume ϕ_k is the clause (x_i). This means that the edge $x_i^+ \overset{-1}{---} \phi_k$ (or $x_i'^+ \overset{-1}{---} \phi_k$) must be in P. Thus, the edge $x_i^+ \overset{0}{---} x_i^-$ (or $x_i'^+ \overset{0}{---} x_i'^-$) is not in P. This means that for some clause ϕ_l, the edge $x_i^- \overset{-1}{---} \phi_l$ (or $x_i'^- \overset{-1}{---} \phi_l$) is in P. If ϕ_j corresponds to the clause $(\neg x_i \vee x_j)$ or $(\neg x_i \vee \neg x_j)$ for some x_j, then

we add either $(x_i), (\neg x_i \vee x_j) \mid\!\frac{1}{Res}\ (x_j)$ or $(x_i), (\neg x_i \vee \neg x_j) \mid\!\frac{1}{Res}\ (\neg x_j)$ to R read-once unit resolution. If ϕ_l corresponds to the clause $(\neg x_i)$, then we add $(x_i), (\neg x_i) \mid\!\frac{1}{Res}\ \sqcup$ to R.

3. If ϕ_j is a non-unit clause, then we can repeat step 2 from either x_j or $\neg x_j$. This continues until a second unit clause is encountered, completing the refutation. By construction, R is a read-once unit resolution refutation.

□

Observe that the minimum weight perfect matching of an undirected graph having n vertices can be found in $O(n^2 \cdot \log n)$ time using the algorithm in [KV10]. Since in our case, $\mathbf{G}$ has $O(m+n)$ vertices, it follows that we can detect the presence of a negative weight perfect matching in $O((m+n)^2 \cdot \log(m+n))$ time. Hence, using the above reduction, the Unit ROR problems for 2-CNF formulas can be solved in $O((m+n)^2 \cdot \log(m+n))$ time.

Chapter 7

3-CNF Clausal Formulas

7.1 Motivation and Related Work

This chapter is concerned with techniques for checking Not-All-Equal (NAE) satisfiability of propositional formulas in Conjunctive Normal Form (CNF). Briefly, the NAE-SAT problem is concerned with checking if a CNF formula has a satisfying assignment in which each clause has at least one literal set to **false**. It is well-known that the NAE-satisfiability problem for 3CNF formulas (also called NAE3SAT) is **NP-complete** [Sch78]. Indeed, the problem remains **NP-complete**, even when all the literals in each clause are positive. The problem can be solved in polynomial time, when there are at most two literals per clause [MM11, Pap94].

It is not hard to see that the class of CNF formulas which are NAE-satisfiable is a proper subset of CNF formulas which are satisfiable in the ordinary sense. Therefore, proof systems for satisfiability may not be **sound** for checking NAE-satisfiability. Indeed, this is the case with resolution refutation [Rob65], which is **complete** for NAE-satisfiability but not sound. In other words, if a resolution refutation exists for a CNF formula, then the formula is definitely NAE-unsatisfiable (since it is unsatisfiable). However, if a refutation does not exist for a formula, then it may still be NAE-unsatisfiable. In this chapter, we

design a new resolution scheme called NAE-resolution which is simultaneously **sound** and **complete** for the problem of checking NAE-satisfiability in CNF formulas.

Propositional proof complexity is concerned with lengths of proofs (alternatively refutations) in propositional logic [BP98]. In order to discuss lengths of proofs, it is vital that we have a concrete proof system in mind [Urq95]. Several proof systems have been discussed in the literature including Frege Systems, Extended Frege Systems, Resolution and so on. The notion of proof length in various proof systems is discussed in [Bus]. Observe that if it can be established that the length of *any* proof (refutation) of a contradiction must be exponential in the length of the input formula, then we have in fact separated the class **NP** from the class **coNP** [CR74].

Even if we focus on a particular proof system there exist several variants with different computational complexities. For instance, in case of resolution refutations, the commonly studied variants are tree-like proofs, dag-like proofs and read-once proofs [Har09]. Read-once refutations are the simplest from the conceptual perspective, since each clause (original or derived) can be used exactly once.

One of the interesting avenues of research in proof theory is the investigation of **incomplete** proof systems, i.e., proof systems which are not guaranteed to provide a refutation, even if the given formula is unsatisfiable. The idea behind the investigation of such weak systems is the hope that we can find proofs of unsatisfiability more efficiently [IM95]. This chapter focuses on a weak proof system called read-once resolution. It is well-known that read-once resolution is an incomplete proof system [IM95]. Furthermore, even asking if an arbitrary unsatisfiable CNF formula has a read-once refutation is **NP-complete**. As discussed before, read-once refutation is not sound for the purpose of checking NAE-satisfiability. We design a variant of read-once resolution called read-once NAE-resolution which is sound but not complete.

The investigations of this chapter are concerned with properties of read-once NAE-resolutions when applied to the problem of checking NAE-satisfiability in CNF formulas.

7.2 Refutability

7.2.1 The ROR problem for NAE-resolution

Now we focus on applying NAE-resolution to formulas in 3CNF and show that the problem whether for a formula ϕ the formula $\phi \cup \phi^c$ has a read-once resolution refutation is **NP-complete**. Since ROR - the set of formulas in CNF for which a read-once resolution exists - is **NP-complete**, we see that ROR-NAE-3SAT is in **NP**. Therefore, we only have to show **NP-hardness**. This is done by a reduction to the problem whether a formula in 2CNF has a read-once resolution refutation (ROR-2CNF).

***Theorem* 7.2.1.** *ROR-NAE-3SAT is* **NP-complete**.

Proof. Let ϕ be a 2CNF formula. We construct the 3CNF formula ϕ^* as follows:

1. For each variable x_i of ϕ, create the variable x_i for ϕ^*.

2. Create the variable x_0 for ϕ^*.

3. For each clause $\pi \in \phi$, create the clause $(\pi \vee x_0) \in \phi^*$.

We show that $\phi \in$ ROR-2CNF if and only if $\phi^* \in$ ROR-NAE-3SAT.

Assume that $\phi \in$ ROR-2CNF. A read-once resolution refutation $\phi \mid\overline{\scriptstyle RO\text{-}Res}\ \sqcup$ can easily be extended to the read-once NAE-resolution derivation $\phi^* \mid\overline{\scriptstyle RO\text{-}Res}\ x_0$. Thus, by Theorem 6.2.3, ϕ^* is in ROR-NAE-3SAT.

Now suppose that ϕ^* is in ROR-NAE-3SAT. We must show that ϕ has a read-once resolution refutation. We do this by showing that every resolution step done on the 3CNF clauses corresponds to a valid derivation on the 2CNF clauses.

We have the following cases:

1. $(x_i, x_j, x_0) \mid\overline{\scriptstyle Nae\text{-}Res}\ (\neg x_i, \neg x_j, \neg x_0)$: Both of these clauses correspond to the 2CNF clause (x_i, x_j). If (x_i, x_j) is satisfied, then both (x_i, x_j, x_0) and $(\neg x_i, \neg x_j, \neg x_0)$ are NAE-satisfied by setting x_0 to **false**.

2. $(x_i, x_j, x_0), (\neg x_k, \neg x_l, \neg x_0) \mid\!\!\frac{1}{Res}\; (x_i, x_j, \neg x_k, \neg x_l)$: This corresponds to the two CNF clauses $(x_i, x_j, \neg x_k, \neg x_l)$ and $(\neg x_i, \neg x_j, x_k, x_l)$. However, these are made redundant by the 2CNF clauses (x_i, x_j) and (x_k, x_l) which are already derivable from ϕ. Thus, no NAE-resolution refutation of ϕ^* performs a resolution step centered on x_0.

3. $(x_i, x_j, x_0), (\neg x_j, \neg x_k, x_0) \mid\!\!\frac{1}{Res}\; (x_i, \neg x_k, x_0)$: This corresponds to the resolution step $(x_i, x_j), (\neg x_j, \neg x_k) \mid\!\!\frac{1}{Res}\; (x_i, \neg x_k)$. Since $\phi \mid\!\!\frac{}{Res}\; (x_i, x_j)$ and $\phi \mid\!\!\frac{}{Res}\; (\neg x_j, \neg x_k)$, this is a valid derivation from ϕ.

Thus, all steps in the NAE-resolution refutation of the 3CNF formula correspond to steps used in the resolution refutation of the original 2CNF formula. Thus, ϕ^* has a read-once NAE-resolution refutation if and only if ϕ the has a read-once resolution refutation. □

Chapter 8

Horn Clausal Formulas

8.1 Motivation and Related Work

Propositional proof complexity is one of the most widely studied topics in computational complexity [Seg07]. It is primarily concerned with establishing the lengths of proofs in propositional logic. In analyzing the proof complexity of propositions, there are two parameters, which are important, viz., the type of proof system being considered and the manner in which length of proofs is measured. Some of the more common proof systems from classical logic include truth tables, Gentzen proof systems, resolution, Frege systems and Extended Frege systems [Urq95]. Additional proof systems with origins in computational algebra and integer programming include the Nullstellensatz proof system [BIK$^+$94], the polynomial calculus [AR01] and cutting planes [Pud97]. Measures of length include number of symbols used in a proof, number of resolution steps used in a proof, and the total width of the clauses used in a proof [Bus]. One of the principal goals of proof complexity is to separate the classes **NP** and **coNP**. Towards this end, it was shown in [CR74, Rec75] that all propositional tautologies had short proofs if and only if **NP**=**coNP**. Fairly comprehensive discussions on proof complexity can be found in [Urq95], [BP98], and [Kra94].

The resolution proof system is one of the most widely investigated proof systems in proof complexity, on account of its simplicity and ubiquity [BP98]. One of the first lower bounds for resolution was provided by Tseitin [Tse68], where it was shown that a restricted from of resolution had an exponential lower bound. This result was improved by Haken [Hak85]; he established that even general resolution had exponential length. To accomplish this, he showed that the lengths of resolution based proofs of the pigeonhole principle were eponential in the size of the input. Additional lower bounds on the length of resolution proofs are discussed in [BP98, Pud97].

In this chapter, we focus on a restriction of resolution called Read-Once resolution. The first systematic study of the ROR proof system was conducted by Iwama and Miyano in [IM95]. They argued that as proof systems become more powerful, the quest of finding a proof becomes harder. Consequently, it becomes worthwhile to investigate "weakened" proof systems such as ROR. One of their surprising results was that the problem of checking if a 3CNF formula has a read-once refutation is **NP-complete**. They also introduced the notion of copy complexity and showed that the problem of checking if a formula belongs to the differential class $R(k) - R(k-1)$ is $\mathbf{D^P} - \mathbf{complete}$, where the class $R(k)$ represents the set of formulas which have a read-once refutation with k copies of any clause being permitted. This result was generalized in [KZ02], where the authors discuss a number of results related to read-once refutations and minimal unsatisfiable formulas. The application of ROR to MAXSAT is detailed in [HM11], where the investigations are primarily from an empirical perspective. Szeider [Sze01] has studied a variant of ROR called literal-once resolution, which he shows is also **NP-complete**. In [KWS18], we showed that the problem of checking whether a 2-CNF formulas has a read-once refutation is **NP-complete**.

8.2 Refutability

8.2.1 The OLRR problem for resolution

In this sections, we discuss the problem of finding the optimal read-once refutation of a Horn formula.

Let α be an unsatisfiable Horn formula. From [Sze01] we know that α has a read-once refutation. The question whether α has a read-once resolution of length less than k is equivalent to the question whether α contains a minimal unsatisfiable formula consisting of at most k clauses.

***Theorem* 8.2.1.** *Let R denote an OLROR of Φ and let $\Phi_R \subseteq \Phi$ be the set of clauses used by R. R is also an optimal tree-like refutation of Φ and an optimal dag-like refutation of Φ. Additionally, Φ_R is a minimum unsatisfiable subset of Φ.*

Proof. Sine R is a read-once refutation of Φ, R is also a tree-like and a dag-like refutation of Φ.

Assume that T is a tree like refutation of Φ such that $|T| < |R|$. Let $\Phi_T \subseteq \Phi$ be the set of clauses used by T. It follows that $|\Phi_T| \leq |T| + 1 < |R| + 1$

By [Sze01], Φ_T must have a read-once refutation, R_T. However, $|R_T| \leq |\Phi_T| - 1 < |R|$. This contradicts the fact that R is the *optimal* read-once refutation of Φ. Thus, R is also an optimal tree-like refutation of R. Similarly, R is an optimal dag-like refutation of R.

Let $\Phi' \subseteq \Phi$ be an unsatisfiable Horn formula such that $\Phi' < \Phi_R$. By [Sze01], Φ' must have a read-once refutation, R'. However, $|R'| \leq |\Phi'| - 1 < |\Phi_R| - 1 = |R|$. This contradicts the fact that R is the *optimal* read-once refutation of Φ. Thus, Φ_R is a minimum unsatisfiable subset of R. □

We conclude, that for unsatisfiable Horn formulas the length of the shortest resolution refutation equals the length of the shortest read-once and the shortest tree resolution. Moreover, let k be the number of clauses of a minimal unsatisfiable subformula with mini-

mal number of clauses. Then the shortest resolution (read-once resolution, tree resolution) refutations is $k-1$. Note that this only applies to regular resolution refutations.

Let α be a definite Horn formula without unit clauses, x a variable, U a set of variables (unit-clauses), and $k \geq 1$.
Determining if there exists a subset $K \subseteq U$ of at most k variables such that $K \wedge \alpha \models x$ is **NP-complete**. The proof is based on a reduction to the vertex cover problem. Note that this problem asks for the number of variables and not the number of clauses.

The following result is a corollary of Theorem 5.2 in [Lib08]. However, it is included here for completeness.

***Theorem* 8.2.2.** *The problem of deciding whether a Horn formula contains an unsatisfiable sub-formula with at most k clauses is* **NP-complete**.

Proof. We show this by a reduction from Vertex Cover. Let $G = (V,E)$ be an undirected graph where $V = \{v_1, \ldots, v_n\}$ and $E = \{e_1, \ldots, e_m\}$. We associate with G the Horn formula:

$$v_1 \wedge \ldots \wedge v_n \wedge \bigwedge_{1 \leq i \leq m, e_i = (v_{i_1}, v_{i_2})} (v_{i_1} \rightarrow e_i) \wedge (v_{i_2} \rightarrow e_i) \wedge (\neg e_1 \vee \ldots \vee \neg e_m)$$

Then there exists a subset $V' \subseteq V$ such that $|V'| \leq r$ and $V' \cap e_i$ is non-empty for every $1 \leq i \leq m$ if and only if the associated formula contains an unsatisfiable sub-formula with at most $(1+m+r)$ clauses. (the negative clause, the clause $(v \rightarrow e_i)$ for each $1 \leq i \leq m$, and r unit clauses).

Since the problem of finding a vertex cover of size at most k is **NP-complete**, the problem of finding a read-once resolution refutation of length at most k is **NP-complete** for Horn formulas. □

8.2.2 The ROR problem for unit-resolution

In this section, we explore the problem of finding read-once unit resolution refutations for Horn formulas. If we restrict ourselves to unit resolution refutations then we are no

longer guaranteed read-once refutations.

Example 35: Consider the Horn formula

$$(x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_3)$$

This formula has the following read-once refutation:

$$\begin{aligned} (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \quad &|\frac{1}{Res} \quad (\neg x_1 \vee x_3) \\ (x_1) \wedge (\neg x_1 \vee x_3) \quad &|\frac{1}{Res} \quad (x_3) \\ (x_3) \wedge (\neg x_3) \quad &|\frac{1}{Res} \quad \sqcup \end{aligned}$$

However, we will now show that this formula does not have a read-once unit resolution refutation.

There are three possibilities for the final resolution step.

1. The final resolution step is $(x_1) \wedge (\neg x_1) \; |\frac{1}{Res} \; \sqcup$: Note that in this case it is impossible to generate the clause $(\neg x_1)$ by unit resolution.

2. The final resolution step is $(x_2) \wedge (\neg x_2) \; |\frac{1}{Res} \; \sqcup$: To generate the clause (x_2) we need to use the clause (x_1). However, we also need to use this clause to generate $(\neg x_2)$.

3. The final resolution step is $(x_3) \wedge (\neg x_3) \; |\frac{1}{Res} \; \sqcup$: To generate the clause $(\neg x_3)$ via unit resolution, we need to use the clause (x_1) twice. Once to resolve with $(\neg x_1 \vee x_2)$ and once to resolve with $(\neg x_1, \neg x_2, x_3)$.

Thus, It is not always possible to find a read-once unit refutation.

It was shown in [KZ03] that the UROR problem for Horn formulas is **NP-complete**. We now provide an alternative proof that the unit ROR problem is **NP-complete** for Horn formulas. This is done by a reduction from the set packing problem.

***Definition* 8.2.1.** *The* **set packing** *problem is the following: Given a set S, m subsets* $S_1, \ldots, S_m$ *of S, and an integer k, does* $\{S_1, \ldots, S_m\}$ *contain k mutually disjoint sets.*

This problem is known to be **NP-complete** [Kar72].

***Theorem* 8.2.3.** *The unit ROR problem for Horn formulas is* **NP-complete**.

Proof. Let us consider an instance of the set packing problem. We construct the Horn formula Φ as follows.

1. For each $x_i \in S$, create the boolean variable x_i and the clause (x_i).
2. For $j = 1 \ldots k$, create the boolean variable v_j.
3. For each subset S_l, $l = 1 \ldots m$ create the clauses

$$(v_j \vee \bigvee_{x_i \in S_l} \neg x_i) \qquad j = 1 \ldots k.$$

4. Finally create the variable w and the clauses $(w \vee \neg v_1 \vee \ldots \vee \neg v_k)$ and $(\neg w)$.

We now show that Φ has a read-once unit resolution refutation if and only if $\{S_1, \ldots, S_m\}$ contains k mutually disjoint sets.

Suppose that $\{S_1, \ldots, S_m\}$ does contain k mutually disjoint sets. Without loss of generality assume that these are the sets $S_1, \ldots, S_k$.

Let us consider the sets of clauses

$$\Phi_j = \{(v_j \vee \bigvee_{x_i \in S_j} \neg x_i)\} \cup \{(x_i) \,|\, x_i \in S_j\} \qquad j = 1 \ldots k.$$

By the construction of Φ, $\Phi_j \subseteq \Phi$ for $j = 1 \ldots k$. Since the sets $S_1, \ldots, S_k$ are mutually disjoint, so are the sets $\Phi_1, \ldots, \Phi_k$.

It it easy to see that the clause (v_j) can be derived from the set Φ_j by read-once unit resolution. Sine this holds for every $j = 1 \ldots k$ and since the sets $\Phi_1, \ldots, \Phi_k$ are mutu-

ally disjoint, the set of clauses $\{(v_1),\ldots,(v_k)\}$ can be derived from Φ by read-once unit resolution.

Together with the clause $(w \vee \neg v_1 \vee \ldots \vee \neg v_k)$, this set of clauses has a read-once unit derivation of the clause (w). Thus, Φ has a read-once unit derivation of the clause (w). Since Φ contains the clause $(\neg w)$, it follows that Φ has a read-once unit resolution refutation.

Now suppose that Φ has a read-once unit resolution refutation **R**. Note that $\Phi/\{(\neg w)\}$ can be satisfied by setting every variable to **true**. Thus, **R** must use the clause $(\neg w)$.

Let us consider the resolution step that involves the clause $(\neg w)$. By construction, $(\neg w)$ must be resolved with the clause $(w \vee \neg v_1 \vee \ldots \vee \neg v_k)$ or a clause derived from it. We can assume without loss of generality that the clause resolved with $(\neg w)$ has the form $(w \vee \neg v_1 \vee \ldots \vee \neg v_{k'})$ where $k' \leq k$. To derive this clause, the clauses $(v_{k'+1}),\ldots,(v_k)$ must have already been derived.

Thus,

$$(\neg w) \wedge (w \vee \neg v_1 \vee \ldots \vee \neg v_{k'}) \mid\!\!\frac{1}{Res}\ (\neg v_1 \vee \ldots \vee \neg v_{k'}).$$

To eliminate the clause $(\neg v_1 \vee \ldots \vee \neg v_{k'})$, **R** must either derive the clauses $(v_1),\ldots,(v_{k'})$, or reduce it to a unit clause and then resolve it with another clause. Without loss of generality we can assume that this unit clause is $(\neg v_1)$. In either case **R** must derive the clauses $(v_2),\ldots,(v_{k'})$.

Thus, we must derive the clauses $(v_2),\ldots,(v_k)$. Let us consider the clause (v_j), $2 \leq j \leq k$. By the construction of Φ, this clause must be derived from one of the clauses

$$(v_j \vee \bigvee_{x_i \in S_l} \neg x_i) \qquad l = 1 \ldots m.$$

To to this, we must use the set of clauses $\Psi_{l_j} = \{(x_i) \,|\, x_i \in S_{l_j}\}$ for some $l_j \leq m$.

Since the refutation is read-once, the sets Ψ_{l_j} for $j = 2 \ldots k$ are mutually disjoint. Thus, the sets S_{l_j} for $j = 2 \ldots k$ are also mutually disjoint.

If **R** derives the clause (v_1), then, by the construction of Φ, this clause must be derived from one of the clauses

$$(v_1 \vee \bigvee_{x_i \in S_l} \neg x_i) \qquad l = 1 \ldots m.$$

To do this, we must use the set of clauses $\Psi_{l_1} = \{(x_i) \,|\, x_i \in S_{l_j}\}$ for some $l_1 \leq m$.

If **R** reduces $(\neg v_1 \vee \ldots \vee \neg v_{k'})$ to the clause $(\neg v_1)$, then we must resolve $(\neg v_1)$ with one of the clauses

$$(v_1 \vee \bigvee_{x_i \in S_l} \neg x_i) \qquad l = 1 \ldots m.$$

This results in the clause $(\bigvee_{x_i \in S_{l_1}} \neg x_i)$ for some $l_1 \leq m$. To eliminate this clause, we must use the set of clauses $\Psi_{l_1} = \{(x_i) \,|\, x_i \in S_{l_j}\}$.

Since **R** is read-once, the set Ψ_{l_1} does not share any clauses with the sets Ψ_{l_j}, $j = 2 \ldots k$. Thus, the sets S_{l_j} for $j = 1 \ldots k$ are also mutually disjoint. This means that $\{S_1, \ldots, S_m\}$ contains k mutually disjoint sets.

Thus, Φ has a read-once unit resolution refutation if and only if $\{S_1, \ldots, S_m\}$ contains k mutually disjoint sets. As a result of this, the unit ROR problem for Horn formulas is **NP-complete**. □

***Theorem* 8.2.4.** *For a CNF formula Φ, the length of a read-once unit resolution refutation is at most $(m-1)$, where m is number of clauses in the formula.*

Proof. Recall that a read-once resolution step is equivalent to removing the two parent clauses from the formula and adding the resolvent. Thus, each read-once resolution step effectively reduces the number of clauses in the formula by 1. Since Φ initially has m clauses, there can be at most $(m-1)$ such resolution steps. □

8.2.3 The copy complexity of unit-resolution

We now examine the copy complexity of read-once unit resolution for Horn formulas.

***Theorem* 8.2.5.** *The copy complexity of Horn formulas is at most 2^{n-1} where n is the*

number of variables.

Proof. Suppose Φ is an unsatisfiable Horn formula. Note that adding clauses to a system cannot increase the copy complexity. Thus, we can assume without loss of generality that Φ is minimal unsatisfiable. Let $CC(n)$ denote the copy complexity of a minimal unsatisfiable Horn formula with n variables. We will show that $CC(n) \leq 2^{n-1}$. For a clause ϕ_k of Φ let $N_c(\phi_j)$ be the number of copies of ϕ_j needed for a read-once unit resolution refutation.

Let Φ be a minimal unsatisfiable Horn formula with n variables. Thus, Φ has $(n+1)$ clauses. If $n = 1$, then Φ has the form $(x) \wedge (\neg x)$. This formula has a read-once unit resolution refutation. Thus $CC(1) = 1 \leq 2^0$. Also note that $\sum_{\phi_j \in \Phi} N_c(\phi_j) = 2 \leq 2^1$.

Now assume that $CC(k) \leq 2^{k-1}$, and that for each minimal unsatisfiable formula Φ' with k variables, $\sum_{\phi'_j \in \Phi'} N_c(\phi'_j) \leq 2^k$. If $n = k+1$, then Φ has the form $(x) \wedge (\neg x \vee \alpha_1) \wedge \ldots \wedge (\neg x \vee \alpha_t) \wedge \sigma_{t+1} \ldots \wedge \sigma_{k+1}$. A read-once unit resolution refutation needs to use the clause (x) to eliminate each instance of $\neg x$. Thus, we need a copy of the clause (x) for each copy of $(\neg x \vee \alpha_i)$ for $i = 1 \ldots t$. Let Φ' be the formula $\alpha_1 \wedge \ldots \alpha_t \wedge \sigma_{t+1} \ldots \wedge \sigma_{k+1}$. Note that Φ' is a minimal unsatisfiable formula with $n-1 = k$ variables. Thus,

$$\sum_{j=1}^{t} N_c(\alpha_i) \leq \sum_{\phi'_j \in \Phi'} N_c(\phi'_j) \leq 2^k.$$

This means that we need at most 2^k copies of the clause (x). Thus, $CC(k+1) \leq 2^k$ and $\sum_{\phi_j \in \Phi} N_c(\phi_j) \leq 2^k + 2^k = 2^{k+1}$. □

***Theorem* 8.2.6.** *There exists a Horn formula with copy complexity 2^{n-1} where n is the number of variables.*

Proof. Consider the following clauses:

$$\begin{array}{ccc} (\neg x_1 \vee \neg x_2 \vee \ldots \vee \neg x_n) & (x_1 \vee \neg x_2 \vee \ldots \vee \neg x_n) & (x_2 \vee \neg x_3 \vee \ldots \vee \neg x_n) \\ \ldots & (x_{n-1} \vee \neg x_n) & (x_n) \end{array}$$

To eliminate $\neg x_2$ from the clauses $(\neg x_1 \vee \neg x_2 \vee \ldots \vee \neg x_n)$ and $(x_1 \vee \neg x_2 \vee \ldots \vee \neg x_n)$ we need 2 copies of the clause $(x_2 \vee \neg x_3 \vee \ldots \vee \neg x_n)$. However, we now have four instances of $\neg x_3$ so we need to use four copies of the clause $(x_3 \vee \neg x_4 \vee \ldots \vee \neg x_n)$. In general we need to use 2^{i-1} copies of the clause $(x_i \vee \neg x_{i+1} \vee \ldots \vee \neg x_n)$. Thus, we need to use 2^{n-1} copies of the clause (x_n). □

From these two results, it is easy to see that the copy complexity of Horn formulas with respect to read-once unit resolution refutation is 2^{n-1}.

Part III

Polyhedral Constraints: Linear Satisfiability

Chapter 9

Difference Constraint Systems

9.1 Motivation and Related Work

Every infeasible DCS has a refutation that verifies its infeasibility. For a DCS, the refutation is a subset of the difference constraints such that its conjunction results in a contradiction of the form $0 \leq -b$, $b > 0$. The length of a refutation is the number of difference constraints in the subset that proves the infeasibility of the DCS.

For each DCS **D**, there exists a corresponding difference constraint network **G**. **D** is infeasible if and only if **G** contains a simple, negative cost cycle. The length of a negative cost cycle is the number of edges in the negative cost cycle. The shortest negative cost cycle is defined as the negative cost cycle having the fewest number of edges. It follows that the refutation in **D** with the fewest number of constraints corresponds to the length of the shortest negative cost cycle in **G**.

If each difference constraint in a DCS has unit length, then the problem of determining the length of the refutation with the fewest number of constraints is called the optimal length resolution refutation (OLRR) problem. The OLRR problem is motivated by a number of applications, as discussed in [Sub09], including program verification [SLB03], real-time scheduling [HL89], and incremental shortest paths in weighted networks [DI04].

The first polynomial time algorithm for this problem was proposed in [Sub09] and runs in $O(n^3 \cdot \log K)$ time, where n is the number of vertices in **G**, and K is the OLRR. The current fastest algorithm runs in $O(m \cdot n \cdot K)$ time [SWG13], where m is the number of edges in **G**.

In this chapter, we are interested in a weighted DCS (WDCS), where a positive weight is associated with each constraint. We represent the constraint network of a WDCS **D** as a constraint network **G**, where each edge has both a cost and a positive, integral length. Note that the term "weight" is used for a WDCS, while the term "length" is used for the difference constraint network. In the case of a WDCS, the weight of a refutation is defined as the sum of the lengths of the edges in the corresponding negative cost cycle in **G**. The problem of finding the minimum weight refutation in a WDCS is called the weighted optimal length resolution refutation (WOLRR) problem. This problem is known to be **NP-hard** [Sub09].

Difference constraints occur in a wide variety of domains, including but not limited to program verification [NO05, CAMN04], real-time scheduling [GPS95a] and image segmentation [CRY96]. Similarly, UTVPI constraints are used extensively in array bounds checking [LM05] and abstract interpretation [SS10].

In this chapter, we are concerned with short certificates of infeasibility in the form of short resolution refutations. There are several reasons for desiring the shortest infeasibility certificate:

1. In system design, it has been observed that infeasibility typically results from a small subset of infeasible constraints [LTCA89]. When constraints in this subset are relaxed, the constraint system becomes feasible.

2. A significant section of research in Satisfiability (SAT) problems is concerned with the identification of a Minimum Unsatisfiable Core (MUC) of an unsatisfiable CNF formula. [LS04] describes both theoretical and practical milestones in this research. In [Sub09], it was shown that the MUC of a DCS coincides with its OROR. In case of UCS's, the MUC coincides with the OTLR.

The problem of finding short refutations is one of the principal problems in proof com-

plexity [BP98]. Research proceeds along the lines of finding lower bounds on the lengths of refutations for propositional tautologies (contradictions) in proof systems of increasing complexity, with a view towards separating the complexity class **NP** from the class **coNP** [Urq95]. Resolution is one of the weakest proof systems, but even in this proof system it was difficult to obtain lower bounds on the length of proofs. The first non-trivial lower bound on the length of resolution proofs is due to Haken [Hak85], who showed that any resolution proof for the pigeonhole principle required exponentially many steps. [Iwa97] showed that the problem of finding the shortest resolution proofs in arbitrary 3CNF formulas is **NP-complete**. A stronger result was obtained in [ABMP98]; they showed that the problem of finding the shortest resolution proof in Horn formulas is not linearly approximable, unless **P**=**NP**. This result is interesting because it is easy to see that every unsatisfiable Horn formula has a resolution refutation that is quadratic in the number of clauses. It is important to note that the problem of finding minimum witnesses of infeasibility arises in other domains too. For instance see [AAHO98], which analyzes the problem of finding minimum cardinality witnesses of minimum cost flow infeasibility.

On the read-once refutation side, [IM95] showed that the problem of checking if an arbitrary CNF has an ROR is **NP-complete**. [KZ02] strengthened this result by showing that the problem of checking whether a CNF formula has a read-once unit resolution refutation is **NP-complete**.

As we can see, much of the work in finding short refutations focused on discrete domains (CNF formulas). [Sub09] departed from existing work by considering difference constraint systems from the perspective of determining the optimal length resolution refutations. That paper, shows that short refutations exist for difference constraints and also that the optimal length refutation can be determined in polynomial time. It is worth noting that in DCS's, linear and integer feasibility coincide and therefore, the departure is not strict.

Our work in this chapter is motivated by applications in a number of different domains:

1. Program verification - SMT (Satisfiability Modulo Theories) solvers are increasingly being used in program verification procedures [dMOR$^+$04, FS02]. These solvers

are also part of procedures for bounded model checking in infinite state systems and test-case generation [DdM06]. An important subclass of SMT solvers are those devoted to difference logic (also called separation logic or difference constraint logic (DCL)) [SLB03]. Broadly, quantifier-free difference logic refers to the satisfiability of an arbitrary boolean combination of difference constraints. For instance, $(x_1 - x_2 \leq 7) \wedge ((x_3 - x_{10} \leq 4) \vee (x_9 - x_4 \leq 6) \vee \neg(x_1 - x_9 \leq 6))$ is a proposition in difference logic [CAMN04]. It is well-known that difference logic can be used to express bounded reachability for timed automata [ABK$^+$97], existence of timing paths in digital circuits with bounded delays [BS94] and other timing related problems, such as Job Shop Scheduling [ABZ88].

It was shown in [Tse70] that the satisfiability problem in DCL is `NP-complete` and that an arbitrary proposition of DCL can be converted into Conjunctive Normal Form in polynomial time. [SB04] showed that the unsatisfiability of a proposition in DCL is defined by a *conjunction* of difference constraints. In order to convince a user of the unsatisfiability of a proposition, it is necessary to provide him with a certificate; although any negative cost cycle serves as a certificate, in case of conjunctions of difference constraints, it would be preferable to provide the certificate of shortest length, i.e., the negative cost cycle having the least number of edges. We also note that the task of providing certificates is not necessarily unique to SMT solvers; indeed the field of certifying algorithms requires that certificates accompany all algorithmic outputs [KMMS03, WB97].

2. Proof Theory - A secondary motivation for our work arises from proof theory. One of the concerns in proof theory is the establishment of non-trivial lower bounds on the proof lengths of propositional tautologies (alternatively refutation lengths of propositional contradictions). An essential aspect of establishing a lower bound is the proof system used to establish the bound. For instance, super-polynomial bounds for tautologies have been established for weak proof systems such as resolution [Hak85],

while the establishment of such a bound for Frege proof systems would separate the complexity class NP from the complexity class coNP [BP98]. Likewise, establishing that there exist short refutations for all contradictions in a given proof system causes the classes NP and coNP to coincide [CR73]. The work on propositional proofs is easily extensible to other finite, discrete domains such as integer arithmetic [Pud97]. However, FM elimination is not a valid proof system for a system of integer inequalities; [Wil76] provides examples where FM is not sound. Proofs in integer arithmetic usually use cutting plane theory [BE00].

The resolution technique for clausal formulas has a number of variants including tree-like resolution, dag-like resolution, read-once resolution, linear resolution, and so on [Bus98, RV01]. Tree-like resolution and dag-like resolution are complete procedures in that if an input formula is unsatisfiable, then it must have a tree-like resolution proof and dag-like resolution proof. On the other hand, neither linear resolution nor read-once resolution are complete procedures; indeed the question of whether an arbitrary formula has a read-once proof is NP-complete [IM95]. We shall show later that in the case of difference constraint systems, tree-like proofs, dag-like proofs, linear proofs and read-once proofs coincide.

Although the work in this chapter focuses on linear arithmetic, on account of the total unimodularity property, it can be thought of as providing proofs for a subclass of integer arithmetic propositions.

3. Real-Time Scheduling - Whereas traditional scheduling problems are concerned with finding a sequence that optimizes a performance metric [Pin95a], real-time scheduling problems are characterized by the presence of non-constant execution times and complex timing relationships among jobs [Sub05b]. The timing relationships are circular in nature and hence cannot be captured through precedence graphs; indeed, in most cases a constraint network is needed to represent timing relationships [HL89]. If the constraints are infeasible, then there must exist a negative cost cycle under

the appropriate type of clairvoyance [Sub05a]. Empirical evidence [LTCA89] seems to suggest that infeasibility is usually the result of a small infeasible subset. When constraints in this subset are relaxed, the constraint system becomes feasible.

Constraint-based scheduling problems also arise in placement [AB93] and job-shops [BPN95]; if the constraints are unsatisfiable, the goal is to identify the smallest unsatisfiable subset.

4. SAT Research - A significant section of research in Satisfiability (SAT) problems is concerned with the identification of a Minimum Unsatisfiable core of an unsatisfiable CNF formula. [LS04] describes both theoretical and practical milestones in this research. It is important to note that the Minimum Unsatisfiable core problem is *not* the dual of the Maximum Satisfiable Subformula (MSS) problem. For instance, eliminating the minimum unsatisfiable core need not result in a satisfiable subformula. The MSS problem is strongly `NP-complete` for boolean formulas in CNF (MAXSAT) [GJ79]; what is surprising is that it is `NP-complete` for unsatisfiable linear programs as well [JP78].

Our work also finds application in the design of incremental algorithms for shortest paths in an arbitrarily weighted network [DI04].

9.2 Refutability

9.2.1 The OLRR problem (ADD rule)

In this section, we are concerned with finding the shortest read-once refutations of difference constraint systems.

9.2.1.1 Extracting a negative cycle in a directed graph from closed walks

In this section, we describe an algorithm for extracting a negative cost cycle (NCC) in a directed graph from closed walks.

A *walk* from a vertex x_i to a vertex x_j is a directed path commencing at x_i and ending at x_j. Note that the path need not be simple, i.e., a walk is permitted to repeat nodes and edges. If a walk commences and ends on the same vertex, it is said to be *closed*. A walk of k or fewer edges is called a k-walk.

A Bellman-Ford variant can be used to find a minimum cost closed k-walk around x_j in $O(m \cdot k)$ time. If a closed walk W has negative cost, then W contains a negative cost cycle. That is, W can be expressed as the union of cycles, such that at least one of the cycles has a negative cost.

Some observations are in order:

1. Let $d_i^{(k)}(j)$ denote the cost of the minimum cost k-walk from vertex x_i to vertex x_j. (We permit the case, where $i = j$.)

2. Using the principle of optimality, it is easy to see that

$$d_i^{(k+1)}(j) = \min \begin{cases} d_i^{(k)}(j) \\ d_i^{(k)}(r) + c_{r,j} : \quad (r,j) \in \mathbf{E} \end{cases} \tag{9.1}$$

   Given $d_i^{(k)}(j)$ for all $j = 1, 2, \ldots n$, we can compute $d_i^{(k+1)}$ in $O(m)$ time.

3. Pick $x_j \in \mathbf{V}$. We initialize $d_j^{(0)}(i)$ as follows:

$$d_i^{(0)}(j) = \begin{cases} 0, & \text{if } i = j \\ \infty, & \text{otherwise} \end{cases}$$

   A single Bellman-Ford sweep of all the edges in $\mathbf{G}$, with x_j as the source, gives the

minimum cost 1-walk from x_j to every vertex in $\mathbf{G}$ (including x_j). Using the principle of optimality described above, it follows that if we have computed the minimum cost k-walk from x_j to each vertex in the graph, then a subsequent Bellman-Ford sweep will detect the minimum cost $(k+1)$-walk from x_j to all the vertices in $\mathbf{G}$. It follows that the minimum cost closed k-walk around x_j can be computed in $O(m \cdot k)$ time.

4. We maintain a data structure $pred_i$ called the predecessor array for each vertex x_i. The structure $pred_i$ stores the predecessor tree of the single source shortest paths tree from vertex x_i. This data structure is updated during the Bellman-Ford sweep, so that the $pred[j]$ points to the parent of vertex x_j in the shortest paths tree from vertex x_i. To detect a negative cost closed k-walk around x_i, we check if $d_i^{(k)}(i) < 0$. If this is the case, then there must exist a simple negative cycle having at most k edges, in this closed walk. Such a cycle can be recovered in linear time by tracing back from vertex x_i using the $pred[\,]$ structure [AMO93]. Let W_j denote the minimum cost closed walk from x_j to itself. Then, the shortest negative cost cycle in $\mathbf{G}$ is the shortest of the walks W_j, across all vertices $x_j \in V$.

A Bellman-Ford sweep, as described previously, is performed by Algorithm 9.2.1.

Algorithm 9.2.1 An algorithm for performing a Bellman-Ford sweep.

Function BFSWEEP($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$, x_j, $\mathbf{d}$, $pred_j$)

1: Let $\mathbf{d}_{temp}$ be a copy of $\mathbf{d}$.
2: **for** (each edge $(x_{i_1}, x_{i_2}) \in \mathbf{E}$) **do**
3: **if** ($\mathbf{d}_{temp}[x_{i_2}] < \mathbf{d}[x_{i_1}] + c_{i_1,i_2}$) **then**
4: $\mathbf{d}_{temp}[x_{i_2}] \leftarrow \mathbf{d}[x_{i_1}] + c_{i_1,i_2}$.
5: $pred_j[x_{i_2}] \leftarrow x_{i_1}$.
6: **return** ($\mathbf{d}_{temp}$).

Example 36: Consider the graph **G** in Figure 9.1.

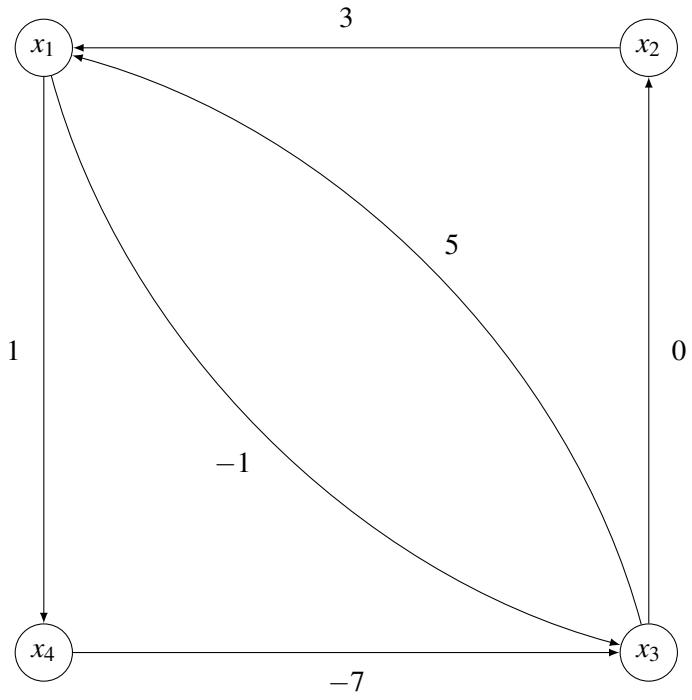

Figure 9.1: Directed Graph

Table 9.1 shows how a single Bellman-Ford, as performed by Algorithm 9.2.1, sweep can calculate $d_1^{(1)}$ from $d_1^{(0)}$.

	x_1	x_2	x_3	x_4
$d_1^{(0)}$	0	∞	∞	∞

edge	$d_1^{(0)}(1)$	$d_1^{(0)}(2)$	$d_1^{(0)}(3)$	$d_1^{(0)}(4)$
(x_1,x_3)	0	∞	-1	∞
(x_1,x_4)	0	∞	-1	1
(x_2,x_1)	0	∞	-1	1
(x_3,x_1)	0	∞	-1	1
(x_3,x_2)	0	∞	-1	1
(x_4,x_3)	0	∞	-1	1

Table 9.1: Bellman-Ford Sweep

Using Algorithm 9.2.1, we can now present Algorithm 9.2.2.

Algorithm 9.2.2 An algorithm for finding an NCC

Function NCC($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$, x_j, ρ)

```
1: Initialize d to be vector of minimum cost 0-walks from x_j to each vertex in G.
2: Initialize pred_j to a vector of vertices in G.
3: for (k = 1 to ρ) do
4:     d ← BFSWEEP(G, x_j, d, pred_j)
5:     if (d[j] < 0) then                ▷ A negative cost cycle of length k has been found.
6:         Find a negative cost simple cycle C in this closed walk.
7:         return (C).
```

This algorithm determines if the vertex x_j is part of a negative cost walk using at most ρ edges. If x_j is past of such a walk, then NCC() will return a negative cycle that is part of that walk.

Example 37: Consider the graph $\mathbf{G}$ in Figure 9.1. Table 9.2 shows how successive Bellman-Ford sweeps can detect the shortest negative cost cycle that uses x_1.

	x_1	x_2	x_3	x_4
$d_1^{(0)}$	0	∞	∞	∞
$d_1^{(1)}$	0	∞	-1	1
$d_1^{(2)}$	0	-1	-6	1
$d_1^{(3)}$	-1	-6	-6	1

Table 9.2: Minimum cost k-walks from x_1 for $k = 0 \ldots 3$.

As can be seen from Table 9.2, $d_1^{(3)}(1) < 0$. Thus, there is a negative cost 3-cycle in $\mathbf{G}$ that uses x_1. This cycle consists of the edges (x_1, x_4), (x_4, x_3), and (x_3, x_1) and has total cost -1.

***Theorem* 9.2.1.** *If x_j is on a negative cost closed walk with at most ρ edges, then* NCC(**G**, x_j, ρ) *will return a negative cost cycle.*

Proof. This follows from the observation that every closed negative walk consists of a union of simple cycles, at least one of which has negative cost. □

***Corollary* 9.2.1.** *Let C^* be a shortest NCC of* **G**. *If x_j is on C^* and $|C^*| \leq \rho$, then* NCC(**G**, x_j, ρ) *will return a shortest NCC of* **G**.

Proof. The key observation is that in this case the first negative closed walk that is detected is the simple negative cycle involving x_j. □

It is worth noting that because of Corollary 9.2.1, we do not need to rely on Theorem 9.2.1. However, if we were to discover a negative cost walk that is not a simple cycle, then it is useful to discover the simple negative cost cycle as well.

9.2.1.2 A simple randomized algorithm for SNCC

In this section, we present the first of our two randomized algorithms for SNCC; Algorithm 9.2.3 represents our strategy.

Algorithm 9.2.3 A randomized algorithm for SNCC

Function SHORTEST-NEGATIVE-CYCLE($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$)

1: Let σ be a randomly generated permutation of $\{1, \ldots, n\}$.
2: $D \leftarrow \text{NCC}(\mathbf{G}, x_{\sigma(1)}, n)$.
3: **if** $(D = \emptyset)$ **then**
4: **return** (-1).
5: **else**
6: $l \leftarrow |D|$.
7:
8: **for** $(j = 2$ **to** $n)$ **do** ▷ For each vertex x_j do
9: $\rho \leftarrow \min\{l, \left\lceil \frac{n}{j-1} \right\rceil\}$.
10: $C \leftarrow \text{NCC}(\mathbf{G}, x_{\sigma(j)}, \rho)$.
11: **if** $(C \neq \emptyset)$ **and** $(|C| < l)$ **then**
12: $D \leftarrow C$.
13: $l \leftarrow |C|$.
14: **return** (l).

Algorithm 9.2.3 initially permutes the vertices $\{x_1, \ldots, x_n\}$ of the input network $\mathbf{G}$ uniformly and at random; that is, each of the $n!$ permutations of the vertex indices is equally likely. Observe that the algorithm requires $\Omega(\log n!) = \Omega(n \log n)$ random bits.

The algorithm then checks if $\mathbf{G}$ has a negative cost cycle reachable from $x_{\sigma(1)}$ and exits in the absence of such a cycle. If a negative cost cycle is found, then its length is recorded in l.

Each vertex is processed in sequence. The algorithm proceeds by generating the minimum cost k-walks from x_j to itself, for all values of k in the set $\{1,2,\ldots,\min\{l,\lceil\frac{n}{j-1}\rceil\}\}$, where l is the length of the shortest negative cycle found thus far. For instance, consider the vertex x_2. All minimum cost closed k-walks around x_2 for each $k \leq \min\{l,n\}$ are generated. Likewise, in the case of vertex x_3, all minimum cost closed k-walks around x_3 having at most $\min\{l,\lceil\frac{n}{2}\rceil\}$ edges are generated and so on.

Algorithm 9.2.3 can be viewed as a *true-biased* Monte-Carlo algorithm. If it returns a value of l for the length of a shortest NCC, then it guarantees that a shortest NCC of $\mathbf{G}$ has no more than l edges. Furthermore, the probability that a shortest NCC has fewer edges is bounded above by $\frac{1}{e}$.

9.2.1.2.1 Running time analysis

Let $T(n,m)$ denote the running time of Algorithm 9.2.3 on a network $\mathbf{G}$ with n vertices and m edges. In this subsection, we show that $T(n,m) = O(m \cdot n \cdot \log n)$.

For Step (1), we can use any of the Bellman-Ford variants described in [AMO93]. All these variants run in $O(m \cdot n)$ time.

The bottleneck operation is Step 6, which is run for each $j = 1$ to n. The running time of NCC($\mathbf{G}$, x_i, ρ) is at most $q \cdot m \cdot \rho$ for some constant $q > 0$. The value of ρ in Step 20 is set in Step 17 to $\rho(j) = \min\{l, \left\lceil\frac{n}{j-1}\right\rceil\}$.

Therefore,

$$\begin{aligned} T(n,m) &\leq \sum_{j=1}^{n} q \cdot m \cdot \min\left\{l, \left\lceil\frac{n}{j-1}\right\rceil\right\} \\ &\leq q \cdot m \cdot \left(n + \sum_{j=2}^{n} \left\lceil\frac{n}{j-1}\right\rceil\right) \\ &\leq 2 \cdot q \cdot m \cdot n \cdot H_n, \end{aligned}$$

where H_n is the n^{th} Harmonic number, and is at most $(1+\ln n)$.

It follows that $T(n,m) = O(m \cdot n \cdot \log n)$.

9.2.1.2.2 Space analysis

In order to store the input graph, we require $\Theta(m+n)$ space. The distance labels from a given vertex require $\Theta(n)$ space and so does the predecessor structure.

Observe that once a vertex has been processed, we no longer need its distance labels. Likewise, we need to maintain its predecessor structure, only if it holds the current shortest negative cost cycle. If not, it can be discarded as well.

It follows that the algorithm requires at most $O(n)$ additional space and $O(m+n)$ space overall. This is a significant improvement over the quadratic space requirements of all known deterministic algorithms [Sub09, SWG13].

9.2.1.2.3 Analysis of error bounds

We now establish that Algorithm 9.2.3 returns a shortest NCC of $\mathbf{G}$ with probability at least $(1-\frac{1}{e})$.

We assume that $\mathbf{G}$ has at least one negative cost cycle. Otherwise, Algorithm 9.2.3 returns (-1). In this case, the error probability is 0.

Let C^* be a shortest NCC of $\mathbf{G}$.

Let us consider a vertex $a \in C^*$.

We compute the probability that C^* is not discovered when vertex a is examined.

Assume that the label of a is r under the randomized permutation.

First observe that if $r = 2$, then C^* will clearly be discovered, since for the first vertex we examine paths up to length l, where l is the number of edges in the current candidate for the shortest negative cost cycle.

The only way in which C^* is not discovered when a is processed, is if the paths generated from a are of length at most $|C^*|-1$. In other words, we must have $\lceil \frac{n}{r-1} \rceil < |C^*|$, which in turn implies that $r > \left\lceil \frac{n}{|C^*|} \right\rceil$. Since a was chosen arbitrarily, the above observation

holds for all vertices in C^*. Thus, for C^* to go undiscovered in the **for** loop commencing on Step 6, none of the first $\left\lceil \frac{n}{|C^*|} \right\rceil$ vertices in the random permutation can belong to C^*.

The probability that none of the first $\alpha = \left\lceil \frac{n}{|C^*|} \right\rceil$ vertices belong to C^* can be computed as:

$$\begin{aligned}
\frac{n-|C^*|}{n} \cdot \frac{n-|C^*|-1}{n-1} \cdot \ldots \cdot \frac{n-|C^*|-\alpha+1}{n-\alpha+1} &= \prod_{k=1}^{\alpha} \frac{n-|C^*|-k+1}{n-k+1} \\
&\leq \left(\frac{n-|C^*|}{n}\right)^{\alpha} \\
&\leq \left(1-\frac{1}{\alpha}\right)^{\alpha} \\
&\leq \frac{1}{e}
\end{aligned}$$

It follows that Algorithm 9.2.3 succeeds with probability at least $(1-\frac{1}{e}) > 0.63$.

Clearly, we can boost the probability of success by running Algorithm 9.2.3 multiple times and taking the negative cycle with the smallest number of edges. In particular, if we run it p times, the error probability will drop to e^{-p}, while the running time will increase to $O(m \cdot n \cdot p \cdot \log n)$.

9.2.1.3 Reducing the number of random bits

In this section, we describe a new randomized algorithm for the SNCC problem with the goal of reducing the number of random bits. We recall that the algorithm discussed in previous section requires
$\Omega(\log n!) = \Omega(n \cdot \log n)$ bits.

Our ideas are similar to those used in the design of skip lists (see [Pug92]).

Instead of randomly permuting the vertices, we associate with each vertex v, a random variable $level(v)$. This variable is determined as follows:

1. For each vertex v, we choose random bits until a bit with value 1 is chosen.

2. Let $level(v)$ represent the number of bits chosen this way.

For each vertex v we require, on average, 2 random bits. Thus, on average, we require $O(n)$ bits; in fact, the bound is $O(l)$, where l is the length of the shortest negative cost cost cycle found during the initialization phase (Step 3 of Algorithm 9.2.4). This is an improvement over the $\Omega(n \cdot \log n)$ required by the previous algorithm.

From each vertex v, the new algorithm searches for negative cycles of length at most $p \cdot 2^{level(v)+1}$. This approach is detailed in Algorithm 9.2.4. Note that Algorithm 9.2.4 is parameterized by input parameter p in that the probability that it finds the shortest NCC in the input graph $\mathbf{G}$ is at least $(1 - e^{-p})$.

Algorithm 9.2.4 A new randomized algorithm for SNCC

Function SHORTEST-NEGATIVE-CYCLE($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle, p$)

```
1:  D ← NCC(G, x_1, n).
2:  if (D = ∅) then
3:      return (−1).
4:  else
5:      l ← |D|.
6:
7:  for (each vertex v ∈ V) do
8:      t ← level(v).
9:      ρ ← min{l, p · 2^{t+1}}.
10:     C ← NCC(G, v, ρ).
11:     if (C ≠ ∅) and (|C| < l) then
12:         D ← C.
13:         l ← |C|.
14: return (l)
```

9.2.1.3.1 Running time and space analysis

In this subsection, we show that the expected running time of Algorithm 9.2.4 on a network $\mathbf{G}$ with n vertices and m edges is $O(m \cdot n \cdot p \cdot \log n)$

After initialization, we have a negative cycle of length l. Let T_v denote the random variable that represents the time taken by Algorithm 9.2.4 to execute the **for** loop on Line

5 when processing vertex v. We are interested in computing $\mathbf{E}[T_v]$. Let $i = level(v)$. Algorithm 9.2.4 finds walks of length $p \cdot 2^{i+1}$ or of length l from v, whichever is less (see Step 7). Let $q \cdot m$ denote the time taken for a single Bellman-Ford sweep of $\mathbf{G}$, where $q > 0$ is a fixed constant.

Accordingly, we have,

$$\begin{aligned}
\mathbf{E}[T_v] &\leq \sum_{i=1}^{\lceil \log l \rceil - 1} q \cdot m \cdot p \cdot 2^{i+1} \cdot \mathbf{Pr}[level(v) = i] + q \cdot m \cdot l \cdot \mathbf{Pr}[level(v) = \lceil \log l \rceil] \quad (9.2) \\
&\leq 2 \cdot q \cdot m \cdot p \cdot \log l + q \cdot m \cdot l \cdot \frac{1}{l} \quad (9.3) \\
&\leq q' \cdot m \cdot p \cdot \log l, \text{ for some constant } q' \quad (9.4)
\end{aligned}$$

Summing over all the vertices in $\mathbf{G}$, it follows that the expected running time of Algorithm 9.2.4 is $O(m \cdot n \cdot p \cdot \log l)$, i.e., $O(m \cdot n \cdot p \cdot \log n)$.

It is not hard to see that Algorithm 9.2.4 can be implemented in $O(m+n)$ space as well.

9.2.1.3.2 Analysis of error bounds

We now establish that Algorithm 9.2.4 returns a shortest NCC of $\mathbf{G}$ with probability at least $(1 - e^{-p})$.

We assume that $\mathbf{G}$ has at least one negative cost cycle. Otherwise, Algorithm 9.2.4 returns (-1). In this case, the error probability is 0.

Let C^* be a shortest NCC of $\mathbf{G}$. For each vertex v, $\rho \geq 4 \cdot p$. Thus, if $|C^*| \leq 4 \cdot p$, then Algorithm 9.2.4 is guaranteed to succeed in finding the shortest negative cost cycle.

Let us consider a vertex $v \in C^*$.

We compute the probability that C^* is not discovered when vertex v is examined.

This can only happen if $p \cdot 2^{level(v)+1} \leq \rho < |C^*|$.

Let $\beta = \left\lfloor \log \frac{|C^*|}{p} \right\rfloor$. We have that C^* will not be discovered when v is examined only if $level(v) \leq \beta$.

The probability of this occurring is $(1-2^{\beta}) \leq (1-\frac{p}{|C^*|})$.

Thus, the probability of failing when checking each vertex $v \in C^*$ can be computed as:

$$\left(1-\frac{p}{|C^*|}\right)^{|C^*|} \leq e^{-p}$$

It follows that Algorithm 9.2.4 succeeds with probability at least $(1-e^{-p})$.

9.2.2 The WOLRR problem (ADD rule)

In this section, we are concerned with weighted difference constraints. A constraint $\mathscr{C}$ of the form $x_i - x_j \leq b_{ij}$ is called a *weighted difference constraint* if $\mathscr{C}$ is associated with a weight $l_{ij} > 0$, where $\mathbf{l} : \mathbf{E} \rightarrow \mathbb{Z}^+$ is the weight function. Observe that if $\mathbf{E}' \subseteq \mathbf{E}$ is a set of edges in $\mathbf{G}$, then $l(\mathbf{E}') = \sum_{e_{ij} \in \mathbf{E}'} l_{ij}$ is defined as the sum of the lengths of all edges in $\mathbf{E}'$. A conjunction of weighted difference constraints is called a *weighted difference constraint system* (WDCS). Constructing the corresponding constraint network of a WDCS $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b}, \mathbf{l} \rangle$ is similar to constructing the constraint network of a DCS. The key difference is that for each constraint $\mathscr{C} : x_i - x_j \leq b_{ij}$ with weight l_{ij}, we add the edge $e_{ji} = (v_j, v_i)$ with cost b_{ij} and *length* $l(\mathscr{C}) = l_{ij}$. We denote the length of a path P_{ij} from vertex v_i to vertex v_j as $l(P_{ij})$. Note that for constraint networks, we use the term "length" rather than "weight."

We already know that if a DCS is unsatisfiable, then there must exist a simple negative cost cycle in the corresponding constraint network [Sub09]. The same applies for a WDCS. Therefore, the refutation with the smallest total weight corresponds to the negative cost cycle ("no"-certificate) with the smallest total length. We call the length of such a negative cost cycle the *weighted optimal length resolution refutation* (WOLRR).

Using the terminology above, we define the WOLRR problem as follows: *Given a WDCS* $\mathbf{D} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$*, where the weight of each constraint is a positive integer, find the weight of a refutation having the smallest total weight.*

Alternatively, based on the equivalence between difference constraints and constraint networks, we define the WOLRR problem as: *Given a network* $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b}, \mathbf{l} \rangle$*, where* $\mathbf{b}$ *is the set of real edge costs and* $\mathbf{l}$ *is the set of positive integral edge lengths, find the length of a negative cost cycle having the smallest total length.*

9.2.2.1 A Pseudo-Polynomial Time Algorithm

In this section, we present a pseudo-polynomial time algorithm for computing the WOLRR in a WDCS. Recall that an algorithm runs in pseudo-polynomial time if the running time is bounded by both the *size* (i.e., number of bits) and *magnitude* (i.e., value) of the input. Note that pseudo-polynomial time algorithms may run in exponential time in the worst case scenario. However, they may run in polynomial time if the input is bounded by a polynomial function.

Consider the difference constraint network $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b}, \mathbf{l} \rangle$. Our approach applies the pseudo-polynomial time algorithm described in [GRKL01]. This algorithm computes the shortest path from source vertex v_1 to all other vertices $v_j \in \mathbf{V}$ for networks with positive integral edge costs and having a transition time at most T. Note that [GRKL01] denotes the cost of an edge with l_{ij}, while we use b_{ij}. Furthermore, [GRKL01] uses the term "delay" (denoted as t_{ij}), while we use the term "length" (denoted as l_{ij}) to define the same property.

Assume the vertices are enumerated from 1 to n, where v_1 denotes the source vertex. Let $L(j,t)$ denote the cost of the shortest path from vertex v_1 to vertex v_j with length at most t. We compute $L(j,t)$ using the following dynamic program [GRKL01]:

$$L(j,t) = \begin{cases} 0 & j = 1 \quad t = 0,\ldots,T \\ \infty & j = 2,\ldots,n \quad t = 0 \\ \min\left\{ L(j,t-1), \min\limits_{k \mid t_{kj} \le t, e_{kj} \in \mathbf{E}} \{L(k,t-t_{kj}) + b_{kj}\} \right\} & j = 2,\ldots,n \quad t = 1,\ldots,T. \end{cases}$$

Our algorithm modifies the dynamic program for networks with real edge costs. We use the notation $D(j,t)$ rather than $L(j,t)$. This is to differentiate our modified dynamic program from the dynamic program in [GRKL01]. We initialize $D(j,t)$ to 0 when $j =$

1. However, we compute $D(j,t) = \min\left\{D(j,t-1), \min_{k|t_{kj}\leq t, e_{kj}\in \mathbf{E}}\{D(k,t-t_{kj})+b_{kj}\}\right\}$ when $j = 1,\ldots n$. Note that our algorithm does not apply to networks where the edge lengths may be zero. Otherwise, $D(j,t)$ could be defined in terms of itself if $t_{kj} = 0$.

After computing $D(j,t)$ for all $v_j \in \mathbf{V}$ and a single value of t, we check if $D(1,t) < 0$. If this is true, then there exists a negative cost cycle from vertex v_1 to itself with length t. Otherwise, we repeat the computation for $t+1$, where $t+1 \leq T$.

To compute the WOLRR, we apply the above dynamic program for all vertices. For each source vertex v_s, let $D_s(j,t)$ be the shortest path from v_s to v_j with length t. We compute $D_s(j,t)$ for all values of s, j, and a single value of t. We then check if $D_s(s,t) < 0$ for any $v_s \in \mathbf{V}$. If this is true, we immediately halt the algorithm and return t as the WOLRR. Otherwise, we repeat the calculations for $t+1$, where $t+1 \leq T$. If L denotes the largest length of any edge in $\mathbf{G}$, then we set $T = n \cdot L$, which is the largest possible length for any negative cost cycle.

The above observations are summarized in Algorithm 9.2.5 and Algorithm 9.2.6. Observe that Algorithm 9.2.6 gives us only the weight of the shortest refutation. The actual negative cost cycle can be obtained by using a predecessor subgraph.

Algorithm 9.2.5 Single Vertex WOLRR Algorithm

Function SINGLE-VERTEX-WOLRR($\mathbf{G}, v_s, t$)

1: Enumerate the vertices such that v_s is v_1.
2: **for** ($j = 1$ to n) **do**
3: $\quad D_1(j,t) = \min\left\{D_1(j,t-1), \min_{k|l_{kj}\leq t, e_{kj}\in \mathbf{E}}\{D_l(k,t-l_{kj})+b_{kj}\}\right\}$.
4: **if** ($D_1(1,t) < 0$) **then**
5: $\quad$ **return** (**True**).
6: **return** (**False**).

9.2.2.1.1 Analysis

From [GRKL01], we know that Algorithm 9.2.5 takes $O(m)$ time because we scan each edge exactly once.

We now analyze the running time of Algorithm 9.2.6. The **for** loop at line 4 clearly has

Algorithm 9.2.6 Pseudo-Polynomial Time Algorithm for WOLRR

Function PSEUDO-WOLRR(**G**)

```
1:  n = |V|.
2:  L = max_{v_i,v_j ∈ V}{l_ij}.
3:  T = n · L.
4:  for (each vertex s ∈ V) do
5:      for (t = 0 to T) do
6:          D_s(s,t) = 0.
7:          for (v_j ∈ V − {s}) do
8:              D_s(j,t) = ∞.
9:  for (t = 1 to T) do
10:     for (each vertex v_s ∈ V) do
11:         SINGLE-VERTEX-WOLRR(G, v_s, t).
12:         if (SINGLE-VERTEX-WOLRR returned True) then
13:             return ("The WOLRR is t").
```

$O(n)$ iterations. Observe that the **for** loop at line 5 takes $O(T) = O(n \cdot L)$ time, where L is the largest length among all edges. Furthermore, the **for** loop at line 7 takes $O(n)$ time. Therefore, the **for** loop at line 4 runs in $O(n^2 \cdot L)$ time.

To analyze the **for** loop at line 9, observe that line 11 takes $O(m)$ time since this is Algorithm 9.2.5. The **for** loop at line 10 has $O(n)$ iterations, and the **for** loop at line 9 has $O(T) = O(n \cdot L)$ iterations. Therefore, the **for** loop at line 9 takes $O(n \cdot L \cdot n \cdot m) = O(m \cdot n^2 \cdot L) = O(n^4 \cdot L)$ time.

9.2.2.1.2 Correctness

We now prove the correctness of our pseudo-polynomial time algorithm. We first address the correctness of the dynamic program. Observe that [GRKL01] proves that the dynamic program correctly computes the shortest paths from vertex v_j to all other vertices with length t for $j = 2, \ldots, n$ and $t = 1, \ldots, T$, where $T = n \cdot L$. The key difference with our algorithm is that we include $j = 1$ in the computation. Thus, it must be the case that the dynamic program correctly computes the shortest paths from the source vertex $v_j = v_1$ to all other vertices.

This implies that when we update the value of $D(j,t)$ for $j = 1$, $D(j,t)$ represents the

cost of the shortest cycle containing vertex v_1 whose length is at most t. The smallest t, for which $D_s(s,t) < 0$, represents the length of the shortest negative cost cycle $\mathbf{C}$ containing v_s. We need to show that $\mathbf{C}$ is a simple negative cost cycle.

Suppose $\mathbf{C}$ is not a simple negative cost cycle, and there exists a vertex $v \in \mathbf{V}$ that appears in $\mathbf{C}$ more than once. Consider the path along $\mathbf{C}$ from v to itself. This path forms a cycle, which we denote as $\mathbf{C}_1$. Furthermore, $\mathbf{C} \backslash \mathbf{C}_1$ forms a second cycle, denoted as $\mathbf{C}_2$. Observe that the total cost of $\mathbf{C}$ is the sum of the costs of $\mathbf{C}_1$ and $\mathbf{C}_2$. Likewise, the total length of $\mathbf{C}$ is the sum of the lengths of $\mathbf{C}_1$ and $\mathbf{C}_2$.

Since $\mathbf{C}$ has a negative cost, at least $\mathbf{C}_1$ or $\mathbf{C}_2$, or both, must also have a negative cost. Without loss of generality, assume that $\mathbf{C}_1$ is the negative cost cycle. Since all edge lengths are strictly positive, the total length of $\mathbf{C}_1$ is less than the total length of $\mathbf{C}$. This contradicts the fact that $\mathbf{C}$ is the negative cost cycle with the smallest length. Therefore, $\mathbf{C}$ must be a simple negative cost cycle.

9.2.2.2 A Fully Polynomial-Time Approximation Scheme

In this section, we present a fully polynomial time approximation scheme (FPTAS) for computing the WOLRR of a DCS.

9.2.2.2.1 Preprocessing phase

The first phase of our algorithm converts $\mathbf{G}$ into a simpler network by erasing a carefully selected subset of edges. This phase preserves the WOLRR of $\mathbf{G}$.

Algorithm 9.2.7 removes the edges of $\mathbf{G}$ one-by-one in descending order with respect to the lengths until $\mathbf{G}$ does not have a negative cost cycle. Let e_{uv} be the last edge removed in this manner. Observe that the length of any negative cost cycle in $\mathbf{G}$ is at least l_{uv}. This is because any negative cost cycle has to contain at least one edge whose length is at least l_{uv}. Therefore, l_{uv} is a lower bound for the WOLRR of $\mathbf{G}$.

Consider the moment immediately before the algorithm removes e_{uv} from $\mathbf{G}$. l_{uv} is an upper bound for the lengths of the remaining edges in $\mathbf{G}$ at that time moment since the

Algorithm 9.2.7 Preprocessing Step

Function PRE-PROCESS(**G**)

1: Let A be a vector of edges initialized as $\mathscr{A} = \emptyset$.
2: **while** (**G** has a negative cost cycle) **do**
3: Let e_{ij} denote the edge of **G** with the largest length.
4: Remove e_{ij} from **G**.
5: Add e_{ij} to $\mathscr{A}$.
6: Let e_{uv} be the last edge added to $\mathscr{A}$.
7: **for** (each edge e_{st} in $\mathscr{A}$ such that $l_{st} \leq n \cdot l_{uv}$) **do**
8: Add e_{st} back to **G**.

algorithm removes the edges in descending order with respect to their lengths. Since **G** has a negative cost cycle at that moment, and a simple cycle can have at most n edges, $(n \cdot l_{uv})$ is an upper bound for the WOLRR of **G**. In other words, if $|OPT|$ is the length of the negative cost cycle with the smallest length in **G**, then $|OPT| \leq n \cdot l_{uv}$.

Algorithm 9.2.7 then inserts the edges with length at most $(n \cdot l_{uv})$ back into **G**. This means that when the algorithm terminates, the only edges that are pruned are the ones whose lengths are more than $(n \cdot l_{uv})$. Note that the transformation made by Algorithm 9.2.7 on **G** preserves the WOLRR. Since one can check the existence of a negative cost cycle on a constraint network in $O(m \cdot n)$ time, the running time of Algorithm 9.2.7 is $O(m^2 \cdot n) = O(n^5)$.

9.2.2.2.2 An FPTAS for WOLRR

We next present the main part of our algorithm. Let $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{b}, \mathbf{l} \rangle$ be the constraint network after the preprocessing step, and let $\varepsilon > 0$. We let $P = \frac{\varepsilon \cdot l_{uv}}{n}$, where $\varepsilon > 0$ is arbitrarily chosen. For each edge e_{ij} remaining in **G**, we set l'_{ij} to be $\left\lceil \frac{l_{ij}}{P} \right\rceil$. We then apply Algorithm 9.2.6 on $\mathbf{G}' = \langle \mathbf{V}, \mathbf{E}, \mathbf{b}, \mathbf{l}' \rangle$, and the resulting WOLRR is our approximation. The above observations are summarized in Algorithm 9.2.8.

Let OPT denote the negative cost cycle with the smallest length in **G**. Let OPT' denote the negative cost cycle with the smallest length after running Algorithm 9.2.8. Let $|OPT|$ and $|OPT'|$ denote the lengths of their respective negative cost cycles. Our algorithm re-

Algorithm 9.2.8 FPTAS for WOLRR

Function WOLRR-FPTAS($\mathbf{G}$)

1: PRE-PROCESS().
2: Let $\mathbf{G}$ be the resulting constraint network.
3: Let $P = \frac{\varepsilon \cdot l_{uv}}{n}$.
4: **for** (each edge $e_{ij} \in \mathbf{E}$) **do**
5: $\quad l'_{ij} = \left\lceil \frac{l_{ij}}{P} \right\rceil$.
6: Define: $\mathbf{G}' = \langle \mathbf{V}, \mathbf{E}, \mathbf{b}, \mathbf{l}' \rangle$
7: Let OPT' denote the resulting negative cost cycle with the smallest length from running PSEUDO-WOLRR($\mathbf{G}'$).
8: **return** ($|OPT'|$).

turns $|OPT'|$ at termination. In order to prove that our algorithm is an FPTAS, we will show that $|OPT'| \leq (1 + 2 \cdot \varepsilon) \cdot |OPT|$. Clearly, this will prove our claim since $\varepsilon > 0$ is chosen arbitrarily and 2 is a constant.

Recall that for each edge $e_{ij} \in \mathbf{E}$, we have $l'_{ij} = \left\lceil \frac{l_{ij}}{P} \right\rceil < \frac{l_{ij}}{P} + 1$. We claim that $l_{ij} < P \cdot l'_{ij} + P$. If $l_{ij} \geq P \cdot l'_{ij} + P$, then $\frac{l_{ij}}{P} \geq l'_{ij} + 1$, and therefore, $l'_{ij} = \left\lceil \frac{l_{ij}}{P} \right\rceil \geq \frac{l_{ij}}{P} \geq l'_{ij} + 1$, which is a contradiction.

Let $l'(\mathbf{C}) = \sum_{e_{ij} \in \mathbf{C}} l'_{ij}$ be defined as the sum of the scaled and rounded lengths of the edges in $\mathbf{C}$. If we add the above inequalities for all edges e_{ij} that lie in OPT', we have $l(OPT') < P \cdot l'(OPT') + P \cdot n$. Here we used the fact that OPT' contains at most n edges. Now, observe that OPT' is a negative cost cycle with the smallest length in $\mathbf{G}'$. Hence, it must be the case that $l'(OPT') \leq l'(OPT)$. Thus, we will have $l(OPT') < P \cdot l'(OPT) + P \cdot n$. Taking into account that $l'_{ij} < \frac{l_{ij}}{P} + 1$ and $\varepsilon \cdot l_{uv} = n \cdot P$, we get

$$\begin{aligned} |OPT'| &= l(OPT') < P \cdot l'(OPT) + P \cdot n < P \cdot (\tfrac{l(OPT)}{P} + n) + P \cdot n \\ &= l(OPT) + 2 \cdot P \cdot n = |OPT| + 2 \cdot \varepsilon \cdot l_{uv}. \end{aligned}$$

Recall that e_{uv} is the last edge added to $\mathbf{G}$ such that the absence of e_{uv} would result in $\mathbf{G}$ having no negative cost cycles. This means that any negative cost cycle in $\mathbf{G}$ must include an edge of length at least l_{uv}. Hence, the length of any negative cost cycle in $\mathbf{G}$ must be at least l_{uv}. Therefore, $l_{uv} \leq |OPT|$. Thus, we have $|OPT| + 2 \cdot \varepsilon \cdot l_{uv} \leq |OPT| + 2 \cdot \varepsilon \cdot |OPT| =$

$(1+2\cdot\varepsilon)\cdot|OPT|$. Therefore, we can conclude that $|OPT'| \leq (1+2\cdot\varepsilon)\cdot|OPT|$.

We now analyze the running time of Algorithm 9.2.8. As previously stated, Line 1 takes $O(n^5)$ time. The **for** loop at line 4 takes $O(m) = O(n^2)$ time. For line 7, recall that Algorithm 9.2.6 takes $O(n^4 \cdot L)$ time, where L is the length of the largest edge length. However, in this case, the pseudo-polynomial time algorithms takes $O(n^4 \cdot L')$ time, where $L' = \left\lceil \frac{L}{P} \right\rceil$. Hence, the total running time is $O(n^4 \cdot L')$.

We distinguish two cases. In the first case, $\frac{L}{P} < 1$. This means that $O\left(n^4 \cdot \left\lceil \frac{L}{P} \right\rceil\right) = O\left(n^4\right)$. In the second case, $\frac{L}{P} \geq 1$. This implies that $\left\lceil \frac{L}{P} \right\rceil \leq \left\lfloor \frac{L}{P} \right\rfloor + 1 \leq \frac{L}{P} + 1 \leq 2 \cdot \frac{L}{P}$. Therefore,

$$\begin{aligned} O\left(n^4 \cdot \left\lceil \tfrac{L}{P} \right\rceil\right) &\leq O\left(n^4 \cdot \tfrac{L}{P}\right) = O\left(n^4 \cdot \tfrac{n \cdot L}{\varepsilon \cdot l_{uv}}\right) = O\left(n^4 \cdot n \tfrac{L}{\varepsilon \cdot l_{uv}}\right) \\ &\leq O\left(n^5 \cdot \tfrac{n \cdot l_{uv}}{\varepsilon \cdot l_{uv}}\right) = O\left(n^6 \cdot \tfrac{1}{\varepsilon}\right) \end{aligned}$$

Observe that if $\varepsilon < 1$, then $O(n^6 \cdot \frac{1}{\varepsilon})$ dominates the running time of our algorithm. Since the running time is polynomial in both $(1/\varepsilon)$ and the size of the input instance, the above algorithm is an FPTAS.

Chapter 10

UTVPI Constraint Systems

10.1 Motivation and Related Work

Unit Two Variable Per Inequality (UTVPI) constraints arise in a number of problem domains, including but not limited to, program verification [LM05], abstract interpretation [Min06, CC77], real-time scheduling [GPS95a] and operations research.

The focus of this chapter is on proofs of linear infeasibility in UCSs. Such proofs are very important from the perspective of designing certifying algorithms. In a certifying algorithm, both positive and negative answers must be accompanied by "certificates" which attest to the validity of the answer. For general linear programs, strong duality (Farkas' lemma) enables us to derive proofs of infeasibility.

Proofs of infeasibility are also referred to as refutations. There exist a number of refutation types, depending upon how the input constraints can be used in the construction of a proof of infeasibility. Our focus is on a class of refutations called resolution refutations. In resolution refutations, there is only one inference rule, viz., the transitive inference rule. The three major types of (resolution) refutations are **read-once**, **tree-like** and **dag-like** [Iwa97, IM95]. As already established in the previous section, read-once proofs are not **complete** for the purpose of refuting linear feasibility in UCSs. However, both tree-like

and dag-like proof systems are complete. We will focus exclusively on tree-like refutations.

Optimal length proofs (refutations) of various types and for various constraint systems have been studied extensively in the literature. In [Sub04b], optimal-length tree-like proofs were studied for 2CNF formulae. In [Sub09], it was established that read-once, tree-like, and dag-like proofs coincide for difference constraints systems.

On the read-once refutation side, [IM95] showed that the problem of checking if an arbitrary CNF has an ROR is **NP-complete**. This result was strengthened in [KZ02], where it was shown that the problem of checking whether a CNF formula has a read-once unit resolution refutation is **NP-complete**. In [Sze01], it was shown that the problem of finding literal-once resolution refutations for CNF formulas is **NP-complete**. The problem of finding read-once refutations in 2CNF formulas is discussed in [KWS18]; this problem was shown to be **NP-complete**. In this chapter, we examine the read-once refutation and literal-once refutation problems on continuous (as opposed to discrete) variables.

As we can see, much of the work in finding short refutations focused on discrete domains (CNF formulas). In a departure from existing work, [Sub09] considered difference constraint systems from the perspective of determining the optimal length resolution refutations. That paper showed that short refutations exist for difference constraints and also that the optimal length refutation can be determined in polynomial time. The algorithm therein is based on dynamic programming and runs in time $O(n^3 \cdot \log n)$ on a DCS with n variables. In [SWG13], a different dynamic program was used to achieve a time of $O(m \cdot n \cdot k)$, where m is the number of constraints and k is the length of the shortest refutation. It is worth noting that in DCSs, linear and integer feasibility coincide and therefore, the departure is not strict. Furthermore, as pointed out in [Sub09], every minimal refutation (i.e., a refutation without redundant constraints) is necessarily read-once and literal-once, since every minimal refutation corresponds to a simple negative cost cycle in the corresponding constraint network [CLRS01]. In this paper though, we consider the problem of read-once refutations in UCSs. Unlike DCSs, linear feasibility does not imply integer feasibility in UCSs [SW17b]. UTVPI constraints occur in a number of problem domains including but not

limited to program verification [LM05], abstract interpretation [Min06, CC77], real-time scheduling [GPS95a] and operations research [HN94].

10.2 Refutability

10.2.1 The OLTR problem (ADD rule)

In this section, we discuss the problem of finding the shortest tree-like refutation of a system of UTVPI constraints.

10.2.1.1 A dynamic programming approach

In this section, we design a dynamic programming algorithm for the OTLR problem. Let $\mathbf{U}$ be a system of UTVPI constraints, and let $\mathbf{G}$ be the corresponding constraint network constructed as described in Section 2.2.2. We have that every negative cost gray cycle in $\mathbf{G}$ corresponds to a tree-like refutation of $\mathbf{U}$ of the same length. Thus, a negative cost gray cycle with the fewest edges in $\mathbf{G}$ corresponds to an OTLR of $\mathbf{U}$.

The algorithm (Algorithm 10.2.1) utilizes a variant of matrix multiplication to construct matrices, $\mathbf{D}^{(k)}$, which store the costs of the shortest k-path from each node to itself. This is done as follows:

1. Let $\mathbf{D}^{(k)}$ be an $n \times n$ matrix of shortest paths of length at most k. Each element of $\mathbf{D}^{(k)}$ has four values ($d_{ij}^{(k,\square)}$, $d_{ij}^{(k,\blacksquare)}$, $d_{ij}^{(k,\blacksquarel)}$, and $d_{ij}^{(k,\blacksquarer)}$) where each value represents the cost of the shortest k-path of the corresponding type from node x_i to node x_j.

2. Let $\mathbf{D}^{(1)}$ be the distance matrix corresponding to $\mathbf{G}$.

3. Using the edge reductions described in [SW17b], we can compute $\mathbf{D}^{(k+l)} = \mathbf{D}^{(k)} \cdot \mathbf{D}^{(l)}$

as follows,

$$
\begin{aligned}
d_{ij}^{(k+l,\square)} &= \min \begin{cases} d_{ir}^{(k,\square)} + d_{rj}^{(l,\blacksquare)}, & r = 1 \ldots n \\ d_{ir}^{(k,\blacksquare)} + d_{rj}^{(l,\square)}, & r = 1 \ldots n \end{cases} \\
d_{ij}^{(k+l,\blacksquare)} &= \min \begin{cases} d_{ir}^{(k,\blacksquare)} + d_{rj}^{(l,\blacksquare)}, & r = 1 \ldots n \\ d_{ir}^{(k,\blacksquare)} + d_{rj}^{(l,\square)}, & r = 1 \ldots n \end{cases} \\
d_{ij}^{(k+l,\blacksquare)} &= \min \begin{cases} d_{ir}^{(k,\blacksquare)} + d_{rj}^{(l,\blacksquare)}, & r = 1 \ldots n \\ d_{ir}^{(k,\blacksquare)} + d_{rj}^{(l,\blacksquare)}, & r = 1 \ldots n \end{cases} \\
d_{ij}^{(k+l,\blacksquare)} &= \min \begin{cases} d_{ir}^{(k,\blacksquare)} + d_{rj}^{(l,\blacksquare)}, & r = 1 \ldots n \\ d_{ir}^{(k,\square)} + d_{rj}^{(l,\blacksquare)}, & r = 1 \ldots n \end{cases}
\end{aligned}
\tag{10.1}
$$

4. $\mathbf{G}$ has a negative cost gray cycle of length k or less if and only if $d_{ii}^{(k,\blacksquare)} < 0$ for any $i = 1 \ldots n$.

Algorithm 10.2.1 Dynamic Programming Algorithm for UTVPI Constraints

Function OTLR-DYNAMIC-PROGRAM($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$)

1: **for** ($k = 2$ **to** $(2 \cdot n + 2)$) **do**
2: Compute $\mathbf{D}^{(k)}$ from $\mathbf{D}^{(k-1)} \cdot \mathbf{D}^{(1)}$ as described by System (10.1).
3: **for** ($i = 1$ **to** n) **do**
4: **if** ($d_{ii}^{(k,\blacksquare)} < 0$) **then**
5: **return** (k).
6: **return** (-1). ▷ No negative cost gray cycle was detected.

10.2.1.1.1 Resource analysis

We now analyze the running time of Algorithm 10.2.1. Line 2 takes $(c_1 \cdot n^3)$ time, where c_1 is some constant. This is because it takes $(c_1 \cdot n^3)$ time to compute $\mathbf{D}^{(k)} = \mathbf{D}^{(k-1)} \cdot \mathbf{D}^{(1)}$.

Checking if $d_{ii}^{(k,\blacksquare)} < 0$ takes constant time. This means that the **for** loop at line 3 takes $(c_2 \cdot n)$ time, where c_2 is some constant.

We now need to address the **for** loop at line 1. Since the loop iterates up to $2n+2$ times, it would appear that we have $(c_3 \cdot n)$ iterations, where c_3 is some constant. However, observe that the algorithm terminates at line 5 as soon as we find the OTLR. Since k is the value of the OTLR, the **for** loop at line 1 actually has $(c_3 \cdot k)$ iterations.

Therefore, the running time of Algorithm 10.2.1, denoted as $T(n)$, is

$$\begin{aligned} T(n) &\leq (c_3 \cdot k) \cdot ((c_1 \cdot n^3) + (c_2 \cdot n)) \\ &= (c_1 \cdot c_3) \cdot n^3 \cdot k + (c_2 \cdot c_3) \cdot n \cdot k \\ &= O(n^3 \cdot k). \end{aligned}$$

We now analyze the space requirements for our algorithm. The key observation is that for each $l < k$, once we compute $\mathbf{D}^{(l+1)}$, we no longer need $\mathbf{D}^{(l)}$. This means that we only need to keep $\mathbf{D}^{(l+1)}$ and $\mathbf{D}^{(1)}$. Therefore, Algorithm 10.2.1 requires $O(n^2)$ space.

10.2.1.1.2 Correctness

We now prove the correctness of Algorithm 10.2.1.

***Theorem* 10.2.1.** *Algorithm 10.2.1 always returns either an OTLR of a UCS* **U** *corresponding to the input network* **G** *or* -1 *if* **U** *is feasible.*

Proof. Suppose Algorithm 10.2.1 returns some value $k \neq -1$. This means that there exists some node $x_i \in \mathbf{V}$ such that $d_{ii}^{(k,\blacksquare)} < 0$. This also means that x_i is part of a negative cost gray cycle of length k. This means that **U** has a tree-like refutation of length k.

We also know that for all nodes $x_j \in \mathbf{V}$, $d_{jj}^{(l,\blacksquare)} \geq 0$ for all $0 < l < k$. Otherwise, there would exist a different negative cost gray cycle with a length smaller than k. This means that **U** has no tree-like refutations of length less than k. Therefore, k is an OTLR of **U**.

Suppose Algorithm 10.2.1 returns -1. This happens only when $d_{ii}^{(k,\blacksquare)} \geq 0$ for all $x_i \in \mathbf{V}$ and all $k \leq n$. In this case, there are no negative cost gray cycles in **G**. From [SW17b], this implies that **U** is feasible. □

10.2.1.1.3 Improved dynamic programming algorithm

We now discuss how we can improve the running time of Algorithm 10.2.1. Instead of computing each matrix $\mathbf{D}^{(1)}$ through $\mathbf{D}^{(k)}$, we can repeatedly square the matrices. In this case, we would compute $\mathbf{D}^{(1)}$, $\mathbf{D}^{(2)}$, $\mathbf{D}^{(4)}$, ... We continue using repeated squaring until we compute a matrix that indicates the presence of a negative cost gray cycle. Let $\mathbf{D}^{(h)}$ be this matrix. Note that h is not necessarily the OTLR. However, we do know that the OTLR is between $\frac{h}{2}$ and h. Thus, we can utilize the matrices constructed during the repeated squaring procedure to find the length of the OTLR by performing a binary search.

Instead of constructing each of the matrices $\mathbf{D}^{(1)}$ through $\mathbf{D}^{(k)}$ we can instead repeatedly square the matrix (constructing the matrices $\mathbf{D}^{(1)}$, $\mathbf{D}^{(2)}$, $\mathbf{D}^{(4)}$, ...) to reduce the number of matrix multiplications required. This process of repeated squaring continues until a matrix indicating the existence of negative cost gray cycle is found. Let $\mathbf{D}^{(h)}$ be this matrix. Note that h is not the length of the OTLR, however the length of the OTLR is between $\frac{h}{2}$ and h. Thus, we can utilize the matrices constructed during the repeated squaring procedure to find the length of the OTLR by performing a binary search on the interval $\left[\frac{h}{2}+1, h\right]$.

To perform this binary search we first construct the matrix $\mathbf{D}^{\left(\frac{3}{4}\cdot h\right)}$ from $\mathbf{D}^{\left(\frac{1}{2}\cdot h\right)}$ and $\mathbf{D}^{\left(\frac{1}{4}\cdot h\right)}$. If this matrix indicates the existence of a negative cost gray cycle, then we repeat this process on the interval $[\frac{1}{2}\cdot h+1, \frac{3}{4}\cdot h]$. Otherwise, we repeat on the interval $[\frac{3}{4}\cdot h+1, h]$. When we find a k such that the matrix $\mathbf{D}^{(k)}$ indicates the existence of a negative cost gray cycle but the matrix $\mathbf{D}^{(k-1)}$ does not, then we know that k is the OTLR.

This approach is described in Algorithm 10.2.2.

Note that Algorithm 10.2.2 needs to store the intermediary distance matrices so that they do not need to be recomputed during the binary search procedure. This increases the space needed from $O(n^2)$ to $O(n^2 \cdot \log k)$. However, we now perform only $O(\log k)$ matrix multiplications instead of $O(k)$ matrix multiplications. Therefore, the running time of Algorithm 10.2.2 is $O(n^3 \cdot \log k)$.

Algorithm 10.2.2 Dynamic Programming Algorithm for UTVPI Constraints

Function IMPROVED-OTLR-DYNAMIC-PROGRAM($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$)

1: **for** ($k = 1$ **to** $\lceil \log(2 \cdot n + 2) \rceil$) **do**
2: Compute $\mathbf{D}^{(2^k)}$ from $\mathbf{D}^{(2^{k-1})} \cdot \mathbf{D}^{(2^{k-1})}$ as described by System (10.1).
3: **for** ($i = 1$ **to** n) **do**
4: **if** ($d_{ii}^{(2^k, \blacksquare)} < 0$) **then**
5: $h \leftarrow 2^k$.
6: $l \leftarrow 2^{k-1}$.
7: Break from **for** loop on line 1.
8: **if** (a negative cost gray cycle was detected) **then**
9: **for** ($k = (\log l) - 1$ **to** 0) **do**
10: Compute $\mathbf{D}^{(l+2^k)}$ from $\mathbf{D}^{(2^k)} \cdot \mathbf{D}^{(l)}$ as described by System (10.1).
11: **if** (for any $i = 1 \ldots n$, $d_{ii}^{(l+2^k, \blacksquare)} < 0$) **then**
12: $h \leftarrow l + 2^k$.
13: **else**
14: $l \leftarrow l + 2^k$.
15: **return** (h).
16: **else**
17: **return** (-1).

10.2.1.2 A path following approach

In this section, we exploit the observations in Section 2.2.2 to design a simple, path following algorithm for the OTLR problem. A negative cost gray cycle in $\mathbf{G}$ corresponds to a tree-like refutation of the original UCS. Thus, the shortest such cycle in $\mathbf{G}$ corresponds to an OTLR of the original UCS. The following observations result in Algorithms 10.2.3 and 10.2.4.

1. Let $d_i^{(k,t)}(j)$ denote the length of the shortest path of type t from node x_i to node x_j with at most k edges.

2. Let $d_i^{(k)}()$ contain $d_i^{(k,t)}(j)$ for all $j = 1 \ldots n$ and $t \in \{ \square , \blacksquare , \blacksquare . \blacksquare \}$.

3. We initially set $d_i^{(0,t)}(i) = 0$ and $d_i^{(0,t)}(j) = \infty$ for each $t \in \{ \square , \blacksquare , \blacksquare , \blacksquare \}$ and $j \neq i$.

4. From [SW17b], we have that,

$$
\begin{aligned}
d_i^{(k+1,\,\square)}(j) &= \min \begin{cases} d_i^{(k,\,\square)}(r) + c(x_r \,\blacksquare\!\square\, x_j), & r \text{ is a neighbor of } j \\ d_i^{(k,\,\blacksquare)}(r) + c(x_r \,\square\, x_j), & r \text{ is a neighbor of } j \end{cases} \\
d_i^{(k+1,\,\blacksquare)}(j) &= \min \begin{cases} d_i^{(k,\,\blacksquare)}(r) + c(x_r \,\blacksquare\!\square\, x_j), & r \text{ is a neighbor of } j \\ d_i^{(k,\,\blacksquare)}(r) + c(x_r \,\square\, x_j), & r \text{ is a neighbor of } j \end{cases} \\
d_i^{(k+1,\,\blacksquare)}(j) &= \min \begin{cases} d_i^{(k,\,\blacksquare)}(r) + c(x_r \,\square\!\blacksquare\, x_j), & r \text{ is a neighbor of } j \\ d_i^{(k,\,\blacksquare)}(r) + c(x_r \,\blacksquare\, x_j), & r \text{ is a neighbor of } j \end{cases} \\
d_i^{(k+1,\,\square)}(j) &= \min \begin{cases} d_i^{(k,\,\square)}(r) + c(x_r \,\square\!\blacksquare\, x_j), & r \text{ is a neighbor of } j \\ d_i^{(k,\,\square)}(r) + c(x_r \,\blacksquare\, x_j), & r \text{ is a neighbor of } j \end{cases}
\end{aligned}
\tag{10.2}
$$

5. If we have a negative cost gray cycle of length k centered around an arbitrary node x_i, then $d_i^{(k,\,\blacksquare)}(i) < 0$. Observe that any path which reduces to the edge $(x_i \overset{b_i}{\blacksquare\!\square} x_i)$ also reduces to the edge $(x_i \overset{b_i}{\square\!\blacksquare} x_i)$ since these are the same edge. This means that $d_i^{(k,\,\blacksquare)}(i) = d_i^{(k,\,\square)}(i)$. Thus, it is only necessary to check one of these values.

10.2.1.2.1 Resource analysis

We first analyze the running time of Algorithm 10.2.3.

***Lemma* 10.2.1.** *Given $d_i^{(k-1)}()$, Algorithm 10.2.3 computes $d_i^{(k)}()$ in $O(m')$ time, where m' is the number of edges in the constraint network.*

Proof. Observe that Algorithm 10.2.3 implements the recurrence relation defined by System (10.2). Computing $d_i^{(k,t)}$ is accomplished by relaxing all of the edges in $\mathbf{G}$. Relaxing an edge takes $O(1)$ time, and $\mathbf{G}$ has m' edges. Therefore, the running time of Algorithm 10.2.3 is $O(m')$. □

We now analyze the running time of Algorithm 10.2.4. Let $T(n,m')$ denote the running time of Algorithm 10.2.4 on a network with $(n+1)$ nodes and m' edges. Also let $q \in O(1)$

Algorithm 10.2.3 Shortest Path Computation for UTVPI Constraints

Function SHORTEST-PATH-UTVPI($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle, d_i^{(k)}(), d_i^{(k-1)}()$)

1: **for** ($j = 0$ to n) **do**
2: **for** ($t \in \{ \square , \blacksquare , \blacksquarel , \blacksquarer \}$) **do**
3: $d_i^{(k,t)}(j) \leftarrow \infty$.
4: **for** (each edge $(x_r \blacksquarer x_j) \in \mathbf{E}$) **do**
5: $d_i^{(k,\square)}(j) \leftarrow \min\{d_i^{(k,\square)}(j), d_i^{(k-1,\square)}(r) + c(x_r \blacksquarer x_j)\}$
6: $d_i^{(k,\blacksquarer)}(j) \leftarrow \min\{d_i^{(k,\blacksquarer)}(j), d_i^{(k-1,\blacksquarer)}(r) + c(x_r \blacksquarer x_j)\}$
7: **for** (each edge $(x_r \square x_j) \in \mathbf{E}$) **do**
8: $d_i^{(k,\square)}(j) \leftarrow \min\{d_i^{(k,\square)}(j), d_i^{(k-1,\blacksquarel)}(r) + c(x_r \square x_j)\}$
9: $d_i^{(k,\blacksquarer)}(j) \leftarrow \min\{d_i^{(k,\blacksquarer)}(j), d_i^{(k-1,\blacksquare)}(r) + c(x_r \square x_j)\}$
10: **for** (each edge $(x_r \blacksquarel x_j) \in \mathbf{E}$) **do**
11: $d_i^{(k,\blacksquare)}(j) \leftarrow \min\{d_i^{(k,\blacksquare)}(j), d_i^{(k-1,\blacksquare)}(r) + c(x_r \blacksquarel x_j)\}$
12: $d_i^{(k,\blacksquarel)}(j) \leftarrow \min\{d_i^{(k,\blacksquarel)}(j), d_i^{(k-1,\blacksquarel)}(r) + c(x_r \blacksquarel x_j)\}$
13: **for** (each edge $x_r \blacksquare x_j \in \mathbf{E}$) **do**
14: $d_i^{(k,\blacksquare)}(j) \leftarrow \min\{d_i^{(k,\blacksquare)}(j), d_i^{(k-1,\blacksquarer)}(r) + c(x_r \blacksquare x_j)\}$
15: $d_i^{(k,\blacksquarel)}(j) \leftarrow \min\{d_i^{(k,\blacksquarel)}(j), d_i^{(k-1,\square)}(r) + c(x_r \blacksquare x_j)\}$

Algorithm 10.2.4 Deterministic Algorithm for UTVPI Constraints

Function SHORTEST-NEGATIVE-GRAY-CYCLE($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$)

1: **for** ($k = 1$ **to** $(2 \cdot n + 2)$) **do**
2: **for** ($i = 0$ **to** n) **do**
3: SHORTEST-PATH-UTVPI($\mathbf{G}, d_i^{(k)}(), d_i^{(k-1)}()$).
4: **if** ($d_i^{(k,\blacksquarer)}(i) < 0$) **then**
5: **return** (k).
6: **return** (-1). ▷ No negative cost gray cycle was detected.

denote the time for a single edge relaxation. Observe that the **for** loop in lines 2 to 5 has $O(n)$ iterations. From Lemma 10.2.1, we know that line 3 takes $O(m')$ time. Therefore, the total running time is:

$$\begin{aligned} T(n,m') &\leq \sum_{i=1}^{k}\sum_{j=0}^{n} q \cdot m' \\ &= q \cdot m' \cdot (n+1) \cdot k \\ &\in O(m' \cdot n \cdot k) \\ &= O(m \cdot n \cdot k) \end{aligned}$$

10.2.1.2.2 Correctness

We now prove the correctness of Algorithm 10.2.4. First we need the following lemma from [SW17b].

***Lemma* 10.2.2.** *If the UCS* **U** *is infeasible, then the corresponding constraint network* **G** *has a negative cost gray cycle with at most* $(2 \cdot n + 2)$ *edges.*

The proof of this lemma is found in [SW17b].

***Theorem* 10.2.2.** *Algorithm 10.2.4 always returns an OTLR of the UCS* **U** *corresponding to the input network* **G** *or* -1 *if* **U** *is feasible.*

Proof. We first address the correctness of Algorithm 10.2.3. As stated in Lemma 10.2.1, the algorithm is an implementation of System (10.2). From [SW17b], System (10.2) correctly calculates $d_i^{(k+1)}()$ from $d_i^{(k)}()$.

If Algorithm 10.2.4 returns $k \neq -1$, then for some node x_i, $d_i^{(k,\blacksquare)}(i) < 0$. This means that x_i is located on a negative cost gray cycle of length k. This means that **U** has a tree-like refutation of length k.

We also know that for all nodes x_j, $d_j^{(l,\blacksquare)}(j) \geq 0$ for all $0 < l < k$. Thus, **G** has no negative cost gray cycles of length less than k. This means that **U** has no tree-like refutation of length less than k. Thus, k is the length of an OTLR of **U**.

If Algorithm 10.2.4 returns -1, then for all nodes x_j, $d_j^{(l,\blacksquare)}(j) \geq 0$ for all $0 < l \leq (2 \cdot n + 2)$. Thus, $\mathbf{G}$ has no negative cost gray cycles with $(2 \cdot n + 2)$ or fewer edges. By Lemma 10.2.2, $\mathbf{U}$ must be feasible. □

10.2.1.3 A randomized approach

In this section, we propose a randomized algorithm for an OTLR problem in UCSs. This algorithm is a generalization of the randomized algorithm for finding shortest negative cost cycles in a directed graph [OSW18].

In each iteration, the algorithm (Algorithm 10.2.5) processes a randomly chosen node. Let v_r denote the node chosen by the r^{th} iteration of this process. The algorithm proceeds by generating the shortest paths from v_r to every node in the network $\mathbf{G}$ having at most $\lceil \frac{2 \cdot n+2}{r} \rceil$ edges (see Lemma 10.2.2).

Algorithm 10.2.5 represents our strategy to find the shortest negative cost gray cycle in a UTVPI constraint network with arbitrarily costed edges.

Algorithm 10.2.5 Randomized Algorithm for UTVPI Constraints

Function SHORTEST-NEGATIVE-GRAY-CYCLE($\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$)

1: **if** ($\mathbf{G}$ has a negative cost gray cycle) **then**
2: Let l be the number of edges in the cycle.
3: **else**
4: **return** (-1).
5: **for** $(r = 1$ **to** $(n+1))$ **do**
6: Let x_i be a node in $\mathbf{V}$ chosen uniformly and at random.
7: **for** $(k = 1$ **to** $\lceil \frac{2 \cdot n+2}{r} \rceil)$ **do**
8: SHORTEST-PATH-UTVPI($\mathbf{G}, d_i^{(k)}(), d_i^{(k-1)}()$).
9: **if** $(d_i^{(k,\blacksquare)}(i) < 0)$ **and** $(k < l)$ **then**
10: $l \leftarrow k$.
11: **return** (l).

Resource Analysis: Let $T(n, m')$ denote the running time of Algorithm 10.2.5 on a network with $(n+1)$ nodes and m' edges. Also let $q_1 \in O(1)$ denote the amount of time taken by a single edge relaxation. The check on line 1 can be accomplished in $O(m' \cdot n)$

time [SW17b]. Indeed, some negative cost gray cycle (not necessarily the one having the fewest number of edges) is returned by the linear feasibility algorithm in [SW17b]. Let $q_2 \cdot m' \cdot (n+1)$ be the amount of time taken by this process. The **for** loop on lines 5 has $O(n)$ iterations. From Lemma 10.2.1, it follows that each iteration of the **for** loop on line 7 takes $O(m')$ time. Thus, we have that:

$$\begin{aligned}
T(n,m') &\leq q_2 \cdot m' \cdot (n+1) + \sum_{r=1}^{n+1} \sum_{k=1}^{\lceil \frac{2\cdot n+2}{r} \rceil} q_1 \cdot m' \\
&= q_2 \cdot m' \cdot (n+1) + q_1 \cdot m' \cdot \left(\sum_{r=1}^{n+1} \sum_{k=1}^{\lceil \frac{2\cdot n+2}{r} \rceil} 1 \right) \\
&= q_2 \cdot m' \cdot (n+1) + q_1 \cdot m' \cdot \left(\sum_{r=1}^{n+1} \left\lceil \frac{2\cdot n+2}{r} \right\rceil \right) \\
&\leq q_2 \cdot m' \cdot (n+1) + q_1 \cdot m' \cdot 2 \cdot (n+1) \cdot (H_{n+1}) \\
&\in O(m' \cdot n \cdot \log n) \\
&= O(m \cdot n \cdot \log n)
\end{aligned}$$

where H_n is the n^{th} harmonic number.

10.2.1.3.1 Correctness

We now establish that Algorithm 10.2.5 returns an OTLR of **G** with high probability.

***Theorem* 10.2.3.** *Algorithm 10.2.5 returns the OTLR with probability at least* $(1 - \frac{1}{e})$.

Proof. If **G** has no negative cost gray cycles, then Algorithm 10.2.5 returns -1. Thus, Algorithm 10.2.5 always returns the correct answer in this case.

If **G** has a negative cost gray cycle, then let C denote an OTLR of **G**. Let N_C be the number of nodes in C and $|C|$ be the length of C. Note that $|C| \leq 2 \cdot N_C$ [SW17b].

Let us compute the probability that C is discovered during the r^{th} iteration of the **for** loop on line 3. Let x_i be the vertex chosen this iteration. If $r \leq \left\lceil \frac{n+1}{N_C} \right\rceil$, then $r \leq \left\lceil \frac{2\cdot n+2}{2\cdot N_C} \right\rceil \leq \left\lceil \frac{2\cdot n+2}{|C|} \right\rceil$. This means that $|C| \leq \lceil \frac{2\cdot n+2}{r} \rceil$. Thus, C will be discovered if x_i lies on C. This has a probability of $\frac{N_C}{n+1}$.

Let E_r be the event that the r^{th} node processed by the algorithm is not a node of C. Note that these events are independent. We have that C not being discovered corresponds to the event $\bigcap_{r=1}^{n} E_r$ [MR95]. Thus, the probability that C is not discovered by Algorithm 10.2.5 is

$$\begin{aligned} P\left(\bigcap_{r=1}^{n} E_r\right) &\leq P\left(\bigcap_{r=1}^{\left\lceil \frac{n+1}{N_C} \right\rceil} E_r\right) \\ &= \prod_{r=1}^{\left\lceil \frac{n+1}{N_C} \right\rceil} P(E_r) \\ &\leq \left(1 - \frac{N_C}{n+1}\right)^{\left\lceil \frac{n+1}{N_C} \right\rceil} \\ &\leq \frac{1}{e} \end{aligned}$$

It follows that Algorithm 10.2.5 succeeds with probability at least $(1 - \frac{1}{e}) = 0.632$. □

Clearly, we can boost the probability of success by running the same procedure multiple times. Indeed, the expected number of runs before finding an OTLR is 2.

10.2.2 The WOLTR problem (ADD rule)

In this section, we introduce the weighted optimal length tree-like refutation problem (WOTLR). This problem is concerned with weighted UTVPI constraints, which are defined below:

***Definition* 10.2.1.** *A UTVPI constraint of the form $a_i \cdot x_i + a_j \cdot x_j \leq b_{ij}$ and $a_i, a_j \in \{1, -1\}$ is called a weighted UTVPI constraint if there exists a positive, integral weight $l_{ij} > 0$, where $\mathbf{l} : \mathbf{E} \rightarrow \mathbb{N}$ is the weight function.*

Note that if $\mathbf{E}' \subseteq \mathbf{E}$ is a set of edges in $\mathbf{G}$, then $l(\mathbf{E}) = \sum_{e_{ij}} \in \mathbf{E} l_{ij}$. A conjunction of weighted UTVPI constraints is called a weighted UTVPI constraint system (WUCS).

Constructing the corresponding constraint network of a WUCS is similar to constructing the constraint network of a UCS. The key difference is we assign each edge $e_{ij} \in \mathbf{V}$

a length l_{ij}. Note that the term "length" is used for the constraint network rather than "weight." This is because the term "weight" is commonly used in networks to define the cost of an edge. Thus, to prevent confusion, we use the term "length" for the constraint network.

As previously described in [SW17b], if a UCS is unsatisfiable, then there must exist a negative cost gray cycle in the corresponding constraint network. Clearly, this observation holds for a WUCS. Therefore, the tree-like refutation with the smallest total weight corresponds to the negative cost gray cycle with the smallest total length. We call the length of such a negative cost gray cycle the *weighted optimal length tree-like refutation* (WOTLR).

Using the terminology above, we define the WOTLR problem as follows:

Given a WUCS $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$*, where the weight of each constraint is a positive integer, find the weight of a tree-like refutation having the smallest total weight.*

Alternatively, we define the WOTLR problem as follows:

Given a network $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c}, \mathbf{l} \rangle$ *with real edge costs and positive integral edge lengths, find the length of the negative cost gray cycle having the smallest total length.*

Note that every difference constraint is a UTVPI constraint. Thus, the WOLTR problem for UTVPI constraints is a more general version of the WOLTR problem for difference constraints. The WOLTR problem for difference constraints is **NP-hard** [Sub09]. Thus, so is the WOLTR problem for UTVPI constraints.

10.2.2.1 A pseudo-polynomial time algorithm

In this section, we present a pseudo-polynomial time algorithm for computing the WOTLR in a WUCS. This algorithm is a key subroutine for our FPTAS. Consider the UTVPI constraint network $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c}, \mathbf{l} \rangle$. We apply the pseudo-polynomial time algorithm in [GRKL01]. This algorithm computes the shortest path from a source node v_1 to all other nodes $v_{\in}\mathbf{V}$ for networks with positive integral edge costs and having a transition time at most T. Note that [GRKL01] uses the term "length" to define the cost of an edge e_{ij}, while we use the term "cost." Additionally, [GRKL01] uses the term "delay" (denoted

as t_{ij}), while we use the term "length" (denoted as l_{ij}) to define the same property.

Assume the vertices are enumerated from 1 to n, where v_1 denotes the source node. Let $L(j,t)$ denote the cost of the shortest path from node v_1 to node v_j with length at most t. For each $j = 1 \ldots n$ and $T = 0 \ldots T$, we compute $L(j,t)$ using the dynamic program from [GRKL01]:

$$L(j,t) = \begin{cases} 0 & j = 1 \quad t \geq 0 \\ \infty & j \geq 2 \quad t = 0 \\ \min\left\{ L(j,t-1), \min\limits_{k \mid t_{kj} \leq t, e_{kj} \in \mathbf{E}} \{L(k,t-t_{kj}) + b_{kj}\} \right\} & j \geq 2 \quad t \geq 1 \end{cases}$$

We modify the dynamic program for networks with real edge costs. We use the notation $D(j,t)$ rather than $L(j,t)$. This is to differentiate our modified dynamic program from the dynamic program in [GRKL01]. We apply the dynamic program for each edge type in the WUCS. Let $D^{(\square)}(j,t)$, $D^{(\blacksquare)}(j,t)$, $D^{(\blacksquarel)}(j,t)$, and $D^{(\blacksquarer)}(j,t)$ denote the shortest path from node x_1 to x_j with length at most t for the respective edge types. We initialize $D^{(\square)}(j,t)$, $D^{(\blacksquare)}(j,t)$, $D^{(\blacksquarel)}(j,t)$, and $D^{(\blacksquarer)}(j,t)$ to 0 when $j = 1$. We let x_r, instead of x_k, denote the neighboring node of x_j. This is because k is already used to denote the value of the WOTLR. Instead of computing $\min\left\{L(j,t-1), \min_{k \mid t_{kj} \leq t, e_{kj} \in \mathbf{E}} \{L(k,t-t_{kj}) + b_{kj}\}\right\}$, we apply the dynamic programs from System (10.2) in Section 10.2.1.2. We formally define our dynamic programs below.

$$D^{(\,\square\,)}(j,t) = \begin{cases} 0 & j=1 \quad t=0 \\ \infty & j \geq 2 \quad t=0 \\ \min \begin{cases} D^{(\,\square\,)}(j,t-1) \\ \min_{r|l_{rj}\leq t} \begin{cases} D^{(\,\square\,)}(r,t-l_{rj}) + c(v_r \,\blacksquare\, v_j) \\ D^{(\,\blacksquare\,)}(r,t-l_{rj}) + c(v_r \,\square\, v_j) \end{cases} \end{cases} & j \geq 1 \quad t \geq 1 \end{cases}$$

$$D^{(\,\blacksquare\,)}(j,t) = \begin{cases} 0 & j=1 \quad t=0 \\ \infty & j \geq 2 \quad t=0 \\ \min \begin{cases} D^{(\,\blacksquare\,)}(j,t-1) \\ \min_{r|l_{rj}\leq t} \begin{cases} D^{(\,\blacksquare\,)}(r,t-l_{rj}) + c(v_r \,\blacksquare\, v_j) \\ D^{(\,\blacksquare\,)}(r,t-l_{rj}) + c(v_r \,\square\, v_j) \end{cases} \end{cases} & j \geq 1 \quad t \geq 1 \end{cases}$$

$$D^{(\,\blacksquare\,)}(j,t) = \begin{cases} 0 & j=1 \quad t=0 \\ \infty & j \geq 2 \quad t=0 \\ \min \begin{cases} D^{(\,\blacksquare\,)}(j,t-1) \\ \min_{r|l_{rj}\leq t} \begin{cases} D^{(\,\blacksquare\,)}(r,t-l_{rj}) + c(v_r \,\blacksquare\, v_j) \\ D^{(\,\blacksquare\,)}(r,t-l_{rj}) + c(v_r \,\blacksquare\, v_j) \end{cases} \end{cases} & j \geq 1 \quad t \geq 1 \end{cases}$$

$$D^{(\,\blacksquare\,)}(j,t) = \begin{cases} 0 & j=1 \quad t=0 \\ \infty & j \geq 2 \quad t=0 \\ \min \begin{cases} D^{(\,\blacksquare\,)}(j,t-1) \\ \min_{r|l_{rj}\leq t} \begin{cases} D^{(\,\blacksquare\,)}(r,t-l_{rj}) + c(v_r \,\blacksquare\, v_j) \\ D^{(\,\square\,)}(r,t-l_{rj}) + c(v_r \,\blacksquare\, v_j) \end{cases} \end{cases} & j \geq 1 \quad t \geq 1 \end{cases}$$

After computing $D^{(type)}(j,t)$ for all $j \in \mathbf{V}$, for all $type \in \{\,\square\,,\,\blacksquare\,,\,\blacksquare\,,\,\blacksquare\,\}$, and a single value of t, we check if $D^{(\,\blacksquare\,)}(1,t) < 0$ (or $D^{(\,\blacksquare\,)}(1,t) < 0$). If this is true, then there exists

a negative cost gray cycle from node v_1 to itself with length t. Otherwise, we repeat the computation for $(t+1) \leq T$.

We apply the dynamic programs for all nodes. For each source node v_s, $D_s^{(type)}(j,t)$ is the shortest path from v_s to v_j with length t, where $type \in \{ \square , \blacksquare , \blacksquare , \blacksquare \}$. We compute $D_s^{(type}(j,t)$ for all values of v_s, v_j, $type$, and a single value of t. We then check if $D_s^{(\blacksquare)}(s,t) < 0$ (or $D_s^{(\blacksquare)}(s,t) < 0$) for any $v_s \in \mathbf{V}$. If this is true, then t is the WOTLR. Otherwise, we repeat the calculations for $(t+1) \leq T$. If L denotes the largest edge length of any edge in $\mathbf{G}$, then we set $T = (2 \cdot n + 2) \cdot L$, which is the largest possible length for any negative cost gray cycle.

The above observations are summarized in Algorithm 10.2.6 and Algorithm 10.2.7.

Algorithm 10.2.6 Single Node WOTLR Algorithm for UTVPI Constraints

Function SINGLE-NODE-WOTLR($\mathbf{G}, v_s, t$)

1: Enumerate the nodes such that v_s is v_1.
2: **for** $j = 1$ to n **do**
3: **for** ($type \in \{ \square , \blacksquare , \blacksquare , \blacksquare \}$) **do**
4: Compute $D_1^{(type)}(j,t)$ as described above.
5: **if** ($D_1^{(\blacksquare)}(j,t) < 0$) **then**
6: **return** (**True**).
7: **return** (**False**).

10.2.2.1.1 Resource analysis

From [GRKL01], we know that Algorithm 10.2.6 takes $O(m)$ time because we scan each edge exactly once.

We now analyze the running time of Algorithm 10.2.7. The **for** loop at line 4 has $(c_1 \cdot n)$ iterations, where c_1 is some constant. The **for** loop at line 5 takes $(c_2 \cdot T) = (c_2 \cdot (2 \cdot n + 2) \cdot L)$ time, where L is the largest length among all edges and c_2 is some constant. The **for** loop at line 8 takes $(c_3 \cdot n)$ time, where c_3 is some constant. Therefore, the running time of the **for** loop at line 4, which we denote as $T_1(n)$, is

Algorithm 10.2.7 Pseudo-Polynomial Time WOTLR Algorithm

Function PSEUDO-WOTLR(**G**)

1: $n = |\mathbf{V}|$.
2: $L = \max_{v_i, v_j \in \mathbf{V}}\{l_{ij}\}$.
3: $T = (2 \cdot n + 2) \cdot L$.
4: **for** (each node $v_s \in \mathbf{V}$) **do**
5: **for** ($t = 0$ to T) **do**
6: **for** ($type \in \{ \square , \blacksquare , \blacksquare , \blacksquare \}$) **do**
7: $D_s^{(type)}(s,t) = 0$.
8: **for** ($v_j \in \mathbf{V} - \{v_s\}$) **do**
9: $D_s^{(type)}(j,t) = \infty$.
10: **for** ($t = 1$ to T) **do**
11: **for** (each node $v_s \in \mathbf{V}$) **do**
12: SINGLE-NODE-WOTLR(**G**, v_s, t).
13: **if** (SINGLE-NODE-WOTLR returned **True**) **then**
14: **return** ("The WOTLR is t").

$$\begin{aligned} T_1(n) &\leq (c_1 \cdot n) \cdot (c_2 \cdot (2 \cdot n + 2) \cdot L) \cdot (c_3 \cdot n) \\ &= (c_1 \cdot n) \cdot (2 \cdot c_2 \cdot n \cdot L + 2 \cdot c_2 \cdot L) \cdot (c_3 \cdot n) \\ &= (2 \cdot c_1 \cdot c_2 \cdot c_3) \cdot n^3 \cdot L + (2 \cdot c_1 \cdot c_2 \cdot c_3) \cdot n^2 \cdot L \\ &= O(n^3 \cdot L). \end{aligned}$$

We will now analyze the **for** loop at line 14. Observe that line 16 takes $(c_4 \cdot m) = O(m)$ time, where c_4 is a constant, since this is Algorithm 10.2.6. The **for** loop at line 15 has $(c_5 \cdot n)$ iterations, and the **for** loop at line 14 has $(c_6 \cdot T) = (c_6 \cdot (2 \cdot n + 2) \cdot L)$ iterations, where c_5 and c_6 are constants. This means that the running of the **for** loop at line 14, which we denote as $T_2(m,n)$, is

$$\begin{aligned} T_2(m,n) &\leq (c_6 \cdot (2 \cdot n + 2) \cdot L) \cdot (c_5 \cdot n) \cdot (c_4 \cdot m) \\ &= (2 \cdot c_6 \cdot n \cdot L + 2 \cdot c_6 \cdot L) \cdot (c_5 \cdot n) \cdot (c_4 \cdot m) \\ &= (2 \cdot c_4 \cdot c_5 \cdot c_6) \cdot (n \cdot n \cdot m \cdot L) + (2 \cdot c_4 \cdot c_5 \cdot c_6) \cdot (n \cdot m \cdot L) \\ &= O(n^4 \cdot L). \end{aligned}$$

Therefore, the running time of Algorithm 10.2.7, denoted as $T(m,n)$, is

$$\begin{aligned} T(m,n) &\leq T_1(n) + T_2(m,n) \\ &\leq d_1 \cdot (n^3 \cdot L) + d_2 \cdot (n^4 \cdot L) \\ &\leq d_3 \cdot (n^4 \cdot L) \\ &= O(n^4 \cdot L), \end{aligned}$$

where d_1, d_2, and d_3 are constants.

10.2.2.1.2 Correctness

We now prove the correctness of our pseudo-polynomial time algorithm. We first address the correctness of the dynamic program in Algorithm 10.2.6. Consider the dynamic program $D_s^{(\,\square\,)}(j,t)$ for computing the shortest white path from v_s to v_j. Suppose $2 \leq j \leq n$ and $1 \leq t \leq T$. This means that we compute the dynamic program:

$$D_s^{(\,\square\,)}(j,t) = \min \begin{cases} D_s^{(\,\square\,)}(j,t-1) \\ \min_{r|l_{rj} \leq t} \begin{cases} D_s^{(\,\square\,)}(r,t-l_{rj}) + c(v_r \blacksquare v_j) \\ D_s^{(\,\blacksquare\,)}(r,t-l_{rj}) + c(v_r \square v_j) \end{cases} \end{cases}$$

Observe that this dynamic program uses the same logic as the dynamic program from [GRKL01]. We already know that [GRKL01] correctly computes shortest paths from v_s to v_j with length t for $j = 2,\ldots,n$ and $t = 1,\ldots,T$, where $T = (2 \cdot n + 2) \cdot L$. There are two key differences with this dynamic program. First, we also find the shortest white path from a node to itself. In other words, we include $j = 1$ in the computation. Second, we use System (10.2) for the second argument of our min operation. Section 10.2.1.2.2 already proves that System (10.2) correctly computes the shortest paths of increasing length. Therefore, it must be the case that the dynamic program correctly computes the shortest white paths from v_s to all other nodes v_j. Similar arguments can be used to prove that $D_s^{(\,\blacksquare\,)}(j,t)$, $D_s^{(\,\blacksquare\,)}(j,t)$, and $\overset{\blacksquare}{D}_s(j,t)$ correctly compute their respective shortest paths in a WUCS.

This implies that when we update the values of $D_s^{(type)}(s,t)$ for $type \in \{ \square , \blacksquare , \blacksquare , \blacksquare \}$, $D_s^{(\blacksquare)}(s,t)$ represents the cost of the shortest gray cycle containing node v_s whose length is at most t. The smallest t, for which $D_s^{(\blacksquare)}(s,t) < 0$, represents the length of the shortest negative cost gray cycle $\mathbf{C}_s$ containing v_s. The smallest length among all $\mathbf{C}_i$ for all $v_i \in \mathbf{V}$ must give the WOTLR.

10.2.2.2 An FPTAS for the WOTLR problem

In this section, we present a fully polynomial time approximation scheme (FPTAS) for computing the WOTLR of a WUCS.

***Definition* 10.2.2.** *An FPTAS is an algorithm that takes an instance of an optimization problem and a parameter $\varepsilon > 0$ and produces a solution that is within a factor $(1+\varepsilon)$ (or $(1-\varepsilon)$ for maximization problems) of the optimal solution. The running time of the algorithm is to be polynomial in both the problem size and $(1/\varepsilon)$.*

10.2.2.2.1 Preprocessing phase

We first need to convert $\mathbf{G}$ into a simpler network by erasing a carefully selected subset of edges.

Algorithm 10.2.8 Preprocessing Step

Function PRE-PROCESS()

1: Let A be a vector of edges initialized as $\mathscr{A} = \emptyset$.
2: **while** ($\mathbf{G}$ has a negative cost gray cycle) **do**
3: Let e_{ij} denote the edge of $\mathbf{G}$ with the largest length.
4: Remove e_{ij} from $\mathbf{G}$.
5: Add e_{ij} to $\mathscr{A}$.
6: Let e_{uv} be the last edge added to $\mathscr{A}$.
7: **for** (each edge e_{st} in $\mathscr{A}$ such that $l_{st} \leq (2 \cdot n + 2) \cdot l_{uv}$) **do**
8: Add e_{st} back to $\mathbf{G}$.

Algorithm 10.2.8 removes the edges of $\mathbf{G}$ one-by-one in descending order with respect to the lengths until $\mathbf{G}$ does not have a negative cost gray cycle. Let e_{uv} be the last edge

removed. Observe that the length of any negative cost gray cycle in $\mathbf{G}$ is at least l_{uv}. This is because any negative cost gray cycle has to contain at least one edge whose length is at least l_{uv}. Therefore, l_{uv} is a lower bound for the WOTLR of $\mathbf{G}$.

Consider the moment immediately before the algorithm removes e_{uv} from $\mathbf{G}$. l_{uv} is an upper bound for the lengths of the remaining edges in $\mathbf{G}$ at that moment since the algorithm removes the edges in descending order with respect to their lengths. Since $\mathbf{G}$ has a negative cost gray cycle at that moment, and a gray cycle can have at most $(2 \cdot n + 2)$ edges, $((2 \cdot n + 2) \cdot l_{uv})$ is an upper bound for the WOTLR of $\mathbf{G}$. In other words, if $|OPT|$ is the length of the negative cost gray cycle with the smallest length in $\mathbf{G}$, then

$$|OPT| \leq (2 \cdot n + 2) \cdot l_{uv}.$$

Algorithm 10.2.8 then inserts the edges with length at most $((2 \cdot n + 2) \cdot l_{uv})$ back into $\mathbf{G}$. This means that when the algorithm terminates, the only edges that are removed are the ones whose lengths are more than $((2 \cdot n + 2) \cdot l_{uv})$. Observe that the transformation made by Algorithm 10.2.8 on $\mathbf{G}$ preserves the WOTLR. We can check the existence of a negative cost gray cycle on a constraint network in $(c_1 \cdot m \cdot n) = O(m \cdot n)$ time [SW17b], where c_1 is some constant. Furthermore, observe that lines 3 to 5 takes $(c_2 \cdot m)$ time, and the **for** loop at line 8 takes $(c_3 \cdot m)$ time, where c_2 and c_3 are constants. Therefore, the running time of Algorithm 10.2.8, denoted as $T_1(m,n)$, is

$$\begin{aligned} T_1(m,n) &\leq (c_1 \cdot m \cdot n) \cdot (c_2 \cdot m) + (c_3 \cdot m) \\ &= (c_1 \cdot c_2) \cdot (m^2 \cdot n) + (c_3 \cdot m) \\ &\leq (c_1 \cdot c_2) \cdot (n^4 \cdot n) + (c_3 \cdot m) \\ &= (c_1 \cdot c_2) \cdot n^5 + (c_3 \cdot m) \\ &= O(n^5). \end{aligned}$$

10.2.2.2.2 A description of our FPTAS

We next present the main part of our algorithm. For simplicity purposes, let $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c}, \mathbf{l} \rangle$ be the constraint network after the preprocessing step, and let $\varepsilon > 0$.

We let $P = \frac{\varepsilon \cdot l_{uv}}{2 \cdot n + 2}$, where $\varepsilon > 0$ is arbitrarily chosen. For each edge e_{ij} remaining in $\mathbf{G}$, we set l'_{ij} to be $\left\lceil \frac{l_{ij}}{P} \right\rceil$. We then apply Algorithm 10.2.7 on $\mathbf{G}' = \langle \mathbf{V}, \mathbf{E}, \mathbf{c}, \mathbf{l}' \rangle$, and the resulting WOTRR is our approximation. The above observations are summarized in Algorithm 10.2.9.

Algorithm 10.2.9 FPTAS for WOLRR

Function WOTLR-FPTAS()

1: PRE-PROCESS().
2: Let $\mathbf{G}$ be the resulting constraint network.
3: Let $P = \frac{\varepsilon \cdot l_{uv}}{2 \cdot n + 2}$.
4: **for** (each edge $e_{ij} \in \mathbf{E}$) **do**
5: $\quad l'_{ij} = \left\lceil \frac{l_{ij}}{P} \right\rceil$.
6: Define: $\mathbf{G}' = \langle \mathbf{V}, \mathbf{E}, \mathbf{c}, \mathbf{l}' \rangle$
7: Let OPT' denote the resulting negative cost gray cycle with the smallest length from running PSEUDO-WOTLR($\mathbf{G}'$).
8: **return** ($|OPT'|$).

Let OPT denote the negative cost gray cycle with the smallest length in $\mathbf{G}$. Let OPT' denote the negative cost gray cycle with the smallest length after running Algorithm 10.2.9. Let $|OPT|$ and $|OPT'|$ denote the lengths of their respective negative cost gray cycles. Our algorithm returns $|OPT'|$ at termination. In order to prove that our algorithm is an FPTAS, we will show that $|OPT'| \leq (1 + 2 \cdot \varepsilon) \cdot |OPT|$. Clearly, this will prove our claim since $\varepsilon > 0$ is chosen arbitrarily and 2 is a constant.

Recall that for each edge $e_{ij} \in \mathbf{E}$, we have $l'_{ij} = \left\lceil \frac{l_{ij}}{P} \right\rceil < \frac{l_{ij}}{P} + 1$. We claim that

$$l(e_{ij}) < P \cdot l'_{ij} + P.$$

If $l(e_{ij}) \geq P \cdot l'_{ij} + P$, then $\frac{l(e_{ij})}{P} \geq l'_{ij} + 1$, and therefore

$$l'_{ij} = \left\lceil \frac{l_{ij}}{P} \right\rceil \geq \frac{l(e_{ij})}{P} \geq l'_{ij} + 1$$

which is a contradiction.

Let $l'(\mathbf{C}) = \sum_{e_{ij} \in \mathbf{C}} l'_{ij}$ be defined as the sum of the scaled and rounded lengths of the edges in $\mathbf{C}$. If we add the above inequalities for all edges e_{ij} that lie in OPT', we have

$$l(OPT') < P \cdot l'(OPT') + P \cdot (2 \cdot n + 2).$$

Here we used the fact that OPT' contains at most $(2 \cdot n + 2)$ edges. Now, observe that OPT' is a negative cost gray cycle with the smallest length in $\mathbf{G}'$. Hence, it must be the case that $l'(OPT') \leq l'(OPT)$. Thus, we have

$$l(OPT') < P \cdot l'(OPT) + P \cdot (2 \cdot n + 2).$$

Taking into account that $l'_{ij} < \frac{l_{ij}}{P} + 1$ and $\varepsilon \cdot l_{uv} = (2 \cdot n + 2) \cdot P$, we get

$$\begin{aligned}
|OPT'| &= l(OPT') \\
&< P \cdot l'(OPT) + P \cdot (2 \cdot n + 2) \\
&< P \cdot \left(\frac{l(OPT)}{P} + (2 \cdot n + 2) \right) + P \cdot (2 \cdot n + 2) \\
&= l(OPT) + 2 \cdot P \cdot (2 \cdot n + 2) \\
&= |OPT| + 2 \cdot \varepsilon \cdot l_{uv}
\end{aligned}$$

Recall that e_{uv} is the last edge added to $\mathbf{G}$ such that the absence of e_{uv} would result in $\mathbf{G}$ having no negative cost gray cycles. This means that any negative cost gray cycle in $\mathbf{G}$ must include an edge of length at least l_{uv}. Hence, the length of any negative cost gray

cycle in **G** must be at least l_{uv}. Therefore,

$$l_{uv} \leq |OPT|.$$

Thus, we have

$$\begin{aligned} |OPT| + 2 \cdot \varepsilon \cdot l_{uv} &\leq |OPT| + 2 \cdot \varepsilon \cdot |OPT| \\ &= (1 + 2 \cdot \varepsilon) \cdot |OPT| \end{aligned}$$

Therefore, we can conclude that $|OPT'| \leq (1 + 2 \cdot \varepsilon) \cdot |OPT|$.

We now analyze the running time of Algorithm 10.2.9. Let $T(n)$ denote the running time of Algorithm 10.2.9. As previously stated, line 1 takes $O(n^5)$ time. The **for** loop from lines 4 to 6 takes $(c \cdot m) \leq (c \cdot n^2) = O(n^2)$ time, where c is some constant. For line 8, recall that Algorithm 10.2.7 takes $O(n^4 \cdot L)$ time, where L is the length of the largest edge length. However, in this case the pseudo-polynomial time algorithm takes $O(n^4 \cdot L')$ time, where $L' = \left\lceil \frac{L}{P} \right\rceil$. Hence, the total running time is $O(n^4 \cdot L')$. We distinguish two cases.

Case 1: $\frac{L}{P} < 1$. In this case,

$$\begin{aligned} T(n) &\leq (d_1 \cdot n^5) + (c \cdot n^2) + d_2 \cdot (n^4 \cdot L') \\ &= (d_1 \cdot n^5) + (c \cdot n^2) + d_2 \cdot \left(n^4 \cdot \left\lceil \frac{L}{P} \right\rceil \right) \\ &\leq (d_3 \cdot n^5) \\ &= O\left(n^5\right), \end{aligned}$$

where d_1, d_2, and d_3 are constants.

Case 2: $\frac{L}{P} \geq 1$. Then,

$$\begin{aligned} \left\lceil \frac{L}{P} \right\rceil &\leq \left\lfloor \frac{L}{P} \right\rfloor + 1 \\ &\leq \frac{L}{P} + 1 \\ &\leq 2 \cdot \frac{L}{P}. \end{aligned}$$

Therefore, the running time of our FPTAS is

$$\begin{aligned}
T(n) &\leq (d_1 \cdot n^5) + (c \cdot n^2) + d_2 \cdot (n^4 \cdot L') \\
&= (d_1 \cdot n^5) + (c \cdot n^2) + d_2 \cdot \left(n^4 \cdot \left\lceil \frac{L}{P} \right\rceil \right) \\
&\leq (d_1 \cdot n^5) + (c \cdot n^2) + d_2 \cdot \left(n^4 \cdot 2 \cdot \frac{L}{P} \right) \\
&= (d_1 \cdot n^5) + (c \cdot n^2) + 2 \cdot d_2 \cdot \left(n^4 \cdot \frac{(2 \cdot n + 2) \cdot L}{\varepsilon \cdot l_{uv}} \right) \\
&= (d_1 \cdot n^5) + (c \cdot n^2) + 2 \cdot d_2 \cdot \left(n^4 \cdot (2 \cdot n + 2) \cdot \frac{L}{\varepsilon \cdot l_{uv}} \right) \\
&\leq (d_1 \cdot n^5) + (c \cdot n^2) + 2 \cdot d_2 \cdot \left(d_3 \cdot n^5 \cdot \frac{L}{\varepsilon \cdot l_{uv}} \right) \\
&\leq (d_1 \cdot n^5) + (c \cdot n^2) + 2 \cdot d_2 \cdot \left(d_3 \cdot n^5 \cdot \frac{(2 \cdot n + 2) \cdot l_{uv}}{\varepsilon \cdot l_{uv}} \right) \\
&\leq (d_1 \cdot n^5) + (c \cdot n^2) + 2 \cdot d_2 \cdot \left(d_3 \cdot d_4 \cdot n^6 \cdot \frac{1}{\varepsilon} \right) \\
&\leq d_5 \cdot \left(n^6 \cdot \frac{1}{\varepsilon} \right) \\
&= O\left(n^6 \cdot \frac{1}{\varepsilon} \right),
\end{aligned}$$

where d_1, d_2, d_3, d_4, and d_5 are constants.

Observe that if $\varepsilon < 1$, then $O(n^6 \cdot \frac{1}{\varepsilon})$ dominates the running time of our algorithm. Since the running time is polynomial in both $(1/\varepsilon)$ and the size of the input, the above algorithm is an FPTAS.

10.2.3 The LOR problem (ADD rule)

In this section, we investigate the LOR problem in UTVPI constraints. Recall that in a literal-once refutation (LOR), no literal can be used more than once. We shall show that such refutations, if they exist, can be identified in polynomial time via a reduction to the MWPM problem.

10.2.3.1 Construction

Assume that we are given a UCS $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ with n variables and m constraints.

We construct the undirected graph $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$ as follows:

1. For each variable x_i in $\mathbf{U}$, add the vertices x_i^+ and x_i^- to $\mathbf{V}$. Additionally, add the edge $x_i^- \xrightarrow{0} x_i^+$ to $\mathbf{E}$.
2. Add the vertices x_0^+ and x_0^- to $\mathbf{V}$. Additionally, add the edge $x_0^- \xrightarrow{0} x_0^+$ to $\mathbf{E}$.
3. For each constraint l_k of $\mathbf{U}$, add the vertices l_k and l_k' to $\mathbf{V}$, and the edge $l_k \xrightarrow{0} l_k'$ to $\mathbf{E}$. Additionally:
   (a) If l_k is of the form $x_i + x_j \leq b_k$, add the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ and $x_j^+ \xrightarrow{\frac{b_k}{2}} l_k'$ to $\mathbf{E}$.
   (b) If l_k is of the form $x_i - x_j \leq b_k$, add the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ and $x_j^- \xrightarrow{\frac{b_k}{2}} l_k'$ to $\mathbf{E}$.
   (c) If l_k is of the form $-x_i + x_j \leq b_k$, add the edges $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ and $x_j^+ \xrightarrow{\frac{b_k}{2}} l_k'$ to $\mathbf{E}$.
   (d) If l_k is of the form $-x_i - x_j \leq b_k$, add the edges $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ and $x_j^- \xrightarrow{\frac{b_k}{2}} l_k'$ to $\mathbf{E}$.
   (e) If l_k is of the form $x_i \leq b_k$, add the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$, $x_0^+ \xrightarrow{\frac{b_k}{2}} l_k'$, and $x_0^- \xrightarrow{\frac{b_k}{2}} l_k'$ to $\mathbf{E}$.
   (f) If l_k is of the form $-x_i \leq b_k$, add the edges $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$, $x_0^+ \xrightarrow{\frac{b_k}{2}} l_k'$, and $x_0^- \xrightarrow{\frac{b_k}{2}} l_k'$ to $\mathbf{E}$.
4. The weights on the edges in the above construction constitute the cost function $\mathbf{c}$ of $\mathbf{G}$.

Note that if $\mathbf{U}$ has m constraints over n variables, then $\mathbf{G}$ has $(2 \cdot n + 2 \cdot m + 2)$ vertices and $(n + 3 \cdot m + N_a + 1)$ edges where N_a is the number of absolute constraints in $\mathbf{U}$. Since $N_a \leq m$, we can conclude that $|\mathbf{V}| \leq c_1 \cdot (m+n)$, for some constant $c_1 \geq 1$ and $|\mathbf{E}| \leq c_2 \cdot (m+n)$ for some constant $c_2 \geq 1$.

Example 38: Let us consider UCS (10.3).

$$l_1 : \ -x_1 + x_2 \ \leq \ -2 \qquad l_2 : \ x_1 + x_3 \ \leq \ -2 \qquad l_3 : \ -x_2 - x_3 \ \leq \ 2 \tag{10.3}$$

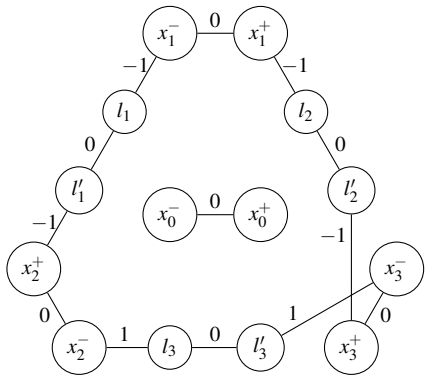

Figure 10.1: Undirected graph corresponding to UCS (10.3).

Applying the construction discussed above to UCS (10.3), we get the undirected graph in Figure 10.1.

It can be shown that the minimum weight perfect matching in this graph is $x_1^+ \xrightarrow{-1} l_2$, $l_2' \xrightarrow{-1} x_3^+$, $x_3^- \xrightarrow{1} l_3'$, $l_3 \xrightarrow{1} x_2^-$, $x_2^+ \xrightarrow{-1} l_1'$, and $l_1 \xrightarrow{-1} x_1^-$. This matching has weight -2 and corresponds to the literal-once refutation obtained by summing constraints l_1, l_2, and l_3 (see Theorem 10.2.4).

10.2.3.2 Correctness

We will now complete the reduction, by establishing the correctness of the above construction.

Observe that x_0 is represented by only two nodes (viz., x_0^+ and x_0^-), although there could be more than two absolute constraints in the system. We argue that these two nodes are sufficient from the perspective of preserving literal-once refutations.

***Lemma* 10.2.3.** *Let* $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ *denote a UCS. If* $\mathbf{U}$ *has a read-once refutation using absolute constraints, then* $\mathbf{U}$ *has a read-once refutation using zero or two absolute constraints.*

Proof. Let R be a read-once refutation of $\mathbf{U}$ with the minimum number of absolute constraints, and let $|R|_a$ represent the number of absolute constraints in R.

If R has an odd number of absolute constraints, then the total number of literals in R would also be odd. Thus, summing the constraints in R would not result in a constraint of

the form $0 \leq b$ where $b < 0$, which is a requirement of a read-once refutation. Thus, R must have an even number of absolute constraints.

Assume that $|R|_a > 2$. Let $l_0 : a_i \cdot x_i \leq b_0$ be an absolute constraint in R. Since R is a read-once refutation, there must be a constraint l_1 with the term $-a_i \cdot x_i$.

If l_1 is an absolute constraint, then the sum of l_0 and l_1 is a constraint of the form $0 \leq b$. If $b < 0$, then the constraints l_0 and l_1 form a refutation using fewer absolute constraints than R. If $b \geq 0$, then the remaining constraints form a read-once refutation using fewer absolute constraints than R. Both cases contradict the assumption that R is a read-once refutation with the fewest absolute constraints.

If l_1 is not an absolute constraint, then the sum of l_0 and l_1 is a constraint of the form $a_j \cdot x_j \leq b$. Thus, we can continue this process, always eliminating the only variable in the derived constraint, until either:

1. No constraints remaining in R can eliminate the variable. In this case, the sum of the constraints in R is not of the form $0 \leq b < 0$. Thus, R is not a read-once refutation.

2. The variable can be eliminated by an absolute constraint. In this case we again derive a constraint of the form $0 \leq b$. As before, this contradicts the assumption that R is a read-once refutation with the fewest absolute constraints.

All possible cases lead to a contradiction, thus we must have that $|R|_a \leq 2$. Since $|R|_a$ is even, this means that $|R|_a \in \{0, 2\}$. □

Note that the proof of Lemma 10.2.3 applies to both read-once and literal-once refutations, since every literal-once refutation is a read-once refutation.

Theorem 10.2.4 relates literal-once refutations to negative weight perfect matchings.

***Theorem* 10.2.4.** *Let* $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ *denote a UCS and let* $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$ *denote the graph constructed from* $\mathbf{U}$*, as described in subsection 10.2.3.1.*

Then, $\mathbf{U}$ *has a literal-once refutation if and only if* $\mathbf{G}$ *has a negative weight perfect matching.*

Proof. First assume that **U** has a literal-once refutation L. Without loss of generality (see Lemma 10.2.3), we can assume that L has 0 or 2 absolute constraints. Since L is a literal-once refutation, it is also a read-once refutation. Therefore, if we sum up the constraints in L, we get $0 \leq b$, where $b < 0$.

We can construct a negative weight perfect matching P of **G** as follows:

1. For each variable x_i in **U**, if L does not use x_i add the edge $x_i^+ \xrightarrow{0} x_i^-$ to P.

2. For each two variable constraint l_k in **U**:

   (a) If $l_k \notin L$, add the edge $l_k \xrightarrow{0} l'_k$ to P.

   (b) If $l_k \in L$ is of the form $x_i + x_j \leq b_k$, add the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ and $x_j^+ \xrightarrow{\frac{b_k}{2}} l'_k$ to P.

   (c) If $l_k \in L$ is of the form $x_i - x_j \leq b_k$, add the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ and $x_j^- \xrightarrow{\frac{b_k}{2}} l'_k$ to P.

   (d) If $l_k \in L$ is of the form $-x_i + x_j \leq b_k$, add the edges $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ and $x_j^+ \xrightarrow{\frac{b_k}{2}} l'_k$ to P.

   (e) If $l_k \in L$ is of the form $-x_i - x_j \leq b_k$, add the edges $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ and $x_j^- \xrightarrow{\frac{b_k}{2}} l'_k$ to P.

3. If L has no absolute constraints, add the edge $x_0^+ \xrightarrow{0} x_0^-$ to P.

4. If L has absolute constraints, as per Lemma 10.2.3, it must have exactly two of them, say l_k and l_r, $k < r$.

   We first process l_k:

   (a) If l_k is of the form $x_i \leq b_k$, add the edge $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ to P.

   (b) If l_k is of the form $-x_i \leq b_k$, add the edge $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ to P.

   Add the edge $x_0^+ \xrightarrow{\frac{b_k}{2}} l'_k$ to P.

   We then process l_r:

   (a) If l_r is of the form $x_i \leq b_r$, add the edge $x_i^+ \xrightarrow{\frac{b_r}{2}} l_r$ to P.

(b) If l_r is of the form $-x_i \leq b_r$, add the edge $x_i^- \xrightarrow{\frac{b_r}{2}} l_r$ to P.

Add the edge $x_0^- \xrightarrow{\frac{b_r}{2}} l'_r$ to P.

We first establish that P is a perfect matching by showing that each vertex in $\mathbf{G}$ is incident upon exactly one edge in P. Observe that:

1. If there are no absolute constraints in L, then the only edge in P to use x_0^+ and x_0^- is $x_0^+ \xrightarrow{0} x_0^-$.

2. If there are two absolute constraints in L, then let l_k and l_r denote these constraints, with $k < r$. Thus, the only edge in P to use x_0^+ is $x_0^+ \xrightarrow{\frac{b_k}{2}} l'_k$ and the only edge in P to use x_0^- is $x_0^- \xrightarrow{\frac{b_r}{2}} l'_r$.

3. For each variable x_i:

   (a) If x_i is not used by L, then the only edge in P to use x_i^+ and x_i^- is $x_i^+ \xrightarrow{0} x_i^-$.

   (b) If x_i is used by L, then there exists exactly one constraint $l_k \in L$ which uses the literal x_i and exactly one constraint $l_r \in L$ which uses the literal $-x_i$. Thus, the only edge in P to use x_i^+ is $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ (or $x_i^+ \xrightarrow{\frac{b_k}{2}} l'_k$), depending on whether x_i is the first variable or second variable respectively, in the constraint l_k. Likewise, the only edge in P to use x_i^- is $x_i^- \xrightarrow{\frac{b_r}{2}} l_r$ (or $x_i^- \xrightarrow{\frac{b_r}{2}} l'_r$), depending on whether x_i is the first variable or second variable respectively, in the constraint l_r.

4. For each constraint l_k:

   (a) If $l_k \notin L$, then the only edge in P to use l_k and l'_k is $l_k \xrightarrow{0} l'_k$.

   (b) If $l_k \in L$, then

      i. If l_k is of the form $a_i \cdot x_i + a_j \cdot x_j \leq b_k$, where $a_i, a_j \in \{1, -1\}$, then the only edge in P to use l_k is $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ (if $a_i = 1$) or $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ (if $a_i = -1$). Likewise, the only edge in P to use l'_k is $x_j^+ \xrightarrow{\frac{b_k}{2}} l'_k$ (if $a_j = 1$) or $x_j^- \xrightarrow{\frac{b_k}{2}} l'_k$ (if $a_j = -1$).

ii. If l_k is of the form $a_i \cdot x_i \leq b_k$, where $a_i \in \{1, -1\}$, then the only edge in P to use l_k is $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ (if $a_i = 1$) or $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ (if $a_i = -1$). Likewise, the only edge in P to use l'_k is $x_0^+ \xrightarrow{\frac{b_k}{2}} l'_k$, if l_k is the first absolute constraint or $x_0^- \xrightarrow{\frac{b_k}{2}} l'_k$, if l_k is the second absolute constraint.

This means that every vertex in $\mathbf{G}$ is an endpoint of exactly one edge in P and P is a perfect matching.

Observe that only constraints in L contribute non-zero weight edges to the matching P. Furthermore, each constraint in l_k in L contributes exactly $(\frac{b_k}{2} + \frac{b_k}{2}) = b_k$ to the weight of the matching. It therefore follows that

$$\sum_{e \in P} \mathbf{c}(e) = \sum_{l_k \in L} (\frac{b_k}{2} + \frac{b_k}{2}) = \sum_{l_k \in L} b_k < 0.$$

The negativity of the sum of the weights follows from the fact that L is a literal-once refutation. It follows that P is a negative weight perfect matching.

Now assume that $\mathbf{G}$ has a negative weight perfect matching P. We construct a literal-once refutation L as follows:

For each constraint l_k in $\mathbf{U}$, if P does not use the edge $l_k \xrightarrow{0} l'_k$, then add the constraint l_k to L.

We make the following observations:

1. If the literal x_i appears in a constraint in L, then so does the literal $-x_i$ - Let $l_k \in L$ and let literal x_i appear in l_k. The edge $l_k \xrightarrow{0} l'_k$ is clearly not part of the matching P, since l_k would not be in L otherwise. It follows that either $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ is in P or $x_i^+ \xrightarrow{\frac{b_k}{2}} l'_k$ is in P depending on whether x_i is the first or second variable in l_k. Then, it must be the case that the edge $x_i^+ \xrightarrow{0} x_i^-$ is not in P. This forces one of the edges $x_i^- \xrightarrow{\frac{b_r}{2}} l_r$ or $x_i^- \xrightarrow{\frac{b_r}{2}} l'_r$ to be in P, for some r. This means that the edge $l_r \xrightarrow{0} l'_r$ is not part of the

matching either. It follows that the constraint $l_r \in L$, which forces the literal $-x_i$ to be in L.

2. If the literal $-x_i$ appears in a constraint in L, then so does the literal x_i - Let $l_k \in L$ and let literal $-x_i$ appear in l_k. The edge $l_k \xrightarrow{0} l'_k$ is clearly not part of the matching P, since l_k would not be in L otherwise. It follows that either $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ is in P or $x_i^- \xrightarrow{\frac{b_k}{2}} l'_k$ is in P depending on whether x_i is the first or second variable in l_k. Then, it must be the case that the edge $x_i^+ \xrightarrow{0} x_i^-$ is not in P. This forces one of the edges $x_i^+ \xrightarrow{\frac{b_r}{2}} l_r$ or $x_i^+ \xrightarrow{\frac{b_r}{2}} l'_r$ to be in P, for some r. This means that the edge $l_r \xrightarrow{0} l'_r$ is not part of the matching either. It follows that the constraint $l_r \in L$, which forces the literal x_i to be in L.

3. Each literal can appear in at most one constraint in L - Assume the contrary and let the literal x_i appear in two distinct constraints l_k and l_r in L. As argued above, since x_i appears in l_k, then either $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ is in P or $x_i^+ \xrightarrow{\frac{b_k}{2}} l'_k$ is in P depending on if x_i is the first or second variable in l_k. Likewise, since x_i appears in l_r, either $x_i^+ \xrightarrow{\frac{b_r}{2}} l_r$ is in P or $x_i^+ \xrightarrow{\frac{b_r}{2}} l'_r$ is in P. In either case, x_i is matched to two distinct vertices in P, which is not possible. An identical argument establishes that the literal $-x_i$ cannot appear in two distinct constraints in L.

4. Summing all the constraints in L results in a contradiction of the form $0 \leq b$, $b < 0$ - Based on the discussion above, we know that if a literal x_i appears in a constraint in L, then so does the literal $-x_i$. Furthermore, each literal can appear in at most one constraint in L. It follows that if we sum the constraints, we get 0 on the left hand side, since each literal is canceled by it counterpart.

   If a constraint $l_k \in L$, then the edge $l_k \xrightarrow{0} l'_k$ is not in P. Thus, P must contain one edge from l_k and one edge from l'_k, which are distinct. By construction, these edges have weight $\frac{b_k}{2}$.

   Conversely, if the constraint $l_k \notin L$, then the edge $l_k \xrightarrow{0} l'_k$ is in P. Thus, none of the

edges of weight $\frac{b_k}{2}$ from l_k and l'_k are in P.

By construction, the remaining edges in P have weight 0. This means that

$$\begin{aligned} \sum_{l_k \in L} b_k &= \sum_{l_k \in L} (\frac{b_k}{2} + \frac{b_k}{2}) && (10.4) \\ &= \sum_{e \in P} \mathbf{c}(e) && (10.5) \\ &< 0 && (10.6) \end{aligned}$$

Note that Equation (10.4) follows from the fact that constraint $l_k \in L$ contributes two edges to the matching P. One of these edges is from vertex l_k and the other edge is from vertex l'_k. Furthermore, both these edges have weight $\frac{b_k}{2}$. Equation (10.5) follows from the fact all edges in P which did not result in the corresponding constraint being included in L have weight 0. Finally, relation (10.6) follows from the fact that P is a negative weight perfect matching. We thus have, $0 \leq \sum_{l_k \in L} b_k = \sum_{e \in P} \mathbf{c}(e) < 0$, since summing the constraints in L results in a sum of 0 on the left hand side.

Based on the four observations above, it is clear that L is a literal-once refutation of $\mathbf{U}$. □

Note that this method cannot be used to identify read-once refutations that reuse literals.

Example 39: Consider the UCS described by System (10.7).

$$\begin{array}{lll} l_1: \quad x_1 + x_2 \leq -2 & l_2: \quad x_1 + x_3 \leq -2 & l_3: \quad -x_1 - x_4 \leq -2 \\ l_4: \quad -x_1 - x_5 \leq -2 & l_5: \quad -x_2 - x_3 \leq 2 & l_6: \quad x_4 + x_5 \leq 2 \end{array} \tag{10.7}$$

This corresponds to the undirected graph in Figure 10.2.

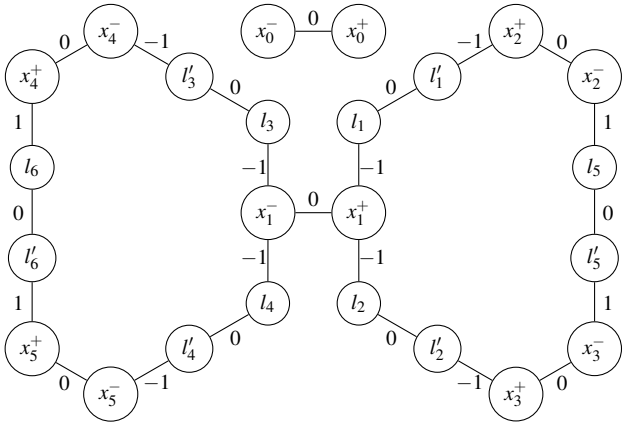

Figure 10.2: Undirected graph corresponding to UCS (10.7)

It is not hard to see that the minimum weight perfect matching in this graph is $x_1^+ \overset{0}{—} x_1^-$, $x_2^+ \overset{0}{—} x_2^-$, $x_3^+ \overset{0}{—} x_3^-$, $x_4^+ \overset{0}{—} x_4^-$, $x_5^+ \overset{0}{—} x_5^-$, $l_1' \overset{0}{—} l_1''$, $l_2' \overset{0}{—} l_2''$, $l_3' \overset{0}{—} l_3''$, $l_4' \overset{0}{—} l_4''$, $l_5' \overset{0}{—} l_5''$, and $l_6' \overset{0}{—} l_6''$. This matching has weight 0 and indicates that UCS (10.7) has no literal-once refutation.

However, UCS (10.7) still has a read-once refutation obtained by summing all six constraints.

10.2.3.3 Resource analysis

To construct $\mathbf{G}$, we need to process each variable and each constraint in $\mathbf{U}$. Each variable and each constraint can be processed in constant time. Thus, the reduction can be performed in $O(m+n)$ time. The minimum weight perfect matching of $\mathbf{G}$ can be found in $O(|\mathbf{E}| \cdot |\mathbf{V}| + |\mathbf{V}|^2 \cdot \log |\mathbf{V}|)$ time [Gab90]. As discussed in Subsection 10.2.3.1, $\mathbf{G}$ has $O(m+n)$ vertices and $O(m+n)$ edges. Thus, by using this reduction, the LOR problem for UTVPI constraints can be solved in $O((m+n)^2 \cdot \log(m+n))$ time.

10.2.4 The ROR problem (ADD rule)

In this section, we show that read-once refutations in systems of UTVPI constraints (if they exist) can be found in polynomial time.

To find these refutations, we modify the undirected graph construction described in Section 10.2.3. The undirected graph construction in Section 10.2.3 does not allow for the reuse of variables. However, as demonstrated in Example 10.2.3.2, a read-once refutation may need to use a variable more than once. In our new construction, we increase the number of vertices representing each variable.

10.2.4.1 Construction

We convert the UCS $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ into an undirected graph $\mathbf{G}' = \langle \mathbf{V}', \mathbf{E}', \mathbf{c}' \rangle$ as follows:

1. For each variable x_i in $\mathbf{U}$, add the vertices x_i^+, ${x'_i}^+$, x_i^-, and ${x'_i}^-$ to $\mathbf{V}'$. Additionally, add the edges $x_i^- \xrightarrow{0} x_i^+$ and ${x'_i}^- \xrightarrow{0} {x'_i}^+$ to $\mathbf{E}'$.

2. Add the vertices x_0^+ and x_0^- to $\mathbf{V}'$. Additionally, add the edge $x_0^- \xrightarrow{0} x_0^+$ to $\mathbf{E}'$.

3. For each constraint l_k of $\mathbf{U}$, add the vertices l_k and l'_k to $\mathbf{V}'$ and the edge $l_k \xrightarrow{0} l'_k$ to $\mathbf{E}'$. Additionally:

   (a) If l_k is of the form $x_i + x_j \leq b_k$, add the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$, ${x'_i}^+ \xrightarrow{\frac{b_k}{2}} l_k$, $x_j^+ \xrightarrow{\frac{b_k}{2}} l'_k$, and ${x'_j}^+ \xrightarrow{\frac{b_k}{2}} l'_k$ to $\mathbf{E}'$.

   (b) If l_k is of the form $x_i - x_j \leq b_k$, add the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$, ${x'_i}^+ \xrightarrow{\frac{b_k}{2}} l_k$, $x_j^- \xrightarrow{\frac{b_k}{2}} l'_k$, and ${x'_j}^- \xrightarrow{\frac{b_k}{2}} l'_k$ to $\mathbf{E}'$.

   (c) If l_k is of the form $-x_i + x_j \leq b_k$, add the edges $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$, ${x'_i}^- \xrightarrow{\frac{b_k}{2}} l_k$, $x_j^+ \xrightarrow{\frac{b_k}{2}} l'_k$, and ${x'_j}^+ \xrightarrow{\frac{b_k}{2}} l'_k$ to $\mathbf{E}'$.

   (d) If l_k is of the form $-x_i - x_j \leq b_k$, add the edges $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$, ${x'_i}^- \xrightarrow{\frac{b_k}{2}} l_k$, $x_j^- \xrightarrow{\frac{b_k}{2}} l'_k$, and ${x'_j}^- \xrightarrow{\frac{b_k}{2}} l'_k$ to $\mathbf{E}'$.

   (e) If l_k is of the form $x_i \leq b_k$, add the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$, ${x'_i}^+ \xrightarrow{\frac{b_k}{2}} l_k$, $x_0^+ \xrightarrow{\frac{b_k}{2}} l'_k$, and $x_0^- \xrightarrow{\frac{b_k}{2}} l'_k$ to $\mathbf{E}'$.

(f) If l_k is of the form $-x_i \leq b_k$, add the edges $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$, $x_i'^- \xrightarrow{\frac{b_k}{2}} l_k$, $x_0^+ \xrightarrow{\frac{b_k}{2}} l'_k$, and $x_0^- \xrightarrow{\frac{b_k}{2}} l'_k$ to $\mathbf{E}'$.

In this (new) construction, each variable is represented by a pair of 0-weight edges. As we shall see, this permits each vertex to be used twice by a read-once refutation. However, we still only have one 0-weight edge for each constraint, which prevents a read-once refutation from reusing edges.

Note that if $\mathbf{U}$ has m constraints over n variables, then $\mathbf{G}'$ has $(4 \cdot n + 2 \cdot m + 2)$ vertices and $(2 \cdot n + 5 \cdot m + 1)$ edges. In other words, $\mathbf{G}'$ has $O(m+n)$ vertices and $O(m+n)$ edges.

Example 40: Let us return to UCS represented by System (10.7). As shown in Section 10.2.3, UCS (10.7) does not have a literal-once refutation. However, it does have a read-once refutation. The new undirected graph corresponding to UCS (10.7) is shown in Figure 10.3.

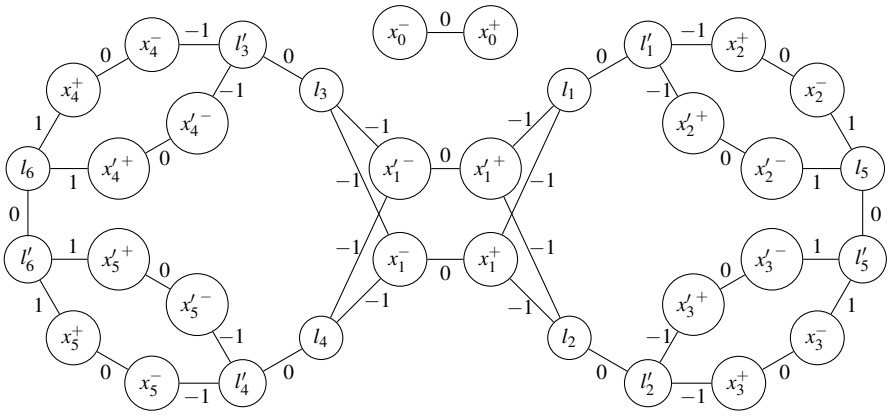

Figure 10.3: Undirected graph corresponding to UCS (10.7)

It can be shown that the minimum weight perfect matching in this graph is $x_1^+ \xrightarrow{-1} l_2$, $l'_2 \xrightarrow{-1} x_3^+$, $x_3^- \xrightarrow{1} l'_5$, $l_5 \xrightarrow{1} x_2^-$, $x_2^+ \xrightarrow{-1} l'_1$, $l_1 \xrightarrow{-1} x_1'^+$, $x_1'^- \xrightarrow{-1} l_3$, $l'_3 \xrightarrow{-1} x_4^-$, $x_4^+ \xrightarrow{1} l_6$, $l'_6 \xrightarrow{1} x_5^+$, $x_5^- \xrightarrow{-1} l'_4$, $l_4 \xrightarrow{-1} x_1^-$, $x_2'^+ \xrightarrow{0} x_2'^-$, $x_3'^+ \xrightarrow{0} x_3'^-$, $x_4'^+ \xrightarrow{0} x_4'^-$, and $x_5'^+ \xrightarrow{0} x_5'^-$. This matching has weight -4 and corresponds to the read-once refutation obtained by summing all six constraints (see Theorem 10.2.5).

10.2.4.2 Correctness

We now proceed to argue the correctness of the above construction.

We first establish a structural property of certain read-once refutations in a UCS (see Lemma 10.2.5). This property will be used in Theorem 10.2.5.

Let R be a read-once refutation of $\mathbf{U}$ which uses the fewest number of constraints, i.e., a shortest read-once refutation. Let x_i be a variable used by R. Let $|R|_i$ be the number of constraints in R that use the literal x_i. Since R is a read-once refutation, $|R|_i$ is also the number of constraints that use the literal $-x_i$.

Assume that $|R|_i \geq 3$. By Lemma 10.2.3, we can assume without loss of generality that R has at most 2 absolute constraints. Thus R must have a non-absolute constraint that uses x_i. Let $l_0 : x_i + a_j \cdot x_j \leq b_{ij}$ be one such non-absolute constraint in R. l_0 is called the *current constraint*. In what follows, we will proceed through a sequence of stages eliminating the non-x_i variable in the current constraint, until eventually x_i itself is eliminated or a contradiction results.

Algorithm 10.2.10 represents our approach.

***Lemma* 10.2.4.** PROCESS-REFUTATION(R, l_0) *cannot return any value.*

Proof. Recall that R is a shortest read-once refutation of $\mathbf{U}$ and that $|R|_i \geq 3$. Assume that PROCESS-REFUTATION(R, l_0) returns a value of u. We will show that every value of u results in a contradiction.

1. $u = -1$ - In this case, there is a non-x_i literal that cannot be canceled by any of the the remaining constraints in R. Thus, the sum of the constraints in R is not of the form $0 \leq b < 0$. It follows that R is not a read-once refutation.

2. $u = -2$ - Note that every time a constraint l_s is added to *sum* it cancels the non-x_i variable in $\sum_{l_h \in sum} l_h$. Thus, $\sum_{l_h \in sum} l_h$ is always a UTVPI constraint of the form $a_i \cdot x_i + a_j \cdot x_j \leq b$, $a_i, a_j \in \{1, -1\}$, where x_j is the non-canceled variable in l_s. The only exception to this is if l_s is an absolute constraint. However, in this case lines

Algorithm 10.2.10 Shortest refutation property

Function PROCESS-REFUTATIONR, l_0

1: $sum := \{l_0\}$. ▷ The set of constraints processed thus far.
2: $num_i := 1$.
3: $R' := R \setminus \{l_0\}$.
4: **while** $(R' \neq \emptyset)$ **do**
5: Let $l_s \in R'$ be a constraint which cancels the non-x_i variable in $\sum_{l_h \in sum} l_h$.
6: **if** (l_s does not exist) **then**
7: **return**(-1).
8: $R' := R' \setminus \{l_s\}$.
9: $sum := sum \cup \{l_s\}$.
10: **if** (l_s is an absolute constraint) **then**
11: Let l'_s be the remaining absolute constraint in R.
12: $R' := R' \setminus \{l'_s\}$. $l_s = l'_s$.
13: $sum := sum \cup \{l_s\}$.
14: **if** (l_s uses the literal $-x_i$) **then**
15: **if** $(num_i = 1)$ **then**
16: **return**(-2).
17: **else**
18: **return**(-3).
19: **if** (l_s uses the literal x_i) **then**
20: **if** $(num_i = 1)$ **then**
21: $num_i := 2$.
22: Let $l'_s \in R$ be a non-absolute constraint with the literal $-x_i$.
23: $R' := R' \setminus \{l'_s\}$. $l_s = l'_s$.
24: $sum' := sum$.
25: $sum := \{l_s\}$.
26: **else**
27: **return**(-2).
28: **return**(-4).

13 though 15 are executed and the other absolute constraint in R (viz., l'_s) is added to *sum*. This returns $\sum_{l_h \in sum} l_h$ to the desired form.

Algorithm 10.2.10 can return -2 from either line 19 or line 32. Thus, we need to consider the following cases:

(a) Line 19 was executed - In this case, $num_i = 1$ and the condition in the **if** statement on line 17 was satisfied by a constraint l_s which uses the literal $-x_i$. Since $num_i = 1$, lines 26 through 30 have not been executed. Thus, the first constraint added to *sum* is l_0 which contains the literal x_i (cf. line 30). The non-canceled literal in l_s is $-x_i$. Thus, $\sum_{l_h \in sum} l_h$ is a constraint of the form $x_i - x_i \leq b$. Note that the literal x_i is used only once by the constraints in *sum*. Hence, $sum \subset R$. If $b < 0$, then the constraints in *sum* form a read-once refutation of **U**. If $b \geq 0$, then the constraints in $R \setminus sum$ form a read-once refutation of **U**. It follows that R is not the shortest read-once refutation of **U**.

(b) Line 32 was executed - In this case, $num_i = 2$ and the condition in the **if** statement on line 24 was satisfied by a constraint l_s which uses the literal x_i. Since $num_i = 2$, lines 26 through 30 have been executed. Thus, the first constraint added to *sum* is the constraint l'_s found on line 27 which contains the literal $-x_i$. l'_s is guaranteed to exist because there are at least three constraints in R that use the literal $-x_i$ and at most two of them can be absolute constraints. The non-canceled literal in l_s is x_i. Thus, $\sum_{l_h \in sum} l_h$ is a constraint of the form $-x_i + x_i \leq b$. Note that the literal x_i is used only once by the constraints in *sum*. Thus, $sum \subset R$. If $b < 0$, then the constraints in *sum* form a read-once refutation of **U**. If $b \geq 0$, then the constraints in $R \setminus sum$ form a read-once refutation of **U**. It follows that R is not the shortest read-once refutation of **U**.

3. $u = -3$ - In this case, $num_i = 2$ and the condition in the **if** statement on line 17 was satisfied by a constraint l_s which uses the literal $-x_i$. Since $num_i = 2$, some constraint l_k satisfied the condition in the **if** statement on line 24. Furthermore, lines

26 through 30 have been executed. Let $l_{k'}$ be the constraint found on line 27. Thus, *sum* contains the constraints l'_k and l_s both of which use the literal $-x_i$. As argued previously, all other literals have been canceled. Thus, $\sum_{l_h \in sum} l_h$ is a constraint of the form $-x_i - x_i \leq b$. Similarly, sum' contains the constraints l_0 and l_k both of which use the literal x_i. Thus, $\sum_{l_h \in sum'} l_h$ is a constraint of the form $x_i + x_i \leq b'$. This means that $\sum_{l_h \in sum \cup sum'} l_h$ is a constraint of the form $0 \leq b + b'$.

Note that *sum* contains no constraints that use x_i and that sum' contains two constraints that use x_i. Thus, $sum \cup sum' \subset R$, since as per the hypothesis R has three constraints which use the literal x_i. If $b < 0$, then the constraints in $sum \cup sum'$ form a read-once refutation of $\mathbf{U}$. If $b \geq 0$, then the constraints in $R \setminus (sum \cup sum')$ form a read-once refutation of $\mathbf{U}$. It follows that R is not the shortest read-once refutation of $\mathbf{U}$.

4. $u = -4$ - In this case, we have processed every constraint in R. In partuclar, we have processed every constraint in R that uses the literal x_i. Let $l_1, l_2 \in R$ be the first two such constraints processed, i.e., both l_1 and l_2 use the literal x_i and are processed before any other constraint in R that also uses literal x_i. Assume that l_1 is processed before l_2. Since $|R|_i \geq 3$, the constraints l_1 and l_2 are guaranteed to exist.

   Since both l_1 and l_2 use the literal x_i, Algorithm 10.2.10 will execute the portion between lines 25 and 34. When processing l_1, it must be the case that $num_i = 1$ and hence, Algorithm 10.2.10 sets num_i to 2. Now, when constraint l_2 is processed, num_i is already 2 and hence line 32 is executed and a value of -2 is returned.

□

***Lemma* 10.2.5.** *Let* $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ *denote an infeasible UCS. If* $\mathbf{U}$ *has a read-once refutation, then it has a read-once refutation, in which each literal is used at most twice.*

Proof. Let R be the shortest read-once refutation of $\mathbf{U}$. Assume that $|R|_i \geq 3$, for some literal x_i, where $|R|_i$ is the number of constraints in R that use the literal x_i (as discussed before). Recall that R must have a non-absolute constraint l_0 of the form $x_i - a_j \cdot x_j \leq b_{ij}$.

From Lemma 10.2.4, PROCESS-REFUTATION(R, l_0) cannot return any value. However, R is finite. Thus, PROCESS-REFUTATION(R, l_0) must eventually halt and return a value. This is a contradiction. It follows that we must have $|R|_i \leq 2$, for every literal x_i. □

Theorem* 10.2.5.** *Let* $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$ *denote a UCS and let* $\mathbf{G}' = \langle \mathbf{V}', \mathbf{E}', \mathbf{c}' \rangle$ *denote the graph constructed from* **U, as described in subsection 10.2.4.1.*

Then, **U** *has a read-once refutation if and only if* $\mathbf{G}'$ *has a negative weight perfect matching.*

Proof. First assume that **U** has a read-once refutation R. As argued in Lemma 10.2.5, we can assume that each literal in **U** occurs in R at most twice.

We construct a negative weight perfect matching P of $\mathbf{G}'$ as follows:

1. For each variable x_i in **U**:
   (a) If R does not use the literal x_i, add the edges $x_i^+ \xrightarrow{0} x_i^-$ and $x_i'^+ \xrightarrow{0} x_i'^-$ to P.
   (b) If R uses the literal x_i only once, add the edge $x_i'^+ \xrightarrow{0} x_i'^-$ to P.
   (c) If R uses the literal x_i twice, do not add any edges to P.
2. Assume that the constraints in **U** are assigned an arbitrary order. For constraint l_k in **U**:
   (a) If $l_k \notin R$, add the edge $l_k \xrightarrow{0} l_k'$ to P.
   (b) If $l_k \in R$ is of the form $a_i \cdot x_i + a_j \cdot x_j \leq b_k$ such that $a_i, a_j \in \{1, -1\}$:
      i. If l_k is the first edge to use the literal x_i, add the edge $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ to P. If it is the second, add the edge $x_i'^+ \xrightarrow{\frac{b_k}{2}} l_k$.
      ii. If l_k is the first edge to use the literal $-x_i$, add the edge $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ to P. If it is the second, add the edge $x_i'^- \xrightarrow{\frac{b_k}{2}} l_k$.
      iii. If l_k is the first edge to use the literal x_j, add the edge $x_j^+ \xrightarrow{\frac{b_k}{2}} l_k'$ to P. If it is the second, add the edge $x_j'^+ \xrightarrow{\frac{b_k}{2}} l_k'$.

iv. If l_k is the first edge to use the literal $-x_j$, add the edge $x_j^- \xrightarrow{\frac{b_k}{2}} l'_k$ to P. If it is the second, add the edge ${x'_j}^+ \xrightarrow{\frac{b_k}{2}} l'_k$.

(c) If $l_k \in R$ is of the form $a_i \cdot x_i \leq b_k$, such that $a_I \in \{1, -1\}$:

i. If l_k is the first edge to use the literal x_i, add the edge $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ to P. If it is the second, add the edge ${x'_i}^+ \xrightarrow{\frac{b_k}{2}} l_k$.

ii. If l_k is the first edge to use the literal $-x_i$, add the edge $x_i^- \xrightarrow{\frac{b_k}{2}} l_k$ to P. If it is the second, add the edge ${x'_i}^- \xrightarrow{\frac{b_k}{2}} l_k$.

iii. If l_k is the first absolute constraint, then add the edge $x_0^+ \xrightarrow{\frac{b_k}{2}} l'_k$ to P.

iv. If l_k is the second absolute constraint, then add the edge $x_0^- \xrightarrow{\frac{b_k}{2}} l'_k$ to P.

From Lemma 10.2.3, we can assume without loss of generality that there are at most two absolute constraints in R.

(d) If R has no absolute constraints, then add the edge $x_0^+ \xrightarrow{0} x_0^-$ to P.

We make the following observations:

1. For each variable x_i:

(a) Assume that the literal x_i is not used by R. Since R is a read-once refutation, the literal $-x_i$ cannot be used by R either. In this case, the only edge in P to use the vertices x_i^+ and x_i^- is $x_i^+ \xrightarrow{0} x_i^-$. Similarly, the only edge in P to use the vertices ${x'_i}^+$ and ${x'_i}^-$ is ${x'_i}^+ \xrightarrow{0} {x'_i}^-$.

(b) Assume that the literal x_i is used by a single constraint, say $l_k \in R$. It follows that the literal $-x_i$ is also used by a single constraint, say $l_r \in R$, since R is a read-once refutation. Thus, the only edge in P to use the vertex x_i^+ is $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$, the only edge in P to use the vertex x_i^- is $x_i^- \xrightarrow{\frac{b_r}{2}} l_r$, and the only edge in P to use the vertices ${x'_i}^+$ and ${x'_i}^-$ is ${x'_i}^+ \xrightarrow{0} {x'_i}^-$.

(c) Assume that the literal x_i is used in two constraints, say $l_{k_1}, l_{k_2} \in R$ with $k_1 < k_2$. As argued above, the literal $-x_i$ must also be used by two constraints, say $l_{r_1}, l_{r_2} \in R$ with $r_1 < r_2$. Thus, the only edge in P to use the vertex x_i^+ is $x_i^+ \overset{\frac{b_{k_1}}{2}}{---} l_{k_1}$, the only edge in P to use the vertex x_i^- is $x_i^- \overset{\frac{b_{r_1}}{2}}{---} l_{r_1}$, the only edge in P to use the vertex $x_i'^+$ is $x_i'^+ \overset{\frac{b_{k_2}}{2}}{---} l_{k_2}$, and the only edge in P to use the vertex $x_i'^-$ is $x_i'^- \overset{\frac{b_{r_2}}{2}}{---} l_{r_2}$.

2. For each constraint l_k:

   (a) If $l_k \in R$, then, by construction, P contains two edges of weight $\frac{b_k}{2}$. One of these edges is between l_k and a vertex representing a literal and the other edge is between l'_k and a vertex representing a literal. Additionally, P does not contain the edge $l_k \overset{0}{---} l'_k$. It follows that both l_k and l'_k are the endpoints of exactly one edge in P.

   (b) If $l_k \notin R$, then, by construction, P contains the edge $l_k \overset{0}{---} l'_k$ and none of the weight $\frac{b_k}{2}$ edges from either l_k or l'_k. Thus both l_k and l'_k are the endpoints of exactly one edge in P.

3. By Lemma 10.2.3, we can assume without loss of generality that R contains 0 or 2 absolute constraints. Thus:

   (a) If R contains no absolute constraints, then the only edge in P to use the vertices x_0^+ and x_0^- is $x_0^+ \overset{0}{---} x_0^-$.

   (b) If R contains two absolute constraints l_{k_1} and l_{k_2} with $k_1 < k_2$, then the only edge in P to use the vertex x_0^+ is $x_0^+ \overset{\frac{b_{k_1}}{2}}{---} l'_{k_1}$ and the only edge in P to use the vertex x_0^- is $x_0^- \overset{\frac{b_{k_2}}{2}}{---} l'_{k_2}$.

Thus, every vertex in $\mathbf{G}'$ is an endpoint of exactly one edge in P. It follows that P is a perfect matching. Additionally, for each constraint $l_k \in R$, P has two edges of weight $\frac{b_k}{2}$

and all other edge in P have weight 0. Thus,

$$\begin{aligned} \sum_{e \in P} \mathbf{c}'(e) &= \sum_{l_k \in R} \left(\frac{b_k}{2} + \frac{b_k}{2} \right) && (10.8) \\ &= \sum_{l_k \in R} b_k && (10.9) \\ &< 0 && (10.10) \end{aligned}$$

Equation (10.8) follows from the fact that the edges of $\mathbf{G}'$ in P, which do not correspond to constraints in R have zero weight. Relation (10.10) follows from the fact that R is a read-once refutation.

It follows that P has negative weight.

Now assume that $\mathbf{G}'$ has a negative weight perfect matching P. Construct the set R as follows: For each constraint l_k in $\mathbf{U}$, if P does not use the edge $l_k \xrightarrow{0} l'_k$, then add the constraint l_k to R.

In order to show that R is a read-once refutation, we need to establish that each constraint in $\mathbf{U}$ is used at most once and that the summation of the constraints in R results in a contradiction of the form $0 \leq b$, $b < 0$.

From the construction of R, it is clear that each constraint in $\mathbf{U}$ occurs in R at most once. In order to establish that the summation of the constraints in R results in a contradiction, we show that the number of times a literal (say x_i) appears in R is equal to the number of times its complement ($-x_i$) appears in R.

P is a perfect matching. Thus, for each variable x_i, we have the following:

1. If a constraint $l_k \in R$ uses the literal x_i, then either:

   (a) One of the edges $x_i^+ \xrightarrow{\frac{b_k}{2}} l_k$ or $x_i^+ \xrightarrow{\frac{b_k}{2}} l'_k$ is in P. Thus, the edge $x_i^+ \xrightarrow{0} x_i^-$ is not in P. This means that for some r, one of the edges $x_i^- \xrightarrow{\frac{b_r}{2}} l_r$ or $x_i^- \xrightarrow{\frac{b_r}{2}} l'_r$ is in P. Thus, the edge $l_r \xrightarrow{0} l'_r$ is not in P. It follows that l_r is a constraint R that uses the literal $-x_i$.

(b) One of the edges $x_i'^{+} \xrightarrow{\frac{b_k}{2}} l_k$ or $x_i'^{+} \xrightarrow{\frac{b_k}{2}} l_k'$ is in P. Thus, the edge $x_i'^{+} \xrightarrow{0} x_i'^{-}$ is not in P. This means that for some r, one of the edges $x_i'^{-} \xrightarrow{\frac{b_r}{2}} l_r$ or $x_i'^{-} \xrightarrow{\frac{b_r}{2}} l_r'$ is in P. Thus, the edge $l_r \xrightarrow{0} l_r'$ is not in P. It follows that l_r is a constraint R that uses the literal $-x_i$.

Note that if there are two constraints in R that use the literal x_i, then, since P is a perfect matching, one constraint corresponds to an edge in P that uses x_i^+ and the other constraint corresponds to an edge in P that uses $x_i'^{+}$. Thus, one constraint corresponds to case (a) above and the other corresponds to case (b). This means that each constraint that uses x_i corresponds to a distinct constraint in R that uses the literal $-x_i$.

If an additional (third) constraint in R used the literal x_i, then this constraint would correspond to another edge in P that uses x_i^+ or $x_i'^{+}$. Thus, either two edges in P use x_i^+ or two edges in P use $x_i'^{+}$. However, this cannot happen since P is a perfect matching. It follows that the literal x_i is used by at most two constraints in R.

2. Arguing in similar fashion, we can show that if one or two constraints in R use the literal $-x_i$, then the same number of constraints in R use the literal x_i.

Thus, the number of constraints in which the literal x_i appears in R is equal to the number of constraints in R in which the literal $-x_i$ appears.

If a constraint $l_k \in R$, then the edge $l_k \xrightarrow{0} l_k'$ is not in P. Thus, P contains two edges of weight $\frac{b_k}{2}$ one with end point l_k and one with endpoint l_k'. Conversely, if the constraint $l_k \notin R$, then the edge $l_k \xrightarrow{0} l_k'$ is in P. It follows that none of the edges of weight $\frac{b_k}{2}$ from l_k and l_k' are in P. Thus, for each constraint $l_k \in R$, P has two edges of weight $\frac{b_k}{2}$ and all other edge in P have weight 0

Thus, summing the constraints in R yields

$$\begin{aligned} 0 &\leq \sum_{l_k \in R} b_k \\ &= \sum_{l_k \in R} \left(\frac{b_k}{2} + \frac{b_k}{2} \right) \\ &= \sum_{e \in P} \mathbf{c}'(e) \\ &< 0, \text{ since } P \text{ is a negative weight matching.} \end{aligned}$$

As discussed before, each constraint appears at most once in R. Thus, R is a read-once refutation of $\mathbf{U}$. □

10.2.4.3 Resource analysis

To construct $\mathbf{G}'$, we need to process each variable and each constraint in $\mathbf{U}$. Each variable and each constraint can be processed in constant time. Thus, the reduction can be performed in $O(m+n)$ time. The minimum weight perfect matching of $\mathbf{G}'$ can be found in $O(|\mathbf{E}'| \cdot |\mathbf{V}'| + |\mathbf{V}'|^2 \cdot \log |\mathbf{V}'|)$ time [Gab90]. As discussed in Subsection 10.2.4.1, $\mathbf{G}'$ has $O(m+n)$ vertices and $O(m+n)$ edges. Thus, by using this reduction, the ROR problem for UTVPI constraints can be solved in $O((m+n)^2 \cdot \log(m+n))$ time.

10.2.5 The NLROR problem (ADD rule)

In this section, we study a variant of read-once refutation called non-literal read-once refutation (NLROR). Recall that an NLROR is a read-once refutation that does not contain a literal-once refutation (see subsection 3.4.3).

By definition, an NLROR must reuse one or more literals. However, the mere fact that a read-once refutation reuses a literal does not imply that it is an NLROR.

Example 41: Consider the UCS in System (10.11).

$$\begin{array}{lllllllllll} l_1: & x_1+x_2 & \leq & -1 & l_2: & -x_2-x_3 & \leq & 0 & l_3: & x_3-x_1 & \leq & -1 \\ l_4: & x_1+x_4 & \leq & 1 & l_5: & -x_4-x_5 & \leq & -1 & l_6: & x_5-x_1 & \leq & 1 \end{array} \tag{10.11}$$

Summing all 6 constraints in UCS (10.11) results in the constraint $0 \leq -1$. Thus, all 6 constraints form a read-once refutation that reuses the literal x_1.

However, summing the constraints l_1, l_2, and l_3 results in the constraint $0 \leq -2$. Thus, these 3 constraints form a literal-once refutation contained within the original read-once refutation. This means that the original refutation is not an NLROR despite reusing a literal.

We show that the problem of checking if a UCS has an NLROR is **NP-complete**.

We first show that the NLROR problem for UCSs is in **NP**.

***Lemma* 10.2.6.** *The NLROR problem for UCSs is in* **NP**.

Proof. Let **U** be a UCS with m constraints over n variables, and let R be a read-once refutation of **U** that does not contain a literal-once refutation. We show that this property of R can be verified in polynomial time.

Since R is a read-once refutation, it has at most m constraints. Thus, it is polynomially sized in terms of **U**. To show that R is a read-once refutation of **U**, we need to show that no constraint is used twice. Thus, for each constraint l_i, we verify that $l_i \in \mathbf{U}$ and that l_i does not occur among the remaining constraints in R. We then sum the constraints in R to verify that the result is a contradiction of the form $0 \leq b$, $b < 0$. This can be accomplished in $O(m^2)$ time.

From Section 10.2.3, we know that checking if R contains a literal-once refutation can be accomplished in polynomial time. Thus, the entire process of checking if R is a non-literal read-once refutation can be accomplished in polynomial time. This means that the NLROR problem for UCSs is in **NP**. □

Before proving the **NP-hardness** of the NLROR problem, we prove the following result on the structure of non-literal read-once refutations.

***Lemma* 10.2.7.** *Let* $\mathbf{U}$ *be a UCS* $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b}$. *If* $\mathbf{U}$ *has an NLROR, then for some variable* x_i, $\mathbf{U}$ *has an NLROR* R *that can be partitioned into sets* R_1 *and* R_2 *such that:*

1. R_1 *can be used to derive the constraint* $2 \cdot x_i \leq b_1$.
2. R_2 *can be used to derive the constraint* $-2 \cdot x_i \leq b_2$.
3. *There exists no variable* x_j *other than* x_i, *such that* x_j *is used in* R_1 *and* R_2.

Proof. Let R be an NLROR of $\mathbf{U}$ with the fewest constraints. If there exists a read-once refutation $R' \subset R$, then R' must also be an NLROR. Thus, R is a minimal read-once refutation of $\mathbf{U}$.

Since R is not a literal-once refutation, there must exist a literal x_i or $-x_i$ reused by R. Since R is a read-once refutation, the literals x_i and $-x_i$ occur the same number of times in R.

We can assume without loss of generality that the literal x_i is reused by R. By Lemma 10.2.5, the literal x_i is used at most twice by R. Since x_i is reused, it must be the case that x_i is used twice by R. Thus, there are four constraints in R that use the variable x_i (two which use the literal x_i and two which use the literal $-x_i$). By Lemma 10.2.3, R has at most 2 absolute constraints. Thus, at least one of the four constraints which use x_i is not an absolute constraint. Let l_0 denote a non-absolute constraint with variable x_i. Assume without loss of generality that l_0 uses the literal x_i.

We run PROCESS-REFUTATION(R, l_0) (Algorithm 10.2.10). Observe that the **return** statements in Algorithm 10.2.10 are in the following places:

1. inside the **if** statement on line 7 (line 8),
2. inside the **if** statement on line 17 (line 19 and line 20),
3. inside the **if** statement on line 24 (line 32), and

4. outside the **while** loop (line 36).

Since R is a read-once refutation. we can always find a l_s on line 6 of Algorithm 10.2.10 and hence line 8 is never executed. This means that either l_s is assigned to be a constraint which satisfies the condition in one of the **if** statements on line 17 or line 24 or every constraint in $R \setminus \{l_0\}$ is processed by the algorithm.

$R \setminus \{l_0\}$ contains at least one constraint with the literal x_i and at least one constraint with the literal $-x_i$. Thus, PROCESS-REFUTATION(R, l_0) will eventually assign l_s to be one of these constraints. This will satisfy the condition in either the **if** statement on line 17 (l_s contains the literal $-x_i$) or the condition in the **if** statement on line 24 (l_s contains the literal x_i).

If the condition in the **if** statement on line 17 is satisfied, then summing the constraints in *sum* results in a constraint of the form $x_i - x_i \leq b_1$. Likewise, we can sum the constraints in $R \setminus sum$ to obtain a constraint of the form $x_i - x_i \leq b_2$.

If the condition in the **if** statement on line 24 is satisfied, then summing the constraints in *sum* results in a constraint of the form $2 \cdot x_i \leq b_1$. Likewise, we can sum the constraints in $R \setminus sum$ to obtain a constraint of the form $-2 \cdot x_i \leq b_2$.

Thus, we can partition R into sets R_1 and R_2 such that either:

1. R_1 can be used to derive the constraint $x_i - x_i \leq b_1$ and R_2 can be used to derive the constraint $x_i - x_i \leq b_2$.

2. R_1 can be used to derive the constraint $2 \cdot x_i \leq b_1$ and R_2 can be used to derive the constraint $-2 \cdot x_i \leq b_2$.

In the first case, either $b_1 < 0$ or $b_2 < 0$, since $(b_1 + b_2) < 0$. Thus, either R_1 or R_2 is a read-once refutation of **U**. This contradicts the minimality of R. Thus, the second case must hold.

If R_1 and R_2 share a variable $x_j \neq x_i$, then we can decompose R_1 into the sets R_1^+ and R_1^- such that the constraints in R_1^+ sum to produce the constraint $x_i + x_j \leq b_1^+$ and the

constraints in R_1^- sum to produce the constraint $x_i - x_j \leq b_1^-$. This follows from previous reasoning about the functioning of Algorithm 10.2.10 and Lemma 6 in [SW17b].

We can similarly decompose R_2 into the sets R_2^+ and R_2^- such that the constraints in R_2^+ sum to produce the constraint $-x_i + x_j \leq b_2^+$ and the constraints in R_2^- sum to produce the constraint $-x_i - x_j \leq b_2^-$.

Together, the constraints in the sets R_1^+ and R_2^- sum to produce the constraint $0 \leq b_1^+ + b_2^-$. Similarly, the constraints in the sets R_1^- and R_2^+ sum to produce the constraint $0 \leq b_1^- + b_2^+$. Since $b_1^+ + b_1^- + b_2^+ + b_2^- = b_1 + b_2 < 0$, we must have that $b_1^+ + b_2^- < 0$ or $b_1^- + b_2^+ < 0$. Thus, either $R_1^+ \cup R_2^-$ or $R_1^- \cup R_2^+$ is a read-once refutation of **U**. This contradicts the minimality of R. Thus, R_1 and R_2 cannot share any variable, except for x_i. □

The **NP-hardness** of the NLROR problem is established via a reduction from the vertex-disjoint path problem.

***Definition* 10.2.3.** *Given a directed graph G and four distinct vertices s_1, t_1, s_2, and t_2, the* **vertex-disjoint path problem** *consists of determining if G has a pair of vertex-disjoint paths, one from s_1 to t_1 and the other from s_2 to t_2.*

The problem is known to be **NP-complete** [FHW80].

***Theorem* 10.2.6.** *The NLROR problem for UCSs is* **NP-complete**.

Proof. Let $\mathbf{G} = \langle \mathbf{V}, \mathbf{E} \rangle$ be a directed graph with source vertices s_1 and s_2 and destination vertices t_1 and t_2. We construct the UCS **U** as follows:

1. For each vertex $x_i \in \mathbf{V}$, add the variable x_i to **U**.
2. Add the variable x_0 to **U**.
3. For each edge $(x_i, x_j) \in \mathbf{E}$, add the constraint $x_j - x_i \leq 0$ to **U**.
4. Add the constraints $x_0 + s_1 \leq -1$, $x_0 - t_1 \leq -1$, $-x_0 + s_2 \leq -1$, and $-x_0 - t_2 \leq -1$ to **U**.

Consider the directed graph in Figure 10.4.

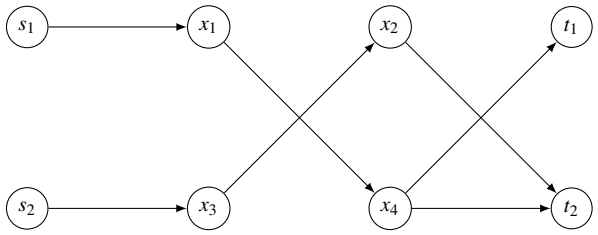

Figure 10.4: Directed graph corresponding to UCS (10.12).

The graph in Figure 10.4 corresponds to the UCS in System (10.12).

$$\begin{array}{llllll} l_1: & x_1 - s_1 \leq 0 & l_2: & x_4 - x_1 \leq 0 & l_3: & t_1 - x_4 \leq 0 \\ l_4: & t_2 - x_4 \leq 0 & l_5: & x_3 - s_2 \leq 0 & l_6: & x_2 - x_3 \leq 0 \\ l_7: & t_2 - x_2 \leq 0 & l_8: & x_0 + s_1 \leq -1 & l_9: & x_0 - t_1 \leq -1 \\ l_{10}: & -x_0 + s_2 \leq -1 & l_{11}: & -x_0 - t_2 \leq -1 & & \end{array} \tag{10.12}$$

The graph in Figure 10.4 has the following vertex-disjoint paths:

$$p_1: \quad s_1 \rightarrow x_1 \rightarrow x_4 \rightarrow t_1$$

$$p_2: \quad s_2 \rightarrow x_3 \rightarrow x_2 \rightarrow t_2$$

These paths correspond to an NLROR of UCS (10.12) consisting of the constraints l_1, l_2, l_3, l_5, l_6, l_7, l_8, l_9, l_{10}, and l_{11}.

Assume that $\mathbf{G}$ has a pair of vertex-disjoint paths from s_1 to t_1 and from s_2 to t_2. We will show that $\mathbf{U}$ has a refutation of the desired type.

Let p_1 be the path from s_1 to t_1 in $\mathbf{G}$. By construction, p_1 corresponds to a set of constraints that sum together to produce $t_1 - s_1 \leq 0$. When summed together with the constraints $x_0 + s_1 \leq -1$ and $x_0 - t_1 \leq -1$, we get the constraint $2 \cdot x_0 \leq -2$. Let R_1 be the set of constraints used in this derivation.

Let p_2 be the path from s_2 to t_2 in $\mathbf{G}$. By construction, p_2 corresponds to a set of constraints that sum together to produce $t_2 - s_2 \leq 0$. When summed together with the constraints $-x_0 + s_2 \leq -1$ and $-x_0 - t_2 \leq -1$,we get the constraint $-2 \cdot x_0 \leq -2$. Let R_2 be the set of constraints used in this derivation.

The paths p_1 and p_2 are vertex-disjoint. This means that the only variable common to both R_1 and in R_2 is x_0. Let $R = R_1 \cup R_2$. Summing the constraints in R, results in the constraint $0 \leq -4$. Thus, R is a read-once refutation of $\mathbf{U}$. Note that if we remove any constraint from R_1, then we can no longer derive $2 \cdot x_0 \leq -2$. Similarly, if we remove any constraint from R_2, then we can no longer derive $-2 \cdot x_0 \leq -2$. Thus, R is a minimal read-once refutation. This means that R does not contain a literal-once refutation.

Now assume that $\mathbf{U}$ contains an NLROR. By Lemma 10.2.7, $\mathbf{U}$ has an NLROR R that can be partitioned into sets R_1 and R_2 such that for some variable x_i:

1. R_1 can be used to derive the constraint $2 \cdot x_i \leq b_1$.

2. R_2 can be used to derive the constraint $-2 \cdot x_i \leq b_2$.

3. There exists no variable x_j other than x_i, such that x_j is used in R_1 and R_2.

Note that R_1 must contain a constraint with two positive literals. By construction, the only constraint with two positive literals in $\mathbf{U}$ is $x_0 + s_1 \leq -1$. Thus, the constraint $x_0 + s_1 \leq -1$ is in R_1.

Similarly, R_2 must contain a constraint with two negative literals. By construction, the only constraint with two negative literals in $\mathbf{U}$ is $-x_0 - t_2 \leq -1$. Thus, the constraint $-x_0 - t_2 \leq -1$ is in R_2. This means that x_0 must be used by constraints in both R_2 and R_2. Thus, $x_i = x_0$.

This means that R_1 can be decomposed into:

1. The constraint $x_0 + s_1 \leq -1$.

2. A set of constraints which sum together to produce the constraint $t_1 - s_1 \leq 0$. This corresponds to a path p_1 from s_1 to t_1 in $\mathbf{G}$.

3. The constraint $x_0 - t_1 \leq -1$.

Similarly, R_2 can be decomposed into:

1. The constraint $-x_0 + s_2 \leq -1$.

2. A set of constraints which sum together to produce the constraint $t_2 - s_2 \leq 0$. This corresponds to a path p_2 from s_2 to t_2 in $\mathbf{G}$.

3. The constraint $-x_0 - t_2 \leq -1$

Since the sets R_1 and R_2 do not share any variables other than x_0, the paths p_1 and p_2 in $\mathbf{G}$ are vertex-disjoint. □

Chapter 11

Horn Constraint Systems

11.1 Motivation and Related Work

Horn constraint systems define a class of polyhedra that find applications in a number of disparate domains [Tru03, CS13]. A refutation of an unsatisfiable constraint system (not necessarily Horn) is a negative certificate that attests to the infeasibility of the system. When an algorithm produces a certificate to accompany the output, it is called a certifying algorithm [MMNS11]. The literature is replete with certifying algorithms for a number of problems in combinatorial optimization, especially as they relate to graphical structures [KMMS03, DFK$^+$03, KN09]. Likewise, there exist a number of combinatorial algorithms for Horn constraint systems that do not produce negative certificates [CS13, SW15b]. This chapter discusses negative certificates for Horn constraint systems in a number of interesting proof systems.

The focus of this chapter is on read-once refutations in various proof systems. In a read-once refutation, each constraint defining the polyhedron can be used at most *once* in an inference step of the proof system. The advantage of read-once refutations is that they are short by definition [IM95]. In general, read-once proof systems are not complete in that there could exist unsatisfiable constraint systems for which read-once refutations do

not exist. A variant of read-once refutation called *input refutation* is discussed in [Hoo89].

There are many different types of refutation techniques. These include cutting planes and resolution. Resolution refutations can be further divided into linear, tree-style, dag-style, and read once. Each of these has different rules regarding how clauses can be used in the refutations. For example, in a read-once refutation each clause can be used only once. In this chapter we will be looking at read-once resolution refutations for several different problems.

For a given problem, P, two refutation systems are **P**-equivalent if for any instance of P a refutation under one system can be transformed into a refutation under the other system with at most a polynomial increase in size.

11.2 Refutability

11.2.1 The ROR problem (ADD rule)

In this section we study systems of Horn constraints. Note that not every system of Horn constraints has a read-once refutation.

Example 42: Consider the system of Horn constraints

$$\begin{aligned}
l_1: &\quad x_1 &\geq\ & 1\\
l_2: &\quad -x_1+x_2 &\geq\ & 1\\
l_3: &\quad -x_1-x_2+x_3 &\geq\ & 1\\
l_4: &\quad -x_1-x_2-x_3+x_4 &\geq\ & 1\\
l_5: &\quad -x_1-x_2-x_3-x_4 &\geq\ & -14
\end{aligned}$$

The constraint l_5 is necessary for any refutation. To cancel each x_i, $i = 1\ldots4$, the constraint l_i, $i = 1\ldots4$ must also be used in the refutation. Thus, all five constraints need to be used in any refutation.

However to get a positive number on the right-hand side of the resultant constraint we need to use some of the constraints l_1 through l_4 more than once. Thus, this system does not have a read-once linear refutation.

We now show that ROR(ADD) is **NP-complete** for HCSs. This is done by a reduction from the set packing problem.

***Definition* 11.2.1.** *The* **set packing** *problem is the following: Given a set S, m subsets $S_1, \ldots, S_m$ of S, and an integer k, does $\{S_1, \ldots, S_m\}$ contain k mutually disjoint sets.*

This problem is known to be **NP-complete** [Kar72].

***Theorem* 11.2.1.** *ROR(ADD) for Horn constraints is* **NP-hard**.

Proof. Let us consider an instance of the set packing problem. We construct the system of Horn constraints **H** as follows.

1. For each $x_i \in S$, create the variable x_i and the constraint $x_i \geq 1$.
2. For $j = 1 \ldots k$, create the variable v_j.
3. For each subset S_l, $l = 1 \ldots m$ create the constraints

$$v_j - \sum_{x_i \in S_l} x_i \geq 1 - |S_l| \qquad j = 1 \ldots k.$$

4. Finally create the variable w and the constraints $w - v_1 - \ldots - v_k \geq 1 - k$ and $-w \geq 0$.

We now show that **H** is in $ROR(ADD)$ if and only if $\{S_1, \ldots, S_m\}$ contains k mutually disjoint sets.

Suppose that $\{S_1, \ldots, S_m\}$ does contain k mutually disjoint sets. Without loss of generality assume that these are the sets $S_1, \ldots, S_k$.

Let us consider the sets of clauses

$$\mathbf{H}_j = \{v_j - \sum_{x_i \in S_j} x_i \geq 1 - |S_j|\} \cup \{x_i \geq 1 \,|\, x_i \in S_j\} \qquad j = 1 \ldots k.$$

By the construction of $\mathbf{H}$, we have that $\mathbf{H}_j \subseteq \mathbf{H}$ for $j = 1 \ldots k$. Since the sets $S_1, \ldots, S_k$ are mutually disjoint, so are the sets $\mathbf{H}_1, \ldots, \mathbf{H}_k$.

It it easy to see that the constraint $v_j \geq 1$ can be derived by summing all of the constraints in $\mathbf{H}_j$. Sine this holds for every $j = 1 \ldots k$ and since the sets $\mathbf{H}_1, \ldots, \mathbf{H}_k$ are mutually disjoint, we have that the set of constraints $\{v_1 \geq 1, \ldots, v_k \geq 1\}$ can be derived from $\mathbf{H}$ by read-once linear resolution.

Together with the constraint $w - v_1 - \ldots - v_k \geq 1 - k$, this set of constraints sums together to derive the constraint $w \geq 1$. Thus, $\mathbf{H}$ has a read-once linear derivation of the constraint $w \geq 1$. Since $\mathbf{H}$ contains the clause $-w \geq 0$, it follows that $\mathbf{H}$ has a read-once linear refutation.

Now suppose that $\mathbf{H}$ has a read-once linear refutation $\mathbf{R}$. Note that $\mathbf{H}/\{-w \geq 0\}$ can be satisfied by setting every variable to 1. Thus, $\mathbf{R}$ must use the constraint $-w \geq 0$.

By construction, to cancel $-w$ we must use the constraint $w - v_1 - \ldots - v_k \geq 1 - k$. We must now cancel $-v_1, \ldots, -v_k$. Let us consider $-v_j$, $1 \leq j \leq k$. By the construction of $\mathbf{H}$, to cancel this term we must use one of the constraints

$$v_j - \sum_{x_i \in S_l} x_i \geq 1 - |S_l| \qquad l = 1 \ldots m.$$

To cancel the $-x_i$ terms introduced by this constraint, we must use the set of constraints $\mathbf{F}_{l_j} = \{x_i \geq 1 \,|\, x_i \in S_{l_j}\}$ for some $l_j \leq m$.

Since the refutation is read-once we must have that the sets $\mathbf{F}_{l_j}$ for $j = 1 \ldots k$ are mutually disjoint. Thus, the sets S_{l_j} for $j = 1 \ldots k$ are also mutually disjoint. This means that $\{S_1, \ldots, S_m\}$ contains k mutually disjoint sets.

Thus, $\mathbf{H}$ is in $ROR(ADD)$ if and only if $\{S_1, \ldots, S_m\}$ contains k mutually disjoint sets. As a result of this, the linear ROR problem for systems of Horn constraints is **NP-hard**. □

11.2.1.1 An exact exponential algorithm

In this section we describe an exact exponential time algorithm for the problem of fining a read-once ADD refutation for a system of Horn constraints.

Let **H** be a system of horn constraints. Let k be the maximum number of constraints that use any literal and let C be the largest absolute value of any defining constant of a constraint in **H**.

Thus, any constraint derivable from summing a subset of the constraints in **H** can be represented by an $(n+1)$ tuple $(a_1, a_2, \ldots, a_n, b)$ where:

1. $-k \leq a_i \leq k$, is the coefficient of x_i in the derived constraint.

2. $-m \cdot C \leq b \leq m \cdot C$, is the defining constant of the derived constraint.

This results in $2 \cdot C \cdot m \cdot (2 \cdot k+1)^n$ possible derived constraints. Note that any constraint that proves infeasibility, $0 \geq b > 0$, corresponds to a tuple $(0, 0, \ldots, 0, b)$, where $b > 0$.

Thus, we have the following algorithm for determining if **H** has a read-once refutation using only the ADD rule.

Algorithm 11.2.1 Algorithm for ROR(ADD).

Function ROR-ADD (HCS **H**)

Let D be an $m \times 2 \cdot C \cdot m \cdot (2 \cdot k+1)^n$ boolean array which stores if a constraint can be derived from the first i constraints of **H**.
Set every value in D to **false**.
$D(1, l_1) :=$ **true**.
for ($i = 2$ **to** m) **do**
    $D(i, l_i) :=$ **true**.
    for (each possible derived constraint l) **do**
        $D(i,l) := D(i-1,l) \vee D(i-1, l-l_i)$.
        if ($D(i,l) =$ **true** and l is of the form $0 \geq b$, $b > 0$) **then**
            return (**true**)
return (**false**)

11.2.1.1.1 Correctness

We now show that Algorithm 11.2.1 correctly determines if an HCS **H** has a read-once

refutation using only the ADD rule.

***Theorem* 11.2.2.** ROR-ADD(**H**) *returns* **true** *if and only if* **H** *has a read-once refutation using only the ADD rule.*

Proof. If **H** has a read-once refutation using only the add rule, then we can derive a contradiction by summing a subset $\mathbf{H}'$ of the constraints in **H**. Let j be the last constraint in $\mathbf{H}'$ reached by Algorithm 11.2.1. We have that $D(j, \sum_{l \in \mathbf{H}'} l) = \mathbf{true}$. Thus, Algorithm 11.2.1 returns **true**.

If Algorithm 11.2.1 returns **true**, then for some j and constraint l of the form $0 \geq b$, $b > 0$, we have $D(j,l) = \mathbf{true}$. Note that l is derived by summing a subset of the constraints in **H**. Thus **H** has a read-once refutation using only the ADD rule. □

11.2.1.1.2 Resource analysis

The array D has $O(C \cdot m^2 \cdot (2 \cdot k + 1)^n)$ entries, thus initialization takes $O(C \cdot m^2 \cdot (2 \cdot k + 1)^n)$ time. Each iteration of the **for** loop on line 6 takes constant time. Thus, all iterations of this **for** loop take a total of $O(C \cdot m \cdot (2 \cdot k + 1)^n)$ time. Since this **for** loop is run $(m-1)$ times, we have that the total running time of Algorithm 11.2.1 is $O(C \cdot m^2 \cdot (2 \cdot k + 1)^n)$.

11.2.1.2 Horn clausal constraint systems

We now examine the complexity of determining if an HClCS has a read-once refutation using only the ADD rule.

***Lemma* 11.2.1.** *If Φ is an unsatisfiable system of Horn clauses, then the HClCS $S(\Phi) \in$ CP(ADD).*

Proof. We do this by showing that a linear refutation can simulate positive unit resolution. Consider a single resolution step. Let (x) and $(\neg x \vee \neg x_1 \vee \ldots \vee \neg x_s \vee y)$ be two clauses. The resolvent is $(\neg x_1 \vee \ldots \vee \neg x_s \vee y)$.

The constraints corresponding to the original clauses are $x \geq 1$ and $-x - x_1 - \ldots - x_s + y \geq 1 - s$. Summing these inequalities results in $-x_1 - \ldots - x_s + y \geq 1 - (s-1)$. This is the inequality corresponding to the resolvent. □

***Lemma* 11.2.2.** *If Φ has a read-once unit resolution refutation, then the HClCS $S(\Phi) \in$ CP-RO(ADD) Moreover, $S(\Phi)$ has a read-once unit refutation under the ADD rule.*

This is a direct consequence of Lemma 11.2.1.

From [KZ03], we know that determining if a Horn formula has a read-once unit resolution refutation is **NP-complete**. Thus we have the following result.

***Corollary* 11.2.1.** *The CP-RO(ADD) problem for HClCSes is* **NP-complete**.

Part IV

Polyhedral Constraints: Integer Satisfiability

Chapter 12

UTVPI Constraint Systems

12.1 Motivation and Related Work

In this section, we briefly review some of the important milestones in the design of algorithms for checking integer feasibility in UTVPI constraint systems.

The first known decision procedure for checking the integer feasibility of a system of UTVPI constraints is detailed in [JMSY94]. This algorithm processes a set of UTVPI constraints with the goal of finding its transitive and tightening closure. Such a closure is essentially a finite representation of all possible UTVPI constraints that can be inferred from the input set of constraints (also see [BHZ09]). In other words, it finds all possible deductions from the initial set of constraints, including rounded constraints which can be forced into integral solutions. It then checks to see if the system of constraints thus generated, is feasible by virtue of having no contradictions. The algorithm runs in $O(m \cdot n^2)$ time and uses $O(n^2)$ space. Furthermore, it is not certifying. [HS97] improves on the approach in [JMSY94] from an ease-of-implementation standpoint, by combining the transitive and tightening closures into a single step. However, the additional wrinkle does not improve the asymptotic complexity of the algorithm in [JMSY94]; nor does it provide certificates.

A rather different approach was used in [Sub04a] to decide integer feasibility in UTVPI

systems, while also producing a model. This algorithm uses Fourier-Motzkin elimination [DE73] to project the polyhedral representation of a system of UTVPI constraints to a single variable in a solution-preserving manner, thereby determining bounds for that variable. The algorithm then works in reverse order to assign values to the rest of the variables. The algorithm takes $O(n^3)$ time and $O(n^2)$ space.

The algorithm in [LM05] (henceforth, the Lahiri algorithm) is the fastest known algorithm to date, for deciding integer feasibility in UTVPI systems. We will elaborate on their method, in order to provide the proper background to contrast our procedures.

The Lahiri algorithm begins by converting each constraint into a pair of difference constraints with positive and negative versions of each involved variable. For instance, a sum constraint, say, $x_i + x_j \leq c_{ij}$ is converted into the following difference constraint pair: $x_i^+ - x_j^- \leq c_{ij}$ and $x_j^+ - x_i^- \leq c_{ij}$. Once all constraints are thus converted, the converted constraint system is represented by a constraint network as detailed in [CLRS01]. For instance, the constraint $x_j^- - x_i^- \leq c_{ij}$ results in an edge $x_j^- \overset{c_{ij}}{\blacksquare} x_i^-$. The resulting edges are then tightened by converting edges of the form $x_i \overset{c_{ii}}{\blacksquare} x_i$, where c_{ii} is odd to $x_i \overset{c_{ii}-1}{\blacksquare} x_i$, in order to ensure integral solutions. A negative cycle detection subroutine (such as the Bellman-Ford algorithm) then determines whether the system is satisfiable.

We note that in order for the Lahiri algorithm to produce a model, it must compute the transitive and tightening closure of the original constraint system, *even* when such a set of constraints is known to be satisfiable. Indeed, it uses a procedure similar to the one in [JMSY94] and [HS97] to find bounds for all variables and assign values to them. A naive implementation of this algorithm runs in $O(n^3)$ time and uses $O(n^2)$ space. Utilizing Johnson's algorithm for implementing the transitive closure [CLRS01], the time complexity can be improved to $O(m \cdot n + n^2 \cdot \log n)$, while maintaining $O(n^2)$ space complexity. However, even the improved algorithm is more expensive (asymptotically) to the ideal $O(m \cdot n)$ time and $O(m+n)$ space complexity of the non-model generating decision algorithm.

Recently, there has been some work on *incremental* satisfiability of UTVPI constraints. For instance, [SS10] describes an algorithm for incremental (integer) satisfiability check-

ing in UTVPI constraints. Their algorithm adds a single constraint to a set of UTVPI constraints in $O(m + n \cdot \log n)$ time. Incremental algorithms are extremely important from the perspective of SAT Modulo Theories [NOT04].

12.2 Satisfiability

12.2.1 Scaling algorithm

In this section, we provide a bit-scaling algorithm for UTVPI constraints.

Our algorithm requires the transformation of the input UCS into a constraint network as described in [LM05].

This transformation is handled by Algorithm 12.2.1.

Algorithm 12.2.1 MAKE-GRAPH

Function MAKE-GRAPH (System of UTVPI constraints S)

1: Let $\mathbf{G}$ be a constraint network.
2: **for** $i = 1 \ldots n$ **do**
3: Add the vertices x_i^+ and x_i^- to $\mathbf{G}$.
4: **for** Every constraint e in S **do**
5: **if** e is of the form $x_i + x_j \leq c_k$ **then**
6: Add the edge $x_i^- \rightarrow x_j^+$ to $\mathbf{G}$ with weight c_k.
7: Add the edge $x_j^- \rightarrow x_i^+$ to $\mathbf{G}$ with weight c_k.
8: **if** e is of the form $x_i - x_j \leq c_k$ **then**
9: Add the edge $x_i^- \rightarrow x_j^-$ to $\mathbf{G}$ with weight c_k.
10: Add the edge $x_j^+ \rightarrow x_i^+$ to $\mathbf{G}$ with weight c_k.
11: **if** e is of the form $-x_i + x_j \leq c_k$ **then**
12: Add the edge $x_i^+ \rightarrow x_j^+$ to $\mathbf{G}$ with weight c_k.
13: Add the edge $x_j^- \rightarrow x_i^-$ to $\mathbf{G}$ with weight c_k.
14: **if** e is of the form $-x_i - x_j \leq c_k$ **then**
15: Add the edge $x_i^+ \rightarrow x_j^-$ to $\mathbf{G}$ with weight c_k.
16: Add the edge $x_j^+ \rightarrow x_i^-$ to $\mathbf{G}$ with weight c_k.
17: **if** e is of the form $x_i \leq c_k$ **then**
18: Add the edge $x_i^- \rightarrow x_i$ to $\mathbf{G}$ with weight $2 \cdot c_k$.
19: **if** e is of the form $-x_i \leq c_k$ **then**
20: Add the edge $x_i^+ \rightarrow x_i^-$ to $\mathbf{G}$ with weight $2 \cdot c_k$.
21: **return** $\mathbf{G}$ as a constraint network.

[LM05] transforms the input UTVPI system into a constraint network as follows:

Consider the following constraint system.

$$\begin{array}{rcl@{\qquad}rcl@{\qquad}rcl} x_1 + x_3 & \leq & 0 & x_2 - x_3 & \leq & -7 & x_4 - x_2 & \leq & 3 \\ -x_1 - x_4 & \leq & 5 & x_1 & \leq & 6 & & & \end{array} \tag{12.1}$$

For each variable, two vertices (a positive version and a negative version) are added to the constraint network. For instance, corresponding to the variable x_i, we create the vertices x_i^+ and x_i^-. Each constraint is replaced by a pair of equivalent constraints. For instance, a difference constraint $x_i - x_j \leq c$ is replaced by the two constraints $x_i^+ - x_j^+ \leq c$ and $x_j^- - x_i^- \leq c$. The exception is for absolute constraints, each of which is simply converted to a single equivalent constraint. For instance, $x_i \leq c$ yields $x_i^+ - x_i^- \leq 2 \cdot c$. Once all the equivalent constraints have been determined, they are represented in a directed graph, as discussed in [CLRS01]. It is thus seen that the constraint network constructed as per [LM05] has $2 \cdot n$ vertices (assuming n variables in the constraint system) and up to $2 \cdot m$ edges (assuming m constraints in the original constraint system). The resultant constraint network is called the potential graph. Figure 12.1 shows the potential graph, corresponding to System (12.1).

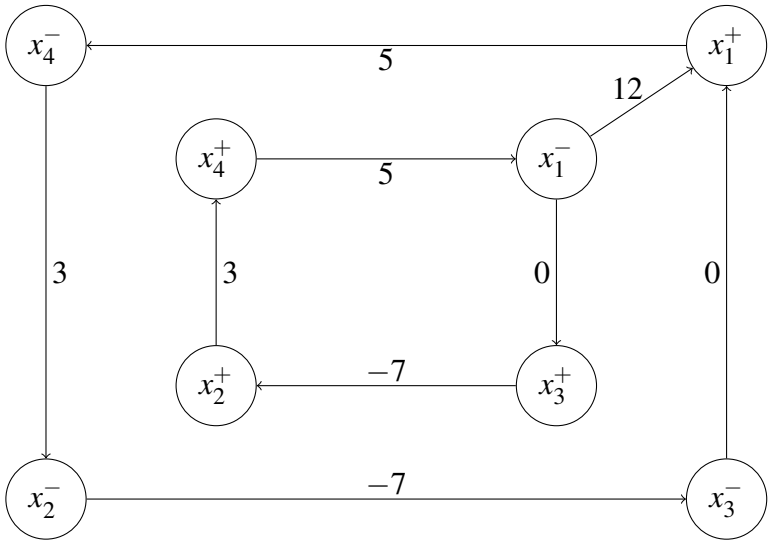

Figure 12.1: Potential graph corresponding to System (12.1)

We are now ready to present our bit-scaling algorithm.

Algorithm 12.2.2 UTVPI-SCALING

Function UTVPI-SCALING (System of UTVPI constraints S)

1: Let $\mathbf{d}$ denote a linear solution to S.
2: Let $\mathbf{y}$ denote an integer solution to S.
3: $\mathbf{G} \leftarrow$ MAKE-GRAPH(S).
4: $\mathbf{f} \leftarrow$ GOLDBERG($\mathbf{G}$).
5: **if** $\mathbf{f}$ is a feasible price function for $\mathbf{G}$ **then**
6: **for** $i = 1 \ldots n$ **do**
7: $d_i \leftarrow \frac{f_{2 \cdot i-1} - f_{2 \cdot i}}{2}$.
8: **else**
9: **return** S is not linear feasible. ▷ Thus, S is not integer feasible.
10: $\Phi \leftarrow$ MAKE-2CNF(S, $\mathbf{d}$).
11: **if** Φ is satisfiable **then**
12: $\mathbf{v} \leftarrow$ satisfying assignment to Φ.
13: **else**
14: **return** S is not integer feasible.
15: **for** $i = 1 \ldots n$ **do**
16: **if** $d_i \in \mathbb{Z}$ **then**
17: $y_i \leftarrow d_i$
18: **else if** v_i is **true then**
19: $y_i \leftarrow d_i + \frac{1}{2}$
20: **else**
21: $y_i \leftarrow d_i - \frac{1}{2}$
22: **return** $\mathbf{y}$ as an integer solution to S.

Algorithm 12.2.2 divides the process of obtaining an integer solution to a system of UTVPI constraints into several steps.

First the system of UTVPI constraints, $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$, is converted into a constraint network. This is the same process used in [LM05] and is described in greater detail in Algorithm 12.2.1. Note that $\mathbf{G}$ has two vertices, x_i^+ and x_i^-, corresponding to each variable. Thus, $\mathbf{f}$ will have $2 \cdot n$ values with $f_{2 \cdot i-1}$ as the price of vertex x_i^+ and $f_{2 \cdot i}$ as the price of vertex x_i^-.

Linear feasibility is then determined using Goldberg's Bit-Scaling Algorithm. This is the same process used in [Gol95]. We refer to this algorithm as GOLDBERG($\mathbf{G}$). If $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ is not linearly feasible, then it is not integer feasible and it is returned as such. However, if it is linearly feasible, then we can construct a linear solution $\mathbf{d}$. Note that every element of $\mathbf{d}$ is an integer multiple of $\frac{1}{2}$, thus $\mathbf{d}$ is a *half-integral* solution.

$\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ is then transformed into a system of 2CNF clauses, $\Phi(\mathbf{v})$. This process is done by Algorithm 12.2.3.

From the original system and half-integral solution $\mathbf{d}$, we can construct a new system of UTVPI constraints, $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$, as follows:

Algorithm 12.2.3 MAKE-2CNF

Function MAKE-2CNF (System of UTVPI constraints S, feasible half-integer solution $\mathbf{d}$)

Let Φ denote the 2CNF formula corresponding to S.
for Every constraint $e \in S$ **do**
  if e is of the form $x_i + x_j \leq c_k$ **then**
    if $(d_i \notin \mathbb{Z} \wedge d_j \notin \mathbb{Z} \wedge c_k = d_i + d_j)$ **then**
      ▷ This becomes $x_i + x_j \leq 0$ in $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$.
      Add the clause $(\neg v_i \vee \neg v_j)$ to Φ.
  if e is of the form $x_i - x_j \leq c_k$ **then**
    if $(d_i \notin \mathbb{Z} \wedge d_j \notin \mathbb{Z} \wedge c_k = d_i - d_j)$ **then**
      ▷ This becomes $x_i - x_j \leq 0$ in $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$.
      Add the clause $(\neg v_i \vee v_j)$ to Φ.
  if e is of the form $-x_i + x_j \leq c_k$ **then**
    if $(d_i \notin \mathbb{Z} \wedge d_j \notin \mathbb{Z} \wedge c_k = -d_i + d_j)$ **then**
      ▷ This becomes $-x_i + x_j \leq 0$ in $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$.
      Add the clause $(v_i \vee \neg v_j)$ to Φ.
  if e is of the form $-x_i - x_j \leq c_k$ **then**
    if $(d_i \notin \mathbb{Z} \wedge d_j \notin \mathbb{Z} \wedge c_k = -d_i - d_j)$ **then**
      ▷ This becomes $-x_i - x_j \leq 0$ in $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$.
      Add the clause $(v_i \vee v_j)$ to Φ.
return Φ as a system of 2CNF clauses.

1. Replace each constraint $x_i + x_j \leq c_k$ with $x_i + x_j \leq c_k - (d_i + d_j)$.

2. Replace each constraint $x_i - x_j \leq c_k$ with $x_i - x_j \leq c_k - (d_i - d_j)$.

3. Replace each constraint $-x_i + x_j \leq c_k$ with $-x_i + x_j \leq c_k - (-d_i + d_j)$.

4. Replace each constraint $-x_i - x_j \leq c_k$ with $-x_i - x_j \leq c_k - (-d_i - d_j)$.

Note that, by construction, $\mathbf{c}' = \mathbf{c} - \mathbf{A} \cdot \mathbf{d} \geq \mathbf{0}$. This corresponds to the process of re-weighting difference constraints with a potential function.

We can now reduce the number of constraints by focusing on the constraints of the form $\pm x_i \pm x_j \leq 0$ in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$ such that $d_i, d_j \notin \mathbb{Z}$. Let $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$ be the system of these constraints.

From $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$, we can construct a system, $\Phi(\mathbf{v})$, of 2CNF clauses which is satisfiable if and only if $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ is integer feasible. $\Phi(\mathbf{v})$ also has the property that any proof of

unsatisfiability for $\Phi(\mathbf{v})$ can be easily converted into a proof of integer infeasibility of the same length for $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ and vice-versa.

Note that, $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$ and $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$ are not actually constructed by Algorithm 12.2.3. However, these systems are used to prove the correctness of the algorithms.

In $\Phi(\mathbf{v})$, each v_i corresponds to an x_i which assumes a non-integer value in the linear solution $\mathbf{d}$. v_i being **true** corresponds to x_i being rounded up, while **false** corresponds to x_i being rounded down. This action is performed by the final step of Algorithm 12.2.2.

If $\Phi(\mathbf{v})$ is feasible, then the values of each v_i correspond to a rounding needed to make an integral solution to the original system $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$. Similarly, if $\Phi(\mathbf{v})$ infeasible, then no such rounding is possible and $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ is integer infeasible.

12.2.1.1 Resource analysis

Algorithm 12.2.2 can be broken up into several parts. The complexity of each part can be considered independently.

1. First, Algorithm 12.2.2 finds a linear solution. This is accomplished by running Goldberg's Bit-Scaling Algorithm on the constraint network construction in [LM05]. This takes $O(\sqrt{n} \cdot m \cdot \log C)$ time [Gol95].

2. Then, Algorithm 12.2.3 converts the system into a system of 2CNF clauses. This consists of checking each constraint in the system and performing a series of constant time operations to generate the 2CNF clause. Thus, this takes $O(m)$ time.

3. Then, Algorithm 12.2.2 generates a feasible solution to the 2-SAT system or declares the system infeasible. This can be done in $O(n+m)$ time [BA79].

4. Finally, Algorithm 12.2.2 generates a feasible integer solution to the UTVPI system or declares the system not integer feasible. This is done by utilizing the 2-SAT solution and initial linear solution and runs in $O(n)$ time.

Thus, Algorithm 12.2.2 generates a feasible integer solution to the UTVPI system or declares the system not integer feasible in $O(\sqrt{n} \cdot m \cdot \log C)$ time.

12.2.1.2 Proof of correctness

We now establish the correctness of the reduction from linearly feasible UTVPI to 2-SAT.

We first show that the limitations on the constraints used in the reduction do not eliminate any proofs of integer infeasibility. Note that all proof in this section apply only to linearly feasible systems of UTVPI constraints.

***Theorem* 12.2.1.** *If $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ has a proof of integer infeasibility, then the constraints forming that proof correspond to constraints of the form $\pm x_i \pm x_j \leq 0$ in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$.*

Proof. Let $\mathbf{d}$ be a half-integral solution to $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$, and let x_i be a variable such that $d_i \notin \mathbb{Z}$. Since $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ is not integer feasible there exist no solutions with $x_i = \lceil d_i \rceil$ or $x_i = \lfloor d_i \rfloor$. Thus, we can derive the constraints $x_i + x_i \leq 2 \cdot d_i$ and $-x_i - x_i \leq -2 \cdot d_i$.

Consider the constraints in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ added together to obtain $x_i + x_i \leq 2 \cdot d_i$. When we add the corresponding constraints in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$ we obtain

$$x_i + x_i \leq 2 \cdot d_i - (d_i + d_i) = 0.$$

All constraints in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$ have $c'_k \geq 0$. Thus, every constraint involved in this new sum must have $c'_k = 0$. The same holds true for the constraints used to derive $-x_i - x_i \leq -2 \cdot d_i$. Thus, the constraints used to establish the integer infeasibility of $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ correspond to constraints in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$ such that $c'_k = 0$. □

***Theorem* 12.2.2.** *If $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ has a proof of integer infeasibility, then the constraints in that proof involve only variables x_j such that $d_j \notin \mathbb{Z}$.*

Proof. From Theorem 12.2.1, we have that every constraint involved in the proof of integer infeasibility must correspond to a constraint with $c'_k = 0$ in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$.

For any constraint $x_j + x_l \leq 0$ in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$, d_j and d_l must both be integral or both be non-integral. Otherwise,

$$c'_k = c_k \pm d_j \pm d_l \notin \mathbb{Z}.$$

A proof of integer infeasibility for $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ consists of establishing bounds on a variable x_i with $d_i \notin \mathbb{Z}$. Thus, all constraints in that proof must involve only variables x_j such that $d_j \notin \mathbb{Z}$. □

Together these two theorems imply that to find a proof of integer infeasibility we only need to focus on constraints in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$ such that $c'_k = 0$ and involving variables x_i for which d_i is non-integral.

***Theorem* 12.2.3.** *A 2CNF clause can be resolved from* $\Phi(\mathbf{v})$ *if and only if the corresponding UTVPI constraint can be derived from* $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$.

Proof. The inference rule used in the resolution of 2CNF clauses is

$$\frac{(l_i \vee l_j) \qquad\qquad (\neg l_j \vee l_k)}{(l_i \vee l_k)}$$

for literals l_i, l_j and l_k.

Let us consider the case where $l_i = v_i$, $l_j = v_j$, and $l_k = v_j$. If we look at the corresponding constraints in $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$, then we see that, in this case, the clauses correspond to the constraints $-x_i - x_j \leq 0$ and $x_j - x_k \leq 0$ yielding $-x_i - x_k \leq 0$. This is exactly what would be obtained from applying the transitive inference rule. It is easy to see that the reverse also holds.

The cases corresponding to the other possible assignments to the literals l_i, l_j, and l_k are handled similarly. □

We can now establish the correctness of the reduction.

***Theorem* 12.2.4.** $\Phi(\mathbf{v})$ *is satisfiable if and only if* $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ *has an integer solution.*

Proof. Assume that $\Phi(\mathbf{v})$ is unsatisfiable. Thus, we can derive the clauses (v_i) and $(\neg v_i)$ for some variable v_i. These clauses correspond to the constraints $x_i + x_i \leq 0$ and $-x_i - x_i \leq 0$. Thus, from Theorem 12.2.3, these constraints are derivable from $\mathbf{A}' \cdot \mathbf{x}' \leq \mathbf{0}$. Since d_i is an odd multiple of $\frac{1}{2}$, these correspond to the constraints

$$x_i + x_i \leq 2 \cdot d_i = 2 \cdot \lfloor d_i \rfloor + 1 \text{ and } -x_i - x_i \leq -2 \cdot d_i = -2 \cdot \lfloor d_i \rfloor - 1.$$

These constraints are derivable from $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$.

When we tighten these constraints, we get $x_i \leq \lfloor d_i \rfloor$ and $-x_i \leq -\lfloor d_i \rfloor - 1$. Summing these two constraints yields $0 \leq -1$. Thus, showing that $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ is infeasible.

Now assume that $\Phi(\mathbf{v})$ is satisfiable. Let $\mathbf{v}'$ be a boolean vector such that $\Phi(\mathbf{v}')$ is **true**. From $\mathbf{v}'$ and $\mathbf{d}$, we can construct the vector $\mathbf{r}$ as follows:

1. If $d_i \in \mathbb{Z}$, then set $r_i = 0$.
2. If $d_i \notin \mathbb{Z}$ and v'_i is **true**, then set $r_i = \frac{1}{2}$.
3. If $d_i \notin \mathbb{Z}$ and v'_i is **false**, then set $r_i = -\frac{1}{2}$.

We now show that $\mathbf{A} \cdot \mathbf{r} \leq \mathbf{c}'$. Let $\pm x_i \pm x_j \leq c'_k$ be a constraint in $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}'$. Let us examine all possible cases:

1. $c'_k \geq 1$: We have that $\pm r_i \pm r_j \leq \frac{1}{2} + \frac{1}{2} = 1 \leq c'_k$. Thus, the constraint is satisfied by $\mathbf{r}$.
2. $c'_k = 0$ and $d_i \in \mathbb{Z}$: From the proof of Theorem 12.2.2, we know that $d_j \in \mathbb{Z}$. Thus, $r_i = r_j = 0$ and $\pm r_i \pm r_j = 0 = c'_k$. Thus, the constraint is satisfied by $\mathbf{r}$.
3. $c'_k = 0$ and $d_i \notin \mathbb{Z}$: From the proof of Theorem 12.2.2, we know that $d_j \notin \mathbb{Z}$. In this case, we look at each possible constraint individually.
   (a) $x_i + x_j \leq 0$: By construction, the clause $(\neg v_i \vee \neg v_j)$ is in $\Phi(\mathbf{v})$. Thus, v'_i or v'_j

must be **false**. This means that $r_i = -\frac{1}{2}$ or $r_j = -\frac{1}{2}$. In either case, we have that

$$r_i + r_j \leq -\frac{1}{2} + \frac{1}{2} = 0 = c'_k.$$

Thus, the constraint is satisfied by $\mathbf{r}$.

(b) $x_i - x_j \leq 0$: By construction, the clause $(\neg v_i \vee v_j)$ is in $\Phi(\mathbf{v})$. Thus, v'_i must be **false** or v'_j must be **true**. This means that $r_i = -\frac{1}{2}$ or $r_j = \frac{1}{2}$. In either case, we have that

$$r_i - r_j \leq -\frac{1}{2} + \frac{1}{2} = 0 = c'_k.$$

Thus, the constraint is satisfied by $\mathbf{r}$.

(c) $-x_i + x_j \leq 0$: By construction, the clause $(v_i \vee \neg v_j)$ is in $\Phi(\mathbf{v})$. Thus, v'_i must be **true** or v'_j must be **false**. This means that $r_i = \frac{1}{2}$ or $r_j = -\frac{1}{2}$. In either case, we have that

$$-r_i + r_j \leq -\frac{1}{2} + \frac{1}{2} = 0 = c'_k.$$

Thus, the constraint is satisfied by $\mathbf{r}$.

(d) $-x_i - x_j \leq 0$: By construction, the clause $(v_i \vee v_j)$ is in $\Phi(\mathbf{v})$. Thus, v'_i or v'_j must be **true**. This means that $r_i = \frac{1}{2}$ or $r_j = \frac{1}{2}$. In either case, we have that

$$-r_i - r_j \leq -\frac{1}{2} + \frac{1}{2} = 0 = c'_k.$$

Thus, the constraint is satisfied by $\mathbf{r}$.

By the construction of $\mathbf{r}$, we have $\mathbf{d} + \mathbf{r} \in \mathbb{Z}^n$. We also have that $\mathbf{c}' = \mathbf{c} - \mathbf{A} \cdot \mathbf{d}$. Thus,

$$\mathbf{A} \cdot (\mathbf{d} + \mathbf{r}) = \mathbf{A} \cdot \mathbf{d} + \mathbf{A} \cdot \mathbf{r} \leq \mathbf{A} \cdot \mathbf{d} + \mathbf{c}' = \mathbf{c}.$$

This means that $(\mathbf{d} + \mathbf{r})$ is a valid integer solution to $\mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$. □

12.3 Refutability

12.3.1 Theorem of the alternative

In this section, we present a theorem of the alternative for integer infeasibility in UTVPI constraints.

As before, let $\mathbf{U} : \mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}$ denote a UCS and let $\mathbf{G} = \langle V, E, \mathbf{c} \rangle$ denote the corresponding constraint network.

Let $\mathbf{U}'$ denote the UCS constructed by adding selected absolute constraints of the form $x_i \leq c_i$ and $-x_i \leq c_i$, to $\mathbf{U}$.

Initially, $\mathbf{U}' = \mathbf{U}$. The constraint $x_i \leq c_i$, $c_i \in \mathbb{Z}$ is added to $\mathbf{U}'$ if, there exist constraints in $\mathbf{U}$, which can be summed to produce $x_i + x_i \leq (2 \cdot c_i + 1)$.

The constraint $-x_i \leq c_i$ where $c_i \in \mathbb{Z}$, is added to $\mathbf{U}'$ if, there exist two constraints in $\mathbf{U}$, which can be summed to produce $-x_i - x_i \leq (2 \cdot c_i + 1)$.

Let $\mathbf{X}'$ denote the set of feasible solutions to $\mathbf{U}'$. Let $\mathbf{G}' = \langle V', E' \rangle$ denote the constraint network representation of $\mathbf{U}'$.

We first prove a number of lemmata.

***Lemma* 12.3.1.** *If* $\mathbf{u}$ *is an integer point in* $\mathbf{X}$*, then* $\mathbf{u}$ *is an integer point in* $\mathbf{X}'$ *as well.*

Proof. Observe that every constraint in $\mathbf{U}'$ is either a constraint in $\mathbf{U}$ or an absolute constraint of the form $x_i \leq c_i$ or $-x_i \leq c_i$, which was added as per the discussion above. For our purposes, it suffices to show that any integer point satisfying the constraints in $\mathbf{U}$, satisfies all the absolute constraints that are in $\mathbf{U}'$, but not in $\mathbf{U}$.

Let the constraint $l_1 : x_j \leq c_j$, denote one such constraint, i.e., l_1 is in $\mathbf{U}'$, but not in $\mathbf{U}$. As per the construction of $\mathbf{U}'$, l_1 was added to $\mathbf{U}'$, because the constraint $x_j + x_j \leq (2 \cdot c_j + 1)$ is deducible from the constraints in $\mathbf{U}$. Thus, every integer point in $\mathbf{X}$, must also satisfy the constraint $x_j + x_j \leq (2 \cdot c_j + 1)$. It follows that every integer point in $\mathbf{X}$, also satisfies the constraint $x_j \leq \lfloor \frac{2 \cdot c_j + 1}{2} \rfloor = c_j$.

Since the constraint l_1 was picked arbitrarily, the same argument applies for every constraint of the form $x_i \leq c_i$, which is present in $\mathbf{U}'$. but not in $\mathbf{U}$. Furthermore, the above argument can easily be generalized to the case where the absolute constraint in question has the form $-x_i \leq c_i$ by changing the sign on all of the variables.

Based on the above discussion, it follows that the integer points in $\mathbf{X}$ satisfy all the constraints in $\mathbf{U}'$, i.e., they satisfy all the constraints in $\mathbf{U}$ and the additional absolute constraints that define $\mathbf{U}'$. It follows that every integer point in $\mathbf{X}$ is also an integer point in $\mathbf{X}'$. □

Observe that the converse of Lemma 12.3.1 is trivially true since $\mathbf{U}'$ is constructed by adding constraints to $\mathbf{U}$, and thus, every integer point in $\mathbf{X}'$ is also in $\mathbf{X}$.

***Lemma* 12.3.2.** *If $\mathbf{G}'$ has a negative weight gray cycle, then $\mathbf{X}$ contains no integer points.*

Proof. From [SW17b], we know that if $\mathbf{G}'$ contains a negative weight gray cycle, then $\mathbf{X}'$ is empty. It follows that $\mathbf{X}'$ does not contain any integer points. By Lemma 12.3.1, it follows that $\mathbf{X}$ cannot contain any integer points either. □

We have now shown that if $\mathbf{G}'$ contains a path from a vertex x_i to itself that can be reduced to a single gray edge of negative weight, then the constraint system $\mathbf{U}$ does not enclose an integer point. The following lemmata will help us establish the converse.

We need the following definition from [JMSY94].

***Definition* 12.3.1.** *The tightened transitive closure $\mathbf{U}^*$ of a UCS $\mathbf{U}$, is the set of* **all** *UTVPI constraints (including absolute constraints) that are derivable from $\mathbf{U}$ by the transitive and tightening inference rules.*

We make the following observations regarding the tightened transitive closure, $\mathbf{U}^*$:

1. The set of constraints in $\mathbf{U}$ is a subset of the set of constraints in $\mathbf{U}^*$.

2. Every constraint in $\mathbf{U}^*$ is either a constraint in $\mathbf{U}$, or obtained by the application of either the tightening rule or the transitive rule to constraints in $\mathbf{U}^*$.

3. $\mathbf{U}^*$ is closed under the transitive and tightening inference rules. This means that application of either the tightening rule or the transitive rule to constraints in $\mathbf{U}^*$ will not result in a UTVPI constraint that is not already in $\mathbf{U}^*$.

4. $\mathbf{U}$ has no integral solution, if and only if $\mathbf{U}^*$ contains a contradiction, i..e, a constraint of the form $0 \leq a$, where $a < 0$ [JMSY94].

***Lemma* 12.3.3.** *If* $\mathbf{X}$ *contains no integer points, then* $\mathbf{X}'$ *is empty, i.e.,* $\mathbf{X}'$ *is linearly infeasible.*

Proof. In this proof, we will establish that the set $\mathbf{X}'$ of solutions to the constraint system $\mathbf{U}'$ (constructed as per the discussion prior to Theorem 12.3.1) is a subset of $\mathbf{X}^*$, where $\mathbf{X}^*$ is the set of all solutions to $\mathbf{U}^*$, the tightened transitive closure of $\mathbf{U}$. The lemma follows.

We first observe that every constraint in $\mathbf{U}^*$ is obtained by applications of the transitive and tightening inference rules detailed above. As stated previously, applications of the transitive inference rule correspond to edge reductions and so the constraints added in this fashion to $\mathbf{U}^*$, do not affect its linear feasibility. Thus, we are only concerned with the constraints added through application of the tightening inference rule. We will now show that each constraint added to $\mathbf{U}^*$, as a result of applying the tightening inference rule is also added to $\mathbf{U}'$.

We note that only absolute constraints are added to $\mathbf{U}^*$ through the application of the tightening inference rule. Assume that the constraint $x_i \leq c_i$ is added to $\mathbf{U}^*$ by the tightening rule. It follows that either $x_i + x_i \leq 2 \cdot c_i + 1$ or $x_i + x_i \leq 2 \cdot c_i$ is deducible from the original set of constraints, $\mathbf{U}$. In the first case, the constraint $x_i \leq c_i$ is, by definition, in $\mathbf{U}'$. In the second case, the constraint $x_i \leq c$ is equivalent to the constraint $x_i + x_i \leq 2 \cdot c_i$. Thus, this constraint does not remove any solutions from $\mathbf{X}$.

Likewise, assume that the constraint $-x_i \leq c_i$ is added to $\mathbf{U}^*$ by the tightening rule. It follows that $-x_i - x_i \leq 2 \cdot c_i + 1$ or $-x_i - x_i \leq 2 \cdot c_i$ is deducible from the original constraints, $\mathbf{U}$. In the first case, the constraint $-x_i \leq c$ is, by definition, in $\mathbf{U}'$. In the second case, the constraint $-x_i \leq c$ is equivalent to the constraint $-x_i - x_i \leq 2 \cdot c_i$. Thus, this constraint does

not remove any solutions from $\mathbf{X}$.

We can thus conclude that every constraint added to $\mathbf{U}^*$ that removes solutions from $\mathbf{X}$ is also added to $\mathbf{U}'$. It follows that any vector $\mathbf{u}$ that satisfies $\mathbf{U}'$, also satisfies $\mathbf{U}^*$, i.e., $\mathbf{X}' \subseteq \mathbf{X}^*$. Finally, we note that if $\mathbf{X}$ contains no integer points, then $\mathbf{X}^*$ is empty [JMSY94]. Inasmuch as $\mathbf{X}' \subseteq \mathbf{X}^*$, $\mathbf{X}'$ is also empty. □

***Lemma* 12.3.4.** *If* $\mathbf{X}$ *contains no integer points, then* $\mathbf{G}'$ *contains a path from a vertex* x_i *to itself that can be reduced to a single gray edge of negative weight.*

Proof. By Lemma 12.3.3, if $\mathbf{X}$ contains no integer points, then $\mathbf{X}'$ is empty. Thus, $\mathbf{G}'$ contains a negative weight gray cycle from a vertex x_i to itself. However, such a cycle can be reduced to a single gray edge of negative weight. The lemma follows. □

Having established the preceding lemmata, we now prove the following result.

***Theorem* 12.3.1.** *Either the constraint system* $\mathbf{U}$ *encloses an integer point or (mutually exclusively),* $\mathbf{G}'$ *contains a path from a vertex* x_i *to itself that can be reduced to a single gray edge of negative weight.*

Proof. As shown in Lemma 12.3.2, if $\mathbf{G}'$ contains a path from a vertex x_i to itself that can be reduced to a single gray edge of negative weight, then $\mathbf{X}$ and hence $\mathbf{U}$ do not contain any integer points. Likewise, as per Lemma 12.3.4, if $\mathbf{X}$ contains no integer points, then $\mathbf{G}'$ contains precisely such a path. □

Next we express this result in terms of the original constraint network $\mathbf{G}$. Note that a negative weight gray cycle in $\mathbf{G}$ means that $\mathbf{X}$ is empty [SW17b]. Thus, $\mathbf{X}$ trivially contains no integer points. As a result, in the following lemma we assume that $\mathbf{G}$ has no negative weight gray cycles.

***Lemma* 12.3.5.** $\mathbf{X}$ *contains no integer points if and only if* $\mathbf{U}' \setminus \mathbf{U}$ *contains the constraints* $x_i \leq c_i$ *and* $-x_i \leq -c_i - 1$ *for some* x_i.

Proof. The two constraints sum to produce the constraint $0 \leq -1$. This is clearly a contradiction. Thus, if $\mathbf{U}' \setminus \mathbf{U}$ has the desired constraints, then $\mathbf{X}$ has no integer points.

If $\mathbf{X}$ contains no integer points, then by Lemma 12.3.4, $\mathbf{G}'$ contains a negative weight gray cycle. If $\mathbf{G}'$ has a negative weight gray cycle that uses only edges corresponding to constraints in $\mathbf{U}$, then $\mathbf{G}$ has the same negative weight gray cycle. Thus, any negative weight gray cycle cycle in $\mathbf{G}'$ must use edges corresponding to constraints in $\mathbf{U}' \setminus \mathbf{U}$.

All the constraints in $\mathbf{U}' \setminus \mathbf{U}$ are absolute constraints. Thus, any negative weight gray cycle in $\mathbf{G}'$ must use x_0. Since at least two edges of such a cycle must use x_0, any negative weight gray cycle in $\mathbf{G}'$ must use at least two edges corresponding to absolute constraints.

Let p be the negative weight gray cycle in $\mathbf{G}'$ with the fewest edges. Assume without loss of generality that p contains an edge corresponding to the constraint $x_i \leq c_i \in \mathbf{U}' \setminus \mathbf{U}$ for some x_i. If p has an edge that does not correspond to an absolute constraint, then p must have an edge corresponding to a constraint of the form $-x_i + a_j \cdot x_j \leq c_{ij}$ where $a_j \in \{-1, 1\}$.

The constraint $x_i \leq c_i$ is in $\mathbf{U}' \setminus \mathbf{U}$. Thus, the constraint $x_i + x_i \leq 2 \cdot c_i + 1$ is derivable from $\mathbf{U}$ by the transitive inference rule. Since the constraint $-x_i + a_j \cdot x_j \leq c_{ij}$ is in $\mathbf{U}$, the constraint $a_j \cdot x_j + a_j \cdot x_j \leq 2 \cdot (c_{ij} + c_i) + 1$ is derivable from $\mathbf{U}$ by the transitive inference rule. Thus the constraint $a_j \cdot x_j \leq c_{ij} + c_i$ is in $\mathbf{U}'$. Thus, we can construct a new negative weight gray cycle by replacing the edges in p corresponding to the constraints $x_i \leq c_i$ and $-x_i + a_j \cdot x_j \leq c_{ij}$ with the edge corresponding to the constraint $a_j \cdot x_j \leq c_{ij} + c_i$.

However, this means that there is a negative weight gray cycle in $\mathbf{G}'$ with fewer edges than p. Thus, p cannot have any edges corresponding to non-absolute constraints. This means that the edges in p correspond to the constraints $x_i \leq c_{i1}$ and $-x_i \leq c_{i2}$ where $c_{i1} + c_{i2} < 0$ for some x_i. Without loss of generality assume that $x_i \leq c_{i1} \in \mathbf{U}' \setminus \mathbf{U}$.

Thus, the constraint $x_i + x_i \leq 2 \cdot c_{ij} + 1$ is derivable from $\mathbf{U}$ by the transitive inference rule. If the constraint $-x_i \leq c_{i2} \in \mathbf{U}$, then the constraint

$$0 = x_i + x_i - 2 \cdot x_i \leq 2 \cdot c_{ij} + 1 + 2 \cdot c_{i2} = 2 \cdot (c_{ij} + c_{i2}) + 1 \leq -2 + 1 = -1$$

is derivable from $\mathbf{U}$ by the transitive inference rule. However, this means that $\mathbf{U}$ is infeasible and that $\mathbf{G}$ has a negative weight gray cycle [SW17b]. Thus, we must have that $-x_i \leq c_{i2} \in \mathbf{U}' \setminus \mathbf{U}$. Thus, $-x_i - x_i \leq -2 \cdot c_{i2} + 1$ is derivable from $\mathbf{U}$ by the transitive inference rule. Since $\mathbf{U}$ is feasible, we must have that $0 \leq -c_{i1} - c_{i2} + 1$. Thus, we have that $c_{i1} + c_{i2} = -1$ as desired. □

***Theorem* 12.3.2.** *Either the constraint system* $\mathbf{U}$ *encloses an integer point or (mutually exclusively),*

1. $\mathbf{G}$ *contains a path from a vertex* x_i *to itself that can be reduced to a single gray edge of negative weight or*

2. $\mathbf{G}$ *contains a white path of odd weight from* x_i *to itself and a black path of odd weight from* x_i *to itself with total weight* 0*.*

Proof. If $\mathbf{G}$ contains a path of type (a), then $\mathbf{U}$ contains no rational points and thus no integer points [SW17b].

If $\mathbf{G}$ contains a path of type (b), then for some c_i the constraints $x_i + x_i \leq 2 \cdot c_i + 1$ and $-x_i - x_i \leq -2 \cdot c_i - 1$ are derivable from $\mathbf{U}$ by the transitive inference rule [SW17b]. Thus, the constraints $x_i \leq c_i$ and $-x_i \leq -c_i - 1$ are in $\mathbf{U}'$. This means that $\mathbf{U}'$ (and thus $\mathbf{U}$) contains no integer points.

If $\mathbf{U}$ does not contain an integer point, then there are two cases, either $\mathbf{U}$ contains no rational points or it contains rational, but no integer points.

If $\mathbf{U}$ contains no rational points, then $\mathbf{G}$ must have a path of type (a) [SW17b].

If $\mathbf{U}$ contains rational but no integer points, then by Lemma 12.3.5, $\mathbf{U}' \setminus \mathbf{U}$ contains the constraints $x_i \leq c_i$ and $-x_i \leq -c_i - 1$ for some x_i. Thus, the constraints $x_i + x_i \leq 2 \cdot c_i + 1$ and $-x_i - x_i \leq -2 \cdot c_i - 1$ are derivable from $\mathbf{U}$ by the transitive inference rule. The constraints used to derive $x_i + x_i \leq 2 \cdot c_i + 1$ correspond to a white path of weight $(2 \cdot c_i + 1)$ from x_i to itself [SW17b]. Note that this path has odd weight. The constraints used to derive $-x_i - x_i \leq -2 \cdot c_i - 1$ correspond to a black path of weight $(-2 \cdot c_i - 1)$ from x_i to itself

[SW17b]. Note that this path also has odd weight The total weight of these paths is 0. Thus, **G** has a path of type (b). □

Note that our graphical theorem of the alternative differs from the one in [LM05] in the following ways:

1. Our theorem of the alternative utilizes the constraint network from [SW17b], not the constraint network in [Min06].

2. Our theorem of the alternative is based on the constraint network corresponding to the original system without the need for constraints derived from the tightening inference rule. While the algorithm in [LM05] does use the unmodified graph, this is not explicitly given as a theorem of the alternative.

Our graphical theorem of the alternative cannot be used to find the proof of integer infeasibility with the fewest inferences.

Example 43: Consider the following system of UTVPI constraints:

$$\begin{array}{llllllll} x_1+x_2 & \leq & -1 & x_1-x_2 & \leq & 0 & -x_1-x_3 & \leq & 0 & x_3-x_4 & \leq & 0 \\ x_4-x_5 & \leq & 1 & x_4+x_5 & \leq & 0 & x_5-x_6 & \leq & 0 & x_6-x_1 & \leq & 0 \end{array} \tag{12.2}$$

The constraint network corresponding to System (12.2) can be seen in Figure 12.2.

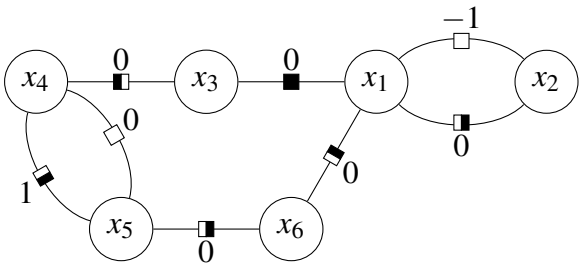

Figure 12.2: Constraint network corresponding to System (12.2) (without vertex x_0)

The shortest pair of cycles is the white cycle $(x_1 \overset{-1}{\square} x_2 \overset{0}{\blacksquare} x_1)$ and the black cycle $(x_1 \overset{0}{\blacksquare} x_3 \overset{0}{\blacksquare} x_4 \overset{1}{\blacksquare} x_5 \overset{0}{\blacksquare} x_6 \overset{0}{\blacksquare} x_1)$. There are 7 edges in total. This corresponds to the following

proof of infeasibility

1. $\dfrac{x_1 + x_2 \leq -1 \qquad x_1 - x_2 \leq 0}{x_1 \leq -1}$
2. $\dfrac{-x_1 - x_3 \leq 0 \qquad x_3 - x_4 \leq 0}{-x_1 - x_4 \leq 0}$
3. $\dfrac{-x_1 - x_4 \leq 0 \qquad x_4 - x_5 \leq 1}{-x_1 - x_5 \leq 1}$
4. $\dfrac{-x_1 - x_5 \leq 1 \qquad x_5 - x_6 \leq 0}{-x_1 - x_6 \leq 1}$
5. $\dfrac{-x_1 - x_6 \leq 1 \qquad x_6 - x_1 \leq 0}{-x_1 \leq 0}$
6. $\dfrac{-x_1 \leq 0 \qquad x_1 \leq -1}{0 \leq -1}$

This proof of infeasibility consists of 6 inferences.

However, we can look at the white cycle $(x_1 \overset{-1}{\square} x_2 \overset{0}{\blacksquare} x_1)$ and the black cycle $(x_1 \overset{0}{\blacksquare} x_3 \overset{0}{\blacksquare} x_4 \overset{1}{\blacksquare} x_5 \overset{0}{\square} x_4 \overset{0}{\blacksquare} x_3 \overset{0}{\blacksquare} x_1)$. There are 8 edges in total. This corresponds to the following proof of infeasibility

1. $\dfrac{x_1 + x_2 \leq -1 \qquad x_1 - x_2 \leq 0}{x_1 \leq -1}$
2. $\dfrac{x_4 - x_5 \leq 1 \qquad x_4 + x_5 \leq 0}{x_4 \leq 0}$
3. $\dfrac{x_3 - x_4 \leq 0 \qquad x_4 \leq 0}{x_3 \leq 0}$
4. $\dfrac{-x_1 - x_3 \leq 0 \qquad x_3 \leq 0}{-x_1 \leq 0}$
5. $\dfrac{-x_1 \leq 0 \qquad x_1 \leq -1}{0 \leq -1}$

This proof of infeasibility consists of only 5 inferences.

12.3.1.1 Integer feasibility algorithm

In this section, we convert the insights of the previous section into an algorithm for deciding integer feasibility in UTVPI constraints. This algorithm complements existing integer feasibility algorithms for UTVPI constraints [Min06, LM05, SW17a].

Let **U** be a UCS and let **G** be the corresponding constraint network. First the algorithm checks to see if **G** has a negative weight gray cycle (path of type (a)). This can be done in $O(m \cdot n)$ time by the algorithm described in [SW17b]. Note that this algorithm will return either a path of type (a) or a feasible linear solution $\mathbf{r}$.

If **G** has a path of type (a) we declare the system infeasible and return path as proof of infeasibility. If **G** does not have a path of type (a), then we use the linear solution returned by the algorithm to construct a re-weighted graph $\mathbf{G}'$ from **G** as follows:

1. Each edge of the form $x_i \overset{c_{ij}}{\square} x_j$ becomes $x_i \overset{c_{ij}-r_i-r_j}{\square} x_j$.
2. Each edge of the form $x_i \overset{c_{ij}}{\blacksquare} x_j$ becomes $x_i \overset{c_{ij}+r_i-r_j}{\blacksquare} x_j$.
3. Each edge of the form $x_i \overset{c_{ij}}{\blacksquare} x_j$ becomes $x_i \overset{c_{ij}-r_i+r_j}{\blacksquare} x_j$.
4. Each edge of the form $x_i \overset{c_{ij}}{\blacksquare} x_j$ becomes $x_i \overset{c_{ij}+r_i+r_j}{\blacksquare} x_j$.

Since $\mathbf{r}$ is a feasible solution to **U**, the weight of each edge in $\mathbf{G}'$ is non-negative. Since any path of type (b) has total weight 0, the corresponding path in $\mathbf{G}'$ must also have total weight 0. Thus, every edge in that path in $\mathbf{G}'$ has weight 0.

If the white sub-path of the path of type (b) has total weight $(2 \cdot c_i + 1)$, then this sub-path corresponds to a constraint of the form $x_i + x_i \leq 2 \cdot c_i + 1$. Additionally, the black sub-path must have total weight $(-2 \cdot c_i - 1)$ and the this sub-path corresponds to a constraint of the form $-x_i - x_i \leq -2 \cdot c_i - 1$. Together these constraints force $x_i = c_i + \frac{1}{2}$. Thus, we have that $r_i = c_i + \frac{1}{2} \notin \mathbb{Z}$.

Thus, to find a path of type (b) in **G**, we need to find an x_i such that:

1. $r_i \notin \mathbb{Z}$

2. x_i is reachable from itself in $\mathbf{G}'$ by a white path using only 0 weight edges.

3. x_i is reachable from itself in $\mathbf{G}'$ by a black path using only 0 weight edges.

For each x_i, a reachability algorithm can be used to determine if the desired white and black paths exist. Thus, it takes $O(m+n)$ time to check if a single x_i satisfies all of these conditions. Thus, the total running time of this part of the algorithm is $O(m \cdot n)$.

Thus, integer feasibility can be established, with proof of infeasibility, in $O(m \cdot n)$ time. Note that unlike the algorithm in [SW17a], this algorithm does not return an integral solution if $\mathbf{U}$ is integer feasible.

12.4 Closure

12.4.1 The closure problem (ADD and DIV rules)

In this section, we describe an algorithm for finding the integer closure of certain systems of UTVPI constraints.

We first describe a modified version of Dijkstra's shortest path algorithm, adapted to handle the network construction introduced in [SW17b]. Dijkstra's shortest path algorithm is a well-known method for solving the single source shortest path problem in graphs with non-negative edge weights. It runs in $O(m+n \cdot \log n)$ time [CLRS01].

In Algorithm 12.4.1 we utilize the weight function $c(e)$ defined in Section 2.2.2.

Algorithm 12.4.1 operates on the same basic principles as Dijkstra's shortest path algorithm. However, it has been adapted to utilize the graph construction described in Section 2.2.2. We use it to find the shortest white, black, and gray paths from x_i to every vertex x_j in $\mathbf{G}$. This gives us $\max(x_i + x_j)$, $\max(-x_i - x_j)$, $\max(x_i - x_j)$, $\max(-x_i + x_j)$, $\max(2 \cdot x_i)$, and $\max(-2 \cdot x_i)$. See Section 12.4.1.1 for a proof of this statement.

If a finite upper bound cannot be established on one of these values, then it is given the default value of ∞. Just like Dijkstra's shortest path algorithm, Algorithm 12.4.1 assumes

Algorithm 12.4.1 UTVPI-DIJKSTRA

Function UTVPI-DIJKSTRA (network **G**, start vertex x_i)

1: Create array $D[x_j,t]$ of distance labels for $t \in \{ \square , \blacksquare , \blacksquarel , \blacksquarer \}$. ▷ Note that $D[x_j,t]$ is the length of a shortest path of type t between x_i and x_j.
2: Create queue of unvisited vertex label pairs Q.
3: **for** (each x_j) **do**
4: $D[x_j,t] \leftarrow \infty$ for $t \in \{ \square , \blacksquare , \blacksquarel , \blacksquarer \}$.
5: Add (x_j,t) to Q for $t \in \{ \square , \blacksquare , \blacksquarel , \blacksquarer \}$.
6: $D[x_i,t] \leftarrow 0$ for $t \in \{ \blacksquarel , \blacksquarer \}$.
7: **while** ($Q \neq \emptyset$) **do**
8: $(y,t) \leftarrow \arg\min_{(x,t)\in Q}(D[x,t])$.
9: **for** (each neighbor x of y) **do**
10: **if** ($t = \square$) **then**
11: ▷ Perform all *valid* edge reductions that start with a white edge.
12: ▷ A table of these reductions can be found in [SW15a].
13: **if** ($D[x, \square] > D[y, \square] + c(y \blacksquarer x)$) **then** ▷ Reduce $x_i \square y \blacksquarer x$ to $x_i \square x$.
14: $D[x, \square] \leftarrow D[y, \square] + c(y \blacksquarer x)$.
15: **if** ($D[x, \blacksquarel] > D[y, \square] + c(y \blacksquare x)$) **then** ▷ Reduce $x_i \square y \blacksquare x$ to $x_i \blacksquarel x$.
16: $D[x, \blacksquarel] \leftarrow D[y, \square] + c(y \blacksquare x)$.
17: $t \in \{ \blacksquare , \blacksquarel , \blacksquarer \}$ are handled according to the rules in [SW15a].
18: Remove (y,t) from Q.
19: **return** Distance labels D.

that all edge weights are non-negative.

If we are given a system of pure difference constraints, then we cannot derive $\max(x_i + x_j)$, $\max(-x_i - x_j)$, $\max(x_i)$, or $\max(-x_i)$ for any x_i, x_j. Thus, for these bounds, Algorithm 12.4.1 returns a value of ∞.

We find the integer closure of a system of UTVPI constraints by converting the system into an equivalent one in each constraint has a non-negative right hand side. We also require that each constraint derivable from the tightening inference rule is included. We run Algorithm 12.4.2 on the modified graph to obtain the integer closure of the original system. This graph conversion is described in Section 12.4.1.1.

The constraints derived by the tightening rule need to be included. Otherwise, the bounds generated only apply to linear solutions, with integer solutions possibly requiring tighter bounds.

Example 44: Let us consider the system, $x_1 + x_2 \leq 1$, $x_1 - x_2 \leq 0$, and $x_2 - x_1 \leq 0$. Without tightening constraints, the best upper bound on $(x_1 + x_2)$ that we can obtain by simply adding constraints is $x_1 + x_2 \leq 1$. However, this bound is only satisfied with equality when $x_1 = x_2 = \frac{1}{2}$. Since no integer solution satisfies this bound with equality, this is not the tightest bound we can obtain.

Instead, if we add the constraints derived by the tightening inference rule, then we see that the two constraints $x_1 \leq 0$ and $x_2 \leq 0$ would be added. This would make the new best upper bound on $(x_1 + x_2)$, $x_1 + x_2 \leq 0$. This bound can be satisfied with equality when $x_1 = x_2 = 0$, which is an integer solution to the original system.

Once we get the system into this form we run Algorithm 12.4.2. From this, we obtain the tightest bounds on each possible UTVPI constraint.

Algorithm 12.4.2 UTVPI-JOHNSON

Function UTVPI-JOHNSON (set **U** of UTVPI constraints)

1: Construct constraint network **G** from **U** according to the rules in Section 2.2.2.
2: Create array $B[x_i, x_j, t]$ of bounds. ▷ Note that $B[x_i, x_j, 0]$ represents $\max(x_i + x_j)$, $B[x_i, x_j, 1]$ represents $\max(x_i - x_j)$, $B[x_i, x_j, 2]$ represents $\min(x_i + x_j)$, and $B[x_i, x_j, 3]$ represents $\min(x_i - x_j)$.
3: Create array $D[x_i]$ of distance labels.
4: **for** (each x_j) **do**
5: $D \leftarrow$UTVPI-DIJKSTRA(G, x_i). ▷ Run Algorithm 12.4.1 from every vertex.
6: **for** (each x_i) **do** ▷ Store the results in B.
7: $B[x_i, x_j, 0] \leftarrow D[x_i, \square]$.
8: $B[x_i, x_j, 1] \leftarrow D[x_i, \blacksquare]$.
9: $B[x_i, x_j, 2] \leftarrow -D[x_i, \blacksquare]$. ▷ Convert $\max(-x_i - x_j)$ to $\min(x_i + x_j)$.
10: $B[x_i, x_j, 3] \leftarrow -D[x_i, \blacksquare]$. ▷ Convert $\max(-x_i + x_j)$ to $\min(x_i - x_j)$.
11: **return** Array of bounds B.

12.4.1.1 The new algorithm

Algorithm 12.4.2 provides a method for obtaining the integer closure when all constraints have non-negative constants, and all results of the tightening inference rule have been found.

To get all constraints to have non-negative constants, it suffices to find an integer solution **d** to the system, and to adjust the constraints accordingly.

That is if the values for x_i and x_j are d_i and d_j, then the adjusted version of the constraint $x_i + x_j \leq c_{ij}$ would be $x_i + x_j \leq c_{ij} - (d_i + d_j)$. Since $d_i + d_j \leq c_{ij}$ we have that $c_{ij} - (d_i + d_j) \geq 0$.

Similarly, $x_i - x_j \leq c_{ij}$ would become $x_i - x_j \leq c_{ij} - (d_i - d_j)$, $-x_i + x_j \leq c_{ij}$ would become $-x_i + x_j \leq c_{ij} - (-d_i + d_j)$, and $-x_i - x_j \leq c_{ij}$ would become $-x_i - x_j \leq c_{ij} - (-d_i - d_j)$.

Example 45: Consider the UTVPI constraint $x_1 + x_2 \leq -5$. We have that the vector $(x_1, x_2) = (-3, -3)$ satisfies this constraint. If this is used as the initial valid solution, then this constraint would become $x_1 + x_2 \leq 1$ in the new system.

To account for all constraints generated by the tightening inference rule, we need to find the tightest bounds on $(x_i + x_i)$ and $(-x_i - x_i)$ for each x_i.

Once these are found, if the tightest bound on $(x_i + x_i)$ corresponds to a constraint $x_i + x_i \leq 2 \cdot a + 1$ for some integer a, then the constraint $x_i \leq a$ can be added to the system as a result of the tightening inference rule.

To make this process easier, finding these bounds can be performed on the adjusted graph where each constraint has non-negative constant. The proof of this is in Section 12.4.1.1.

We need to reduce an arbitrary system of UTVPI constraints to a system in which each constraint has non-negative constant, and all constraints added by the tightening rule are included. This is done in Algorithm 12.4.3.

Analysis of Running Time

Algorithm 12.4.3 consists of two portions. The first converts a general system of UTVPI constraints into one that can be accepted by Algorithm 12.4.2. Then Algorithm 12.4.2 is run on this modified graph. We shall analyze these two portions separately.

Converting the graph into the form required by our fast integer closure algorithm consists of

Algorithm 12.4.3 UTVPI-CLOSURE

Function UTVPI-CLOSURE (set **U** of UTVPI constraints)

1: Find an integer solution **d** to **U**.
2: Create system of UTVPI constraints $\mathbf{U}'$.
3: Create array $B[x_i, x_j, t]$ of bounds. ▷ Note that $B[x_i, x_j, 0]$ represents $\max(x_i + x_j)$, $B[x_i, x_j, 1]$ represents $\max(x_i - x_j)$, $B[x_i, x_j, 2]$ represents $\min(x_i + x_j)$, and $B[x_i, x_j, 3]$ represents $\min(x_i - x_j)$.
4: Create array $D[x_i]$ of distance labels.
5: **for** (each constraint e in **U**) **do**
6: **if** (e is of the form $(x_i + x_j) \leq c_{ij}$) **then**
7: Add the constraint $(x_i + x_j) \leq (c_{ij} - d_i - d_j)$ to $\mathbf{U}'$.
8: **if** (e is of the form $(x_i - x_j) \leq c_{ij}$) **then**
9: Add the constraint $(x_i - x_j) \leq (c_{ij} - d_i + d_j)$ to $\mathbf{U}'$.
10: **if** (e is of the form $(-x_i + x_j) \leq c_{ij}$) **then**
11: Add the constraint $(-x_i + x_j) \leq (c_{ij} + d_i - d_j)$ to $\mathbf{U}'$.
12: **if** (e is of the form $(-x_i - x_j) \leq c_{ij}$) **then**
13: Add the constraint $(-x_i - x_j) \leq (c_{ij} + d_i + d_j)$ to $\mathbf{U}'$.
14: Construct constraint network **G** from $\mathbf{U}'$ according to the rules in Section 2.2.2.
15: **for** (each x_i) **do**
16: $D \leftarrow$UTVPI-DIJKSTRA(G, x_i).
17: **if** ($D[x_i, \square]$ is odd) **then** ▷ $\max(2 \cdot x_i)$ is odd.
18: Add the constraint $x_i \leq \lfloor \frac{D[x_i, \square]}{2} \rfloor$ to $\mathbf{U}'$.
19: **if** ($D[x_i, \blacksquare]$ is odd) **then** ▷ $\max(-2 \cdot x_i)$ is odd.
20: Add the constraint $-x_i \leq \lfloor \frac{D[x_i, \blacksquare]}{2} \rfloor$ to $\mathbf{U}'$.
21: $B \leftarrow$ UTVPI-JOHNSON($\mathbf{U}'$).
22: **for** (each x_i) **do**
23: **for** (each x_j) **do**
24: $B[x_i, x_j, 0] \leftarrow B[x_i, x_j, 0] + d_i + d_j$.
25: $B[x_i, x_j, 1] \leftarrow B[x_i, x_j, 1] + d_i - d_j$.
26: $B[x_i, x_j, 2] \leftarrow B[x_i, x_j, 2] + d_i + d_j$.
27: $B[x_i, x_j, 3] \leftarrow B[x_i, x_j, 3] + d_i - d_j$.
28: **return** Array of bounds B.

- Finding an integer solution: Using the algorithm described in [LM05] this can be done in $O(m \cdot n + n^2 \cdot \log n)$ time.
- Re-weighting the graph: since every edge needs to be re-weighted this portion runs in $O(m)$ time.
- Determining which edges need to be tightened: This consists of n runs of UTVPI-DIJKSTRA(), and so takes $O(m \cdot n + n^2 \cdot \log n)$ time.
- Adding tightened edges: for each x_i we add at most 2 new edges, and so this portion runs in $O(n)$ time.

Thus, the graph conversion procedure runs in $O(m \cdot n + n^2 \cdot \log n)$ time.

Algorithm 12.4.2 which computes the integer closure for certain types of UTVPI systems consists of

- Creating the graph: since each edge and vertex of the graph need to be created, this process takes $O(m+n)$ time.
- Computing the transitive closure of the tightened graph: This consists of n runs of Algorithm 12.4.1, and so takes $O(m \cdot n + n^2 \cdot \log n)$ time.
- Determining $\min(x_i - x_j)$, $\max(x_i - x_j)$, $\max(x_i + x_j)$, $\min(x_i + x_j)$, $\max(x_i)$, and $\min(x_i)$ for each x_i and x_j: There are $O(n^2)$ bounds that need to be determined, so this portion of the algorithm runs in $O(n^2)$ time.

Thus, Algorithm 12.4.2 and the entire integer closure procedure run in $O(m \cdot n + n^2 \cdot \log n)$ time.

Correctness

We first show that Algorithm 12.4.2 generates the correct bounds on integer solutions to **U**. This is done by showing that each of the desired bounds can be obtained from applying the transitive and tightening inference rules.

***Lemma* 12.4.1.** *The bound $x_i - x_j \geq \min(x_i - x_j)$ can be obtained through repeated applications of the transitive and tightening inference rules.*

Proof. Let $c_{ij} = \min(x_i - x_j)$. Thus, there exists a valid solution to $\mathbf{U}$, say $\mathbf{x}'$, such that $x'_i - x'_j \leq c_{ij}$. However, for any $c < c_{ij}$ there is no such value of $\mathbf{x}$. This means that $\mathbf{U} \cup \{x'_i - x'_j \leq c_{ij}\}$ is feasible but $\mathbf{U} \cup \{x'_i - x'_j \leq c_{ij} - 1\}$ is infeasible.

Since this second system is infeasible, repeated applications of the transitive and tightening inference rules are able to produce the constraint $0 \leq b < 0$ [JMSY94].

Note that this proof of infeasibility must use the constraint $x'_i - x'_j \leq c_{ij} - 1$. Thus, removing $x'_i - x'_j \leq c_{ij} - 1$ from the proof of infeasibility, results in a derivation of the constraint $x_i - x_j \geq c'_{ij}$ such that $c_{ij} - 1 < c'_{ij} \leq c_{ij}$. Since c'_{ij} and c_{ij} are both integers, $c'ij = c_{ij}$.

Thus, through repeated applications of the transitive and tightening inference rules we are able to derive the constraint $x_i - x_j \geq \min(x_i - x_j)$. □

The ability to construct the remaining bounds is shown without proof since they are analogous to the proof of Lemma 12.4.1.

The distance labels, $D[x_j, t]$, in Algorithm 12.4.1 are computed by adding edge weights. This corresponds to applications of the transitive inference rule. Thus, we have that Algorithm 12.4.1 always generates valid bounds.

***Lemma* 12.4.2.** *After running Algorithm 12.4.1 from x_i, we have the following:*

(1) $\max(x_i + x_j) = D[x_j, \square]$.

(2) $\max(x_i - x_j) = D[x_j, \blacksquare]$.

(3) $\min(x_i + x_j) = -D[x_j, \blacksquare]$.

(4) $\min(x_i - x_j) = -D[x_j, \blacksquare]$.

(5) $\max(x_i) = \lfloor \frac{D[x_i, \square]}{2} \rfloor$.

(6) $\min(x_i) = -\lfloor \frac{D[x_i, \blacksquare]}{2} \rfloor$.

Proof. The first four cases are handled similarly.

We have that $D[x_j, \blacksquare]$, is the length of the shortest gray path from x_i to x_j in $\mathbf{U}'$. This means that the constraint $-x_i + x_j \leq D[x_j, \blacksquare]$ is derivable from $\mathbf{U}'$ but no constraint of the

form $-x_i + x_j \leq b_{ij} < D[x_j, \blacksquare]$ is derivable from $\mathbf{U}'$.

Thus, the constraint $x_i - x_j \geq -D[x_j, \blacksquare]$ is derivable from $\mathbf{U}'$ but no constraint of the form $x_i - x_j \geq b_{ij} > D[x_j, \blacksquare]$ is derivable from $\mathbf{U}'$. From Lemma 12.4.1, we must have that $\min(x_i - x_j) = -D[x_j, \blacksquare]$. The remaining cases are handled similarly.

Thus, we must have that

$$\max(x_i) = \lfloor \frac{\max(x_i + x_i)}{2} \rfloor = \lfloor \frac{D[x_i, \square]}{2} \rfloor,$$

and

$$\min(x_i) = \lceil \frac{\min(x_i + x_i)}{2} \rceil = \lceil \frac{-D[x_i, \blacksquare]}{2} \rceil = -\lfloor \frac{D[x_i, \blacksquare]}{2} \rfloor.$$

□

As a direct result of Lemma 12.4.2, we have the following.

***Corollary* 12.4.1.** *After running Algorithm 12.4.2, we have the following:*

(1) $B[x_i, x_j, 0] = \max(x_i + x_j)$.

(2) $B[x_i, x_j, 1] = \max(x_i - x_j)$.

(3) $B[x_i, x_j, 2] = \min(x_i + x_j)$.

(4) $B[x_i, x_j, 3] = \min(x_i - x_j)$.

***Lemma* 12.4.3.** *All bounds generated by Algorithm 12.4.2 can be satisfied with equality by valid integer assignments to the variables.*

Proof. Suppose otherwise, thus there exists a system of constraints $\mathbf{U}$ such that the algorithm generates a bound, say $x_i + x_j \leq c$, that can only be satisfied with equality by a non-integer solution to the original system of equations. Because the relaxation steps of DIJKSTRA correspond to the addition of constraints, the constraint $x_i + x_j \leq c$ can be derived from the original system.

If no such constraint can be derived, then no upper bound on $(x_i + x_j)$ can be generated. Thus, there is no white path between x_i and x_j in the constraint network. In this case, the algorithm correctly gives the upper bound as ∞.

Consider the system $\mathbf{U} \cup \{-x_i - x_j \leq -c\}$. This system is linear feasible, but not integer feasible. Thus, through repeated applications of the transitive and tightening inference rules a contradiction can be generated. However, by construction, $\mathbf{U}$ already contains all constraints generated by the tightening inference rule. This means that the contradiction can be obtained by only applying the transitive inference rule.

This would mean that $\mathbf{U} \cup \{-x_1 - x_i \leq -c\}$ is also linearly infeasible. Thus, $\mathbf{U} \cup \{-x_1 - x_i \leq -c\}$ must be integer feasible, and so the constraint $x_1 + x_i \leq c$ can be satisfied with equality by an integer solution.

A similar proof applies to constraints of type $x_1 - x_i \leq c$, $-x_1 + x_i \leq c$, and $-x_1 - x_i \leq c$. □

***Theorem* 12.4.1.** *Algorithm 12.4.2 correctly computes the integer closure* $\mathbf{U}$.

Proof. From Lemma 12.4.1, and the corresponding results for the other constraint types, we know that all bounds generated by Algorithm 12.4.2 are derivable from $\mathbf{U}$. Thus, these bounds are satisfied by all integer solutions to $\mathbf{U}$.

From Lemma 12.4.3, every bound generated by Algorithm 12.4.2 is satisfied with equality by some integer solution of $\mathbf{U}$. Thus, these bounds are tight, and Algorithm 12.4.2 correctly computes the integer closure of $\mathbf{U}$. □

We now show that the graph conversion procedure in Algorithm 12.4.3 generates all constraints derivable from the tightening inference rule.

***Lemma* 12.4.4.** *Re-weighting the graph does not change the parity of the bounds on* $(x_i + x_i)$ *and* $(-x_i - x_i)$.

Proof. When the constraints are re-weighted, the change to the bound on both $(x_i + x_i)$ and $(-x_i - x_i)$ only depends on the value of d_i. We have that d_i is an integer, so the bound changes by $\pm 2 \cdot d_i$. Consequently, the parity of the weight remains unchanged. Thus, re-weighting the graph does not change where the tightening rule is applied. □

Now, we show that only one pass of tightening is required. We do this by showing that adding a constraint generated by the tightening rule does not result in additional applications of the tightening rule.

***Lemma* 12.4.5.** *Adding a constraint generated by the tightening rule does not necessitate adding any additional constraints as a result of the tightening inference rule.*

Proof. Suppose that adding the new constraint $x_i + x_i \leq 2 \cdot c_i$ results in an odd tightest bound for $(x_j + x_j)$.

The path p responsible for this bound must use the constraint $x_i + x_i \leq 2 \cdot c_i$. Thus, the path consists of a path p_1 from x_j to x_i, the newly added edge, and a path p_2 from x_i to x_j.

We have that p_1 corresponds to a constraint of the form $x_j - x_i \leq w_1$, and that the path p_2 corresponds to a constraint of the form $-x_i + x_j \leq w_2$.

Since p is of odd weight, we have that $(2 \cdot c_i + w_1 + w_2)$ is odd. Thus, $(w_1 + w_2)$ is also odd. This means that, since w_1 and w_2 are both integers, either $w_1 < w_2$ or $w_2 < w_1$.

If $w_1 < w_2$, then we can construct the bound $x_j + x_j = x_i + x_i + 2 \cdot (x_j - x_i) \leq 2 \cdot (c_i + w_1) < 2 \cdot c_i + w_1 + w_2$. This contradicts the assumption that $(2 \cdot c_i + w_1 + w_2)$ is the tightest bound on $(x_j + x_j)$

If $w_2 < w_1$, then we can construct the bound $x_j + x_j = x_i + x_i + 2 \cdot (-x_i + x_j) \leq 2 \cdot (c_i + w_2) < 2 \cdot c_i + w_1 + w_2$. This contradicts the assumption that $(2 \cdot c_i + w_1 + w_2)$ is the tightest bound on $(x_j + x_j)$

A similar proof applies when the edge added is $-x_i - x_i \leq 2 \cdot c_i$, or when the odd bound is created for $(-x_j - x_j)$. □

***Theorem* 12.4.2.** *Algorithm 12.4.3 computes all constraints derivable by the tightening inference rule.*

Proof. From Lemma 12.4.4, re-weighting the graph does not change where the tightening rule is applied. Thus, all tightenings performed on $\mathbf{U}'$ correspond to tightenings that can be performed on $\mathbf{U}$. The re-weighting is performed on lines 5 to 18 of Algorithm 12.4.3.

By Lemma 12.4.2, running Algorithm 12.4.1 from x_i is enough to compute $\max(x_i)$ and $\min(x_i)$. Thus, running Algorithm 12.4.1 from x_i allows us to determine if we need to apply the tightening rule to x_i. This is handled by lines 20 to 30 of Algorithm 12.4.3.

By Lemma 12.4.5, one round of tightening, after transitive closure is computed, is sufficient to generate all tightening constraints. Once these constraints are added to the re-weighted system, we have that the system is in the form required by Algorithm 12.4.2.

From Corollary 12.4.1, we have that Algorithm 12.4.2 generates the correct bounds for each pair of variables. All that is left to do is to change the bounds so that they apply to the regular graph. This is done on lines 32 to 39 of Algorithm 12.4.3. □

Chapter 13

Horn Constraint Systems

13.1 Motivation and Related Work

Since the integer feasibility problem is **NP-hard** for general polyhedra ($\mathbf{A} \cdot \mathbf{x} \geq \mathbf{c}$) a fair amount of research has been devoted towards the design of polynomial time algorithms for various special cases, restricting the structure of $\mathbf{A}$.

It is well-known that, if the constraint matrix $\mathbf{A}$ is Totally Unimodular (TUM) and the vector $\mathbf{b}$ is integral, then the system $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{c}$ has integral extreme point solutions [Sch87]. Note that a matrix $\mathbf{A}$ is totally unimodular if the determinant of every square sub-matrix of $\mathbf{A}$ is 0, 1, or -1. Difference constraint systems are a sub-class of TUM systems, in which each constraint has at most one positive entry and one negative entry, with the positive entry being 1 and the negative entry being -1. This problem is the dual of the problem of finding single source shortest paths in directed, real-weighted networks [CLRS01].

A related constraint system is the Unit Two Variables per Inequality (UTVPI) system in which both sum and difference relationships can be expressed. The **IF** feasibility problem for this class was shown to be in **P** [JMSY94]. Unlike DCSs though, in a UTVPI system the answers to the **LF** and **IF** problems do not coincide in that such a system could be linear feasible but not integer feasible. UTVPI systems find applications in a host

of verification-related problems such as abstract interpretation and array-bounds checking [LM05, BHZ09].

Horn Constraint Systems generalize difference constraints in that multiple negative unity entries are permitted in a row. It is easy to see that Horn systems are not TUM. However, a Horn constraint system always has a least element (if it is feasible) and the least element of a Horn system is always integral. It follows that the **LF** and **IF** problems coincide in case of Horn Constraint systems [CS13]. Veinott [VL92, Vei89] has a non-polynomial algorithm for the **LF** problem of Horn type programs where the positive and negative elements can take any value.

Our work is closely related to Lattice programming. Lattice programming is concerned with predicting the direction of change in global optima and equilibria resulting from changing conditions based on problem structure alone without data gathering or computation. Rooted in the theory of lattices, this work is also useful for characterizing the form of optimal and equilibrium policies, improving the efficiency of computation and suggesting desirable properties of heuristics. Applications range widely over dynamic programming, statistical decision-making, cooperative and noncooperative games, economics, network flows, Leontief substitution systems, production and inventory management, project planning, scheduling, marketing, and reliability and maintenance [VW62].

13.2 Refutability

13.2.1 The ROR problem (ADD and DIV rules)

In this section we study refutations of systems of Horn clause constraints that use both the ADD rule and the DIV rule.

With the addition of the DIV rule, additional constraint systems have read-once refutations.

***Lemma* 13.2.1.** *There is a formula Φ such that $S(\Phi) \in$ CP-RO(ADD,DIV) and $S(\Phi) \notin$*

CP-RO(ADD).

Proof. Let $\Phi = (x_1) \wedge (\neg x_1 \vee x_2) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$.

This corresponds to the HClCS

$$\begin{array}{lrcl} l_1: & -x_1 - x_2 + x_3 & \geq & -1 \\ l_2: & -x_1 + x_2 & \geq & 0 \\ l_3: & x_1 & \geq & 1 \\ l_4: & -x_1 - x_2 - x_3 & \geq & -2 \end{array}$$

This system has the following read-once integer refutation.

First sum the constraints l_1 and l_4 to get the constraint $-2 \cdot x_1 - 2 \cdot x_2 \geq -3$. Then apply the division rule (DIV) with $d = 2$ to get the constraint $-x_1 - x_2 \geq -1$. Now sum this constraint with the constraint l_2. This results in the constraint $-2 \cdot x_1 \geq -1$. Then apply the division rule. with $d = 2$ to get the constraint $-x_1 \geq 0$. Finally, we sum this constraint with constraint l_3 to obtain the contradiction $0 \geq 1$.

However $S(\Phi)$ does not have a read-once linear refutation. The formula is minimal unsatisfiable. Therefore we need to use all four constraints. However summing all four constraints results in the constraint $-2 \cdot x_1 - x_2 \geq -2$. Thus, to derive a contradiction we need to use the constraint l_2 an additional time and the constraint l_3 an additional 3 times. □

***Theorem* 13.2.1.** *There is an unsatisfiable Horn formula Φ such that $S(\Phi) \notin$ CP-RO(ADD,DIV)*

Proof. Let $\Phi = (y_1) \wedge (y_2) \wedge (\neg y_1 \vee x_1) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_1 \vee x_2) \wedge (\neg y_1 \vee \neg y_2 \vee \neg x_2)$.

This corresponds to the HClCS:

$$
\begin{array}{lrrrrcr}
l_1: & y_1 & & & & \geq & 1 \\
l_2: & & y_2 & & & \geq & 1 \\
l_3: & -y_1 & & +x_1 & & \geq & 0 \\
l_4: & -y_1 & -y_2 & -x_1 & +x_2 & \geq & -2 \\
l_5: & -y_1 & -y_2 & & -x_2 & \geq & -2
\end{array}
$$

Note that $-y_1$ and $-y_2$ each appear in multiple constraints. Thus, the first applications of the ADD rule cannot use either constraint l_1 or l_2. This means that the first application of the ADD rule must be to either constraints l_3 and l_4, constraints l_3 and l_5, or constraints l_4 and l_5.

1. If we apply the ADD rule to constraints l_3 and l_4, then this results in the constraint $-2 \cdot y_1 - y_2 + x_2 \geq -2$. We cannot apply the DIV rule to this constraint. Additionally, $-y_1$ and $-y_2$ still occur in multiple constraints. Thus, l_1 and l_2 cannot be used in the next application of the ADD rule. This means that the next application of the ADD rule involves the new constraint and l_5. This results in the constraint $-3 \cdot y_1 - 2 \cdot y_2 \geq -4$.

2. If we apply the ADD rule to constraints l_3 and l_5, then this results in the constraint $-2 \cdot y_1 - y_2 + x_1 - x_2 \geq -2$. We cannot apply the DIV rule to this constraint. Additionally, $-y_1$ and $-y_2$ still occur in multiple constraints. Thus, l_1 and l_2 cannot be used in the next application of the ADD rule. This means that the next application of the ADD rule involves the new constraint and l_4. This results in the constraint $-3 \cdot y_1 - 2 \cdot y_2 \geq -4$.

3. If we apply the ADD rule to constraints l_4 and l_5, then this results in the constraint $-2 \cdot y_1 - 2 \cdot y_2 - x_1 \geq -4$. We cannot apply the DIV rule to this constraint. Additionally, $-y_1$ still occurs in multiple constraints and y_2 has a coefficient of 2 in the new constraint. Thus, l_1 and l_2 cannot be used in the next application of the ADD rule. This means that the next application of the ADD rule involves the new constraint and l_3. This results in the constraint $-3 \cdot y_1 - 2 \cdot y_2 \geq -4$.

Note that if the first step is applying the add rule to the constraints l_1 and l_2 the resultant constraint $y_1 + y_2 \geq 2$ encounters the same issues as constraints l_1 and l_2 in the preceding cases.

In all three cases we obtain the constraint $-3 \cdot y_1 - 2 \cdot y_2 \geq -4$. However, we cannot apply the DIV rule to this constraint. Since both $-y_1$ and $-y_2$ have coefficients greater than 1, we cannot use constraints l_1 and l_2 to completely eliminate either of these literals. Thus, the system is not in $CP - RO(ADD, DIV)$. □

We now show that even with the addition of the DIV rule, the problem of determining if an HClCS has a read-once refutation remains **NP-complete**. This is done by a reduction from the set packing problem.

***Theorem* 13.2.2.** *CP-RO(ADD,DIV) is* **NP-complete** *for HClCSs.*

Proof. Consider a system of Horn constraints Φ with m constraints over n variables. Note that each application of the ADD rule effectively reduces the number of constraints by 1. Thus, we can apply the ADD rule at most $(m-1)$ times. For each constraint, either in Φ or derived from the ADD rule, we can apply the DIV rule. We can assume without loss of generality that we never apply the DIV rule to a constraint derived from the DIV rule. Thus, we apply the DIV at most $(2 \cdot m - 1)$ times. Since each intermediate constraint is polynomially sized in terms of the sized of the original system and we have at most $(3 \cdot m - 2)$ intermediate constraints, any read-once refutation using the ADD and DIV rules must be polynomially sized. Thus CP-RO(ADD,DIV) is in **NP** for Horn constraints. We now need to show **NP-hardness**.

Let us consider an instance of the set packing problem. For each instance $k \geq 1$ and $S = \{S_1, \ldots, S_m\}$ where $S_i \subseteq \{x_1, \ldots, x_n\}$ we construct the associated HClCS **H**.

1. Let $p(m)$ be the first prime larger than m. Note that $p(m) \leq 2 \cdot m$ and $p(m)$ can be found in time polynomial in m.

2. Create the variable u and the constraint $u \geq 1$. This constraint corresponds to the clause (u).

3. For each $i = 1 \ldots p(m)$ create the variable u_i and the constraint $u_i - u \geq 0$. This constraint corresponds to the clause $(u_i \vee \neg u)$.

4. For each x_i create the constraint $x_i \geq 1$. This constraint corresponds to the clause (x_i).

5. For $j = 1 \ldots k$, create the variable v_j.

6. For each $j = 1 \ldots k$, $l = 1 \ldots m$, and $i = 1 \ldots p(m)$ create the variable $w_{j,l,i}$.

7. For each subset S_l, $l = 1 \ldots m$ create the constraints

$$v_j - \sum_{x_i \in S_l} x_i - w_{j,l,1} - \ldots - w_{j,l,p(m)} \geq 1 - |S_l| - p(m) \qquad j = 1 \ldots k.$$

   This constraint corresponds to the clause

$$\left(v_j \vee \bigvee_{x_i \in S_l} \neg x_i \vee \neg w_{j,l,1} \vee \ldots \vee \neg w_{j,l,p(m)} \right)$$

8. For each $j = 1 \ldots k$ and $l = 1 \ldots m$ create the variable $z_{j,l}$ and the constraint $z_{j,l} \geq 1$. This constraint corresponds to the clause $(z_{j,l})$.

9. For each $j = 1 \ldots k$, $l = 1 \ldots m$, and $i = 1 \ldots p(m)$ create the constraint $w_{j,l,i} - z_{j,l} \geq 0$. This constraint corresponds to the clause $(w_{j,l,i} \vee \neg z_{j,l})$.

10. Finally create the constraint $-u_1 - \ldots - u_{p(m)} - v_1 - \ldots - v_k \geq 1 - k - p(m)$. This constraint corresponds to the clause $(\neg u_1 \vee \ldots \vee \neg u_{p(m)} \vee \neg v_1 \vee \ldots \vee \neg v_k)$.

11. The resultant HClCS is $\mathbf{H}$.

We now show that $\mathbf{H}$ is in $CP - RO(ADD, DIV)$ if and only if $\{S_1, \ldots, S_m\}$ contains k mutually disjoint sets.

Suppose that $\{S_1, \ldots, S_m\}$ does contain k mutually disjoint sets. Without loss of generality assume that these are the sets $S_1, \ldots, S_k$.

Initially, we use the constraint $-u_1 - \ldots - u_{p(m)} - v_1 - \ldots - v_k \geq 1 - k - p(m)$ together with the constraints $u_i - u \geq 0$ for $i = 1 \ldots p(m)$ to derive the constraint $-p(m) \cdot u - v_1 - \ldots - v_k \geq 1 - k - p(m)$.

Now, let us consider the sets of constraints

$$\begin{aligned} \mathbf{H}_j &= \{v_j - \sum_{x_i \in S_j} x_i - w_{j,j,1} - \ldots - w_{j,j,p(m)} \geq 1 - |S_j| - p(m)\} \\ &\cup \{x_i \geq 1 \,|\, x_i \in S_j\} \\ &\cup \{w_{j,j,i} - z_{j,j} \geq 0 \,|\, i = 1 \ldots p(m)\} \qquad j = 1 \ldots k. \end{aligned}$$

By the construction of $\mathbf{H}$, we have that $\mathbf{H}_j \subseteq \mathbf{H}$ for $j = 1 \ldots k$. Since the sets $S_1, \ldots, S_k$ are mutually disjoint, so are the sets $\mathbf{H}_1, \ldots, \mathbf{H}_k$.

It it easy to see that the constraint $v_j - p(m) \cdot z_{j,j} \geq 1 - p(m)$ can be derived by summing all of the constraints in $\mathbf{H}_j$. Since this holds for every $j = 1 \ldots k$ and since the sets $\mathbf{H}_1, \ldots, \mathbf{H}_k$ are mutually disjoint, we have that the set of constraints $\{v_1 - p(m) \cdot z_{1,1} \geq 1 - p(m), \ldots, v_k - p(m) \cdot z_{k,k} \geq 1\}$ can be derived from $\mathbf{H}$ by read-once applications of the ADD rule.

Together with the constraint $-p(m) \cdot u - v_1 - \ldots - v_k \geq 1 - k - p(m)$, this set of constraints sums together to derive the constraint $-p(m) \cdot u - p(m) \cdot z_{1,1} - \ldots - p(m) \cdot z_{k,k} \geq 1 - (k+1) \cdot p(m)$

Applying the DIV rule results in the constraint $-u - z_{1,1} - \ldots - z_{k,k} \geq -k$

Together with the constraints $u \geq 1$, $z_{1,1} \geq 1$, …, $z_{k,k} \geq 1$ we obtain the contradiction $0 \geq 1$. This is a read once refutation using the ADD and DIV rules.

Now suppose that $\mathbf{H}$ is in $CP - RO(ADD, DIV)$. We have to use the constraint $-u_1 - \ldots - u_{p(m)} - v_1 - \ldots - v_k \geq 1 - k - p(m)$, because without this constraint the system would be satisfied by setting every variable to 1. Let h_0 denote this constraint. Note that to cancel each $-u_i$ from h_0, we must use the constraint $u_i - u \geq 0$. This introduces multiple copes of $-u$ into the derived constraint. Thus, we cannot derive a contradiction or use the constraint $u \geq 1$ until after the DIV rule is used with divisor $d \neq 1$.

We first look at the case where the DIV rule is used on a constraint derived without using h_0. Let $H' \subseteq H \setminus \{h_0\}$, be a set of horn constraints such that summing the constraints in H' results in a constraint C that can have the DIV rule applied with divisor $d \neq 1$.

H' must have the following properties:

1. H' cannot contain any constraint $u_i - u \geq 0$ for $i = 1 \ldots p(m)$ – By construction, this is the only constraint in $h \setminus \{h_0\}$ to use the variable u_i. Thus, if $u_i - u \geq 0$ is in H' then the variable u_i is in C with coefficient 1. This means that the DIV rule can only be applied to C with $d = 1$.

2. H' cannot contain the constraint $u \geq 1$ – We already know that H' cannot contain any of the $u_i - u \geq 0$ constraints. By construction, $u \geq 0$ is the only other constraint in H to use the variable u. Thus, if $u \geq 1$ is in H' then the variable u is in C with coefficient 1. This means that the DIV rule can only be applied to C with $d = 1$.

3. If H' contains the constraint

$$v_j - \sum_{x_i \in S_l} x_i - w_{j,l,1} - \ldots - w_{j,l,p(m)} \geq 1 - |S_l| - p(m)$$

for any $j = 1 \ldots k$ and $l = 1 \ldots m$, then H' must contain the constraints $w_{j,l,1} - z_{j,l} \geq 0$, …, $w_{j,l,p(m)} - z_{j,l} \geq 0$ – Assume that for some $1 \leq i \leq p(m)$ the constraint $w_{j,l,i} - z_{j,l} \geq 0$ is not in H'. Thus, by construction, the only constraint in H' to use the variable $w_{j,l,i}$ is

$$v_j - \sum_{x_i \in S_l} x_i - w_{j,l,1} - \ldots - w_{j,l,p(m)} \geq 1 - |S_l| - p(m).$$

Thus, the variable $w_{j,l,i}$ is in C with coefficient -1. This means that the DIV rule can only be applied to C with $d = 1$.

4. Similarly, if H' contains the constraint $w_{j,l,i} - z_{j,l} \geq 0$ for any $j = 1 \ldots k$, $l = 1 \ldots m$,

and $i = 1 \ldots p(m)$, then H' must contain the constraint

$$v_j - \sum_{x_i \in S_l} x_i - w_{j,l,1} - \ldots - w_{j,l,p(m)} \geq 1 - |S_l| - p(m).$$

5. H' cannot contain the constraint $z_{j,l} \geq 1$ for any $j = 1 \ldots k$ and $l = 1 \ldots m$ – Assume that h' contains this constraint. If H' does not contain the constraint $w_{j,l,i} - z_{j,l} \geq 0$ for any $i = 1 \ldots p(m)$, then $z_{j,l} \geq 1$ is the only constraint in H' to use the variable $z_{j,l}$. Thus, the variable $z_{j,l}$ is in C with coefficient 1. This means that the DIV rule can only be applied to C with $d = 1$.

   However, if H' does contain the constraint $w_{j,l,i} - z_{j,l} \geq 0$ for some $i = 1 \ldots p(m)$, then, from the points above, it must contain all $p(m)$ of those constraints. This means that H' contains all of the constraints in H which use the variable $z_{j,l}$. However, C still contains the variable $z_{j,l}$ with no way to cancel it as part of a read-once refutation.

6. H' cannot contain the constraint

$$v_j - \sum_{x_i \in S_l} x_i - w_{j,l,1} - \ldots - w_{j,l,p(m)} \geq 1 - |S_l| - p(m)$$

   for any $j = 1 \ldots k$ and $l = 1 \ldots m$ – If H' contains this constraint, then it must contain the constraint $w_{j,l,i} - z_{j,l} \geq 0$ for each $i = 1 \ldots p(m)$. Since H' does not contain the constraint $z_{j,l} \geq 1$, the variable $z_{j,l}$ is in C with coefficient $-p(m)$. Since $p(m)$ is prime, for the DIV rule to be applied to C with $d \neq 1$, it must be the case that $d = p(m)$. However, the variable v_j appears in at most m constraints in H' and it has coefficient 1 in each of those constraints. Thus, the v_j must have coefficient a such that $1 \leq a \leq m < p(m)$ in C. This means that the DIV rule cannot be applied to C with $d = p(m)$.

   From the previous points, this also means that H' cannot contain any of the constraints of the form $w_{j,l,i} - z_{j,l} \geq 0$.

7. H' cannot contain any constraint $x_i \geq 1$ for any $i = 1 \ldots n$ – If the constraint $x_i \geq 1$ is in H', then, from the previous point, we know that this is the only constraint in H' that uses the variable x_i. Thus, the variable x_i is in C with coefficient 1. This means that the DIV rule can only be applied to C with $d = 1$.

We have shown that no constraint can be in the set H' thus we cannot use the DIV rule with $d \neq 1$ on a constraint not derived using h_0.

Let h be a constraint derived by summing h_0 with a subset of the constraints in $\mathbf{H} \setminus \{h_0, u \geq 1\}$. h has the following properties:

1. If h contains the variable u_i for any $i = 1 \ldots p(m)$, then the DIV rule has no effect on h – The only way for this to happen is if h contains the term $-u_i$ from l_0. Since the coefficient of this term is -1, the DIV rule has no effect.

2. If h contains no variable u_i for any $i = 1 \ldots p(m)$, then h contains the term $-p(m) \cdot u$ – The only constraint which cancels the term $-u_i$ from h_0 is $u_i - u \geq 0$. Including this constraint in the summation introduces a $-u$ term to h. Doing this for every $i = 1 \ldots p(m)$ results in the desired term.

3. If the DIV rule can be applied to h with $d \neq 1$, then h is a constraint of the form $-p(m) \cdot u - p(m) \cdot z_{1,l_1} - \ldots - p(m) \cdot z_{k,l_k} \geq 1 - (k+1) \cdot p(m)$ – As shown previously, for the DIV rule to be applied to h with $d \neq 1$ h must contain the term $-p(m) \cdot u$. Since $p(m)$ is prime, $d = p(m)$. Thus, all variables in h must have coefficients divisible by $p(m)$. The only variables which are used at least $p(m)$ constraints in H are the variables $z_{j,l}$ for $j = 1 \ldots k$, $l = 1 \ldots m$.

   For the variable $z_{j,l}$ to have coefficient $-p(m)$ in h, each constraint $w_{j,l,i} - z_{j,l} \geq 0$ for $i = 1 \ldots p(m)$ must be included in the summation. The only constraint in H which can cancel each $w_{j,l,i}$ introduced by the constraints is

$$v_j - \sum_{x_i \in S_l} x_i - w_{j,l,1} - \ldots - w_{j,l,p(m)} \geq 1 - |S_l| - p(m).$$

Thus, this constraint must also be in the summation. This constraint cancels the $-v_j$ term in h_0. Since h_0 has k such terms, h must contain k terms of the form $-p(m)\cdot z_{j,l}$. Thus h must have the desired form.

4. If h is of the form $-p(m)\cdot u - p(m)\cdot z_{1,l_1} - \ldots - p(m)\cdot z_{k,l_k} \geq 1-(k+1)\cdot p(m)$, then $\{S_1,\ldots,S_m\}$ contains k mutually disjoint sets – To derive this constraint from h_0, we must do the following:

   (a) For each $i = 1\ldots p(m)$, cancel the term $-u_i$ with the constraint $u_i - u \geq 0$.

   (b) For each $j = 1\ldots k$, cancel the term $-v_j$ with the constraint $v_j - \sum_{x_i\in S_l} x_i \geq 1-|S_l|$ for some $l = 1\ldots m$.

   (c) For each $w_{j,l,i}$ introduced by a constraint of the form $v_j - \sum_{x_i\in S_l} x_i - w_{j,l,1} - \ldots - w_{j,l,p(m)} \geq 1-|S_l|-p(m)$, cancel the term $-w_{j,l,i}$ with the constraint $w_{j,l,i} - z_{j,l} \geq 0$.

   (d) For each x_i introduced by a constraint of the form $v_j - \sum_{x_i\in S_l} x_i - w_{j,l,1} - \ldots - w_{j,l,p(m)} \geq 1-|S_l|-p(m)$, cancel the term $-x_j$ with the constraint $x_i \geq 1$.

   To do the last step each x_i can be used by at most one of the constraints of the form $v_j - \sum_{x_i\in S_l} x_i \geq 1-|S_l|$, otherwise we would not be able to cancel it by using the constraint $x_i \geq 1$ only once. Thus, as in the proof of Theorem 11.2.1, $\{S_1,\ldots,S_m\}$ contains k mutually disjoint sets.

Thus, $\mathbf{H}$ is in $CP-RO(ADD,DIV)$ if and only if $\{S_1,\ldots,S_m\}$ contains k mutually disjoint sets. As a result of this, $CP-RO(ADD,DIV)$ is **NP-complete** for HClCS. □

Part V

Quantified Linear Constraints

Chapter 14

Quantified Linear Programming

14.1 Motivation and Related Work

Modeling uncertainty is one of the principal application areas of QLP. Many application models incorporate the assumption of constancy in data which is neither realistic nor accurate. In scheduling problems [Pin95b], for instance, the execution time of a job is usually considered fixed and known in advance. This simplifying assumption leads to elegant models; however in real-time systems [GPS95b, CA00] such an assumption may lead to dire consequences [SSRB98].

A application field where uncertainty plays a crucial role is *reactive systems* [HP85]. A system is called reactive if its role is to maintain an ongoing interaction with its environment. A reactive system is an event driven system reacting endlessly to external stimuli. For instance, a communications' protocol is a system that must respond to each stimulus, even to a fragment of input, such as a disrupted received message. In several real-world important applications [KSSVS99, KCH01, PW00, PW01, Hal02, Har04], reactive systems handle input that is immense. Reactive systems are widely used to control critical procedures. Air and road traffic control systems, programs navigating robotic devices (e.g., trains, planes) and systems controlling nuclear reactors or chemical plants processes are

typical examples of such applications.

Most importantly, the domain of possible environmental inputs for reactive systems is usually not fixed and thus unpredictable; it changes continuously due to the interplay between the environmental stimuli and the responses of the system, i.e., as a result of events that are initiated by the environment and given as input to the system, and as a result of the responses of the system to this input. Hence, the primary goal of reactive systems is to provide an offline guarantee that the input constraints will be met at run-time, regardless of the actual input that may be given at any time to the system. This constitutes reliability (i.e., failure-free system operation over a specified time in a given environment and for a given purpose) to be of the utmost importance for such systems. Failure may result into loss of critical data (e.g., in communication networks), high economic losses, catastrophic environmental damages (e.g., systems controlling chemical or nuclear plants), injury or even loss of life (e.g., traffic control systems), depending on the system's purpose.

In the attempt to design and implement reliable reactive systems, system developers and software engineers confront two major issues, namely parameter variability and the existence of complex relationships between the input and the reactions of the system. This makes the analysis and design of dependable reactive system an elaborate and crucial task [Cla97, KSSVS99, Mut77, Dru06]. Thus, it highlights the need for modeling tools that are appropriate for incorporating and handling such issues in the specification of reactive systems. In that regard, modeling reactive systems as 2-person games enables the utilization of powerful mathematical and algorithmic tools that can be developed for such games. QLPs provide a unified framework, whose expressive power is ideal for modeling reactive systems as 2-person games. That is, reactive systems can be modeled as QLPs where the environmental input is represented by the values of the universally quantified variables, while the system's response is represented by the values of the existentially quantified variables. Hence, checking feasibility of a QLP (i.e., checking whether a linear polyhedron specified by a linear system of inequalities is nonempty with respect to a specified quantifier string) is equivalent to 'playing out' the corresponding 2-person game and determining

whether the corresponding reactive system will fail or not.

QLPs can also be utilized in software verification [BM07]. For critical systems, it is of paramount importance to make sure that the corresponding software implements its intended purpose. Testing is not sufficient to avoid communication network collapses, huge commercial losses or catastrophic accidents. Software verification is a formal approach to verifying the correctness of the implementation. It is a rigorous technique which analyzes mathematical models that represent the software. The software is formulated in a language with well-defined semantics, pre- and postcondition are formulated in some formal system, and the proof of correctness is carried out in this formal system.

Constraint programming techniques have been explored for software verification [CRV04, CR06, CAS08]. These techniques formulate software as *Constraint Satisfaction Problems* (CSPs). They derive constraint systems from the program and consider their conjunction with the negation of the specifications. Typically, their aim is to prove that the CSP is unsolvable. This would mean that the software is consistent with its specifications. Although this approach is intuitive and straightforward, it may become impractical because of the high number of constraints that are generated. A CSP approach to program verification has also been attempted for rule-based programming [BL08]. 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 verification of rule programs has emerged. Also, in [GSV08] 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 (than Boolean programs) grained abstraction for sequential programs offering an effective model checking procedure [ACM04].

Furthermore, software verification has been studied in the context of reactive systems [BB91]. A minor investigation in the area (and especially of real-time control systems) was done in [BGM90], in which a theorem-prover was used to prove that a simple program keeps a vehicle ‘on course’ in a varying cross wind. A major problem in the verification

of reactive systems is the specification of the non-digital world with which such programs interact. A related area of concern is hardware verification, where timing and interrupt handling are major problems. An initial investigation was conducted in [Wag77], who proved properties of circuits for such basic tasks as counting and multiplying. Formalisms used for specification and verification of reactive systems, and in particular, results on m-calculus, w-automata, and temporal logics, are presented in [Sch04].

Software verification is enormously effort intensive [Cla93], while the verification of complex algorithms written in popular programming languages such as C, C++ or Java is still far beyond the state-of-the-art. However, existing polynomial procedures for QLPs [Sub07] can accommodate the utilization of linear constraints in software verification procedures, leading to the design of polynomial-time software verification methods for special types of programs.

14.2 Satisfiability

14.2.1 Semantics

In this section, we interpret QLP decidability as a 2-person game. Such a game includes an existential player **X**, who chooses values for the existentially quantified variables, and a universal player **Y**, who chooses values for the universally quantified variables. Our analysis focuses on QLPs in general form [Sub07], but also holds for the partially bounded and unbounded variants, discussed in this dissertation.

Consider the generic form of QLP (i.e., QLP (2.4)) and assume, without loss of generality, that $\mathbf{x}_1$ and $\mathbf{y}_n$ are not empty (dummy variables can be added, if necessary). The *initial board configuration* of the game is:

$$\mathbf{A} \cdot \mathbf{x} + \mathbf{N} \cdot \mathbf{y} \leq \mathbf{b} \tag{14.1}$$

The game is played in a sequence of $2 \cdot n$ rounds. Let $i = 1, \ldots, n$. In round $(2 \cdot i - 1)$, **X** makes his i^{th} move (by choosing values for the variables in the set $\mathbf{x}_i$). Then, **Y** makes his i^{th} move (by choosing values for the variables in the set $\mathbf{y}_i$) in round $2 \cdot i$. Hence, **X** and **Y** make their moves by selecting values for their respective variable sets. The moves are strictly alternating: **X** makes his i^{th} move, which is followed by **Y**'s i^{th} move, after which **X** makes his $(i+1)^{th}$ move and so on. After all the moves have been made in the order specified by the quantifier string, if $\mathbf{A} \cdot \mathbf{x} + \mathbf{N} \cdot \mathbf{y} \leq \mathbf{b}$ holds, we say that **X** wins the game; otherwise, we say that **Y** wins the game.

14.2.2 Complexity of UQLP and PQLP

In this section, we examine the computational complexities of PQLP and UQLP. We commence our analysis by showing that PQLP decidability is in **P**.

***Theorem* 14.2.1.** *PQLP decidability is in* **P**.

Proof. Let **L** be the following PQLP:

$$\begin{aligned} &\exists x_n \, \forall y_n \in [0, +\infty) \, \exists x_{n-1} \, \forall y_{n-1} \in [0, +\infty) \, \ldots \, \exists x_1 \\ &\qquad \forall y_1 \in [0, +\infty) \, \exists x_0 \;\; \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{y} \leq \mathbf{c} \end{aligned} \tag{14.2}$$

Let $\mathbf{A} = (\mathbf{a}_n, \mathbf{a}_{n-1}, \ldots, \mathbf{a}_0)$, $\mathbf{B} = (\mathbf{b}_n, \mathbf{b}_{n-1}, \ldots, \mathbf{b}_1)$, $\mathbf{x} = (x_n, x_{n-1}, \ldots, x_0)^T$, and $\mathbf{y} = (y_n, y_{n-1}, \ldots, y_1)^T$. Note that any PQLP can be reduced to the form specified by PQLP (14.2) through the addition of dummy variables. Enforcing this strict alternation of quantifiers will at most double the total number of program variables.

Consider the linear program:

$$LP_n : \;\; \mathbf{A} \cdot \mathbf{x} \leq \mathbf{c}, \tag{14.3}$$

and, for $i = 1, \ldots, n$ consider the linear program:

$$\begin{aligned} LP_{i-1} : \mathbf{A} \cdot \mathbf{x} + \mathbf{b}_i &\leq \mathbf{0} \\ x_i, \ldots, x_n &= 0. \end{aligned} \tag{14.4}$$

We will show that PQLP (14.2) is feasible if and only if LP_n is feasible, and LP_{i-1} is feasible for all $i = 1 \ldots n$.

Assume that the linear program represented by System (14.3) and the linear programs represented by System (14.4) are feasible.

Let $\hat{\mathbf{x}}^n$ denote a solution to LP_n. Furthermore, let $\hat{\mathbf{x}}^{i-1}$ denote a solution to LP_{i-1} for $i = 1, \ldots, n$.

Let $\mathbf{x^s}$ denote the strategy of the existential player $\mathbf{X}$ and, let $\mathbf{y^s}$ denote the strategy of the universal player $\mathbf{Y}$.

Consider the assignment

$$\mathbf{x^s} = \hat{\mathbf{x}}^n + \sum_{i=1}^{n} y_i \cdot \hat{\mathbf{x}}^{i-1} \tag{14.5}$$

Note that in LP_{i-1}, the constraints $x_i, \ldots, x_n = 0$ ensure that x_n through x_i do not depend on the values of y_i through y_1. Hence, the moves made by $\mathbf{X}$ depend only on the moves previously made by $\mathbf{Y}$, i.e., System (14.5) is a strategy for $\mathbf{X}$.

As per the 2-person game semantics of QLPs, the game corresponding to System (14.2) will be played as follows:

1. $\mathbf{X}$ chooses $x_n = x_n^s = \hat{x}_n^n$.

2. $\mathbf{Y}$ chooses $y_n = y_n^s \in [0, \infty)$; $\mathbf{X}$ chooses $x_{n-1} = x_{n-1}^s = \hat{x}_{n-1}^n + y_n \cdot \hat{x}_{n-1}^{n-1}$.

$$\vdots$$

3. $\mathbf{Y}$ chooses $y_{i+1} = y_{i+1}^s \in [0, \infty)$; $\mathbf{X}$ chooses $x_i = x_i^s = \hat{x}_i^n + \sum_{j=i}^{n-1} y_{j+1} \cdot \hat{x}_i^j$.

$$\vdots$$

4. **Y** chooses $y_1 = y_1^s \in [0,\infty)$; **X** chooses $x_0 = x_0^s = \hat{x}_0^n + \sum_{j=0}^{n-1} y_{j+1} \cdot \hat{x}_0^j$.

Observe that all the x_i^s, $i = 0,\ldots,n$ and the y_i^s, $i = 0,\ldots,n$ are numeric vectors. Let $\mathbf{x_T}$ be the numeric vector assigned to $\mathbf{x}$ by the strategy $\mathbf{x^s}$. It follows that $\mathbf{x_T} = \hat{\mathbf{x}}^n + \sum_{i=1}^{n} y_i^{\mathbf{s}} \cdot \hat{\mathbf{x}}^{i-1}$. Likewise, let $\mathbf{y_T} = (y_n^{\mathbf{s}}, y_{n-1}^{\mathbf{s}}, \ldots, y_1^{\mathbf{s}})$. Accordingly, the outcome of the play can be evaluated as:

$$\begin{aligned}
\mathbf{A}\cdot\mathbf{x_T} + \mathbf{B}\cdot\mathbf{y_T} &= \mathbf{A}\cdot\left(\hat{\mathbf{x}}^n + \sum_{i=1}^{n} y_i^{\mathbf{s}}\cdot\hat{\mathbf{x}}^{i-1}\right) + \sum_{i=1}^{n}\mathbf{b}_i\cdot y_i^{\mathbf{s}} \\
&= \mathbf{A}\cdot\hat{\mathbf{x}}^n + \sum_{i=1}^{n} y_i^{\mathbf{s}}\cdot(\mathbf{A}\cdot\hat{\mathbf{x}}^{i-1} + \mathbf{b}_i) \\
&\leq \mathbf{c} + \sum_{i=1}^{n} y_i^{\mathbf{s}}\cdot\mathbf{0} \\
&= \mathbf{c}
\end{aligned}$$

Since **Y**'s strategy was chosen arbitrarily, at the end of any play, **X**'s strategy forces $\mathbf{A}\cdot\mathbf{x} + \mathbf{B}\cdot\mathbf{y} \leq \mathbf{c}$. Thus, PQLP (14.2) is feasible, proving our claim.

Now assume that PQLP (14.2) is feasible. Thus, **X** has a winning strategy $\mathbf{x^s}$.

Consider the case when **Y** plays according to the strategy $\mathbf{y^s} = \mathbf{0}$. Let $\mathbf{x}^*$ be the numeric vector assigned to $\mathbf{x}$ by the strategy $\mathbf{x^s}$ in this situation. It is clear that $\mathbf{x}^*$ satisfies LP_n.

Let us focus on a specific value of i. Let $\bar{\mathbf{y}}$ be an arbitrary non-negative vector, and consider the case when **Y** plays according to the strategy $\bar{\mathbf{y}}$. Let $\bar{\mathbf{x}}$ be the numeric vector assigned to $\mathbf{x}$ by the strategy $\mathbf{x^s}$ in this situation.

Let $\bar{\mathbf{y}}^i = (0,0,\ldots,1,\ldots,0)^T$. Note that $\bar{y}_i^i = 1$ is the only non-zero element of $\bar{\mathbf{y}}^i$. Let $M \in \mathbb{Z}^+$ denote an arbitrary integer constant. Consider the case when **Y** plays according to the strategy $\bar{\mathbf{y}} + M\cdot\bar{\mathbf{y}}^i$. Let $\mathbf{x}'$ be the numeric vector assigned to $\mathbf{x}$ by the strategy $\mathbf{x^s}$ in this situation. Let $\bar{\mathbf{x}}^i$ be the numeric vector such that $\mathbf{x}' = (\bar{\mathbf{x}} + M\cdot\bar{\mathbf{x}}^i)$. Thus $\bar{\mathbf{x}}^i$ represents the change in **X**'s response when **Y**'s play changes by $\bar{\mathbf{y}}^i$.

Until y_i is assigned a value, **Y**'s plays (viz., $\bar{\mathbf{y}}$ and $\bar{\mathbf{y}} + M\cdot\bar{\mathbf{y}}^i$) are indistinguishable. Thus, **X**'s response cannot change until after y_i is assigned a value. This means that $\bar{x}_i^i, \ldots, \bar{x}_n^i = 0$.

We have $\mathbf{A}\cdot\bar{\mathbf{x}}+\mathbf{B}\cdot\bar{\mathbf{y}}\leq\mathbf{c}$ and $\mathbf{A}\cdot(\bar{\mathbf{x}}+M\cdot\bar{\mathbf{x}}^i)+\mathbf{B}\cdot(\bar{\mathbf{y}}+M\cdot\bar{\mathbf{y}}^i)\leq\mathbf{c}$. Since M was chosen arbitrarily, we have:

$$(\forall M)\ \ (\mathbf{A}\cdot\bar{\mathbf{x}}+\mathbf{B}\cdot\bar{\mathbf{y}})+M\cdot(\mathbf{A}\cdot\bar{\mathbf{x}}^i+\mathbf{B}\cdot\bar{\mathbf{y}}^i)\leq\mathbf{c}. \tag{14.6}$$

However, System (14.6) forces $\mathbf{A}\cdot\bar{\mathbf{x}}^i+\mathbf{B}\cdot\bar{\mathbf{y}}^i\leq\mathbf{0}$. Otherwise, System (14.6) would not be satisfied. Thus, $\bar{\mathbf{x}}^i$ must satisfy LP_{i-1}. Since i was chosen arbitrarily, we have that this is true for every $i=1\ldots n$.

Thus, we have successfully reduced the problem of deciding the feasibility of PQLP (14.2) to the problem of deciding the feasibility of LP_n and the feasibility of LP_{i-1} for $i=1\ldots n$. Since the problem of deciding LP feasibility is in **P** [Kha79], it follows that the problem of deciding PQLP feasibility is also in **P**. □

We now show that UQLP decidability is in **P**.

***Theorem* 14.2.2.** *UQLP decidability is in* **P**.

Proof. Consider the following UQLP:

$$\exists\mathbf{x_n}\ \forall\mathbf{y_n}\ \exists\mathbf{x_{n-1}}\ \forall\mathbf{y_{n-1}}\ \ldots\ \exists\mathbf{x_1}\ \forall\mathbf{y_1}\ \exists\mathbf{x_0}\ \ \mathbf{A}\cdot\mathbf{x}+\mathbf{B}\cdot\mathbf{y}\leq\mathbf{c} \tag{14.7}$$

To show that UQLP (14.7) can be solved in polynomial time, we reduce it to a PQLP.

Consider the following PQLP:

$$\begin{aligned}
&\exists\mathbf{x_n}\ \forall\mathbf{y'_n}\in[\mathbf{0},+\infty)\ \forall\mathbf{y''_n}\in[\mathbf{0},+\infty)\ \exists\mathbf{y_n}\ \exists\mathbf{x_{n-1}}\\
&\forall\mathbf{y'_{n-1}}\in[\mathbf{0},+\infty)\ \forall\mathbf{y''_{n-1}}\in[\mathbf{0},+\infty)\ \exists\mathbf{y_{n-1}}\ \ldots\\
&\exists\mathbf{x_1}\ \forall\mathbf{y'_1}\in[\mathbf{0},+\infty)\ \forall\mathbf{y''_1}\in[\mathbf{0},+\infty)\ \exists\mathbf{y_1}\ \exists\mathbf{x_0}\\
&\qquad\qquad\mathbf{A}\cdot\mathbf{x}+\mathbf{B}\cdot\mathbf{y}\ \leq\ \mathbf{c}\\
&\qquad\qquad\mathbf{y_i}\ =\ \mathbf{y'_i}-\mathbf{y''_i},\ i=1,2,\ldots n.
\end{aligned} \tag{14.8}$$

In PQLP (14.8), $\mathbf{y_1}', \ldots, \mathbf{y_n}'$ is a partition of $\mathbf{y}'$ and $\mathbf{y_1}'', \ldots, \mathbf{y_n}''$ is a partition of $\mathbf{y}''$, such that $|\mathbf{y}'_i| = |\mathbf{y}''_i| = |\mathbf{y}_i|$, for $i = 1 \ldots n$.

The key observation is that the universal player can win System (14.7) if and only if he can win System (14.8). To see this, note that the value of $\mathbf{y_i}$ in System (14.8) is completely determined by the universal player's choices for $\mathbf{y}'_\mathbf{i}$ and $\mathbf{y}''_\mathbf{i}$. Since $\mathbf{y}'_i \in [\mathbf{0}, +\infty)$ and $\mathbf{y}''_i \in [\mathbf{0}, +\infty)$, $\mathbf{y}$ can take any value in the interval $(-\infty, +\infty)$. Thus, the quantifier sequence $\forall \mathbf{y_i}$ in System (14.7) $i = 1, 2, \ldots, n$, can be replaced respectively by the quantifier sequence $\forall \mathbf{y}'_\mathbf{i} \in [\mathbf{0}, +\infty)\ \forall \mathbf{y}''_\mathbf{i} \in [\mathbf{0}, +\infty)\ \exists \mathbf{y_i}$ to get System (14.8), without affecting its feasibility. In other words, UQLP (14.7) is feasible if and only is PQLP (14.8) is feasible. From Theorem 14.2.1, it follows that UQLP decidability is in **P**.

□

By proving that UQLP is in **P**, we have essentially shown the following: The computational complexity of deciding a formula in TLA depends on both the number of alternations in the quantifier specification and the syntactic restriction. Observe that in the absence of any syntactic restriction, and in the presence of unbounded alternation, the TLA decidability problem is in 2-**EXPTIME** [FR75, Son85] and **EXPTIME-hard** [BM07]. However, the conjunctive fragment of TLA, even with unbounded alternation is decidable in polynomial time.

Chapter 15

Quantified Linear Implications

15.1 Motivation and Related Work

Quantified Linear Programming represents a rich language that is ideal for expressing schedulability specifications in *real-time scheduling* [GPS95b, CA00]. In a real-time scheduling instance, a dispatcher typically determines whether a set of ordered, non-preemptive jobs can be scheduled within given time frames. Associated with each job is a start time and an execution time. The execution time of a job is a range-bound variable. There exist timing constraints that constrain the execution of jobs.

Now, consider the case where the dispatcher has already obtained a schedule (solution). If new constraints are added to the specification, then the dispatcher may have to recompute the schedule. Alternatively, if it can be concluded that the current schedule does not cause violation of the newly added constraints, then the dispatcher can use the existing schedule. Quantified Linear Implications (QLIs) can be utilized to model the above decision problem.

QLIs can be used to model reactive systems. A system is called *reactive*, if it maintains an ongoing interaction with its environment. A reactive system changes its actions, outputs, and status in response to the input it receives from the environment. Reactive systems are used in several real-world important applications and in various fields (see e.g., [KSSVS99,

PW00, PW01, KCH01, Hal02, Har04]). QLI is an important modeling tool for the design and implementation of reactive systems. The universally quantified variables can be used to represent the environmental input, while the existentially quantified variables can be used to represent the system's response.

A decision procedure for the *full elementary theory of real closed fields* with addition ($+$), multiplication ($\cdot$) and order ($<$, $=$) was established in [Tar48]. Several quantifier elimination methods [DSW98b, Wei88] and efficient-in-practice approaches have been proposed since then [CH91, Bro03, DSW98a, DS97, Rat06, Rat11]. The complexity of these quantifier elimination procedures is, in the worst case, doubly exponential in the number of quantifier alternations and exponential in the number of variables [Wei88, DH88].

Real numbers cannot be fully axiomatized by a first-order theory. Tarski's axiomatization of the reals requires a non-first-order axiom to express the Dedekind completeness of the real numbers (i.e., the property that asks all bounded subsets of real numbers to have a real least upper bound and a real greatest lower bound). The axiom in question involves universal quantification over subsets of the real numbers, which cannot be expressed in first-order logic.

Any field that satisfies all the same first-order properties as the real numbers is called a *real closed field*. Note that although the real algebraic numbers comprise a real closed field, they are not Dedekind complete. It is possible to construct a set of algebraic rational numbers, that has π, which is transcendental, as a supremum.

Several sub-classes of the full elementary theory of the reals have been studied.

The *existential theory of the reals* is obtained by restricting allowable expressions to existentially quantified formulas $\exists \mathbf{x}\, F(\mathbf{x})$ where $F(\mathbf{x})$ is a quantifier-free formula. There exists a decision procedure for this problem that is singly exponential in the number of quantified variables [BPR06]. From the complexity perspective, it is known that this problem is **NP-hard** [Sho91] and in **PSPACE** [Can88].

The *theory of reals with addition and order* (TLA) is obtained by restricting the set of function symbols to $\{+\}$. A quantifier elimination procedure for sentences in this theory

that is singly exponential in space and doubly exponential in time is presented in [FR75]. An exponential time lower bound is shown in [Ber80], where the time and space complexities at various levels of quantifier alternations are also determined.

Consider a formula in the theory of reals with addition and order in prenex normal form with $(k-1)$ quantifier alternations

$$\exists \mathbf{x}_1 \, \forall \mathbf{x}_2 \ldots Q \mathbf{x}_k \, F(\mathbf{x}_1, \ldots, \mathbf{x}_k)$$

where Q is $\exists$ for k odd and $\forall$ for k even, while $F(\mathbf{x}_1, \ldots, \mathbf{x}_k)$ is a quantifier-free formula. This class of formulas has been proven to be log-space complete for $\blacksquare_{\mathbf{k}}\mathbf{P}$ [Son85].

15.2 Satisfiability

15.2.1 Semantics

In this section, we interpret a quantified linear implication problem as a 2-person game. Such a game includes an existential player **X**, who chooses values for the existentially quantified variables, and a universal player **Y**, who chooses values for the universally quantified variables. Consider the generic form of QLI (System (2.5)), and assume, without loss of generality, that $\mathbf{x}_1$ and $\mathbf{y}_n$ are not empty (dummy variables can be added, if necessary). Let the following be the *initial board configuration* of the game:

$$[\mathbf{A} \cdot \mathbf{x} + \mathbf{N} \cdot \mathbf{y} \leq \mathbf{b} \quad \rightarrow \quad \mathbf{C} \cdot \mathbf{x} + \mathbf{M} \cdot \mathbf{y} \leq \mathbf{d}] \tag{15.1}$$

The game is played in a sequence of $2 \cdot n$ rounds. Let $i = 1, \ldots, n$. In round $2 \cdot i - 1$, **X** makes his i^{th} move (by choosing values for variables in $\mathbf{x}_i$). Then, **Y** makes his i^{th} move (by choosing values for $\mathbf{y}_i$) in round $2 \cdot i$. Hence, **X** and **Y** make their moves by selecting values for their respective variables: $\mathbf{x}_i, \mathbf{y}_i \in \Re$, $i = 1, \ldots, n$. The moves are strictly alternating: **X**

will make his i^{th} move, which will be followed by **Y**'s i^{th} move, after which **X** will make his $(i+1)^{th}$ move and so on.

Every move that **X** or **Y** make changes the board configuration of the game by replacing variables (existentially quantified or universally quantified, depending on the round of the game) with their given values. For instance, in the first round, say that the existential player **X** chooses to give to a single variable x_1 the value $x'_1 \in \Re$, i.e., **X** sets $x_1 = x'_1$. Then, the board configuration will change from its initial state (see (15.1)) to the following:

$$[\mathbf{A}' \cdot \mathbf{x}' + \mathbf{N} \cdot \mathbf{y} \leq \mathbf{b} - x'_1 \mathbf{a}_1 \quad \rightarrow \quad \mathbf{C}' \cdot \mathbf{x}' + \mathbf{M} \cdot \mathbf{y} \leq \mathbf{d} - x'_1 \mathbf{c}_1] \tag{15.2}$$

After **X**'s first move, (15.2) is the *current board configuration* of the game. In this configuration, $\mathbf{A}'$ and $\mathbf{C}'$ are derived from **A** and **C** respectively by removing the first column (corresponding to x_1). Vector $\mathbf{x}'$ is derived from **x** by removing variable x_1, that is, $\mathbf{x}' = [x_2, \ldots, x_m]^T$). Note also that $x'_1 \mathbf{a}_1$ and $x'_1 \mathbf{c}_1$ are subtracted from **b** and **d** from the Left-Hand Side (LHS) and the Right-Hand Side (RHS) of the implication respectively. Vectors $\mathbf{a}_1$ and $\mathbf{c}_1$ denote the first column of **A** and **C** respectively, while recall that x'_1 is a constant (since it represents **X**'s choice for his first move).

Each move made by **X** or **Y** depends on the current board configuration and on the previous moves made by the opponent. Hence, the i^{th} move made by **X**, namely $\mathbf{x}_i$, may depend on the first $(i-1)$ moves made by **Y** and the board configuration after round $2 \cdot i - 2$. Similarly, $\mathbf{y}_i$ may depend on the first i moves made by **X** and the board configuration after round $2 \cdot i - 1$.

In any game of this form, the goals of the players are the following: **X** selects the values of the existentially quantified variables so as to violate the constraints in the LHS or to satisfy the constraints in the RHS of the implication. On the other hand, **Y** selects the values of the universally quantified variables so as to satisfy the constraints of the LHS and on the same time to violate the constraints of the RHS of the implication. We say that **X**

wins the game if at the end of the game (i.e., after the $2 \cdot n$ rounds) the board configuration is such that its LHS is false (as a conjunction of inequalities) or its RHS is true. Otherwise, we say that **Y** wins the game (i.e., if the LHS is satisfied and the RHS is falsified).

It is important to note that the game as described above is non-deterministic in nature, in that we have not specified *how* **X** and **Y** make their moves. We say that **X** *has a winning strategy* if it is possible for **X** to win the game, i.e., if there is a sequence of moves such that **X** wins the game. Otherwise, we say that **Y** has a winning strategy. The QLI holds precisely when player **X** has a winning strategy.

***Remark* 15.2.1.** *A QLI holds if and only if the existential player has a winning strategy.*

Let us now show that the proposed game is a conservative extension of the game semantics of QLP problems [Sub07]. There, an existential player **X** and a universal player **Y** also choose their moves according to the order of the variables in the corresponding quantifier string. If, at the end, the instantiated linear system in the QLP is true, then **X** wins the game (and we say that **X** has a winning strategy). Otherwise, **Y** wins the game. Based on these semantics, we explore the relation between QLPs and QLIs. Consider a generic QLP as described by the following system:

$$\begin{aligned} &\exists \mathbf{x}_1 \, \forall \mathbf{y}_1 \in [\mathbf{l}_1, \mathbf{u}_1] \, \ldots \exists \mathbf{x}_n \, \forall \mathbf{y}_n \in [\mathbf{l}_n, \mathbf{u}_n] \\ &\qquad\qquad\qquad\qquad \mathbf{A} \cdot \mathbf{x} + \mathbf{N} \cdot \mathbf{y} \leq \mathbf{b} \end{aligned} \tag{15.3}$$

Now consider the following QLI:

$$\begin{aligned} &\exists \mathbf{x}_1 \, \forall \mathbf{y}_1 \ldots \exists \mathbf{x}_n \, \forall \mathbf{y}_n \\ &\qquad \left.\begin{array}{c} \mathbf{l}_1 \leq \mathbf{y}_1 \leq \mathbf{u}_1 \\ \ldots \\ \mathbf{l}_n \leq \mathbf{y}_n \leq \mathbf{u}_n \end{array}\right\} \rightarrow \mathbf{A} \cdot \mathbf{x} + \mathbf{N} \cdot \mathbf{y} \leq \mathbf{b} \end{aligned} \tag{15.4}$$

where $\mathbf{l}_1, \ldots, \mathbf{l}_n$ and $\mathbf{u}_1, \ldots, \mathbf{u}_n$ are partitions of $\mathbf{l}$ and $\mathbf{u}$ and correspond to the lower and

upper bounds respectively on the variables in $\mathbf{y}_1, \ldots, \mathbf{y}_n$ of $\mathbf{y}$ that appear in the quantifier string of System (15.3).

***Theorem* 15.2.1.** *The existential player has a winning strategy in System (15.4) if and only if the existential player has a winning strategy in System (15.3).*

Proof. Note that the interval constraints on the universal variables that are in the quantifier string of the QLP (see System (15.3)) have been placed within the LHS of (15.4). These bounds are restrictive for the universal player in System (15.4) as well, although they are not within the quantifier string. Recall that the universal player **Y** wants the implication not to hold in order to win the game. Therefore, **Y** must choose values for the universally quantified variables so that the LHS is satisfied (otherwise, the existential player **X** trivially wins the game). Hence, we can safely assume that **Y** will satisfy the interval constraints in the LHS (while also trying to falsify $\mathbf{A} \cdot \mathbf{x} + \mathbf{N} \cdot \mathbf{y} \leq \mathbf{b}$).

If part. Assume that **X** has a winning strategy for System (15.3). Hence, there exists some $\mathbf{x}_0$ such that for any value $\mathbf{y}_0 \in [\mathbf{l}, \mathbf{u}]$ of $\mathbf{y}$, $\mathbf{A} \cdot \mathbf{x}_0 + \mathbf{N} \cdot \mathbf{y}_0 \leq \mathbf{b}$ is true. But then, since the universally quantified variables are restricted by $\mathbf{y} \in [\mathbf{l}, \mathbf{u}]$ due to the interval constraints in the LHS of (15.4), the existential player can choose the exact same values $\mathbf{x}_0$ and satisfy the RHS of the implication, i.e., making System (15.4) hold and hence winning the game.

Only-if part. Assume that **X** has a winning strategy for System (15.4) with $\mathbf{x}$ instantiated to $\mathbf{x}_0$. Since the universal player wants to satisfy the LHS in order to win, this means that $\mathbf{y}$ is instantiated to $\mathbf{y}_0$ such that the LHS necessarily holds (i.e., $\mathbf{y}_0 \in [\mathbf{l}, \mathbf{u}]$). Hence, the same vectors $\mathbf{x}_0$, $\mathbf{y}_0$ can be then used to make **X** win System (15.3). □

Note that the quantifier string of QLPs restricts the possible moves of the universal player through lower and upper bounds. The absence of such bounds in the quantifier string of QLIs follows from the fact that the universal player wants to satisfy the LHS of the implication. Hence, it is the satisfaction of the LHS that restricts the moves of the universal player in QLIs. If explicit interval constraints exist for the universal variables, these can also be placed within the LHS of the implication.

15.2.2 Complexity of QLI

In this section, we examine first the computational complexity of the generic class of QLIs with an arbitrary number of quantifier alternations. These problems are described by System (2.5).

***Theorem* 15.2.2.** *Problem (2.5) is* **PSPACE-hard**.

Proof. We will reduce the Q3SAT problem, which is **PSPACE-complete**, to an instance described by (2.5). Consider a Q3SAT instance $Q(\mathbf{x},\mathbf{y})\ \phi(\mathbf{x},\mathbf{y})$, where $Q(\mathbf{x},\mathbf{y})$ represents the quantifier string, $\mathbf{x}$ is the set of existentially quantified variables, $\mathbf{y}$ is the set of universally quantified variables, and ϕ is a conjunction of 3-literals clauses. We want to produce a corresponding QLI which will hold if and only if $Q(\mathbf{x},\mathbf{y})\ \phi(\mathbf{x},\mathbf{y})$ is satisfiable. Let E represent the set of constraints on the LHS of the implication and F the set of constraints on the RHS of the constructed implication.

For each existentially quantified variable x_i in the instance of Q3SAT, we add an existentially quantified variable x_i and a universally quantified variable r_i. We also add the constraints $r_i \leq x_i$ and $r_i \leq 1 - x_i$ to E and the constraints $r_i \leq 0$ to F. Note that these constraints are equivalent to $r_i \leq \min(x_i, 1-x_i) \rightarrow r_i \leq 0$. Moreover, we add $0 \leq x_i \leq 1$ to F.

For each universally quantified variable y_i in the instance of Q3SAT, we add an existentially quantified variable s_i and a universally quantified variable y_i. We also add the constraints $0 \leq y_i \leq 1$ to E and the constraints $0 \leq s_i \leq 1$, $2y_i - 1 \leq s_i$, and $s_i \leq 2y_i$ to F. Note that these are the only constraints that use y_i variables, since the clause constraints will only use x_i and s_i variables.

For each clause ϕ_j in the instance of Q3SAT, we add the universally quantified variable z_j and the constraint $z_j \geq 1$ to F. Then, depending on the form of the clause ϕ_j, we do one of the following:

1. If $\phi_j = (x_i, y_k, x_l)$, we add the constraint $z_j \geq x_i + s_k + x_l$ to E.

2. If $\phi_j = (x_i, y_k, \bar{x_l})$, we add the constraint $z_j \geq x_i + s_k + (1 - x_l)$ to E.

3. If $\phi_j = (x_i, \bar{y_k}, \bar{x_l})$, we add the constraint $z_j \geq x_i + (1 - s_k) + (1 - x_l)$ to E.

4. If $\phi_j = (\bar{x_i}, \bar{y_k}, \bar{x_l})$, we add the constraint $z_j \geq (1 - x_i) + (1 - s_k) + (1 - x_l)$ to E.

We create the quantifier string of the implication according to $Q(\mathbf{x}, \mathbf{y})$: for $\exists \mathbf{x}_i$ in $Q(\mathbf{x}, \mathbf{y})$, we introduce $\exists \mathbf{x}_i \forall \mathbf{r}_i$; for $\forall \mathbf{y}_i$ in $Q(\mathbf{x}, \mathbf{y})$, we introduce $\forall \mathbf{y}_i \exists \mathbf{s}_i$; finally, we introduce $\forall \mathbf{z}$. It is obvious that the resultant implication is a QLI described by System (2.5). Called $\hat{Q}$ the alternation of quantifiers in $Q(\mathbf{x}, \mathbf{y})$, we have that the alternation of quantifiers in the implication is $\hat{Q}\forall$ if $\hat{Q}$ ends with an existential quantifier, and $\hat{Q}\exists\forall$ if $\hat{Q}$ ends with a universal quantifier.

Now consider the semantics introduced in Section 15.2.1, and specifically the goals of the existential player $\mathbf{X}$. We can safely assume that $\mathbf{X}$ will only choose x_i from the set $\{0, 1\}$. This is because if $x_i \notin [0, 1]$, then at least one of the constraints in F would be violated. But then the implication would not hold (hence the universal player $\mathbf{Y}$ would win the game), since $\mathbf{X}$ cannot cause any constraint in E to be violated. On the other hand, if $x_i \in (0, 1)$, then $\mathbf{Y}$ could choose to set $r_i = \min(x_i, 1 - x_i) > 0$, which would cause the implication not to hold (causing the existential player to lose the game). To sum up, any choice of $x_i \notin \{0, 1\}$ would cause the existential player to lose the game.

Assume also that $\mathbf{Y}$ only chooses y_i from the set $\{0, 1\}$. We will show why this assumption is not restrictive. Suppose that $\mathbf{Y}$ can win by choosing $y_i \notin [0, 1]$. Then at least one constraint of E is violated (i.e., $0 \leq y_i \leq 1$) and the implication holds (i.e., a contradiction). Suppose now that $\mathbf{Y}$ can win by choosing $y_i \in (0, 1)$. If $y_i \in (0, \frac{1}{2}]$, then we have that $s_i \in [0, 2y_i]$. However, if instead $\mathbf{Y}$ had chosen $y_i = 0$, then $s_i \in \{0\} \subseteq [0, 2y_i]$, thus restricting the possible responses of $\mathbf{X}$. Since y_i only appears in the constraints described above, we have that this is a strictly better move for $\mathbf{Y}$. Similarly, choosing $y_i = 1$ is strictly better for $\mathbf{Y}$ than choosing $y_i \in [\frac{1}{2}, 1)$. To sum up, we can safely assume that the universal player would only choose $y_i \in \{0, 1\}$.

Since y_i is in the set $\{0, 1\}$, we have that the existential player is forced to set $s_i =$

y_i. Any other choice of s_i would violate at least one constraint of F, causing (again) the existential player to lose the game. Hence, s_i variables are also restricted in the set $\{0,1\}$.

Let us show that for the QLI obtained from a Q3SAT instance of the form $Q(\mathbf{x},\mathbf{y})\ \phi(\mathbf{x},\mathbf{y})$, the existential player has a winning strategy for the QLI if and only if $Q(\mathbf{x},\mathbf{y})\ \phi(\mathbf{x},\mathbf{y})$ is satisfiable.

Only-if part. Assume the existential player has a winning strategy U for the constructed QLI. This means that for every sequence of moves V made by the universal player, the implication holds. Consider the constraints constructed from the j^{th} clause of $\phi(\mathbf{x},\mathbf{y})$, i.e., ϕ_j, and assume without loss of generality that ϕ_j is of the form (x_i,y_k,x_l). Since the implication holds (for strategy U), we must have that $z_j \geq x_i + s_k + x_l \rightarrow z_j \geq 1$. But this means that $x_i + s_k + x_l \geq 1$. Recall now that x_i, s_k, and x_l are restricted to the set $\{0,1\}$. Therefore, at least one of these variables must be 1. Thus, at least one of the literals in the original clause is true causing ϕ_j to be true as well. The same can be argued for all clauses of $\phi(\mathbf{x},\mathbf{y})$. Hence, $Q(\mathbf{x},\mathbf{y})\ \phi(\mathbf{x},\mathbf{y})$ is satisfiable.

If part. Assume that $Q(\mathbf{x},\mathbf{y})\ \phi(\mathbf{x},\mathbf{y})$ is satisfiable. Then there exist values $\mathbf{x}'$ for $\mathbf{x}$ such that

$$\mathbf{x}' = [c_1, f_1(y_1), f_2(y_1,y_2), \ldots, f_{n-1}(y_1,y_2,\ldots,y_{n-1}]^T$$

and for any values $\mathbf{y}' = [y_1,y_2,\ldots,y_n]^T$ given to $\mathbf{y}$, the Q3SAT expression is satisfied. Note that $f_i()$ are Skolem functions and are used to represent that the values of the elements of $\mathbf{x}'$ depend on the values of the corresponding elements of $\mathbf{y}'$. Consider the constraints constructed from the jth clause of $\phi(\mathbf{x},\mathbf{y})$, i.e., ϕ_j, and assume without loss of generality that ϕ_j is of the form (x_i,y_k,x_l). Since the Q3SAT expression is satisfied (for $\mathbf{x} = \mathbf{x}'$), we must have that at least one of x_i,y_k,x_l is true. This means that at least one of x_i,s_k,x_l is 1. Thus, we have that $x_i + s_k + x_l \geq 1$, and so $z_j \geq x_i + s_k + x_l \rightarrow z_j \geq 1$ holds. The same can be safely argued for all constraints corresponding to clauses. Since the x_i and y_i variables can be restricted to the set $\{0,1\}$ and since we can then restrict $s_i = y_i$, we have that for each i, $r_i \leq x_i$ and $r_i \leq 1 - x_i$ imply that $r_i \leq 0$ (thus satisfying the corresponding constraint

of F). For the same reason, we have that for each i the constraints $0 \leq x_i \leq 1$, $0 \leq y_i \leq 1$, $0 \leq s_i \leq 1$, $2y_i - 1 \leq s_i$, and $s_i \leq 2y_i$ are all satisfied. Thus $E \rightarrow F$ and so the existential player has a winning strategy for the corresponding QLI. □

15.2.3 Complexity of UQLI andd PQLI

We utilize the results of Section 14.2.2 to show that PQLI and UQLI are solvable in polynomial time.

***Corollary* 15.2.1.** *PQLI decidability is in* **P**.

Proof. Consider the following PQLI:

$$\exists \mathbf{x_1}\, \forall \mathbf{y_1}\, \ldots\, \exists \mathbf{x_n}\, \forall \mathbf{y_n}\ [\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b},\ \mathbf{y} \geq \mathbf{0} \rightarrow \mathbf{C} \cdot \mathbf{x} + \mathbf{D} \cdot \mathbf{y} \leq \mathbf{f}] \tag{15.5}$$

We will focus on the following two cases:

1. There exists a point $\mathbf{z}$, such that $\mathbf{A} \cdot \mathbf{z} \not\leq \mathbf{b}$ - It follows that $\forall \mathbf{x}\ (\mathbf{A} \cdot \mathbf{x} \leq \mathbf{b})$ does not hold. In this case, the PQLI is trivially satisfied, since the existential player can guess $\mathbf{z}$ and cause the left hand side of the implication to be falsified.

2. There is no point $\mathbf{z}$ such that $\mathbf{A} \cdot \mathbf{z} \not\leq \mathbf{b}$ - In this case, the PQLI (15.5) reduces to the following PQLI.

$$\exists \mathbf{x_1}\, \forall \mathbf{y_1}\, \ldots\, \exists \mathbf{x_n}\, \forall \mathbf{y_n}\ [\mathbf{y} \geq \mathbf{0} \rightarrow \mathbf{C} \cdot \mathbf{x} + \mathbf{D} \cdot \mathbf{y} \leq \mathbf{f}] \tag{15.6}$$

PQLI (15.6) can in turn be written as:

$$\exists \mathbf{x_1}\, \forall \mathbf{y_1} \in [\mathbf{0}, +\infty) \ldots\, \exists \mathbf{x_n}\, \forall \mathbf{y_n} \in [\mathbf{0}, +\infty)\ \ \mathbf{C} \cdot \mathbf{x} + \mathbf{D} \cdot \mathbf{y} \leq \mathbf{f} \tag{15.7}$$

However, System (15.7) is a PQLP, and hence can be decided in polynomial time by Theorem 14.2.1.

□

***Corollary* 15.2.2.** *UQLI decidability is in* **P**.

Proof. Consider the following UQLI:

$$\exists \mathbf{x_1}\, \forall \mathbf{y_1}\, \ldots\, \exists \mathbf{x_n}\, \forall \mathbf{y_n}\ \ [\mathbf{A}\cdot\mathbf{x} \leq \mathbf{b} \rightarrow \mathbf{C}\cdot\mathbf{x} + \mathbf{D}\cdot\mathbf{y} \leq \mathbf{f}] \tag{15.8}$$

As argued in the proof of Corollary 15.2.1, if $\forall \mathbf{x}\ (\mathbf{A}\cdot\mathbf{x} \leq \mathbf{b})$ does not hold, then UQLI (15.8) is trivially satisfied.

Otherwise, UQLI (15.8) is equivalent to:

$$\exists \mathbf{x_1}\, \forall \mathbf{y_1}\, \ldots\, \exists \mathbf{x_n}\, \forall \mathbf{y_n}\ \ \mathbf{C}\cdot\mathbf{x} + \mathbf{D}\cdot\mathbf{y} \leq \mathbf{f}$$

which is a UQLP, and hence can be decided in polynomial time by Theorem 14.2.2. □

15.2.4 QLI and the polynomial hierarchy

In this section, we prove that for each class of the **PH**, there exists a class of QLI decidability that is complete for that class. This is interesting, since QLIs are comprised of continuous variables, as opposed to the discrete variables comprising QBFs. Hence, we provide a continuous analogue to the results in [Sto77], where the **PH** is generated using QBFs.

Let $\mathbf{B}^{k+1}$ denote a string of $(k+1)$ **B**s.

***Theorem* 15.2.3.** $\langle k, \exists, \mathbf{B}^{k+1} \rangle$ *with k odd, is* $\mathbf{\Sigma_k P}$**-hard**.

Proof. Consider the class of Q3SAT formulas with k quantifiers starting with an existential one, i.e., with the quantifier string of the form $\exists\forall\ldots\exists$. Such a class is Σ_k**P-complete** (the

assumption that k is odd is essential) [Sto77, Theorem 4.1]. The proof of Theorem 15.2.2 reduces such a class to a QLI with a quantifier string obtained by adding a universal quantifier at the end, namely to a $\langle k, \exists, \mathbf{B}^{k+1} \rangle$ formula. Hence, the result. □

***Theorem* 15.2.4.** *$\langle k, \forall, \mathbf{B}^{k+1} \rangle$ with k even, is* **Π_kP-hard**.

Proof. Consider the class of Q3SAT formulas with k quantifiers starting with a universal one, i.e., with the quantifier string of the form $\forall\exists\ldots\exists$. Such a class is **$\Pi_k$P-complete** (the assumption that k is even is essential) [Sto77, Theorem 4.1]. The proof of Theorem 15.2.2 reduces such a class to a QLI with a quantifier string obtained by adding a universal quantifier at the end, namely to a $\langle k, \forall, \mathbf{B}^{k+1} \rangle$ formula. Hence, the result. □

To establish the computational complexities of $\langle k, \exists, \mathbf{B}^{k+1} \rangle$ when k is even, and $\langle k, \forall, \mathbf{B}^{k+1} \rangle$ when k is odd, we first provide a reduction from Q3DNF to QLI.

15.2.4.1 Reduction from Q3DNF to QLI

Consider a Q3DNF instance $\Phi : Q(\mathbf{x}, \mathbf{y})\phi(\mathbf{x}, \mathbf{y})$, where $Q(\mathbf{x}, \mathbf{y})$ represents the quantifier string, $\mathbf{x}$ is the set of existentially quantified variables, and $\mathbf{y}$ is the set of universally quantified variables. Note that $\phi(\mathbf{x}, \mathbf{y}) = \phi_1 \wedge \phi_2 \wedge \ldots \wedge \phi_m$ where each ϕ_i is a *disjunctive* clause. Without loss of generality, we can assume that the innermost quantifier of $Q(\mathbf{x}, \mathbf{y})$ is $\forall$, since if the innermost quantifier is $\exists$, then this variable can be eliminated in polynomial time.

We will produce a QLI $\mathbf{I} : Q'(\mathbf{x}, \mathbf{r}, \mathbf{y}, \mathbf{r}, \mathbf{w})\ E \rightarrow F$, such that the existential player has a winning strategy for $\mathbf{I}$ if and only if the existential player has a winning strategy for Φ. Note that E represent the set of constraints on the LHS of the constructed implication, and F the set of constraints on the RHS of the implication.

For each existentially quantified variable x_i in Φ, we add an existentially quantified variable x_i and a universally quantified variable r_i to $\mathbf{I}$. We also add the constraints $r_i \leq x_i$ and $r_i \leq 1 - x_i$ to E, and the constraint $r_i \leq 0$ to F. Note that these constraints are equivalent to $r_i \leq \min(x_i, 1 - x_i) \rightarrow r_i \leq 0$. Finally, we add the constraint $0 \leq x_i \leq 1$ to F.

For each universally quantified variable y_i in $\mathbf{\Phi}$, we add an existentially quantified variable s_i and a universally quantified variable y_i to $\mathbf{I}$. We also add the constraints $0 \leq y_i \leq 1$ to E, and the constraints $0 \leq s_i \leq 1$, $2 \cdot y_i - 1 \leq s_i$, and $s_i \leq 2 \cdot y_i$ to F. Note that these are the only constraints that use y_i variables, since the clause constraints use only x_i and s_i variables.

For each clause ϕ_j in $\phi(\mathbf{x}, \mathbf{y})$, we add the existentially quantified variable w_j to $\mathbf{I}$, and 3 constraints to F. These constraints ensure that w_j is less than or equal to the existential variables corresponding to the literals in ϕ_j. Note that these constraints contain only existential variables. Even in the case of a universal variable y_i in ϕ_j, the constraint contains the existential variable s_i of the QLI.

For example, if $\phi_j = (x_i, y_k, \bar{x_l})$, we add $w_j \leq x_i$, $w_j \leq s_k$, and $w_j \leq 1 - x_l$ to F, while if $\phi_j = (x_i, \bar{y_k}, \bar{x_l})$, we add $w_j \leq x_i$, $w_j \leq 1 - s_k$, and $w_j \leq 1 - x_l$ to F.

Finally, we add the linear constraint $w_1 + w_2 + \cdots + w_m \geq 1$ to F.

We inductively create the quantifier string $Q'(\mathbf{x}, \mathbf{r}, \mathbf{y}, \mathbf{s}, \mathbf{w})$ of $\mathbf{I}$ based on $Q(\mathbf{x}, \mathbf{y})$: $Q'(\mathbf{x}, \mathbf{r}, \mathbf{y}, \mathbf{s}, \mathbf{w})$ of $\mathbf{I}$ is initialized to ε (the empty string). For each $i = 1, 2, \ldots, n$,

1. If the i^{th} quantifier of $Q(\mathbf{x}, \mathbf{y})$ is $\exists \mathbf{x}_i$, then we append $\exists \mathbf{x}_i \forall \mathbf{r}_i$ to the end of $Q'(\mathbf{x}, \mathbf{r}, \mathbf{y}, \mathbf{s}, \mathbf{w})$.

2. If the i^{th} quantifier of $Q(\mathbf{x}, \mathbf{y})$ is $\forall \mathbf{y}_i$, then we append $\forall \mathbf{y}_i \exists \mathbf{s}_i$ to the end of $Q'(\mathbf{x}, \mathbf{r}, \mathbf{y}, \mathbf{s}, \mathbf{w})$.

Finally, we add $\exists \mathbf{w}$ to the end of $Q'(\mathbf{x}, \mathbf{r}, \mathbf{y}, \mathbf{r}, \mathbf{w})$.

Since the final quantifier of $Q(\mathbf{x}, \mathbf{y})$ is $\forall$, the number of quantifier alternations in $Q'(\mathbf{x}, \mathbf{r}, \mathbf{y}, \mathbf{s}, \mathbf{w})$ is one more than the number of quantifier alternations in $Q(\mathbf{x}, \mathbf{y})$.

Note that the final quantifier of $Q'(\mathbf{x}, \mathbf{r}, \mathbf{y}, \mathbf{r}, \mathbf{w})$ is $\exists$.

Example 46: Let us consider the Q3DNF instance

$$\mathbf{\Phi} = \exists x_1 \, \forall y_1 \, \exists x_2 \, \forall y_2 \; \underbrace{(x_1, \bar{y_1}, y_2)}_{\phi_1} \vee \underbrace{(\bar{x_1}, x_2, y_2)}_{\phi_2}.$$

Under the above construction, Φ becomes an instance of $\langle 4, \exists, \mathbf{BBBBR} \rangle$ with the quantifier string

$$\exists x_1 \, \forall r_1 \, \forall y_1 \, \exists s_1 \, \exists x_2 \, \forall r_2 \, \forall y_2 \, \exists s_2 \, \exists w_1 \, \exists w_2.$$

Note that this is also an instance of $\langle 4, \exists, \mathbf{BBBBB} \rangle$.

The LHS of the implication (E) consists of the following set of constraints:

$$\begin{aligned} r_1 \leq x_1, \qquad & r_1 \leq 1 - x_1, \\ r_2 \leq x_2, \qquad & r_2 \leq 1 - x_2, \\ 0 \leq y_1 \leq 1, \qquad & 0 \leq y_2 \leq 1, \end{aligned}$$

The RHS of the implication (F) consists of the following set of constraints:

$$\begin{aligned} r_1 \leq 0, \qquad & 0 \leq x_1 \leq 1, \\ r_2 \leq 0, \qquad & 0 \leq x_2 \leq 1, \\ 0 \leq s_1 \leq 1, \qquad & 0 \leq s_2 \leq 1, \\ 2 \cdot y_1 - 1 \leq s_1, \qquad & s_1 \leq 2 \cdot y_1, \\ 2 \cdot y_2 - 1 \leq s_2, \qquad & s_2 \leq 2 \cdot y_2, \\ \phi_1 \left\{ \begin{array}{ll} w_1 \leq x_1, & w_2 \leq 1 - x_1, \\ w_1 \leq 1 - s_1, & w_2 \leq x_2, \\ w_1 \leq s_2, & w_2 \leq s_2, \end{array} \right\} & \phi_2 \\ w_1 + w_2 \geq 1. & \end{aligned}$$

We now establish that all the variables in $\mathbf{I}$ can be restricted to $\{0, 1\}$ without affecting its feasibility. To do so, we utilize the game semantics discussed in Section 15.2.1.

***Lemma* 15.2.1.** *The existential player* $\mathbf{X}$ *has a winning strategy, if and only if he has a winning strategy, when each* x_i *is chosen from* $\{0, 1\}$.

Proof. The *if* part is obvious.

Accordingly, we focus on proving the *only if* part. Assume that the existential player **X** has a winning strategy. First, observe that is if $x_i \notin [0,1]$, then at least one of the constraints in F is violated. In particular, the constraint $0 \leq x_i \leq 1$ is violated. Now focus on the constraint $r_i \leq x_i$ in E. Since, r_i is chosen by the universal player **Y** **after** the existential player chooses x_i, this constraint can be violated by **Y**. In other words, the implication does not hold and **X** does not have a winning strategy.

We now consider the case $x_i \in (0,1)$. In this case, **Y** could choose $r_i = \min(x_i, 1-x_i)0$. Note that both x_i and r_i are positive with $r_i \leq x_i$ and $r_i \leq 1-x_i$. Thus the universal player **Y** wins the game, since the constraints involving x_i and r_i in E, viz., $r_i \leq x_i$ and $r_i \leq 1-x_i$ are satisfied and the constraint $r_i \leq 0$ in F is violated.

To sum up, any choice of $x_i \notin \{0,1\}$ would cause **X** to lose the game.

It follows that if **X** has a winning strategy, then he has a winning strategy when each x_i is chosen from $\{0,1\}$. □

***Lemma* 15.2.2.** *The universal player* **Y** *has a winning strategy, if and only if he has a winning strategy, when each y_i is chosen from* $\{0,1\}$.

Proof. The *if* part if obvious. Accordingly, we focus on proving the *only-if* part. It is clear that in order to win, the universal player **Y** must choose $y_i \in [0,1]$. Otherwise, the constraint $0 \leq y_i \leq 1$ in the LHS E of the implication is violated and the existential player **X** wins the game.

Suppose now that **Y** can win by choosing $y_i \in (0,1)$. As per the construction of subsection 15.2.4.1, the only constraints involving s_i and y_i are given by System (15.9).

$$0 \leq \quad s_i \quad \leq 1, 2 \cdot y_i - 1 \leq s_i, s_i \leq 2 \cdot y_i \tag{15.9}$$

If $y_i \in (0, \frac{1}{2}]$, then from System (15.9) it follows that $s_i \in [0,1$. However, if instead **Y** had chosen $y_i = 0$, then $s_i \in \{0\}$. It follows that if **Y** can win by choosing $y_i \in (0, \frac{1}{2}]$, then

Y can win by choosing $y_i = 0$. Similarly, if **Y** can win by choosing $y_i \in [\frac{1}{2}, 1)$, then **Y** can win by choosing $y_i = 1$.

To sum up, we can safely assume that the universal player only chooses $y_i \in \{0,1\}$.

It follows that if **Y** has a winning strategy, then he has a winning strategy when each y_i is chosen from $\{0,1\}$. □

The import of Lemma 15.2.1 and Lemma 15.2.2 is that we can confine our analyses to games in which the existential player **X** and universal player **Y** make moves in $\{0,1\}$ for the x_i and y_i variables respectively. An interesting observation is that if y_i is restricted to $\{0,1\}$, then so is s_i.

The constraints involving s_i and y_i in the RHS F of the implication are described by System (15.9). If $y_i = 0$, the constraints in System (15.9) force s_i to be 0; likewise, if $y_i = 1$, they force s_i to be 1. In other words, **X** is forced to set $s_i = y_i$. Any other choice of s_i would violate at least one constraint of F, causing **X** to lose the game. Hence, the s_i variables are also restricted in the set $\{0,1\}$.

***Theorem* 15.2.5.** **I** *is feasible if and only if* Φ *is feasible.*

Proof. We show that the existential player **X** has a winning strategy for **I** if and only if he has a winning strategy for Φ.

Only if part: Assume that the existential player **X** has a winning strategy for **I**. Consider a play of the game, in which **X** chooses values for the existentially quantified variables and **Y** chooses values for the universally quantified variables. After **X** chooses a value for x_i, **Y** can choose r_i, such that the constraints $r_i \leq x_i$ and $r_i \leq 1 - x_i$ are both satisfied. By our 2-person game semantics, the universal player **Y** will ensure that all the constraints in the LHS are satisfied. In order for **X** to win the game, he has to ensure that all the constraints in the RHS are satisfied as well. In particular, the constraint $w_1 + w_2 + \cdots + w_m \geq 1$ in the RHS of the implication must be satisfied. Consequently, $w_j > 0$, for at least one j. This w_j corresponds to the clause ϕ_j of Φ. Assume that ϕ_j has the form (x_i, y_k, x_l). Since the x_is and s_is are restricted to the set $\{0,1\}$ (see Lemma 15.2.1 and Lemma 15.2.2), the constraints

$w_j \leq x_i$, $w_j \leq s_k$, and $w_j \leq x_l$ force each variable (x_i, s_k and x_l) to be 1. It follows that ϕ_j is satisfied in this play. Similar arguments can be made for other forms of ϕ_j. Since the play was chosen arbitrarily, the same argument applies for all plays, i.e., at least one clause of $\phi(\mathbf{x},\mathbf{y})$ is satisfied in every play. Hence, Φ is feasible.

If part: Assume that the existential player **X** has a winning strategy for Φ.

At the end of any play, $\phi(\mathbf{x},\mathbf{y})$ is satisfied. Thus, at least one clause, say ϕ_j, must be satisfied. Assume that ϕ_j is of the form (x_i, y_k, x_l). Consider the constraints constructed from ϕ_j, viz., $w_j \leq x_i$, $w_j \leq s_k$, and $w_j \leq x_l$. Since ϕ_j is satisfied, x_i, y_k, x_l are all **true**. Assume that **X** sets the variables x_i, s_k, x_l and w_j to 1 and the w variables associated with other clauses to 0. It is clear that the constraints corresponding to the other clauses are trivially satisfied. Likewise, the the aggregate constraint $w_1 + w_2 + \cdots + w_m \geq 1$ is also satisfied. Similar arguments can be made for other forms of ϕ_j.

From Lemma 15.2.1 and Lemma 15.2.2 and the subsequent discussion, x_i and y_i can be restricted to the set $\{0,1\}$ Furthermore, the existential player must choose $s_i = y_i$. Observe that for each i, $r_i \leq x_i$ and $r_i \leq 1 - x_i$ imply that $r_i \leq 0$ (thus satisfying the corresponding constraint in F). Similarly, for any i, the constraints $0 \leq x_i \leq 1$, $0 \leq y_i \leq 1$, $0 \leq s_i \leq 1$, $2 \cdot y_i - 1 \leq s_i$, and $s_i \leq 2 \cdot y_i$ are all satisfied, under this assignment. Hence, $E \rightarrow F$ holds. Since the play was chosen arbitrarily, the same argument applies for all player, i.e., **X** has a winning strategy for **I** and **I** is feasible. □

Theorem (15.2.5) allows us to obtain the following two results.

Corollary* 15.2.3.** *$\langle k, \exists, \mathbf{B}^{k+1} \rangle$ with k even, is Σ_kP-hard**.

Proof. Let Φ denote a Q3DNF formula having k (k even) quantifiers ($(k-1)$ quantifier alternations) starting with $\exists$, i.e., with a quantifier string of the form $\exists\forall\ldots\exists\forall$. Deciding the feasibility of Φ is Σ_k**P-complete**. This is because this problem is the complement of the problem $\forall\exists\ldots\forall\exists\ \phi(\mathbf{x},\mathbf{y})$, where $\phi(\mathbf{x},\mathbf{y})$ is a 3CNF formula, which is Π_k**P-complete** [Pap94] (the assumption that k is even is essential). By Theorem 15.2.5, we can reduce Φ to a QLI, **I**. Note that the quantifier string of **I** has k quantifier alternations, and that the

first and last quantifiers are $\exists$. Thus, $\mathbf{I}$ is a $\langle k, \exists, \mathbf{B}^k\mathbf{R}\rangle$ QLI. This can be trivially reduced to a $\langle k, \exists, \mathbf{B}^{k+1}\rangle$ QLI. Hence, the result follows. □

***Corollary* 15.2.4.** *$\langle k, \forall, \mathbf{B}^{k+1}\rangle$ with k odd, is $\mathbf{\Pi}_k$**P-hard**.*

Proof. Let Φ denote a Q3DNF formula having k (k odd) quantifiers ($(k-1)$ quantifier alternations) starting with $\forall$, i.e., with a quantifier string of the form $\forall\exists\ldots\exists\forall$. Deciding the feasibility Φ belongs to is $\mathbf{\Pi}_k$**P-complete**. This is because this problem is the complement of $\exists\forall\ldots\exists\,\phi(\mathbf{x},\mathbf{y})$, where $\phi(\mathbf{x},\mathbf{y})$ is a 3CNF formula, which is $\mathbf{\Pi}_k$**P-complete** [Pap94] (the assumption that k is odd is essential). By Theorem 15.2.5, we can reduce Φ to a QLI, $\mathbf{I}$. Note that the quantifier string of $\mathbf{I}$ has k quantifier alternations, and that the last quantifiers is $\exists$. Thus, $\mathbf{I}$ is a $\langle k, \forall, \mathbf{B}^k\mathbf{R}\rangle$ QLI. This can be trivially reduced to a $\langle k, \forall, \mathbf{B}^{k+1}\rangle$ QLI. Hence, the result follows. □

***Theorem* 15.2.6.** *$\langle k, \exists, \mathbf{B}^{k+1}\rangle$ is $\mathbf{\Sigma}_k$**P-complete**, $\langle k, \forall, \mathbf{B}^{k+1}\rangle$ is $\mathbf{\Pi}_k$**P-complete**.*

Proof. Given Theorems 15.2.3-15.2.4 and Corollaries 15.2.3-15.2.4, it suffices to show that that each of these problems is also contained within the corresponding complexity class of the polynomial hierarchy. Let $\mathbf{P}$ denote the problem of deciding an arbitrary boolean combination of linear constraints under a quantifier string with a limited number of alternations. Sontag [Son85] showed that problem $\mathbf{P}$ can be solved by an alternating algorithm in which the guesses made by both the $\forall$ player and the $\exists$ player are rational and at most polynomial in the size of the input. QLIs with a finite number of quantifier alternations are clearly sub-problems of $\mathbf{P}$, since they can be rewritten as a quantified disjunction of the RHS constraints and the negation of the LHS constraints. It follows that the feasibility of QLIs with a finite number of alternations is preserved, even if every variable is restricted to values that are polynomially-sized with respect to the input. Consider the problem $\langle k, \exists, \mathbf{B}^{k+1}\rangle$. After k rounds in which the $\exists$ player and $\forall$ player alternate and guess polynomially-sized values, the QLI reduces to either $\langle 0, \exists, \mathbf{B}\rangle$ or $\langle 0, \forall, \mathbf{B}\rangle$ (depending on whether k is even or odd respectively), both of which are in $\mathbf{P}$. Thus, $\langle k, \exists, \mathbf{B}^{k+1}\rangle$ is in $\mathbf{\Sigma}_k\mathbf{P}$. Likewise, $\langle k, \forall, \mathbf{B}^{k+1}\rangle$ is in $\mathbf{\Pi}_k\mathbf{P}$. □

For example, QLIs of the form $\langle 3, \forall, \mathbf{BBBB} \rangle$ are in $\mathbf{\Pi_3 P}$.

The various forms of QLI cover the polynomial hierarchy as shown in Figure 15.1.

We make the following observations on the basis of the theorems derived above:

1. In case of QLPs, if the innermost variable is universally quantified, it can be removed using quantifier elimination techniques, in polynomial time. In case of QLIs, the complexity class to which the problem belongs, does in fact depend upon whether the innermost variable is existentially or universally quantified (see Figure 15.1).

2. There exists a class of QLI that is complete for each class in the **PH**. This is not true, if the underlying formula is in CNF form. For instance, there is no QCNF formula, which is complete for the class **coNP**.

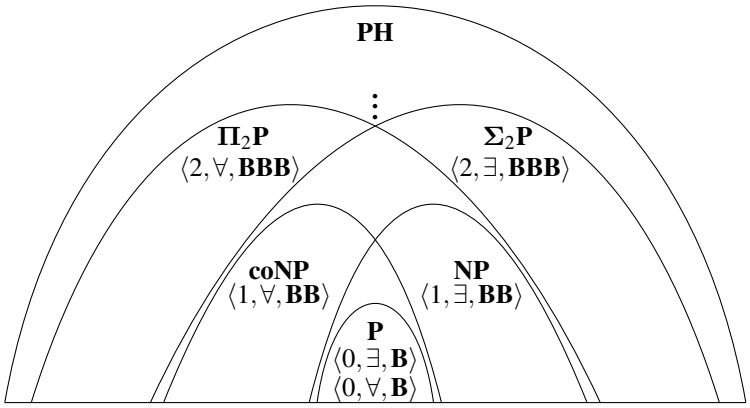

Figure 15.1: QLI and the Polynomial Hierarchy

15.2.5 Complexity with bounded alternation

In this section, we examine the computational complexities of various classes of QLI with one quantifier alternation.

***Lemma* 15.2.3.** $\langle 1, \forall, \mathbf{RB} \rangle$*, i.e., deciding whether* $\forall \mathbf{y}\ \exists \mathbf{x}\ [\mathbf{M} \cdot \mathbf{x} \leq \mathbf{n} \rightarrow \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{y} \leq \mathbf{c}]$ *holds, is in* **P**.

Proof. We will focus on the following two cases:

1. There exists a point $\mathbf{z}$, such that $\mathbf{M} \cdot \mathbf{z} \not\leq \mathbf{n}$ - It follows that $\forall \mathbf{x}\ (\mathbf{M} \cdot \mathbf{x} \leq \mathbf{n})$ does not hold. In this case, the QLI is trivially satisfied, since the existential player can guess $\mathbf{z}$ and cause the left hand side of the implication to be falsified.

2. There is no point $\mathbf{z}$ such that $\mathbf{M} \cdot \mathbf{z} \not\leq \mathbf{n}$ - In this case, the QLI reduces to $\forall \mathbf{y}\ \exists \mathbf{x}\ \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{y} \leq \mathbf{c}$, which is a UQLP and hence can be decided in polynomial time by Theorem 14.2.2.

□

The decision problem for formula $\exists \mathbf{x}\ \forall \mathbf{y}\ [\mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{y} \leq \mathbf{c} \rightarrow \mathbf{M} \cdot \mathbf{y} \leq \mathbf{n}]$ is **NP-complete**. However, if there are no universally quantified variables on the RHS, the problem becomes tractable.

***Lemma* 15.2.4.** $\langle 1, \exists, \mathbf{BL} \rangle$*, i.e., deciding whether* $\exists \mathbf{x}\ \forall \mathbf{y}\ [\mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{y} \leq \mathbf{c} \rightarrow \mathbf{M} \cdot \mathbf{x} \leq \mathbf{n}]$ *holds, is in* **P**.

Proof. First, we check whether $\exists \mathbf{x}\ \mathbf{M} \cdot \mathbf{x} \leq \mathbf{n}$ holds, which can be done in polynomial time [Kha79]. If $\exists \mathbf{x}\ \mathbf{M} \cdot \mathbf{x} \leq \mathbf{n}$ holds, then $\exists \mathbf{x}\ \forall \mathbf{y}\ [\mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{y} \leq \mathbf{c} \rightarrow \mathbf{M} \cdot \mathbf{x} \leq \mathbf{n}]$ trivially holds. If $\exists \mathbf{x}\ \mathbf{M} \cdot \mathbf{x} \leq \mathbf{n}$ does not hold, then the only way in which $\exists \mathbf{x}\ \forall \mathbf{y}\ [\mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{y} \leq \mathbf{c} \rightarrow \mathbf{M} \cdot \mathbf{x} \leq \mathbf{n}]$ can hold is if $\exists \mathbf{x}\ \forall \mathbf{y}\ \mathbf{A} \cdot \mathbf{x} + \mathbf{B} \cdot \mathbf{y} \leq \mathbf{c}$ does not hold. However, the latter formula is a UQLP and hence it can be checked to hold in polynomial time by Theorem 14.2.2. □

Let us turn our attention to the class $\langle 1, \forall, \mathbf{BR} \rangle$ and its super-class $\langle 1, \forall, \mathbf{BB} \rangle$. Both these classes were shown to be **coNP-hard**. Note that these classes are also in **coNP** by Theorem 15.2.6.

Lemma 15.2.3 and Lemma 15.2.4 settle some open problems on the computational complexities of QLIs with one quantifier alternation.

A complete representation of all classes with one quantifier alternation QLIs is given in Figure 15.2 (starting with an existential quantifier) and Figure 15.3 (starting with a universal quantifier).

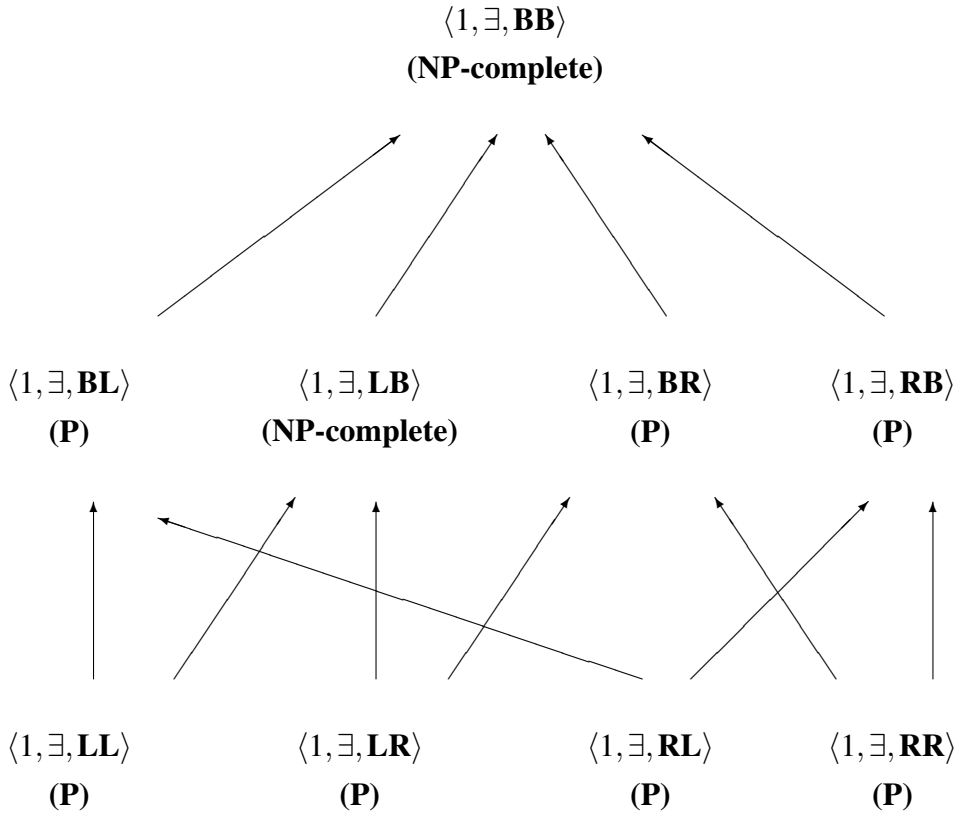

Figure 15.2: Complexity of $\exists\forall$ classes of QLI. Arrows denote inclusions.

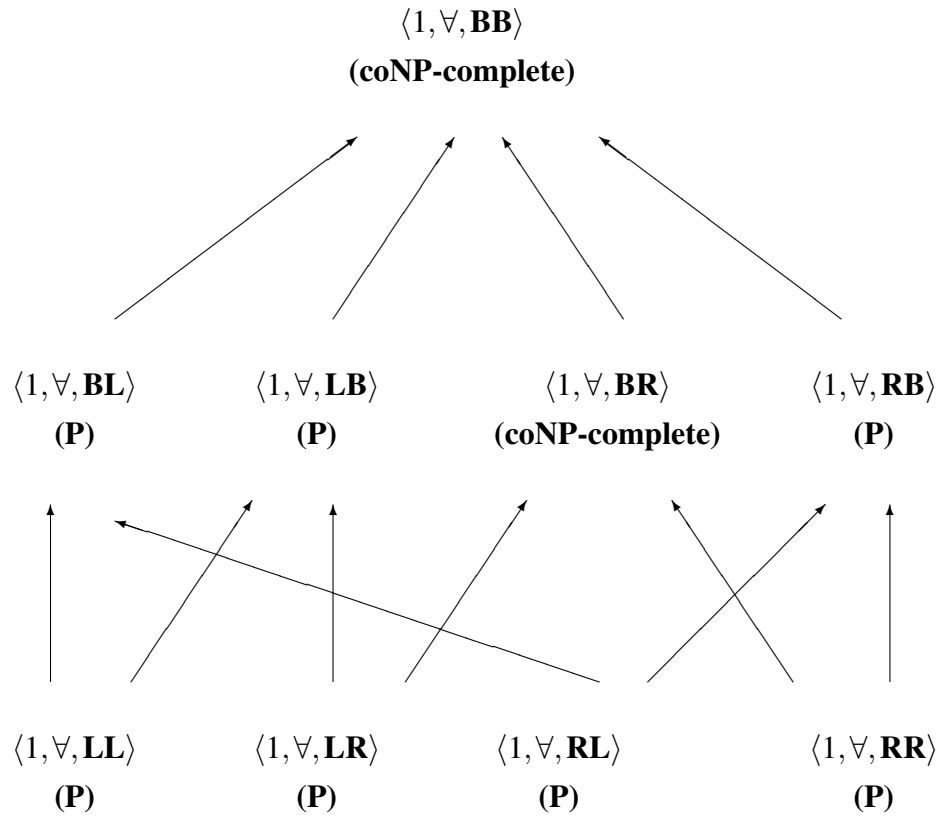

Figure 15.3: Complexity of $\forall\exists$ classes of QLI. Arrows denote inclusions.

Part VI

Conclusion

Chapter 16

Summary of Results

In this chapter we summarize the results in this dissertation.

16.1 Results for Boolean CSPs

First we list out results pertaining to Boolean formulas. This includes results for 2CNF formulas, 3 CNF formulas, and Horn formulas. For 2 CNF formulas, we have the following results:

1. The problem of identifying if a 2CNF formula has a read-once resolution refutation is **NP-complete**.
2. The problem of identifying if a 2CNF formula has a read-once NAE-resolution refutation is in **P**.
3. The problem of finding the shortest NAE-refutation of a 2CNF formula is in **P**.
4. The problem of finding a read-once unit resolution refutation for a 2CNF formula is in **P**.

For 3CNF formulas, we showed that the problem of identifying if a 3 CNF formula has a read-once NAE-resolution refutation is **NP-complete**.

For Horn formulas, we have the following results:

1. The problem of finding the shortest read-once resolution refutation of a Horn formula is **NP-hard**.

2. The problem of identifying if a Horn formula has a read-once unit resolution refutation is **NP-complete**.

3. The copy complexity of Horn formulas with respect to unit resolution is 2^{n-1} where n is the number of variables in the formula.

16.2 Results for Linear Polyhedral CSPs

In this section, we list our results for linear programs. This includes results for difference constraints, UTVPI constraints, and Horn constraints. For systems of difference constraints, we have developed an FPTAS for the problem of finding the shortest weighted read-once refutation.

For systems of Difference constraints, we have the following results:

1. Developing a polynomial time randomized algorithm for finding the shortest read-once linear refutation of a system of difference constraints.

2. Developing a pseudo-polynomial time algorithm for finding the shortest weighted read-once linear refutation of a system of difference constraints.

3. Developing an FPTAS for the problem of finding the shortest weighted read-once refutation of a system of difference constraints.

For systems of UTVPI constraints, we have the following results:

1. Developing several polynomial time algorithms for finding the shortest tree-like linear refutation of a system of UTVPI constraints.

2. Developing a pseudo-polynomial time algorithm for finding the shortest weighted read-once linear refutation of a system of UTVPI constraints.

3. Developing an FPTAS for the problem of finding the shortest weighted tree-like refutation of a system of UTVPI constraints.

4. The problem of identifying if a system of UTVPI constraints has a literal-once linear refutation is in **P**.

5. The problem of identifying if a system of UTVPI constraints has a read-once linear refutation is in **P**.

6. The problem of identifying if a system of UTVPI constraints has a non-literal read-once linear refutation is **NP-complete**.

For systems of Horn constraints, we have the following results:

1. The problem of identifying if a system of Horn constraints has a read-once refutation using the ADD rule is **NP-complete**.

2. The problem of identifying if a system of Horn clausal constraints has a read-once refutation using the ADD rule is **NP-complete**.

3. Developing an exact exponential time algorithm for finding a read-once refutation using the ADD rule for a system of Horn constraints.

16.3 Results for Integer Polyhedral CSPs

In this section, we list our results for integer programs. This includes results for UTVPI constraints and Horn constraints.

For systems of UTVPI constraints, we have the following results:

1. Developing a bit-scaling algorithm for finding an integer refutation of a system of UTVPI constraints.

2. Developing a graphical theorem of the alternative for integer feasibility of UTVPI constraints.

3. Developing a polynomial time algorithm for finding the integer closure of a system of UTVPI constraints.

For systems of Horn constraints, we have the following results:

1. The problem of identifying if a system of Horn constraints has a read-once refutation using the ADD and DIV rules is **NP-complete**.

2. The problem of identifying if a system of Horn clausal constraints has a read-once refutation using the ADD and DIV rules is **NP-complete**.

16.4 Results of Quantified Linear Systems

In this section, we list our results for quantified linear programs and quantified linear implications.

For quantified linear programs we have the following results:

1. The feasibility problem for unbounded quantified linear programs is in **P**.

2. The feasibility problem for partially bounded quantified linear programs is in **P**.

For quantified linear implications we have the following results:

1. The feasibility problem for quantified linear implications is **PSPACE-hard**.

2. The feasibility problem for unbounded quantified linear implications is in **P**.

3. The feasibility problem for partially bounded quantified linear implications is in **P**.

4. There is a variant of quantified linear implication that is complete for every level of the polynomial hierarchy.

Chapter 17

Future Research Directions

In this chapter we discuss possible avenues of future research.

17.1 Research in Boolean CSPs

From the perspective of future research into boolean CSPs, the following avenues appear promising:

1. Minimal unsatisfiability problem for Horn formulas - The Minimal unsatisfiability problem is defined as follows: Given a CNF formula Φ, is Φ unsatisfiable and is every sub-formula of Φ satisfiable. It is well-known that the Minimal unsatisfiability problem is $\mathbf{D^P - complete}$ for 3CNF formulas. This problem is easily seen to be in **P** for Horn formulas, since there exists a satisfiability oracle for Horn formulas that runs in polynomial (in fact, linear) time [DG84]. However, the obvious algorithm of iteratively removing clauses from an unsatisfiable subset of clauses is arguably not very efficient. We plan to investigate hyper-graph based approaches for this problem.

2. Cutting plane proof system - Cutting planes were introduced in [Gom58], as a technique to solve integer programs by repeatedly solving linear programming relaxations. It was shown in [Chv73] that the technique in [Gom58] is complete (with

some modifications). In [CCT87], cutting planes are formally studied under the auspices of a proof system. Exponential lower bounds on proof lengths for various restrictions on cutting planes have been obtained in [Pud97, BPR97, IPU94]. We are currently exploring upper and lower bounds on cutting plane proofs for Horn clausal formulas.

17.2 Research in Polyhedral CSPs

From the perspective of future research into polyhedral CSPs, the following avenues appear promising:

1. The OLRR problem for linear feasibility of systems of UTVPI constraints - The algorithms for UCSs in this dissertation include algorithm for the ROR and OLTR problems. However, none of the algorithms provide a read-once refutation, using the fewest number of constraints, i.e., the shortest read-once refutation. Establishing bounds on the lengths of the shortest proofs of infeasibility is an integral part of research in proof complexity [BP98]. We are therefore interested in this problem. Likewise, we are also interested in the problems of finding the shortest literal-once refutation and finding the shortest dag-like refutation in a system of UTVPI constraints.

2. The ROR problem for integer feasibility of systems of UTVPI constraints - When looking at restricted refutations of UTVPI constraints, this dissertation focused on refutations of linear feasibility. However, these same problems exist for integer feasibility. Of the polyhedral CSPs looked at in this paper, only systems of UTVPI constraints distinguish between linear and integer feasibility. Thus, a possible avenue for future research is applying the same restrictions on refutations of linear feasibility to refutations of integer feasibility.

3. The ROR problem for general linear systems - As discussed before, an infeasible

difference constraint system must have an ROR and an LOR. This paper investigated read-once and literal-once refutations in UTVPI constraint systems. The natural question is: What is the computational complexity of checking whether an arbitrary linear program has a read-once refutation?

4. Minimal refutations - A refutation is said to be *minimal*, if no proper subset contains a refutation. The algorithms discussed in this paper do not produce minimal refutations. However, it is not difficult to obtain minimal refutations (literal-once or read-once), using the algorithms presented here as oracles. Doing so, would add an $O(m)$ factor to the running time. It would be interesting to see if a more efficient approach can be designed.

5. Implementation - We are currently implementing the algorithm for linear feasibility in UTVPI constraints detailed in [SW17b]. However, the refutations returned by this algorithm are not restricted in any of the ways described in this dissertation. Thus, it would be interesting to implement the algorithms described in this dissertation and check their performances against existing similar algorithms on data connected to actual program verification problems.

Chapter 18

List of Publications

18.1 Papers in Refereed Journals

J1: K. Subramani and Piotr Wojciechowski. Read-once certificates of linear infeasibility in UTVPI constraints *Algorithmica*. (**Accepted, In Press**).

J2: Hans Kleine Buning, Piotr Wojciechowski and K. Subramani. Finding read-once resolution refutations in systems of 2CNF clauses. *Theoretical Computer Science (TCS)*, **729**, pp. 42-56, Elsevier Science Publishers, 2018.

J3: K. Subramani and Piotr Wojciechoski. A certifying algorithm for lattice point feasibility in a system of UTVPI constraints. *Journal of Combinatorial Optimization (JCO)*, **35**(2), pp. 389-408, Springer Science Publishers, 2018.

J4: James B. Orlin, K. Subramani and Piotr Wojciechowski. Randomized algorithms for finding the shortest negative cost cycle in networks. *Discrete Applied Mathematics (DAM)*, **236**, pp. 387-394, Elsevier Science Publishers, 2018.

J5: K. Subramani and Piotr Wojciechowski. A combinatorial certifying algorithm for linear feasibility in UTVPI constraints. *Algorithmica*, **78**(1), pp. 166-208, Springer Science Publishers, 2017.

J6: Piotr Wojciechowki, Pavlos Eirinakis and K. Subramani. Analyzing restricted fragments of the theory of linear arithmetic. *Annals of Mathematics and Artificial Intelligence (AMAI)*, **79** (1-3), pp. 245-266, Springer Science Publishers, 2017.

J7: Pavlos Eirinakis, Salvatore Ruggieri, K. Subramani and Piotr Wojciechowski. On Quantified Linear Implication. *Annals of Mathematics and Artificial Intelligence (AMAI)*, **71** (4), pp. 301-325, Springer Science Publishers, 2014.

J8: Salvatore Ruggieri, Pavlos Eirinakis, K. Subramanic and Piotr Wojciechowski. On the Complexity of Quantified Linear Systems. *Theoretical Computer Science (TCS)*, **518** , pp. 128-134, Elsevier Science Publishers, 2014.

J9: Pavlos Eirinakis, Salvatore Ruggieri, K. Subramani and Piotr Wojciechowski. A Complexity Perspective on Entailment of Parameterized Linear Constraints. *Constraints*, **17** (4), pp. 461-487, Springer Science Publishers, 2012.

18.2 Papers in Refereed Conference Proceedings

C1: K. Subramani and Piotr Wojciechowski. Read-Once Certification of Linear Infeasibility in UTVPI Constraints. *Proceedings of the* 15^{th} *Annual Conference on the Theory and Applications of Models of Computation (TAMC)*, pp. 578-593, (Eds.) T. V. Gopal, et. al., *Springer-Verlag*, *Lecture Notes in Computer Science*, vol. 11436, Kitakyushu (Japan), April 2019.

C2: K. Subramani, Piotr Wojciechowski, Zachary Santer and Matthew Anderson. An Empirical Analysis of Feasibility Checking Algorithms for UTVPI Constraints. *Proceedings of the* 12^{th} *International Conference on Algorithmic Aspects of Information Management (AAIM)*, pp. 111–123, (Eds.) Shaojie Tang, Ding-Zhu Du, et. al., *Springer-Verlag, Lecture Notes in Computer Science*, vol. 11343, Dallas (Texas), December 2018.

C3: Bugra Caskurlu, Matthew Williamson, K. Subramani, Vahan Mkrtchyan and Piotr Wojciechowski.

A Fully Polynomial Time Approximation Scheme for Refutations in Weighted Difference Constraint Systems. *Proceedings of the* 4^{th} *Annual Conference on Algorithms and Discrete Applied Mathematics (CALDAM)*,

pp. 45-58, (Eds.) Partha P. Goswami and Bhawani S. Panda, *Springer-Verlag*, *Lecture Notes in Computer Science*, vol. 10743, Guwahati (India), February 2018.

C4: K. Subramani and Piotr J. Wojciechowski, Analyzing lattice point feasibility in UTVPI constraints.

Proceedings of the 23^{rd} *International Conference on the Principles and Practice of Constraint*

Programming (CP), pp. 615-629, (Ed.) Christopher Beck, *Springer-Verlag*, *Lecture Notes in Computer Science*, vol. 10416, Melbourne (Australia), August 2017.

C5: Piotr J. Wojciechowski, R. Chandrasekaran and K. Subramani. On a Generalization

of Horn Constraint Systems. *Proceedings of the* 12^{th} *International Computer Science Symposium in Russia (CSR) on Computer Science - Theory and Applications*, pp. 323-336, (Ed.) Pascal Weil, *Springer-Verlag*, *Lecture Notes in Computer Science*, vol. 10304, Kazan (Russia), June 2017.

C6: Hans Kleine Büning, Piotr J. Wojciechowski and K. Subramani. On the Computational Complexity of Read once Resolution Decidability in 2CNF Formulas. *Proceedings of the* 14^{th} *Annual Conference on the Theory and Applications of Models of Computation (TAMC)*, pp. 362-372, (Eds.) T. V. Gopal, et. al., *Springer-Verlag*, *Lecture Notes in Computer Science*, vol. 10185, Berne (Switzerland), April 2017.

C7: Hans Kleine Büning, Piotr J. Wojciechowski and K. Subramani. The Complexity of Finding Read-Once NAE-Resolution Refutations. *Proceedings of the* 7^{th} *Indian Conference on Logic and Its Applications (ICLA)*,
pp. 64-76, (Eds.) Sujata Ghosh and Sanjiva Prasad, *Springer-Verlag*, *Lecture Notes in Computer Science*,
vol. 10119, Kanpur (India), January 2017.

C8: Alvaro Velasquez, Piotr Wojciechowski, K. Subramani, Steven L. Drager and Sumit Kumar Jha. The cardinality-constrained paths problem: Multicast data routing in heterogeneous communication networks. *Proceedings of the* 15^{th} *IEEE International Symposium on Network Computing and Applications (NCA)*, pp. 126-130, (Eds.) Alessandro Pellegrini, et. al., IEEE Computer Society, Boston, October 2016.

C9: Vahid Hashemi, Holger Hermanns, Lei Song, K. Subramani, Andrea Turrini and Piotr J. Wojciechowski. Compositional Bisimulation Minimization for Interval Markov Decision Processes. *Proceedings of the* 10^{th} *International Conference on Language and Automata Theory and Applications (LATA)*, pp. 114-126, (Eds.) Adrian-Horia Dediu, et. al., *Springer-Verlag*, *Lecture Notes in Computer Science*, vol. 9618, Prague (Czech Republic), March 2016.

C10: K. Subramani and Piotr Wojciechowski. A Graphical Theorem of the Alternative for UTVPI Constraints. *Proceedings of the* 12^{th} *International Colloquium on Theoretical Aspects of Computer Science (ICTAC)*, pp. 328-345, (Eds.) Martin Leucker, Camilo Rueda and Frank Valencia, *Springer-Verlag*, *Lecture Notes in Computer Science*, vol. 9399 Cali (Colombia), October 2015.

C11: Piotr Wojciechowski, Pavlos Eirinakis and K. Subramani. Variants of Quantified Linear Programming and Quantified Linear Implication. *Proceedings of the* 12^{th} *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, pp. $-$, (Eds.) Lisa Hellerstein and Gyorgy Turan, Fort Lauderdale (Florida), January 2014.

C12: Pavlos Eirinakis, Salvatore Ruggieri, K. Subramani and Piotr Wojciechowski. Computational complexity of inclusion queries over polyhedral sets. *Proceedings of the* 11^{th} *International Symposium on Artificial Intelligence and Mathematics (ISAIM)*, pp. $-$, (Eds.) Robert H. Sloan, Fort Lauderdale (Florida), January 2012.

18.3 Papers in Refereed Workshops

W1: Hans Kleine Büning, Piotr Wojciechowski and K. Subramani. Read-Once Resolutions in Horn Formulas *Proceedings of the* 13^{th} *International Frontiers of Algorithmics Workshop (FAW)*, pp. 100-110, (Eds.) Yijia Chen, Xiaotie Deng, and Mei Lu, *Spring-Verlag*, *Lecture Notes in Computer Science*, vol. 11458, Sanya (China), May 2019.

W2: Piotr Wojciechowski, K. Subramani and Matthew Williamson. Optimal length tree-like refutations of linear feasibility in UTVPI constraints *Proceedings of the* 12^{th} *International Frontiers of Algorithmics Workshop (FAW)*, pp. 300-314, (Eds.) Jianer Chen and Pinyan Lu, *Spring-Verlag*, *Lecture Notes in Computer Science*, vol. 10823, Guangzhou (China), May 2018.

W3: K. Subramani and Piotr Wojciechowski. A Bit-Scaling Algorithm for Integer Feasibility in UTVPI Constraints. *Proceedings of the* 27^{th} *International Workshop on Combinatorial Algorithms (IWOCA)*, pp. 321 - 333, (Eds.) Veli Mäkinen, Simon J. Puglisi and Leena Salmela, *Spring-Verlag*, *Lecture Notes in Computer Science*, vol. 9843, Helsinki (Finland), May 2016.

W4: K.Subramani and Piotr Wojciechowski. An optimal algorithm for computing the integer closure of UTVPI constraints. *Proceedings of the* 10^{th} *International Workshop on Algorithms and Computation (WALCOM)*, pp. 154 - 165, (Eds.) Mohammad Kaykobad and Rossella Petreschi, *Spring-Verlag*, *Lecture Notes in Computer Science*, vol. 9627, Kathmandu (Nepal), March 2016.

W5: Pavlos Eirinakis, Salvatore Ruggieri, K. Subramani and Piotr Wojciechowski. On the conjunctive fragments of theory of linear arithmetic. *Proceedings of the Third Workshop on Automated Deduction: Decidability, Complexity, Tractability (ADDCT)*, pp. 51-53, Lake Placid, New York, June 2013.

18.4 Refereed Abstracts

RA1: Piotr Wojciechowski, Pavlos Eirinakis and K. Subramani. Quantifying Linear Programming and Implications. 22^{nd} International Symposium on Mathematical Programming (ISMP), Pittsburgh, Pennsylvania, May 2015.

RA2: Pavlos Eirinakis, Salvatore Ruggieri, K. Subramani and Piotr Wojciechowski. Quantified Linear Programming and Quantified Linear Implications. Proceedings of the 8^{th} Scandinavian Logic Symposium (SLS), page 26, (Eds.) Patrick Blackburn, Klaus Frovin Jorgensen, Neil Jones, Erik Palmgren, Roskilde University, Denmark, August 2012.

Appendix A

Important Related Problems

A.1 The Disjoint Paths Problem

In this section, we briefly discuss the vertex-disjoint path problem for directed graphs.

***Definition* A.1.1.** *Given a directed graph G and pairwise distinct vertexes s_1, t_1, s_2, and t_2, the* **vertex-disjoint path problem** *(2-DPP) consists of finding a pair of vertex-disjoint paths in G, one from s_1 to t_1 and the other from s_2 to t_2.*

The problem is known to be **NP-complete** [FHW80].

Example 47: Consider the directed graph in Figure A.1.

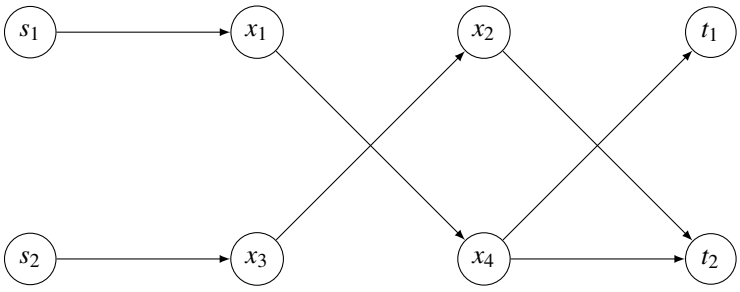

Figure A.1: Directed graph with vertex-disjoint paths.

This graph has the following vertex-disjoint paths.

$$p_1: \quad s_1 \to x_1 \to x_4 \to t_1$$
$$p_2: \quad s_2 \to x_3 \to x_2 \to t_2$$

Now we modify the problem as follows.

***Definition* A.1.2.** *Given a directed graph G and two distinct vertexes s and t, the* **vertex-disjoint cycle problem** *(C-DPP) consists of finding a pair of vertex-disjoint paths in G, one from s to t and the other from t to s.*

Note that the paths are vertex-disjoint, if the inner vertexes of the path from s to t are disjoint from the inner vertexes of the path from t to s.

***Lemma* A.1.1.** *C-DPP is* **NP-complete**.

Proof. Obviously, the problem is in **NP**. We will show **NP-hardness** by a reduction from 2-DPP.

From $G = (V,E)$, s_1, t_1, s_2, and t_2 we construct the new graph

$$G' = (V \cup \{s,t\}, E \cup \{(s,s_1),(t_2,s),(t_1,t),(t,s_2)\}).$$

Assume that G has two vertex-disjoint paths, w_1 from s_1 to t_1, and w_2 from s_2 to t_2. Thus, the paths $(s,s_1),w_1,(t_1,t)$ and $(t,s_2),w_2,(t_2,s)$ in G' are vertex-disjoint. Note that s_1,s_2,t_1,t_2 are pairwise distinct. Thus, G' has the desired vertex-disjoint cycle.

Now assume that G' has two vertex-disjoint paths, w_1 from s to t, and w_2 from t to s. By construction, w_1 must contain a path from s_1 to t_1. Similarly, w_2 must contain a path from s_2 to t_2. Since w_1 and w_2 are vertex-disjoint these new paths must also be vertex-disjoint. Thus, G has the desired vertex-disjoint paths. □ □

We also mention the edge-disjoint cycle problem for directed graphs.

***Definition* A.1.3.** *Given a directed graph G and two distinct vertexes s and t, the* **edge-disjoint cycle problem** *(C-DEP) consists of finding a pair of edge-disjoint paths in G, one from s to t and the other from t to s.*

The problem is **NP-complete**. For two pairs of vertexes, the edge-disjoint path problem is **NP-complete** [EIS76]. We can reduce the edge-disjoint path problem to C-DEP the same way we reduced 2-DPP to C-DPP.

A.2 The Minimum Weight Perfect Matching Problem

In this section, we briefly discuss the problem of finding the minimum weight perfect matching (MWPM) in an undirected, weighted graph. This digression is necessitated by the fact that both Section 10.2.3 and Section 10.2.4 involve reduction to the MWPM problem.

Let $\mathbf{G} = \langle \mathbf{V}, \mathbf{E}, \mathbf{c} \rangle$ denote an undirected graph, with vertex set $\mathbf{V}$, edge set $\mathbf{E}$ and edge cost function $\mathbf{c}$. Let $n = |\mathbf{V}|$ and let $m = |\mathbf{E}|$. A *matching* is any collection of vertex-disjoint edges. A *perfect matching* is a matching in which each vertex $v \in \mathbf{V}$ is matched. Without loss of generality, we assume that n is even, since $\mathbf{G}$ cannot have a perfect matching, otherwise.

Example 48: Consider the undirected graph in Figure A.2.

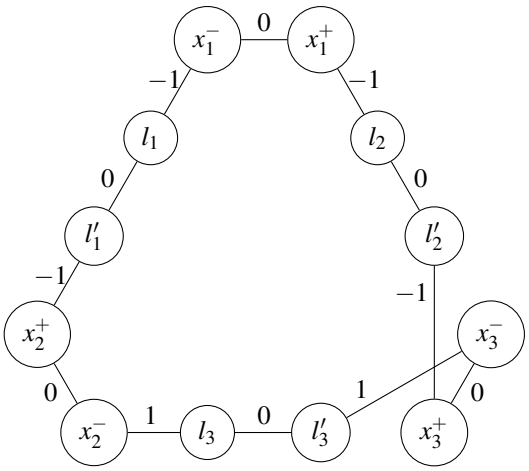

Figure A.2: Undirected graph

The graph in Figure A.2 has a matching of weight 0. This can be seen in Figure A.3.

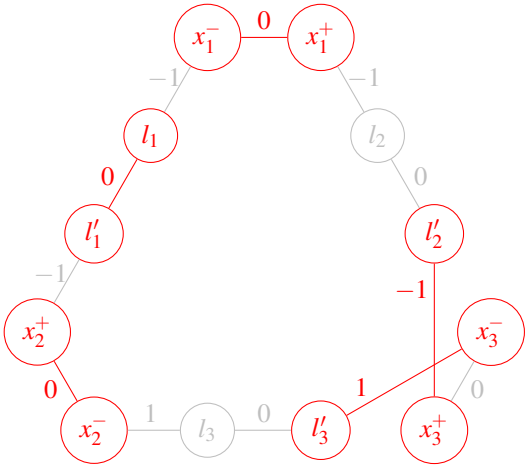

Figure A.3: A matching in the graph in Figure A.2

The graph in Figure A.2 has a perfect matching of weight 0. This can be seen in Figure A.4.

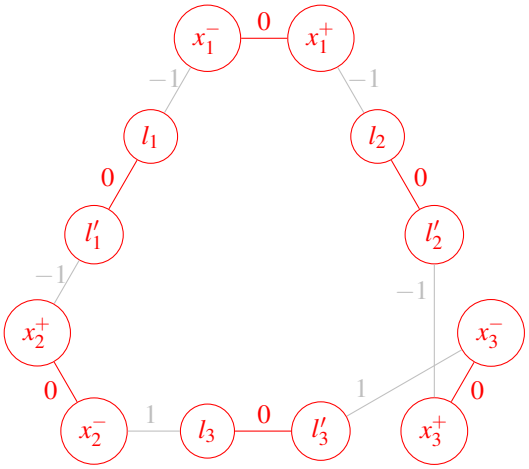

Figure A.4: A prefect matching in the graph in Figure A.2

The graph in Figure A.2 has a minimum weight perfect matching of weight -2. This can be seen in Figure A.5.

The MWPM problem is one of the classical problems in combinatorial optimization [KV10]. Over the years, there has been a steady stream of papers documenting improvements in algorithms for this problem [Edm67, Gab76, DPS18].

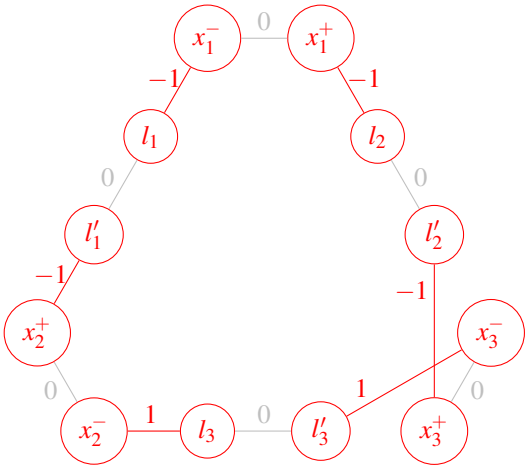

Figure A.5: A minimum weight perfect matching in the graph in Figure A.2

The fastest combinatorial algorithm for the MWPM problem runs in time $O(m \cdot n + n^2 \cdot \log n)$ [Gab90]. It is this bound that we shall be using in our paper.

Bibliography

[AAHO98] Charu C. Aggarwal, Ravindra K. Ahuja, Jianxiu Hao, and James B. Orlin. Diagnosing infeasibilities in network flow problems. *Math. Program.*, 81:263–280, 1998.

[AB93] A Aggoun and N. Beldiceanu. Extending chip to solve complex scheduling and placement problems. *Journal of Mathematical and Computer Modelling*, 17(7):57–73, 1993.

[ABK+97] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data structures for verification of timed automata. In *Proceedings of the Hybrid and Real-Time Systems*, 1997.

[ABMP98] M. Alekhnovich, S. Buss, S. Moran, and T. Pitassi. Minimum propositional proof length is NP-hard to linearly approximate. In *Mathematical Foundations of Computer Science (MFCS)*, pages 176–184. Springer-Verlag, 1998. Lecture Notes in Computer Science.

[ABZ88] J. Adams, E. Balas, and D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34:391–401, 1988.

[ACM04] A. Armando, C. Castellini, and J. Mantovani. Software model checking using linear constraints. In *Lecture Notes in Computer Sciente*, volume 3308, pages 209–223. Springer, 2004.

[Adl13] Ilan Adler. The equivalence of linear programs and zero-sum games. *Int. J. Game Theory*, 42(1):165–177, 2013.

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms and Applications*. Prentice-Hall, 1993.

[AR01] Michael Alekhnovich and Alexander A. Razborov. Lower bounds for polynomial calculus: Non-binomial case. In *42nd Annual Symposium on Foundations of Computer Science, FOCS 2001, 14-17 October 2001, Las Vegas, Nevada, USA*, pages 190–199, 2001.

[BA79] R. E. Tarjan B. Aspvall, M. F. Plass. A linear time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[BB91] A. Benveniste and G. Berry. The synchronous approach to reactive and real-time systems. In *Proceedings of the 1991 IEEE*, volume 79, pages 1270 – 1282, 1991.

[BE00] Alexander Bockmayr and Friedrich Eisenbrand. Combining logic and optimization in cutting plane theory. In *FroCos*, pages 1–17, 2000.

[Ber80] L. Berman. The complexitiy of logical theories. *Theoretical Computer Science*, 11:71–77, 1980.

[BGM90] R.S. Boyer, M.W. Green, and J.S. Moore. *The Use of a Formal Simulator to Verify a Simple Real Time Control Program*. Springer, New York, NY, 1990.

[BHZ09] R. Bagnara, P. M. Hill, and E. Zaffanella. Weakly-relational shapes for numeric abstractions: Improved algorithms and proofs of correctness. *Formal Methods in System Design*, 35(3):279–323, 2009.

[BIK+94] Paul Beame, Russell Impagliazzo, Jan Krajícek, Toniann Pitassi, and Pavel Pudlák. Lower bound on hilbert's nullstellensatz and propositional proofs. In *35th Annual Symposium on Foundations of Computer Science, Santa Fe, New Mexico, USA, 20-22 November 1994*, pages 794–806, 1994.

[BL08] B. Berstel and M. Leconte. Using constraints to verify properties of rule programs. In *Proceedings of the 2010 Int. Conf. on Software Testing, Verification, and Validation Workshops*, pages 349–354, 2008.

[BM07] A. R. Bradley and Z. Manna. *The calculus of computation - decision procedures with applications to verification*. Springer, 1^{st} edition, 2007.

[BP96] Paul Beame and Toniann Pitassi. 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.

[BP97] Buss and Pitassi. Resolution and the weak pigeonhole principle. In *CSL: 11th Workshop on Computer Science Logic*. LNCS, Springer-Verlag, 1997.

[BP98] Paul Beame and Toniann Pitassi. Propositional proof complexity: Past, present, future. *Bulletin of the EATCS*, 65:66–89, June 1998.

[BPN95] P. Baptiste, C. Le Pape, and W. Nuijten. Constraint-based optimization and approximation for job-shop scheduling. In *AAAI-SIGMAN Workshop on Intelligent Manufacturing Systems*, 1995.

[BPR97] Maria Luisa Bonet, Toniann Pitassi, and Ran Raz. Lower bounds for cutting planes proofs with small coefficients. *J. Symb. Log.*, 62(3):708–728, 1997.

[BPR06] Saugata Basu, Richard Pollack, and Marie-Francoise Roy. *Existential Theory of the Reals*, volume 10 of *Algorithms and Computation in Mathematics*. Springer Berlin Heidelberg, 2006.

[Bro03] C. W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *ACM SIGSAM Bullettin*, 37(4):97–108, 2003.

[BS94] J. A. Brzowski and C-J.H. Seger. *Asynchronous Circuits*. Springer, 1994.

[Bus] Samuel R. Buss. Propositional proof complexity: An introduction. `http://www.math.ucsd.edu/~sbuss/ResearchWeb/marktoberdorf97/paper.pdf`.

[Bus98] S. R. Buss, editor. *Handbook of Proof Theory*. Elsevier Science Pub., 1998.

[CA00] S. Choi and A. Agrawala. Dynamic dispatching of cyclic real-time tasks with relative timing constraints. *Real-Time Systems*, 19(1):5–40, 2000.

[CAMN04] Scott Cotton, Eugene Asarin, Oded Maler, and Peter Niebert. Some progress in satisfiability checking for difference logic. In *FORMATS/FTRTFT*, pages 263–276, 2004.

[Can88] J. Canny. Some algebraic and geometric computations in pspace. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 460–467, 1988.

[CAS08] M. Ceberio, C. Acosta, and C. Servin. 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*, pages 225–230, 2008.

[CC77] Patrick Cousot and Radhia Cousot. Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *POPL*, pages 238–252, 1977.

[CCT87] W. Cook, C. R. Coullard, and Gy. Turan. On the complexity of cutting-plane proofs. *Discrete Applied Mathematics*, 18:25–38, 1987.

[CH91] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, 1991.

[Cha81] R. Chandrasekaran. Polynomial algorithms for totally dual integral systems and extensions. *Annals of Discrete Mathematics*, 11:39–52, 1981.

[Cha15] R. Chandrasekaran. Integer farkas lemma. *International Game Theory Review*, 17(1), 2015.

[Chv73] V. Chvatal. Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, 4(10-11):886–904, 1973.

[Cla93] E.M. Clarke. Automatic verification of sequential circuit designs. In *Proceedings of the 1993 International Conference on Computer Hardware Description Languages and their Applications*, volume 32 of *IFIP Transactions A: Computer Science and Technology*, page 163166, 1993.

[Cla97] E.M. Clarke. Model checking. In *FSTTCS*, volume 1346 of *Lecture Notes in Computer Science*, pages 54–56. Springer, 1997.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[CR73] Stephen A. Cook and Robert A. Reckhow. Time bounded random access machines. *Journal of Computer and System Sciences*, 7(4):354–375, August 1973.

[CR74] Stephen A. Cook and Robert A. Reckhow. On the lengths of proofs in the propositional calculus (preliminary version). In *Proceedings of the 6th Annual ACM Symposium on Theory of Computing, April 30 - May 2, 1974, Seattle, Washington, USA*, pages 135–148, 1974.

[CR06] H. Collavizza and M. Reuher. 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*, 2006.

[CRV04] H. Collavizza, M. Reuher, and P. Van Hentenryck. 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, 2004.

[CRY96] I.J. Cox, S.B. Rao, and Y.Zhong. Ratio regions: A technique for image segmentation. In *Proceedings of the International Conference on Pattern Recognition*, pages 557–564. IEEE, August 1996.

[CS13] R. Chandrasekaran and K. Subramani. A combinatorial algorithm for horn programs. *Discrete Optimization*, 10:85–101, 2013.

[Dan51] G. B. Dantzig. A proof of the equivalence of the programming problem and the game problem. In T. C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 330–335. John Wiley & Sons, New York, 1951.

[DdM06] Bruno Duterre and Leonardo de Moura. The yices smt solver. Technical report, SRI International, 2006.

[DE73] G. B. Dantzig and B. C. Eaves. Fourier-Motzkin Elimination and its Dual. *Journal of Combinatorial Theory (A)*, 14:288–297, 1973.

[DFK+03] Marcel Dhiflaoui, Stefan Funke, Carsten Kwappik, Kurt Mehlhorn, Michael Seel, Elmar Schömer, Ralph Schulte, and Dennis Weber. Certifying and repairing solutions to large lps how good are lp-solvers? In *SODA*, pages 255–256, 2003.

[DG84] William F. Dowling and Jean H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. Log. Program.*, 1(3):267–284, 1984.

[DH88] J. H. Davenport and J. Heintz. Real quantifier elimination is doubly exponential. *Journal of Symbolic Computation*, 5(1-2):29–35, 1988.

[DI04] Camil Demtrescu and Giuseppe F. Italiano. A new approach to dynamic all pairs shortest paths. *Journal of the ACM*, 51(6):968–992, 2004.

[dMOR+04] Leonardo de Moura, Sam Owre, Harald Ruess, John M. Rushby, and Natarajan Shankar. The ICS decision procedures for embedded deduction. In *IJCAR*, pages 218–222, 2004.

[DPS18] Ran Duan, Seth Pettie, and Hsin-Hao Su. Scaling algorithms for weighted matching in general graphs. *ACM Trans. Algorithms*, 14(1):8:1–8:35, 2018.

[Dru06] D. Drusinsky. *Modeling and verification using UML statecharts: a working guide to reactive systems design, runtime modeling and execution-based model checking*. Newnes, Burlington, MA, 2006.

[DS97] A. Dolzmann and T. Sturm. REDLOG: Computer algebra meets computer logic. *ACM SIGSAM Bullettin*, 31(2):2–9, 1997.

[DSW98a] A. Dolzmann, T. Sturm, and V. Weispfenning. A new approach for automatic theorem proving in real geometry. *Journal of Automated Reasoning*, 21(3):357–380, 1998.

[DSW98b] A. Dolzmann, T. Sturm, and V. Weispfenning. Real quantifier elimination in practice. In B. H. Matzat, G.-M. Greuel, and G. Hiss, editors, *Algorithmic Algebra and Number Theory*, pages 221–248. Springer, 1998.

[Edm67] Jack Edmonds. An introduction to matching, 1967. Mimeographed notes. Engineering Summer Conference, University of Michigan, Ann Arbor, MI.

[EIS76] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of timetable and multicommodity flow problems. *SIAM J. Comput.*, 5(4):691–703, 1976.

[Far02] Gyula Farkas. Über die Theorie der Einfachen Ungleichungen. *Journal für die Reine und Angewandte Mathematik*, 124(124):1–27, 1902.

[FHW80] Steven Fortune, John E. Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theor. Comput. Sci.*, 10:111–121, 1980.

[FR75] J. Ferrante and C. Rackoff. A decision procedure for the first order theory of real addition with order. *SIAM Journal on Computing*, 4(1):69–76, 1975.

[FS02] Jonathan Ford and Natarajan Shankar. Formal verification of a combination decision procedure. In *CADE*, pages 347–362, 2002.

[Gab76] Harold N. Gabow. An efficient implementation of Edmonds' algorithm for maximum matching on graphs. *Journal of the ACM*, 23(2):221–234, April 1976.

[Gab90] Harold N. Gabow. Data structures for weighted matching and nearest common ancestors with linking. In David Johnson, editor, *Proceedings of the 1st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '90)*, pages 434–443, San Francisco, CA, USA, January 1990. SIAM.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman Company, San Francisco, 1979.

[Gol95] Andrew V. Goldberg. Scaling algorithms for the shortest paths problem. *SIAM Journal on Computing*, 24(3):494–504, June 1995.

[Gom58] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.

[GPS95a] R. Gerber, W. Pugh, and M. Saksena. Parametric dispatching of hard real-time tasks. *IEEE Transactions on Computers*, 44(3):471–479, 1995.

[GPS95b] R. Gerber, W. Pugh, and M. Saksena. Parametric dispatching of hard real-time tasks. *IEEE Transactions on Computing*, 44(3):471–479, 1995.

[GRKL01] A. Goel, K. G. Ramakrishnan, D. Kataria, and D. Logothetis. Efficient computation of delay-sensitive routes from one source to all destinations. In *Proceedings IEEE INFOCOM 2001. Conference on Computer Communications. Twentieth Annual Joint Conference of the IEEE Computer and Communications Society (Cat. No.01CH37213)*, volume 2, pages 854–858 vol.2, 2001.

[GSV08] S. Gulwani, S. Srivastava, and R. Venkatesan. Program analysis as constraint solving. In *Proceedings of the 2008 ACM SIGPLAN Conf. on Programming language design and implementation*, New York, NY, 2008. ACM.

[Hak85] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39(2-3):297–308, August 1985.

[Hal02] R.J. Hall. Specification, validation, and synthesis of email agent controllers: A case study in function rich reactive system design. *Automated Software Engineering*, 9(3):233–261, 2002.

[Har04] D. Harel. A grand challenge for computing: Towards full reactive modeling of a multi-cellular animal. *Verification, Model Checking, and Abstract Interpretation*, 2937:39–60, 2004.

[Har09] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 1^s edition, 2009.

[HL89] C. C. Han and K. J. Lin. Job scheduling with temporal distance constraints. Technical Report UIUCDCS-R-89-1560, University of Illinois at Urbana-Champaign, Department of Computer Science, 1989.

[HM11] Federico Heras and João Marques-Silva. Read-once resolution for unsatisfiability-based max-sat algorithms. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 572–577, 2011.

[HN94] Dorit S. Hochbaum and Joseph (Seffi) Naor. Simple and fast algorithms for linear and integer programs with two variables per inequality. *SIAM Journal on Computing*, 23(6):1179–1192, December 1994.

[Hoo89] John N. Hooker. Input proofs and rank one cutting planes. *INFORMS Journal on Computing*, 1(3):137–145, 1989.

[HP85] D. Harel and A. Pnueli. *On the development of reactive systems*. Springer, New York, NY, 1985.

[HS97] W. Harvey and P. J. Stuckey. A unit two variable per inequality integer constraint solver for constraint logic programming. In *Proceedings of the 20th Australasian Computer Science Conference*, pages 102–111, 1997.

[IM95] K. Iwama and E. Miyano. Intractability of read-once resolution. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory (SCTC '95)*, pages 29–36, Los Alamitos, CA, USA, June 1995. IEEE Computer Society Press.

[IPU94] Russell Impagliazzo, Toniann Pitassi, and Alasdair Urquhart. Upper and lower bounds for tree-like cutting planes proofs. In *Proceedings of the Ninth Annual Symposium on Logic in Computer Science (LICS '94), Paris, France, July 4-7, 1994*, pages 220–228, 1994.

[Iwa97] K. Iwama. Complexity of finding short resolution proofs. *Lecture Notes in Computer Science*, 1295:309–319, 1997.

[Jer85] Robert G. Jeroslow. The polynomial hierarchy and a simple model for competitive analysis. *Mathematical Programming*, 32:146–164, 1985. 10.1007/BF01586088.

[JMSY94] J. Jaffar, M. J. Maher, P. J. Stuckey, and H. C. Yap. Beyond Finite Domains. In *Proceedings of the Second International Workshop on Principles and Practice of Constraint Programming*, 1994.

[JP78] D. S. Johnson and F. P. Preparata. The densest hemisphere problem. *Theoretical Computer Science*, 6(1):93–107, February 1978.

[Kan83] R. Kannan. Polynomial time aggregation of integer programming. *Journal of the ACM*, 30(1):133–145, 1983.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103, New York, 1972. Plenum Press.

[KCH01] N. Kam, I.R. Cohen, and D. Harel. The immune system as a reactive system: modeling t cell activation with statecharts. In *Proceedings of the 2001 IEEE Symposia on Human-Centric Computing Languages and Environments*, pages 15 – 22, 2001.

[Kha79] L. G. Khachiyan. A polynomial algorithm in linear programming. *Doklady Akademiia Nauk SSSR*, 224:1093–1096, 1979. English Translation: *Soviet Mathematics Doklady*, Volume 20, pp. 1093–1096.

[KMMS03] Dieter Kratsch, Ross M. McConnell, Kurt Mehlhorn, and Jeremy Spinrad. Certifying algorithms for recognizing interval graphs and permutation graphs. In *SODA*, pages 158–167, 2003.

[KN09] Haim Kaplan and Yahav Nussbaum. Certifying algorithms for recognizing proper circular-arc graphs and unit circular-arc graphs. *Discrete Applied Mathematics*, 157(15):3216–3230, 2009.

[Kra94] Jan Krajícek. Lower bounds to the size of constant-depth propositional proofs. *J. Symb. Log.*, 59(1):73–86, 1994.

[KSSVS99] T.J. Koo, B. Sinopoli, A. Sangiovanni-Vincentelli, and S. Sastry. A formal approach to reactive system design: unmanned aerial vehicle flight management system design example. In *Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design*, pages 522 – 527, 1999.

[KV10] B. Korte and J. Vygen. *Combinatorial Optimization*. Number 21 in Algorithms and Combinatorics. Springer-Verlag, New York, 4^{th} edition, 2010.

[KWS18] Hans Kleine Büning, Piotr J. Wojciechowski, and K. Subramani. Finding read-once resolution refutations in systems of 2cnf clauses. *Theor. Comput. Sci.*, 729:42–56, 2018.

[KZ02] Hans Kleine Büning and Xishun Zhao. The complexity of read-once resolution. *Ann. Math. Artif. Intell.*, 36(4):419–435, 2002.

[KZ03] Hans Kleine Büning and Xishun Zhao. Read-once unit resolution. In *Theory and Applications of Satisfiability Testing, 6th International Conference, SAT 2003. Santa Margherita Ligure, Italy, May 5-8, 2003 Selected Revised Papers*, pages 356–369, 2003.

[Las04] Jean B. Lasserre. A discrete farkas lemma. *Discrete Optimization*, 1(1):67–75, 2004.

[Lib08] Paolo Liberatore. Redundancy in logic ii: 2cnf and horn propositional formulae. *Artificial Intelligence*, 172(2):265 – 299, 2008.

[LM05] S. K. Lahiri and M. Musuvathi. An Efficient Decision Procedure for UTVPI Constraints. In *Proceedings of the 5^{th} International Workshop on the Frontiers of Combining Systems, September 19-21, Vienna, Austria*, pages 168–183, New York, 2005. Springer.

[LR57] R. D. Luce and H. Raiffa. *Games and Decisions: Introduction and critical survey*. John Wiley & Sons, New York, 1^{st} edition, 1957.

[LS04] Inês Lynce and João P. Marques Silva. On computing minimum unsatisfiable cores. In *SAT*, 2004.

[LTCA89] S. T. Levi, S. K. Tripathi, S. D. Carson, and A. K. Agrawala. The `Maruti` Hard Real-Time Operating System. *ACM Special Interest Group on Operating Systems*, 23(3):90–106, July 1989.

[Man69] O. Mangasarian. *Nonlinear Programming*, McGraw-Hill, New York, 1969.

[Min06] Antoine Miné. The octagon abstract domain. *Higher-Order and Symbolic Computation*, 19(1):31–100, 2006.

[MM11] Cristopher Moore and Stephen Mertens. *The Nature of Computation*. Oxford University Press, 1st edition, 2011.

[MMNS11] R. M. McConnell, K. Mehlhorn, S. Näher, and P. Schweitzer. Certifying algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[MN99] K. Mehlhorn and S. Näher. *The LEDA Platform of Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, 1999.

[MR95] Rajeev Motwani and Prabhakar Raghavan. *Randomized Algorithms*. Cambridge University Press, Cambridge, England, June 1995.

[Mut77] D. Muthiayen. *Real-time reactive system development : a formal approach based on UML and PVS*. Ph.D. Thesis, Concordia University, 1977.

[NO05] Robert Nieuwenhuis and Albert Oliveras. Dpll(t) with exhaustive theory propagation and its application to difference logic. In *CAV*, pages 321–334, 2005.

[NOT04] Robert Nieuwenhuis, Albert Oliveras, and Cesare Tinelli. Abstract dpll and abstract dpll modulo theories. In *LPAR*, pages 36–50, 2004.

[NW99] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons, New York, 1999.

[OSW18] James B. Orlin, K. Subramani, and Piotr Wojciechowski. Randomized algorithms for finding the shortest negative cost cycle in networks. *Discrete Applied Mathematics*, 236(1):387–394, 2018.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison-Wesley, New York, 1994.

[Pin95a] M. Pinedo. *Scheduling: theory, algorithms, and systems*. Prentice-Hall, Englewood Cliffs, 1995.

[Pin95b] M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, Englewood Cliffs, NJ, 1995.

[PSSW14] Stefan Porschen, Tatjana Schmidt, Ewald Speckenmeyer, and Andreas Wotzlaw. XSAT and NAE-SAT of linear CNF classes. *Discrete Applied Mathematics*, 167:1–14, 2014.

[Pud97] Pavel Pudlák. Lower bounds for resolution and cutting plane proofs and monotone computations. *J. Symb. Log.*, 62(3):981–998, 1997.

[Pug92] W. Pugh. The omega test: A fast and practical integer programming algorithm for dependence analysis. *Comm. of the ACM*, 35(8):102–114, August 1992.

[PW00] B. Pfitzmann and M. Waidner. Composition and integrity preservation of secure reactive systems. In *Proceedings of the 2000 ACM Conference on Computer Communications Security*, pages 245–254, 2000.

[PW01] B. Pfitzmann and M. Waidner. A model for asynchronous reactive systems and its application to secure message transmission. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pages 184 – 200, 2001.

[Rat06] Stefan Ratschan. Efficient solving of quantified inequality constraints over the real numbers. *ACM Transactions on Computational Logic*, 7(4):723–748, 2006.

[Rat11] Stefan Ratschan. RSOLVER, 2011. http://rsolver.sourceforge.net.

[Rec75] Robert A. Reckhow. *On the lengths of proofs in the propositional calculus*. PhD thesis, University of Toronto, Ontario, Canada, 1975.

[Rob65] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, 1965.

[RS06] Vojtech Rödl and Mark H. Siggers. Color critical hypergraphs with many edges. *Journal of Graph Theory*, 53(1):56–74, 2006.

[RV01] A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning*. Elsevier, 2001.

[Sab] Ashish Sabharwal. Lecture notes in proof complexity.

[SB04] Sanjit A. Seshia and Randal E. Bryant. Deciding quantifier-free presburger formulas using parameterized solution bounds. In *LICS*, pages 100–109, 2004.

[Sch78] T.J. Schaefer. The complexity of satisfiability problems. In Alfred Aho, editor, *Proceedings of the 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, New York City, NY, 1978. ACM Press.

[Sch87] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley and Sons, New York, 1987.

[Sch04] K. Schneider. *Verification of reactive systems*. Formal Methods and Algorithms, Texts in Theoretical Computer Science. Springer, New York, NY, 2004.

[Seg07] Nathan Segerlind. The complexity of propositional proofs. *Bulletin of Symbolic Logic*, 13(4):417–481, 2007.

[SG11] K. Subramani and Xiaofeng Gu. Absorbing random walks and the NAE2SAT problem. *Int. J. Comput. Math.*, 88(3):452–467, 2011.

[Sho91] P. W. Shor. Stretchability of pseudolines is np-hard. *DIMACS*, 4:531–534, 1991.

[SLB03] Sanjit A. Seshia, Shuvendu K. Lahiri, and Randal E. Bryant. A hybrid sat-based decision procedure for separation logic with uninterpreted functions. In *DAC*, pages 425–430, 2003.

[Son85] Eduardo D. Sontag. Real addition and the polynomial hierarchy. *Inf. Process. Lett.*, 20(3):115–120, 1985.

[SS10] Andreas Schutt and Peter J. Stuckey. Incremental satisfiability and implication for utvpi constraints. *INFORMS Journal on Computing*, 22(4):514–527, 2010.

[SSRB98] J. A. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline Scheduling for Real-Time Systems*. Kluwer Academic, Dordrecht, 1998.

[Sto77] Larry J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1977.

[Sub04a] K. Subramani. On deciding the non-emptiness of 2SAT polytopes with respect to first order queries. *Mathematical Logic Quarterly*, 50(3):281–292, 2004.

[Sub04b] K. Subramani. Optimal length tree-like resolution refutations for 2sat formulas. *ACM Transactions on Computational Logic*, 5(2):316–320, 2004.

[Sub05a] K. Subramani. An analysis of totally clairvoyant scheduling. *Journal of Scheduling*, 8(2):113–133, 2005.

[Sub05b] K. Subramani. A comprehensive framework for specifying clairvoyance, constraints and periodicty in real-time scheduling. *The Computer Journal*, 48(3):259–272, 2005.

[Sub07] K. Subramani. On a decision procedure for quantified linear programs. *Annals of Mathematics and Artificial Intelligence*, 51(1):55–77, 2007.

[Sub09] K. Subramani. Optimal length resolution refutations of difference constraint systems. *Journal of Automated Reasoning (JAR)*, 43(2):121–137, 2009.

[SW15a] K. Subramani and Piotr J. Wojciechowski. A graphical theorem of the alternative for UTVPI constraints. In *Theoretical Aspects of Computing - ICTAC 2015 - 12th International Colloquium Cali, Colombia, October 29-31, 2015, Proceedings*, pages 328–345, 2015.

[SW15b] K. Subramani and James Worthington. Feasibility checking in horn constraint systems through a reduction based approach. *Theor. Comput. Sci.*, 576:1–17, 2015.

[SW17a] K. Subramani and Piotr J. Wojciechowski. Analyzing lattice point feasibility in UTVPI constraints. In *Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings*, pages 615–629, 2017.

[SW17b] K. Subramani and Piotr J. Wojciechowski. A combinatorial certifying algorithm for linear feasibility in UTVPI constraints. *Algorithmica*, 78(1):166–208, 2017.

[SWG13] K. Subramani, Matthew Williamson, and Xiaofeng Gu. Improved algorithms for optimal length resolution refutation in difference constraint systems. *Formal Aspects of Computing*, 25(2):319–341, 2013.

[Sze01] Stefan Szeider. Np-completeness of refutability by literal-once resolution. In *Automated Reasoning, First International Joint Conference, IJCAR 2001, Siena, Italy, June 18-23, 2001, Proceedings*, pages 168–181, 2001.

[Tar48] A. Tarski. A decision method for elementary algebra and geometry. Technical Report R-109, Rand Corporation, 1948.

[Tru03] Klaus Truemper. Personal communication, 2003.

[Tse68] G. S. Tseitin. On the complexity of derivation in the propositional calculus, 1968.

[Tse70] G. Tseitin. On the complexity of derivation in propositional calculus. In *Studies in Constructive Mathematics and Mathematical Logic*, pages 115–125, 1970.

[Urq95] Alasdair Urquhart. The complexity of propositional proofs. *The Bulletin of Symbolic Logic*, 1(4):425–467, December 1995.

[VD68] A.F. Veinott and G.B. Dantzig. Integral extreme points. *SIAM Review*, 10:371–372, 1968.

[Vei89] Arthur F. Veinott. Representation of general and polyhedral subsemilattices and sublattices of product spaces. *Linear Algebra and its Applications*, 114/115:681–704, 1989.

[VL92] Arthur F. Veinott and Marco LiCalzi. Subextremal functions and lattice programming, July 1992. Unpublished Manuscript.

[Voh06] Rakesh V. Vohra. The ubiquitous farkas' lemma. In *Perspectives in Operations Research*, volume 36 of *Operations Research/Computer Science Interfaces Series*, pages 199–210. Springer-Verlag, New York, 2006.

[VW62] A. F. Veinott and H. M. Wagner. Optimal capacity scheduing: Parts i and ii. *Operations Research*, 10:518–547, 1962.

[Wag77] T.J. Wagner. *Hardware Verification*. Ph.D. Thesis, Stanford University, 1977.

[WB97] Hal Wasserman and Manuel Blum. Software reliability via run-time result-checking. *J. ACM*, 44(6):826–849, 1997.

[Wei88] V. Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 4(1-2):3–27, 1988.

[Wil76] H. P. Williams. Fourier-Motzkin elimination extension to integer programming. *J. Combinatorial Theory*, 21:118–123, 1976.