The present work was submitted to the LuFG Theory of Hybrid Systems

MASTER OF SCIENCE THESIS

A NOVEL QUANTIFIER ELIMINATION ALGORITHM FOR A PSEUDO-LINEAR A*E FRAGMENT OF REAL ALGEBRA

Kai Hilgers

Examiners:
Prof. Dr. Erika Ábrahám
Prof. Dr. Thomas Noll

Additional Advisor:
Valentin Promies

Aachen, 27.11.2023

Abstract

To compute minimizing reachability probabilities in rectangular automata with random clocks, one must compute solutions for formulas from a specific fragment of real algebra. This fragment appears when computing the set of initial states from which a set of goal stats is reached unavoidably, under any possible non-deterministic choice of evolution options. In our work, we formalized this A^*E fragment and devised a quantifier elimination algorithm to compute the solution as a single linear constraint system. Since this field is still developing, no standardized benchmarks have been established yet. Consequently, to quantify the performance of multiple different implementations of such algorithms, we have developed a method to generate benchmarks automatically. Upon evaluating our algorithm, the algorithm generates up to $(\frac{m}{2})^2$ constraints, where m is the number of goal constraints. Additionally, based on our benchmarks, the runtime of our best algorithm is mainly affected by the number of goal constraints and is not directly influenced by the dimensionality of the problem.

Erklärung

Hiermit versichere ich, dass ich die vorgelegte Arbeit selbstständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe. Alle benutzten Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate wurden als solche gekennzeichnet.

Kai Hilgers
Aachen, den 27. November 2023

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Hybrid stochastic automata are a modeling formalism for systems with mixed discrete-continuous behavior that is influenced by uncertainties. Examples are regulating power plants, controlling robotics movements, managing aircraft control systems, and operating medical systems (see for example [HKD98]; [PSR21]). Such an automaton describes states that encapsulate the current values of all variables in the system. One can then label some of these states as safe when they correspond to a safe state of the underlying system. The state of the automaton evolves over time and the change in each variable is described by the *rates*. These rates can either be a fixed value or chosen non-deterministically. Additionally, one usually extends the automaton to multiple *location*, that each have different rates, introducing guards, jumps and labels to connect them. However, for our application, we will only require to analyse a single location. A state is called reachable if there exists a run of the automaton that reaches this state. This process of describing a (safety) property of the system and then proving that the system indeed satisfies the property is called *formal verification*. It is imperative to formally verify safety-critical systems. Failure states, such as an aircraft stalling due to erroneous engine control or a radiology machine administering lethal doses of radiation due to software errors, must be prevented. In addition to avoiding such failure states, it is necessary to provide formal proof that the system is safe. Therefore, we can no longer develop systems by just modeling the systems' dynamics. Instead, we must additionally formalize and verify their safety-critical properties. Hybrid (stochastic) automata have proven to be a suitable modeling language for this verification (see [CABls]). Though restricting the expressivity of the dynamics of the systems, they provide the necessary robustness to verify the safety of the system. While previous work has already found practical solutions to the reachability problem (see [ACH$^+$95]) - i.e., deciding whether a given bad state is reachable for some initial value with a suitable specific dynamic - the problem of whether a given state is reachable for some initial value under any of the possible dynamics is still under investigation. In particular, we would like to describe the set of states that reach a specified goal under any possible rate by a preferably small set of constraints. A satisfying solution to this fragment allows for two key concepts. First of all, the fragment describes a kind of certainty. Executions that start at any state inside this set will eventually reach the goal. One can leverage this fact to compute a

(possibly restricted) set of initial values from which the system will ultimately reach the goal regardless of the chosen actual dynamics. This way, we can greedily choose initial values under some objective function while ensuring the system eventually reaches the goal. Conversely, one can also specify a bad state as the goal and use the solution to the fragment to compute all points that lead to this unsafe state. One can then leverage this information to modify the system in a way that these points are not reachable, as they would only lead to a bad state regardless. Concretely, an algorithm for computing the maximal reachability probabilities (see [DSÁR23]) has been successfully developed. The sub-problem we solve in this thesis will be required for the related problem of computing minimal reachability probabilities. We are interested in a single location of the automaton, where the rates are specified by a set of linear constraints, and a goal is defined similarly as a set of linear constraints. The problem of this minimal reachability can then be expressed as a formula in *real algebra*. However, this formula still contains quantifiers that need to be eliminated for the next steps of the algorithm. Therefore, we aim to develop a quantifier-elimination algorithm for this specific subset of formulas in Real Algebra.

1.2 Problem Statement

The current state of a hybrid automaton can be described by a point in $p \in \mathbb{R}^n$, where $\mathbb{R}$ denotes the real numbers and n is the number of variables of the automaton. Now in a given *location* of the automaton, the possible change in p called *flow* is described by the *rectangular rates* of the specific location. The suitable rates $r \in \mathbb{R}^n$ are described by a formula $\mathfrak{B}$, consisting of a conjunction of linear constraints. Now we are given some *goal* which we want to reach at some time point $t \in \mathbb{R}_{\geq 0}$, where $\mathbb{R}_{\geq 0}$ denotes the non-negative real numbers. This goal has to be convex and is described by a formula $\mathfrak{A}$. We aim to compute a set of linear equations that restrict all points p such that for each of them, for any rate r that satisfies $\mathfrak{B}$, some timepoint $t' \in \mathbb{R}_{\geq 0}$ exists, where the goal is reached. Specifically, we aim to compute a representation for the set of solution points for formulae of the following form:

$$\psi := \forall \mathbf{r}.(\underbrace{\mathbf{B}\mathbf{r} \leq \mathbf{b}}_{\mathfrak{B}} \implies \exists t.0 \leq t \wedge \underbrace{\mathbf{A}(\mathbf{x} + t\mathbf{r}) \leq \mathbf{c}}_{\mathfrak{A}})$$

with $\mathbf{B} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{j \times n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{c} \in \mathbb{R}^j$ for some m,n,$j \in \mathbb{N}$, where $\mathbb{N}$ represents the natural numbers.

While this formula is not linear, its solution set is. We therefore call the problem *pseudo-linear*. To find this representation, we need to eliminate the quantifiers $\exists t$ and $\forall \mathbf{r}$. This step is called *quantifier elimination*. We will need to eliminate $\exists t$ once in the beginning and then eliminate $\forall \mathbf{r}$. Since $\mathbf{r}$ is a vector of n variables we will need n steps to then eliminate each component of $\mathbf{r}$. Therefore, our quantifier elimination method handles a specific A^*E fragment of such formulas from *real algebra*.

An example input could look like this.

$$\psi = \forall \mathbf{r}.(\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \mathbf{r} \leq \begin{bmatrix} 2 \\ -1 \\ 2 \\ 2 \end{bmatrix} \implies \exists t.0 \leq t \wedge \begin{bmatrix} -1 & 0 \\ 0 & 1 \\ -1 & 1 \\ 1 & 0 \\ 4 & -1 \end{bmatrix} (\mathbf{x} + t\mathbf{r}) \leq \begin{bmatrix} -4 \\ -2 \\ 6 \\ 6 \\ 18 \end{bmatrix}).$$

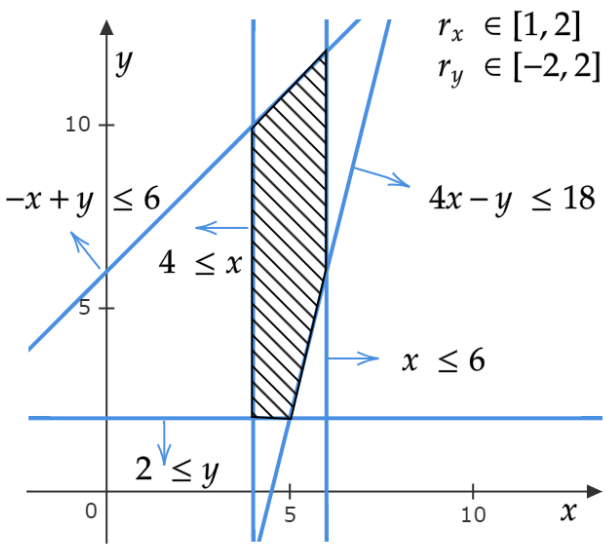


Figure 1.1: Introductory example depicting the goal as the intersection of five linear goal constraints shaded in the striped area. Additionally, the rates for each dimension are given as intervals.

Where the rates are given as intervals $r_x \in [1,2], r_y \in [-2,2]$.The goal is specified by a set of linear constraints depicted in Figure 1.1. We need to find all points that will reach this goal for some later timepoint t, no matter which actual rates are selected for r_x,r_y.

1.3 Research Objectives

The main objectives of this research are as follows:

1. To create a quantifier elimination algorithm designed explicitly for the pseudo-linear A*E fragment of real algebra.
2. Implement the algorithm and assess its computational efficiency and complexity.
3. Provide a way to generate benchmarks to test the algorithm's performance.
4. Evaluate the algorithm under the benchmarks and asses its performance critically.

1.4 Thesis Organization

The rest of the thesis is organized as follows:

- **Chapter 2: Theoretical Framework** introduces the theoretical models and algorithms we will use in the development of our algorithm and introduces the

main theoretical framework of linear arithmetic. Additionally, we will give a brief overview of related work.

- **Chapter 3: Proposed Quantifier Elimination Algorithm** contains the main part of the thesis. Our algorithm is introduced, and its correctness is proven. Additional improvements and their foundations are given.
- **Chapter 4: Experimental Evaluation** explains how the benchmarks are generated, which assumptions were made, and gives an overview of the experimental setup. These results are then displayed, evaluated, and discussed.
- **Chapter 5: Conclusion** summarizes the thesis's contributions, discusses the findings' implications, and discusses ideas for future work.

Chapter 2

Theoretical Framework

We begin by providing a brief overview of the notation, models, and algorithms used in this thesis.

2.1 Theoretical Models and Algorithms

Our work focuses on a special fragment of Real Algebra.

Definition 1 (Real Algebra)**.** *Real Algebra is the first-order theory over the real numbers* $\mathbb{R}$ *described by the following context-free grammar.*

$$\begin{aligned} \textit{Terms } t &:= 0 \mid 1 \mid x \mid t + t \mid t - t \mid t * t \\ \textit{Constraints } c &:= t \leq t \\ \textit{Formulas } \varphi &:= c \mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi \end{aligned}$$

Real Algebra is also called *non-linear real arithmetic* as opposed to *linear real arithmetic* that does not allow multiplication between terms.

The solutions or *models* of such formulas are defined as follows.

Definition 2 (Models)**.** *Let* ϕ_1,ϕ_2 *be two formulas. If the formula* $\phi_1 \wedge \neg\phi_2$ *is unsatisfiable, we say that* ϕ_1 *models* ϕ_2 *written as* $\phi_1 \models \phi_2$*.*

Definition 3 (Satisfying Set)**.** *Let* $\phi(\mathbf{x})$ *be a formula of Real Algebra with free variables* $\mathbf{x}$*. Then*

$$Sol(\phi) := \{\mathbf{x} \in \mathbb{R}^n \mid \mathbf{x} \models \phi\}$$

is the set of all $\mathbf{x}$ *that model* ϕ*.*

This thesis will use a fragment of Real Algebra that only allows multiplication between certain kinds of terms. Before we define this pseudo-linear fragment, we will first focus on the linear fragment.

For this, we will make use of matrices and vectors. All matrices will be printed as bold, capitalized letters such as $\mathbf{A}$. Vectors such as $\mathbf{b} \in \mathbb{R}^m$ are shorthand notations for matrices with one column $\mathbf{b} \in \mathbb{R}^{m\times 1}$. We denote with $\mathbf{A}_i \in \mathbb{R}^{1\times n}$ the ith row vector of a matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$. Given such a row vector $\mathbf{A}_i \in \mathbb{R}^{1\times n}$ and a column vector $\mathbf{x} \in \mathbb{R}^n$ $\mathbf{A_i x}$ denotes the scalar product of the two vectors and therefore, $\mathbf{A_i x} \in \mathbb{R}$. This scalar product is computed as $\sum_{j=1}^{n} a_{i,j}x_j$, where $a_{i,j}$ is the entry found in the ith row and jth column of $\mathbf{A}$.

Definition 4 (Linear Constraint System *LCS*)**.** *Given some matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and a vector* $\mathbf{b} \in \mathbb{R}^m$*, we write*

$$\mathbf{Ax} \leq \mathbf{b}$$

to denote the conjunction

$$\mathbf{A_1 x} \leq b_1 \wedge \mathbf{A_2 x} \leq b_2 \wedge \ldots \mathbf{A_m x} \leq b_m.$$

We call such a formula a linear constraint system (LCS) with free variables $\mathbf{x}$*.*

Additionally, we refer to LCS as sets of linear constraints and to their solution sets as *polytopes*. One such constraint of the form $\mathbf{A_i x} \leq b_i$ is also called a *half-space*, as it divides the solution space into the two areas that satisfy or do not satisfy the constraint.

Solving LCS is a broad topic encapsulated by the field of linear programming. We will only briefly sketch linear programs and assume that some black-box solver exists that computes the correct solution. One such method is the Simplex algorithm [Dan90], which we do not introduce here.

Definition 5 (Linear Programming *LP*)**.** *A linear program is given as.*

$$\begin{aligned} \underset{\mathbf{x}}{\textit{minimize}} \quad & \mathbf{a}^T \mathbf{x} \\ \textit{subject to} \quad & \mathbf{Bx} \leq \mathbf{b} \end{aligned}$$

Where one wants to find a vector $\mathbf{x}$ *that minimizes the given target function* $\mathbf{a}^T \mathbf{x}$ *under some set of linear constraints* $\mathbf{Bx} \leq \mathbf{b}$*.*

Definition 6 (Vertices of a LCS)**.** *In the n-dimensional space, the intersection of n linearly independent halfspaces defines a point. If such a point satisfies all other half-spaces of the LP, we say it is a vertex. We define the set of points* $\mathbf{x}$ *that satisfies these two conditions to be vertices*$(\mathbf{Ax} \leq b)$*.*

When one is interested in the values obtainable by applying functions to intervals of values, interval arithmetic can be used. For this thesis, we are primarily interested in linear combinations of intervals.

Definition 7 (Interval Arithmetic)**.** *Given an interval* $x = [x_l, x_h]$ *where* x_l,$x_h \in \mathbb{Q}$*, with* $x_l \leq x_h$ *and some real number a, the multiplication* $a * x = a * [x_l, x_h]$ *is defined as*

$$a * x = [min(a * x_l, a * x_h), max(a * x_l, a * x_h)].$$

Additionally, for some second interval $y = [y_l, y_h]$ *where* $y_l, y_h \in \mathbb{Q}$ *and* $y_l \leq y_h$ *their addition is defined to be*

$$[x_l,x_h] + [y_l,y_h] = [x_l + y_l, x_h + y_h].$$

For these simple operations, all values of the resulting interval are indeed obtainable, i.e., for all $z \in \mathbb{R}$ we have that $z \in x + y \iff \exists x'.\exists y'.x_l \leq x' \leq x_h \wedge y_l \leq y' \leq y_h \wedge z = x' + y'$. The same holds for the multiplication with constant factors. Extensions such as exponential function, logarithm, etc., exist; however, one might over-approximate the solution in these cases.

As linear real arithmetic is not expressive enough for our problem, and non-linear real arithmetic is too complex to solve as a whole, we investigate a certain *pseudo-linear A*E ragment*. Such a fragment allows us to restrict real algebra to formulas for which we can develop algorithms using the existing methods of linear real arithmetic.

Definition 8 (Pseudo-linear)**.** *Given some formula ϕ with free variables $\mathbf{x}$ we call ϕ* pseudo-linear *if there exists a LCS $\mathbf{Ax} \leq \mathbf{c}$ such that $\mathbf{Ax} \leq \mathbf{c} \equiv \phi$. I.e., the solutions to ϕ can be equivalently described by a set of linear constraints.*

Definition 9 (Our Pseudo-linear A*E Fragment of Real Algebra)**.** *We say that a formula is an instance of the pseudo-linear A*E fragment of Real Algebra if it can be rewritten to have the form of*

$$\psi := \forall \mathbf{r}.(\underbrace{\mathbf{Br} \leq \mathbf{b}}_{\mathfrak{B}} \implies \exists t.0 \leq t \wedge \underbrace{\mathbf{A}(\mathbf{x} + t\mathbf{r}) \leq \mathbf{c}}_{\mathfrak{A}})$$

with $\mathbf{B} \in \mathbb{R}^{m \times n}$, $\mathbf{b} \in \mathbb{R}^m$, $\mathbf{A} \in \mathbb{R}^{j \times n}$, $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{c} \in \mathbb{R}^j$ for some m,n,$j \in \mathbb{N}^{>0}$. Additionally, we require that $\mathbf{Br} \leq \mathbf{b}$ be bounded in every direction. I.e. $\forall \mathbf{x_0}.\forall \mathbf{r}.\exists t.\mathbf{B}(\mathbf{x_0} + \mathbf{rt}) \not\leq \mathbf{b}$, with $\mathbf{x_0}$,$\mathbf{r} \in \mathbb{R}^n$ and $t \in \mathbb{R}_{\geq 0}$.

Example 1. *The formula*

$$\forall r_x, \forall r_y (r_x \leq 2 \wedge r_y \leq 4 \implies \exists t.0 \leq t \wedge (x_1 + tr_x \leq 10 \wedge x_2 + tr_y \leq 20))$$

is an instance of this fragment.

This fragment allows multiplication only for the particular variable t and the rate variables. We will show that such formulas are indeed pseudo-linear. Additionally, after a simple transformation to prenex normal form, these formulas will have the form of $\forall^*$ followed by a single $\exists$. Hence, the complete name of the pseudo-linear A*E fragment of Real Algebra.

2.1.1 Algorithms

Due to the pseudo-linear nature of the given problem, we can adapt or outright use some of the algorithms developed for linear real arithmetic. We will outline these algorithms here and make extensive use of them in our algorithm.

Quantifier Elimination Quantifier elimination simplifies logical formulas by eliminating quantifiers ($\forall$ and $\exists$). It is a transformation of a formula with quantifiers into one that is equivalent without quantifiers. Equivalent in this sense means that both formulas have the same solution space, i.e., any valid solution to either formula also satisfies the other. This is a necessary step in many algorithms that can not directly work on formulas with quantifiers. In our case, we will prove that the results of our problem will always be describable by a set of linear constraints.

Fourier-Motzkin Elimination (FM) [Fou27] Given a set of linear constraints and a target variable to eliminate, the FM approach first divides the constraint set into lower and upper bounds on the variable. Given a single constraint $\mathbf{A_i x} \leq b_i$ we can rewrite this as $\sum_j a_{i,j} x_j \leq b_i$. If we now want to eliminate x_k, we obtain $x_k \leq \frac{b_i - \sum_{j \neq k} a_{i,j} x_j}{a_{i,k}}$ for $a_{i,k} > 0$ and $x_k \geq \frac{b_i - \sum_{j \neq k} a_{i,j} x_j}{a_{i,k}}$ for $a_{i,k} < 0$. In the case of $a_{i,k} = 0$, the constraint does not depend on the target variable we want to eliminate. Finally, we combine every lower and upper bound pairwise to obtain an equivalent formula. For $x_k \leq \frac{b_i - \sum_{j \neq k} a_{i,j} x_j}{a_{i,k}}$ and $x_k \geq \frac{b_l - \sum_{j \neq k} a_{l,j} x_j}{a_{l,k}}$ we obtain the combined formula to be $\frac{b_l - \sum_{j \neq k} a_{l,j} x_j}{a_{l,k}} \leq x_k \leq \frac{b_i - \sum_{j \neq k} a_{i,j} x_j}{a_{i,k}}$ and

therefore also $\frac{b_l - \sum_{j \neq k} a_{l,j} x_j}{a_{l,k}} \leq \frac{b_i - \sum_{j \neq k} a_{i,j} x_j}{a_{i,k}}$. While this does give the correct result, the runtime is doubly exponential when eliminating multiple variables. If we need to eliminate d variables in n constraints, we obtain at most $4(n/4)^{2^d}$, leading to a combinatorial blow-up. In our case, we will use the main ideas of FM sparingly, using it only for a single elimination. This then only generates a polynomial increase in constraints.

Virtual Substitution [CÁ11] As a quantifier elimination method, Virtual Substitution introduces test candidates into the formulas to differentiate sign invariant regions. By successively eliminating quantifiers, the result will eventually be obtained as a large disjunction. However, this poses a disadvantage, as disjunctions are generally more costly to check for satisfiability. Additionally, we want to receive a polytope as the solution for our application, which should be a conjunction of half-spaces instead.

2.2 Critical Analysis of Existing Solutions

Since we do not have only linear constraints, but also allow multiplication with the time variable t, we can not apply FM directly. However, as a Quantifier Elimination Method, the Virtual Substitution algorithm is applicable to our problem. It can, however, not leverage the structure of the problem to speed up the algorithm or reduce the complexity of the solution. This becomes apparent in the first steps of the algorithm. When eliminating the t variable, the Virtual Substitution algorithm introduces a test case for each constraint of the goal. Therefore, even after only eliminating t, the resulting formula will have grown to a factor of m^2, where m is the number of goal constraints. The elimination of the rate variables then increases the size of the formula exponentially in the number of eliminated variables. Therefore, using the Virtual Substitution approach on larger problem instances is not advisable.

Chapter 3

Proposed Quantifier Elimination Algorithm

3.1 Initial Observations

Definition 10 (Problem Definition)**.** *Given a formula ψ of the form*

$$\psi := \forall \mathbf{r}.(\underbrace{\mathbf{Br} \leq \mathbf{b}}_{\mathfrak{B}} \implies \exists t.0 \leq t \wedge \underbrace{\mathbf{A}(\mathbf{x} + t\mathbf{r}) \leq \mathbf{c}}_{\mathfrak{A}})$$

where $\mathbf{x} \in \mathbb{R}^n, \mathbf{B} \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{k \times n}, \mathbf{c} \in \mathbb{R}^k$ *compute an equivalent LCS* $\mathbf{Hx} \leq \mathbf{h}$ *such that* $\mathbf{Hx} \leq \mathbf{h} \equiv \psi(\mathbf{x})$

In this chapter, $\mathbf{A}$, $\mathbf{B}$, $\mathbf{b}$, $\mathbf{c}$, $\mathbf{x}$, $\mathbf{r}$ and $\psi(\mathbf{x})$ will always be defined according to the previous definition. Therefore, $\mathbf{A}$, $\mathbf{c}$ refer to the *goal* of the problem, $\mathbf{B}$, $\mathbf{b}$ give the restrictions on the *rates* $\mathbf{r}$ and lastly, $\mathbf{x}$ are the solution points that we are interested in.

Additionally, $\mathfrak{A}$ describes a formula, with a vector of free variables $\mathbf{x}'$, such that $\mathfrak{A}(\mathbf{x}')$ holds, iff $\mathbf{Ax}' \leq \mathbf{c}$. Similarly, we define $\mathfrak{B}$ to be a formula with exactly one vector of free variables $\mathbf{r}'$ such that $\mathfrak{B}(\mathbf{r}')$ holds iff $\mathbf{Br}' \leq \mathbf{b}$

Finally, in the following sections, we refer to instantiated problems of this form as *problem instance*.

We begin by proving that the quantifier elimination problem for our particular fragment is worth investigating. We want to prove that the solution to our problem will always be convex and, therefore, have a representation consisting of linear constraints - without any quantifiers. While this does not guarantee that we will find a finite definition, it does prove to be a good starting point. We will later show that using our algorithm will give a finite representation.

Lemma 1. *The set* $\{\mathbf{x} \in \mathbb{R}^n \mid \forall \mathbf{r}.(\mathbf{Br} \leq \mathbf{b} \implies \exists t.0 \leq t \wedge \mathbf{A}(\mathbf{x} + t\mathbf{r}) \leq \mathbf{c})\}$ *is convex.*

Proof. Given a problem instance, as

$$\psi = \forall \mathbf{r}.(\mathbf{Br} \leq \mathbf{b} \implies \exists t.0 \leq t \wedge \mathbf{A}(\mathbf{x} + t\mathbf{r}) \leq \mathbf{c}).$$

If the problem is unsatisfiable, the solution will be empty and thus convex. Therefore, assume two points $\mathbf{p_1}$,$\mathbf{p_2} \in \mathbb{R}^n$ that satisfy ψ. To prove the statement, we have to show, for any $0 < \lambda < 1$ with $\lambda \in \mathbb{R}$ that $\lambda\mathbf{p_1} + (1 - \lambda)\mathbf{p_2}$ satisfies ψ.

Given a rate vector $\mathbf{r} \in \mathbb{R}^n$ with $\mathbf{Br} \leq \mathbf{b}$ we define $\mathbf{p_1'}$,$\mathbf{p_2'} \in \mathbb{R}^n$ to be two arbitrary points that satisfy $\mathbf{Ap_i}' \leq \mathbf{c}$ and $\mathbf{p_i}' = \mathbf{p_i} + t_i\mathbf{r}$ for some $t_i \geq 0$ for $i \in \{1,2\}$. Since $\mathbf{p_1}$,$\mathbf{p_2}$ are solutions to the instance, the existence of $\mathbf{p_1'}$,$\mathbf{p_2'}$ is guaranteed.

We now need to show that there exists some $t \in \mathbb{R}^{\geq 0}$, such that

$$\mathbf{A}(\lambda\mathbf{p_1} + (1-\lambda)\mathbf{p_2} + t\mathbf{r}) \leq c.$$

For this, we use the convexity of the original constraints $\mathbf{Ax} \leq \mathbf{c}$. We will show that $t := \lambda t_1 + (1-\lambda)t_2$ is a valid choice. Since t_1,$t_2 \geq 0$ and $0 < \lambda < 1$ it holds that $\lambda t_1 + (1-\lambda)t_2 \geq 0$

$$\begin{aligned}\lambda\mathbf{p_1} + (1-\lambda)\mathbf{p_2} + t\mathbf{r} &= \lambda\mathbf{p_1} + (1-\lambda)\mathbf{p_2} + (\lambda t_1 + (1-\lambda)t_2)\mathbf{r} \\ &= \lambda(\mathbf{p_1} + t_1\mathbf{r}) + (1-\lambda)(\mathbf{p_2} + t_2\mathbf{r}) \\ &= \lambda\mathbf{p_1}' + (1-\lambda)\mathbf{p_2'}\end{aligned}$$

As our goal $\mathfrak{A}$ is convex, and we have a linear combination of two solution points, the resulting solution is valid - proving the statement. □

There are some trivial cases we want to exclude from our future proofs, such as when no valid rates exist or when the goal is unsatisfiable.

Lemma 2. *If $\mathfrak{B}$ is unsatisfiable then $\psi \equiv$ True.*

Therefore, we assume $\mathfrak{B}$ to be satisfiable from now on.

Lemma 3. *If $\mathfrak{A}$ is unsatisfiable (and $\mathfrak{B}$ is satisfiable) then $\psi \equiv$ False.*

Therefore, we assume $\mathfrak{A}$ will also be satisfiable.

Lemma 4. *Every solution to the goal $\mathfrak{A}$ is already a solution to ψ: $\forall \mathbf{x} \in \mathbb{R}^n \mathfrak{A}(\mathbf{x}) \implies \psi(\mathbf{x})$.*

Proof.

$$\begin{aligned} & \text{Since } \mathfrak{A}(\mathbf{x}) \text{ holds, we can choose } t = 0 \text{ to obtain} \\ & \exists t.0 \leq t \wedge \mathbf{A}(\mathbf{x} + t\mathbf{r}) \leq \mathbf{c}, \\ \implies \quad & \forall \mathbf{r}.(\mathbf{Br} \leq \mathbf{b} \implies \exists t.0 \leq t \wedge \mathbf{A}(\mathbf{x} + t\mathbf{r}) \leq \mathbf{c}) \\ \implies \quad & \psi(\mathbf{x}) \end{aligned}$$

□

One last useful observation can be obtained when considering each constraint of the goal. If it is possible to move away from every constraint (i.e. for each row i there exists a rate $\mathbf{r}$ such that $\mathbf{A_i r} \geq 0$), then increasing t will not help in satisfying the problem.

Lemma 5. *If for each $i \in \{1,\ldots,m\}$ there is $\mathbf{r} \in \mathbb{R}^\mathbf{n}$ with $\mathfrak{B}(\mathbf{r})$ and $\mathbf{A_i r} \geq 0$, then $\psi(\mathbf{x}) \equiv \mathfrak{A}(\mathbf{x})$.*

Proof. $\mathfrak{A}(\mathbf{x}) \models \psi(\mathbf{x})$ follows from Lemma 4.

For the $\psi(\mathbf{x}) \models \mathfrak{A}(\mathbf{x})$ direction let us consider an arbitrary row $\mathbf{A_i}$ of $\mathbf{A}$

Since $\mathbf{A_i r} \geq 0$, is satisfiable with some $\mathbf{r}'$ such that $\mathfrak{B}(\mathbf{r}')$ we can observe for this row of $\mathfrak{A}$

$$\begin{aligned} & \psi \models \exists t.0 \leq t \wedge \mathbf{A_i}(\mathbf{x} + t\mathbf{r}') \leq c_i \\ \Leftrightarrow \quad & \psi \models \exists t.0 \leq t \wedge \mathbf{A_i x} + \mathbf{A_i} t\mathbf{r}' \leq c_i \\ \implies \quad & \psi \models \mathbf{A_i x} \leq c_i \end{aligned}$$

Since this holds for any row $\mathbf{A_i}$ it follows that $\psi(\mathbf{x}) \implies \mathfrak{A}(\mathbf{x})$ □

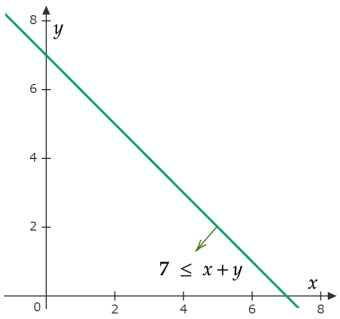


Figure 3.1: Goal with constraint $7 \leq x + y$

3.1.1 Handling a Single Constraint

To understand the concept of the algorithm, let us first consider the case where the goal is described by just one inequality $7 \leq x + y$ (see Figure 3.1).

We are now interested in finding all points x,y such that

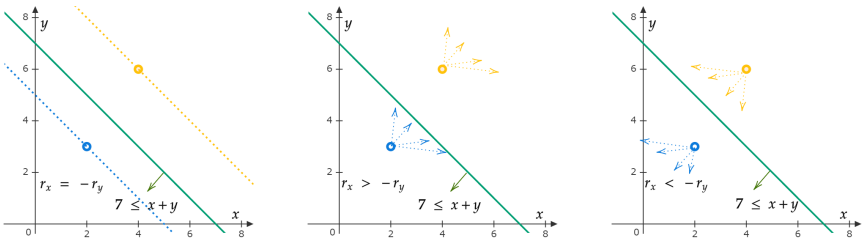
$$\forall r.(\mathbf{Br} \leq \mathbf{b} \implies \exists t.0 \leq t \wedge 7 \leq x + r_x t + y + r_y t). \tag{3.1}$$

We deliberately ignore the restriction of $\mathbf{Br} \leq \mathbf{b}$ for now, such that all possible cases are generated. While the original constraint was a lower bound on x and y, it is not necessarily a lower bound on t. Instead, after rewriting, we obtain

$$0 \leq t \wedge 7 - x - y \leq t(r_x + r_y). \tag{3.2}$$

Now, depending on the sign of $r_x + r_y$, we might obtain a lower or an upper bound on t. Or if $r_x + r_y = 0$, then t appears only in $0 \leq t$, which is satisfiable by e.g. the value 0 for t.

These are the three possible cases to consider (see Figure 3.2).



(a) (b) (c)

Figure 3.2: The three possible cases for a single constraint and some undetermined rates. a) represents the case $r_x = -r_y$ where the states evolve parallel to the goal constraint. b) shows rates $r_x > -r_y$ where every state evolve towards the goal. c) shows the opposite case $r_x < -r_y$ where the states evolve away from the goal.

1. $r_x + r_y = 0$ (see Figure 3.2 a)

   The simplest case is obtained when $r_x = -r_y$. This case corresponds to the rate being parallel to the hyperplane of the goal constraint. Hence, when we keep increasing t, we will move parallel to the goal. Therefore, we will only reach the goal if we start within the goal. Moreover, neither will we leave the goal if we start in the goal initially. The only solutions for $\mathbf{x}$ are those already satisfying the goal constraint $7 \leq x + y$. Inequality 3.2 also reflects this result, as substituting $r_x = -r_y$ returns the original constraint $7 \leq x + y$.

2. $r_x + r_y > 0$ (see Figure 3.2 b)

   Next is the intuitive case of the constraint being a lower bound for t. For this to hold, we require $r_x > -r_y$. In the visual representation, this equates to all directions that point toward the hyperplane of the given goal constraint. We obtain $\frac{7-x-y}{r_x-r_y} \leq t$. Notice that, for only one constraint, this inequality will always have a valid solution, as we can keep increasing the value of t until the goal's constraint is satisfied.

3. $r_x + r_y < 0$ (see Figure 3.2 b)

   The last case is obtained for $r_x < -r_y$. The visual representation is that the rate moves away from the hyperplane of the goal. Therefore, at some point, the goal will not be satisfied anymore. However, this is not an issue since we only require that the goal be reached at some point. Instead, the restrictive part becomes that any point outside the goal will never reach the goal. Therefore, this case also returns the original constraint. It is, however, computed differently. First, the new case is generated: $t \leq \frac{7-x-y}{r_x+r_y}$. Now, we use FM to combine this case with the requirement of $0 \leq t$ to obtain $0 \leq \frac{y-x+7}{r_x-ry}$ and after rewriting: $7 \leq x+y$.

In general, it will not always be necessary to generate all three of these cases. Instead, the rate restrictions $\mathbf{Br} \leq \mathbf{b}$ that we ignored in this example will determine which cases need to be considered.

3.1.2 Handling Pairs of Constraints

We will later show that the problem reduces to an intersection over solutions for pairs of constraints. We will now use similar observations from the previous section to combine two arbitrary constraints.

The FM approach already points us to the relevant cases, as in FM, we only combine lower and upper bounds. Therefore, when handling two constraints, they can only interact with each other if they form opposite bounds on t.

Let us focus on the ith row ($\mathbf{A_i}(\mathbf{x} + t\mathbf{r}) \leq c_i$) of $\mathfrak{A}$ and consider the second constraint later. After distributing, we obtain $\mathbf{A_i x} + \mathbf{A_i} t\mathbf{r} \leq c_i$. Now to apply FM, we need to isolate t. This poses a problem since we need to know the sign of $\mathbf{A_i r}$ as the inequality might flip when dividing by $\mathbf{A_i r}$. Therefore, our single constraint can pose different kinds of bounds (lower, upper, or both) depending on the sign of $\mathbf{A_i r}$. We introduce up to three new cases to distinguish these so-called *precondition*.

Case 1. $\mathbf{A_i r} > 0$ we obtain an upper bound $t \leq \frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}}$.

Case 2. $\mathbf{A_i r} = 0$ does not involve t and we obtain $\mathbf{A_i x} \leq c_i$.

Case 3. $\mathbf{A_i r} < 0$ we obtain a lower bound $\frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}} \leq t$.

However, we need to generate only the cases where the precondition is achievable under the given rate restriction $\mathfrak{B}(\mathbf{r})$. In the general case, we will need to solve the corresponding LCS to determine, if each of the preconditions is indeed obtainable. However, for the common case, where the rate restrictions $\mathfrak{B}(\mathbf{r})$ define a box (each dimension does not depend on any other dimension and is given by an interval, for example $r_x \in [-1,2], r_y \in [1,2]$), we can use interval arithmetic. We can obtain one interval representing all possible values of $\mathbf{A_i r}$ by substituting the intervals for the rates and adding their linear combinations. We can now test whether this interval contains negative or positive values and check if 0 is contained.

In the second case, we notice that t is eliminated already. We can immediately add the resulting inequality to our solution constraints. Again we observe that the added constraint is indeed one of the original goal constraints.

We will now focus on the upper bounds on t, which we obtained from the first case $\mathbf{A_i r} > 0$ with $t \leq \frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}}$. Remember that $0 \leq t$ is also one of our constraints. By applying FM, we can obtain $0 \leq \frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}}$. Making use of the case condition we obtain $0 \leq c_i - \mathbf{A_i x}$. Hence, we obtain the original constraint again:

$$\mathbf{A_i x} \leq c_i.$$

Finally, when considering the lower-bounds $\frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}} \leq t$ for $\mathbf{A_i r} < 0$, we will have to also consider all upper bounds on t. For this, we show the combination with a second row of $\mathfrak{A}$, say $\mathbf{A}_j(\mathbf{x} + t\mathbf{r} \leq c_j)$ where $\mathbf{A_j r} > 0$ and thus $t \leq \frac{c_j - \mathbf{A_j x}}{\mathbf{A_j r}}$. We can now combine the two results using FM to obtain

$$\frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}} \leq \frac{c_j - \mathbf{A_j x}}{\mathbf{A_j r}}.$$

Using the respective case conditions, we can derive the following result

$$(c_j - \mathbf{A_j x})\mathbf{A_i r} \leq (c_i - \mathbf{A_i x})\mathbf{A_j r}. \tag{3.3}$$

Additionally, it will also be useful to present the inequality in a slightly different form of

$$\frac{\mathbf{A_i r}}{\mathbf{A_j r}}(c_j - \mathbf{A_j x}) \leq (c_i - \mathbf{A_i x}). \tag{3.4}$$

We have successfully eliminated all occurrences of t. However, we must still eliminate the rate variable $\mathbf{r}$ in the linear combinations $\mathbf{A_i r}$ and $\mathbf{A_j r}$.

To obtain a first result, we will add the constraints $\mathbf{A_i r} < 0, \mathbf{A_j r} > 0$ to our rate restriction $\mathfrak{B}$ and add for each of the resulting vertices a new constraint to our solution, where we substitute the values r with the corresponding vertex values. This will add exponentially many constraints for each combination. In a later section, we will discuss how to drastically lower this number. However, this can be the basis for our correctness proof.

3.2 Basic Algorithm

We briefly sketch how this first version of our algorithm works. We get as input $\mathbf{B} \in \mathbb{R}^{m \times n}, \mathbf{b} \in \mathbb{R}^n, \mathbf{A} \in \mathbb{R}^{k \times n}, \mathbf{c} \in \mathbb{R}^k$. We now compute for each pair of constraints

$\mathbf{A_i}$,$\mathbf{A_j}$ their combination as outlined by the previous two sections. We begin by describing the combination of $i = j$. In this case, we combine the constraint with the additional requirement of $0 \leq t$. Using our previous results, we add to our computed result the constraint $\mathbf{A_i x} \leq c_i$, if $\mathbf{Br} \leq \mathbf{b} \wedge \mathbf{A_i r} \geq 0$ is satisfiable. For the pairs of constraints, where $i \neq j$, we first check which cases are possible to be generated for each constraint. Therefore, we test if $\mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_k r} \sim 0$ is satisfiable, for $\sim \in \{<\ ,\ =\ ,\ >\}$ and $k \in \{i,j\}$. In the common case, where our $\mathfrak{B}(\mathbf{r})$ describes rates obtained by intervals, we can use interval arithmetic to compute this efficiently. Otherwise, we will have to solve the corresponding linear program. Now recall that the $\mathbf{A_k r} > 0$ case induces an upper bound on t and the $\mathbf{A_k r} < 0$ case introduces a lower bound on t. Finally, the $\mathbf{A_k r} = 0$ case gives us back the original constraint, so we can add it to our solution. Now if $\mathbf{A_i r} < 0$ and $\mathbf{A_j r} > 0$ we obtain the constraint $(c_j - \mathbf{A_j x})\mathbf{A_i r} \leq (c_i - \mathbf{A_i x})\mathbf{A_j r}$. We will then substitute the values of $\mathbf{A_i r}$ and $\mathbf{A_j r}$ with every vertex of $\mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} \leq 0 \wedge \mathbf{A_j r} \geq 0$ and add each of these constraints to our solution. In our proof, we will argue why this is correct. Finally, we return the conjunction of all the added constraints as our result.

Algorithm 1 Basic Version of our Algorithm

```
procedure SOLVE(B,b,A,c)
    result ← True
    for (i,j) ∈ {1,...,m} × {1,...,m} do
        if i == j then
            if Br ≤ b ∧ A_i r ≥ 0 satisfiable then
                result ← result ∧ A_i x ≤ c_i
            end if
        else
            if Br ≤ b ∧ A_i r < 0 ∧ A_j r > 0 satisfiable then
                for (v_i,v_j) ∈ vertices(Br ≤ b ∧ A_i r ≤ 0 ∧ A_j r ≥ 0) do
                    result ← result ∧ (c_j − A_j x)v_i ≤ (c_i − A_i x)v_j
                end for
            end if
        end if
    end for
    return result
end procedure
```

Some key assumptions which still need to be substantiated during our proof of correctness are the following:

1. It is admissible to consider only pairs of constraints.

2. Enumerating the vertices and adding a corresponding constraint gives the correct constraints for the combination.

3. We can add the generated constraint to our solution without requiring any preconditions on the rates that define under which assumptions the constraint was generated.

3.3 Proof of Correctness

We will now show that the basic algorithm presented earlier is correct. This means that the solutions which our algorithm produces, are exactly the solutions specified by the given problem instance. We begin by defining this behavior formally and then use these definitions in our proof.

Definition 11 (Solution Set)**.** *We define*

$$Sol_{\mathfrak{B}}(\mathfrak{A}) := \{\mathbf{x} \in \mathbb{R}^n \mid \forall \mathbf{r}.(\mathfrak{B}(\mathbf{r}) \implies \exists t.0 \leq t \wedge \mathfrak{A}(\mathbf{x} + t\mathbf{r}))\}$$

and call it the solution set.

Notice that $Sol_{\mathfrak{B}}(\mathfrak{A}) = Sol(\psi)$. However, the explicit dependency on the notion of $\mathfrak{B}$ and $\mathfrak{A}$ will be useful for our proof.

Additionally, we are ultimately interested in obtaining the LCS that describes the solution set. However, for the correctness proof it will be easier to represent the solution as a set of points instead. We will later explain how to obtain the LCS without any additional computations.

Definition 12 (Row Selection)**.** *Let $i,j \in \{1,\ldots,m\}$. We define*

$$\mathfrak{A}[i,j] := \begin{bmatrix} A_i \\ A_j \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} c_i \\ c_j \end{bmatrix}.$$

Additionally, we define $\mathfrak{A}[i] := \mathfrak{A}[i,i]$.

Definition 13 (Vertex Enumeration)**.** *Given a formula $\phi(\mathbf{r})$ and an LCS $\mathfrak{B}(\mathbf{r})$, where $\mathbf{r} \in \mathbb{R}^n$ is a vector of variable rates, we define the enumeration of $\mathfrak{B}$ to be*

$$Sol_{\mathfrak{B}}(\phi,\mathbf{r}) := Sol\left(\bigwedge_{\mathbf{v} \in vertices(\mathfrak{B})} \phi[\mathbf{r}/\mathbf{v}]\right)$$

where $\phi[\mathbf{r}/\mathbf{v}]$ denotes the replacement of $\mathbf{r}$ in ϕ by $\mathbf{v}$.

Definition 14 (Extended $\mathfrak{B}$)**.** *Given the original rate constraints $\mathfrak{B}$:*

$$\mathbf{Br} \leq \mathbf{b}$$

we define the extension of $\mathfrak{B}$ with respect to the two additional rows $A_i r < 0$, $0 < A_j r$ to be

$$\mathfrak{B} \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r} := \begin{bmatrix} \mathbf{B} \\ \mathbf{A_i} \\ -\mathbf{A_j} \end{bmatrix} \mathbf{r} \leq \begin{bmatrix} \mathbf{b} \\ 0 \\ 0 \end{bmatrix}.$$

Notice that this definition is not just the straightforward process of adding two rows to the constraint system. Initially, we are given strict inequalities and relax these to be non-strict. In general, this is, of course, not equivalent. However, for our problem, we can use the relaxed inequalities due to a particular property of the system which we will explain shortly. This particular syntax will only be used in the following definition.

We define the $Combine_{\mathfrak{B}}$ operator that combines a lower bound on t with an upper bound. The motivation for this is explained in Subsection 3.1.2 and the particular Equation 3.3.

Definition 15 (Combination of two rows)**.** *We define*

$$Combine_{\mathfrak{B}}(\mathbf{A_i},\mathbf{A_j},c_i,c_j,\mathbf{x},\mathbf{r}) := Sol_{\mathfrak{B}\wedge\mathbf{A_i r}<0\wedge 0<\mathbf{A_j r}}\left((c_j-\mathbf{A_j x})\mathbf{A_i r}\leq(c_i-\mathbf{A_i x})\mathbf{A_j r},\mathbf{r}\right).$$

Now, if in the relaxed problem $\mathbf{A_i r}=0$ is achievable, we obtain $0\leq(c_i-\mathbf{A_i x})$. In the strict problem, however, we would only achieve infinitesimally small values. However, in the limit, since we can keep decreasing $\mathbf{A_i r}$ (as otherwise, $\mathbf{A_i r}=0$ would not be achievable in the relaxed problem) we will also reach $0\leq(c_i-\mathbf{A_i x})$ for sufficiently small values. A similar argument holds for the $\mathbf{A_j r}=0$ case. Finally, in the next lemma, we will show that if $\mathbf{A_k r}=0$ for $k\in\{i,j\}$ then we will generate the corresponding constraint, where one side becomes 0 anyways. Ultimately, we will show that we can even get rid of the $Combine_{\mathfrak{B}}$ cases that permit the $\mathbf{A_k r}=0$ cases. Therefore, this quirk in notation is only relevant for the initial proof.

The solution for a single row is the original constraint if a rate exists that does not decrease the distance from the constraint in each time step t, or every point is valid when no such rate exists. We also refer to this as the *single-dimensional problem.*

Lemma 6. *Let $i\in\{1,\ldots,m\}$. The Solution to the single-dimensional Problem $\mathfrak{A}[i]$ is*

$$Sol_{\mathfrak{B}}(\mathfrak{A}[i])=\begin{cases}Sol(\mathbf{A_i x}\leq c_i) & \text{if } \mathfrak{B}(\mathbf{r})\wedge 0\leq\mathbf{A_i r} \text{ satisfiable}\\ \mathbb{R}^n & \text{otherwise}\end{cases}.$$

Proof. If there is $\mathbf{r}\in\mathbb{R}^n$ such that $\mathfrak{B}(\mathbf{r})$ holds and $0\leq\mathbf{A_i r}$ then $Sol_{\mathfrak{B}}(\mathfrak{A}[\mathrm{i}])=Sol(\mathbf{A_i x}\leq c_i)$ immediately follows from Lemma 5.

Otherwise, for all $\mathbf{r}\in\mathbb{R}^n$ with $\mathfrak{B}(\mathbf{r})$ we have $\mathbf{A_i r}<0$ and thus

$$\begin{aligned}
Sol_{\mathfrak{B}}(\mathfrak{A}[i,i]) &= \{\mathbf{x}\in\mathbb{R}^n \mid \forall\mathbf{r}.\mathfrak{B}(\mathbf{r}) \implies \exists t.0\leq t\wedge\mathfrak{A}[i](\mathbf{x}+t\mathbf{r})\}\\
&= \{\mathbf{x}\in\mathbb{R}^n \mid \forall\mathbf{r}.\mathfrak{B}(\mathbf{r}) \implies \exists t.0\leq t\wedge\mathbf{A_i}(\mathbf{x}+t\mathbf{r})\leq c_i\wedge\mathbf{A_i}(\mathbf{x}+t\mathbf{r})\leq c_i\}\\
&= \{\mathbf{x}\in\mathbb{R}^n \mid \forall\mathbf{r}.\mathfrak{B}(\mathbf{r}) \implies \exists t.0\leq t\wedge\mathbf{A_i}(\mathbf{x}+t\mathbf{r})\leq c_i\}\\
&= \{\mathbf{x}\in\mathbb{R}^n \mid \forall\mathbf{r}.\mathfrak{B}(\mathbf{r}) \implies \exists t.0\leq t\wedge\mathbf{A_i x}+t\mathbf{A_i r}\leq c_i\}\\
&= \bigcap_{\mathbf{r}\text{ s.t }\mathfrak{B}(\mathbf{r})}\{\mathbf{x}\in\mathbb{R}^n \mid \exists t.0\leq t\wedge\mathbf{A_i x}+t\mathbf{A_i r}\leq c_i\}\\
&= \bigcap_{\mathbf{r}\text{ s.t }\mathfrak{B}(\mathbf{r})}\{\mathbf{x}\in\mathbb{R}^n \mid \exists t.0\leq t\wedge\frac{c_i-\mathbf{A_i x}}{\mathbf{A_i r}}\leq t\}\\
&= \bigcap_{\mathbf{r}\text{ s.t }\mathfrak{B}(\mathbf{r})}\mathbb{R}^n
\end{aligned}$$

Hence, we obtain

$$Sol_{\mathfrak{B}}(\mathfrak{A}[\mathrm{i}])=\begin{cases}Sol(\mathbf{A_i x}\leq c_i) & \text{if } \mathfrak{B}(\mathbf{r})\wedge 0\leq\mathbf{A_i r} \text{ satisfiable}\\ \mathbb{R}^n \text{ otherwise}\end{cases}$$

□

The solution to a problem instance with two constraints has to satisfy both single-dimensional instances independently, as well as satisfy the *combine* property, which states that if it is possible to find a rate that increases the distance from one constraint while decreasing the distance to the other, then the solution can only consist of those points that satisfy the increasing constraint before violating the decreasing one.

Theorem 1. *Let*

$$Combine_{\mathfrak{B},i,j} := \begin{cases} Combine_{\mathfrak{B}}(\mathbf{A_i}, \mathbf{A_j}, c_i, c_j, \mathbf{x}, \mathbf{r}) & \text{if } \mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r} \text{ satisfiable} \\ \mathbb{R}^n & \text{otherwise} \end{cases}.$$

Then

$$Sol_{\mathfrak{B}}(\mathfrak{A})[i,j] = Sol_{\mathfrak{B}}(\mathfrak{A}[i]) \cap Sol_{\mathfrak{B}}(\mathfrak{A}[j]) \cap Combine_{\mathfrak{B},i,j} \cap Combine_{\mathfrak{B},j,i}$$

Proof.

$$\begin{aligned} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j]) &\overset{\text{Definition 11}}{=} \{\mathbf{x} \in \mathbb{R}^n \mid \forall \mathbf{r}.\mathfrak{B}(\mathbf{r}) \implies \exists t.0 \le t \wedge \mathfrak{A}[i,j](\mathbf{x} + t\mathbf{r})\} \\ &\overset{\text{Definition 12}}{=} \{\mathbf{x} \in \mathbb{R}^n \mid \forall \mathbf{r}.\mathfrak{B}(\mathbf{r}) \implies \exists t.0 \le t \wedge \mathbf{A_i}(\mathbf{x} + t\mathbf{r}) \le c_i \wedge \mathbf{A_j}(\mathbf{x} + t\mathbf{r}) \le c_j)\} \\ &= \bigcap_{\mathbf{r} \text{ s.t } \mathfrak{B}(\mathbf{r})} \{\mathbf{x} \in \mathbb{R}^n \mid \exists t.0 \le t \wedge \mathbf{A_i}(\mathbf{x} + t\mathbf{r}) \le c_i \wedge \mathbf{A_j}(\mathbf{x} + t\mathbf{r}) \le c_j\} \end{aligned}$$

When eliminating t, we must combine every *lower bound* with every *upper bound*. The only way for the $\mathbf{A_i}$ and $\mathbf{A_j}$ row to interact with each other is therefore, when either $\mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r}$ or $\mathbf{A_j r} < 0 \wedge 0 < \mathbf{A_i r}$. In all other cases, $Sol_{\mathfrak{B}}(\mathfrak{A}[i,j]) = Sol_{\mathfrak{B}}(\mathfrak{A}[i]) \cap Sol_{\mathfrak{B}}(\mathfrak{A}[j])$ Therefore, consider the cases

Case 1 $\mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r}$

$$\begin{aligned} &\{\mathbf{x} \in \mathbb{R}^n \mid \exists t.0 \le t \wedge \mathbf{A_i}(\mathbf{x} + t\mathbf{r}) \le c_i \wedge \mathbf{A_j}(\mathbf{x} + t\mathbf{r}) \le c_j\} \\ &= \{\mathbf{x} \in \mathbb{R}^n \mid \exists t.0 \le t \wedge \frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}} \le t \wedge t \le \frac{c_j - \mathbf{A_j x}}{\mathbf{A_j r}}\} && \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r} \\ &= \{\mathbf{x} \in \mathbb{R}^n \mid \frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}} \le \frac{c_j - \mathbf{A_j x}}{\mathbf{A_j r}} \wedge 0 \le \frac{c_j - \mathbf{A_j x}}{\mathbf{A_j r}}\} && \text{FM} \\ &= \{\mathbf{x} \in \mathbb{R}^n \mid \frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}} \le \frac{c_j - \mathbf{A_j x}}{\mathbf{A_j r}}\} \cap Sol_{\mathfrak{B}}(\mathfrak{A}[j]) \\ &= \{\mathbf{x} \in \mathbb{R}^n \mid (c_j - \mathbf{A_j x})\mathbf{A_i r} \le (c_i - \mathbf{A_i x})\mathbf{A_j r}\} \cap Sol_{\mathfrak{B}}(\mathfrak{A}[j]) \end{aligned}$$

We can drop $\cap Sol_{\mathfrak{B}}(\mathfrak{A}[j])$ as we will compute the intersection with it anyway. Hence, we are currently computing

$$\bigcap_{\mathbf{r} \in \mathbb{R}^n \text{ s.t. } \mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r}} \{\mathbf{x} \in \mathbb{R}^n \mid (c_j - \mathbf{A_j x})\mathbf{A_i r} \le (c_i - \mathbf{A_i x})\mathbf{A_j r}\}$$

Since $\mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r}$ is convex, we will only be interested in the vertices of the resulting polytope. The reason for this is as follows. Assume some $\mathbf{r}'$ that is not a vertex of the polytope. Then, since $\mathbf{A_i} \neq \mathbf{A_j}$, a direction d' exists that does not change the value of one of the terms. I.e., either $\mathbf{A_i}(\mathbf{r}' + \mathbf{d}') = \mathbf{A_i r}'$ or $\mathbf{A_j}(\mathbf{r}' + \mathbf{d}') = \mathbf{A_j r}'$. If we follow this direction in any way, either increasing or decreasing one of the values, we will obtain a stronger inequality than what we achieved with r'. We can repeat this until we hit one of the vertices in this projection. Finally, the other dimensions not included in either $\mathbf{A_i}$ or $\mathbf{A_j}$ can be chosen arbitrarily at a vertex since they do not influence the result. Therefore,

it is sufficient to substitute the vertices of $\mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r}$.

$$
\begin{aligned}
&Sol\left(\bigwedge_{\mathbf{v}\in \text{vertices}(\mathfrak{B}(\mathbf{r})\wedge \mathbf{A_i r}<0\wedge 0<\mathbf{A_j r})} (c_j - \mathbf{A_j x})\mathbf{A_i v} \leq (c_i - \mathbf{A_i x})\mathbf{A_j v}\right) \\
&= Sol_{\mathfrak{B}\wedge \mathbf{A_i r}<0\wedge 0<\mathbf{A_j r}}\left((c_j - \mathbf{A_j x})\mathbf{A_i r} \leq (c_i - \mathbf{A_i x})\mathbf{A_j r}, \mathbf{r}\right) && \text{Definition 13} \\
&= Combine_{\mathfrak{B}}(\mathbf{A_i}, \mathbf{A_j}, c_i, c_j, \mathbf{x}, \mathbf{r}) && \text{Definition 15}
\end{aligned}
$$

Case 2 $\mathbf{A_i r} > 0 \wedge 0 > \mathbf{A_j r}$ Same as Case 1, via substituting i with j and vice versa.

Therefore, we obtain

$$\text{Sol}_{\mathfrak{B}}(\mathfrak{A})[i,j] = \text{Sol}_{\mathfrak{B}}(\mathfrak{A}[i]) \cap \text{Sol}_{\mathfrak{B}}(\mathfrak{A}[j]) \cap \text{Combine}_{\mathfrak{B},i,j} \cap \text{Combine}_{\mathfrak{B},j,i}$$

□

So far we have shown how to compute the result for a single constraint and for a pair of constraints. We now want to leverage this and show that it is sufficient to consider only the intersection over all pairs of constraints instead of the whole constraint set.

Theorem 2. $Sol_{\mathfrak{B}}(\mathfrak{A}) = \bigcap_{i,j} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$

Proof. We will show both inclusions to prove the claim.

Case 1 $Sol_{\mathfrak{B}}(\mathfrak{A}) \subseteq \bigcap_{i,j} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$

This direction immediately follows from the definition of $Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$, since we obtain $\mathfrak{A}[i,j]$ by selecting only two constraints from $\mathfrak{A}$ it holds that $\mathfrak{A}(\mathbf{x}) \implies \mathfrak{A}[i,j](\mathbf{x})$. Hence, for any i,j $Sol_{\mathfrak{B}}(\mathfrak{A}) \subseteq Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$ and therefore also

$$Sol_{\mathfrak{B}}(\mathfrak{A}) \subseteq \bigcap_{i,j} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$$

Case 2 $Sol_{\mathfrak{B}}(\mathfrak{A}) \supseteq \bigcap_{i,j} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$

Assume $\mathbf{x} \in \bigcap_{i,j} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$. We will now show that then also $\mathbf{x} \in Sol_{\mathfrak{B}}(\mathfrak{A})$. To demonstrate this, we will use a proof-by-contradiction.

Assume that $\mathbf{x} \notin Sol_{\mathfrak{B}}(\mathfrak{A})$ then $\mathbf{x} \models \neg\psi$ and

$$\exists \mathbf{r}.(\mathbf{Br} \leq \mathbf{b} \wedge \forall t.(t < 0 \vee \neg(\mathbf{A}(\mathbf{x} + t\mathbf{r}) \leq \mathbf{c}))) \text{ holds.}$$

Let $\mathbf{r}'$ be such a rate, then for all $0 \leq t$, we obtain $\neg(\mathbf{A}(\mathbf{x} + t\mathbf{r}') \leq \mathbf{c})$. Let us consider how this can be the case. Consider an arbitrary index i of $\mathfrak{A}$. Since we have a given $\mathbf{r}'$, we can rewrite $\mathbf{A_i}$ into their possible cases.

$$
\begin{aligned}
&\mathbf{A_i r}' > 0 : t \leq \frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}'} \\
&\mathbf{A_i r}' = 0 : \mathbf{A_i x} \leq c_i \\
&\mathbf{A_i r}' < 0 : \frac{c_i - \mathbf{A_i x}}{\mathbf{A_i r}'} \leq t
\end{aligned}
$$

We can exclude the case where all $\mathbf{A_i}r' = 0$, because $\mathfrak{A}$ would be unsatisfiable then. In the other two cases, the variable t is the only one not fixed. This is because c_i, $\mathbf{A_i}$, x, and $\mathbf{r'}$ are either constant or given. Therefore, we can derive numerical upper and lower limits for t. For $\neg\mathfrak{A}$ to be true, there is only one possible scenario: at least two of these bounds must conflict. Furthermore, this can only be the case when there exist two rows, $\mathbf{A_i}, \mathbf{A_j}$ with $\mathbf{A_j r'} > 0$ and $\mathbf{A_i r'} < 0$ with $\frac{c_j - \mathbf{A_j x}}{\mathbf{A_j r'}} < \frac{b_i - \mathbf{A_i x}}{\mathbf{A_i r'}}$ After rewriting, we obtain

$$(c_j - \mathbf{A_j x})\mathbf{A_i r} > (c_i - \mathbf{A_i x})\mathbf{A_j r}. \tag{3.5}$$

But now we arrive at a contradiction, as $\mathbf{x} \in \bigcap_{i,j} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$ then also $\mathbf{x} \in Sol_{\mathfrak{B}}(\mathfrak{A}[\mathrm{i,j}])$ since $i \neq j$ and $\mathfrak{B}(\mathbf{r'}) \wedge \mathbf{A_i r'} < 0 \wedge 0 < \mathbf{A_j r'}$ is satisfiable, it follows that

$$\mathbf{x} \in Combine_{\mathfrak{B}}(\mathbf{A_i}, \mathbf{A_j}, c_i, c_j, \mathbf{x}, \mathbf{r})$$

but then also

$$\mathbf{x} \in Sol_{\mathfrak{B} \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r}}((c_j - \mathbf{A_j x})\mathbf{A_i r} \leq (c_i - \mathbf{A_i x})\mathbf{A_j r}))$$

which immediately contradicts Equation 3.5. Therefore, the assumption $\mathbf{x} \notin Sol_{\mathfrak{B}}(\mathfrak{A})$ leads to a contradiction proving the claim $Sol_{\mathfrak{B}}(\mathfrak{A}) \supseteq \bigcap_{i,j} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$ and finally

$$Sol_{\mathfrak{B}}(\mathfrak{A}) = \bigcap_{i,j} Sol_{\mathfrak{B}}(\mathfrak{A}[i,j])$$

□

Notice that now the solution is described by the pairwise intersection over essentially $Combine_{\mathfrak{B}}$ operations. Which themselves represent the solution of halfspace constraints. Therefore, our algorithm indeed returns an LCS that corresponds to the correct solution.

3.4 Improved Algorithm

A problem with the current implementation is that we generate a lot of constraints. While it is possible to use algorithms that compute a minimal set of constraints that still have the same solution, these algorithms work best on a small set of constraints. Therefore, it is worth considering if we can lower the amount of generated constraints by avoiding some calculations. Currently, in the worst case, we generate for each pair of constraints up to k constraints, where k is the number of vertices defined by our rate restriction $\mathfrak{B}$. Since the intersection of n hyperplanes defines a vertex, we have, in the worst case $\binom{m+2}{n}$ possible vertices to try. The first place to look for a reduction in this amount is to consider if we need all of the k constraints for each pair of constraints in the goal. Consider the following simple example (see Figure 3.3):

$$r_x \in [1,2], \quad r_y \in [-2,2], \quad 4 \leq x, \quad 2 \leq y$$

Now following our algorithm we compute the following bounds: $\frac{-4+x}{-r_x} \leq t$ (since $r_x > 0$ for all cases) and the case distinction for r_y yields:

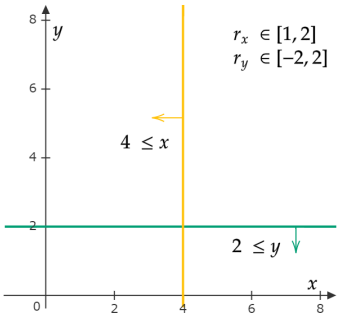


Figure 3.3: Visual representation of a two-dimensional example problem. Given are a set of two constraints and the corresponding rates of the dimensions.

$r_y > 0$ causes $\frac{-2+y}{-r_x} \leq t$,

$r_y = 0$ causes $2 \leq y$,

$r_y < 0$ causes $t \leq \frac{-2+y}{-r_y}$

We will focus on the $Combine_{\mathfrak{B}}$ part for now, hence we only consider the pair $\frac{-4+x}{-r_x} \leq t$ and $t \leq \frac{-2+y}{-r_y}$, where $r_y < 0$. After combining them, we obtain - according to Definition 15

$$\mathrm{Sol}_{\mathfrak{B} \wedge r_y < 0 \wedge 0 < r_x} \left(\frac{-r_y}{-r_x}(-4+x) \leq (-2+y), \mathbf{r} \right).$$

Now, according to Definition 13, we obtain

$$\mathrm{Sol} \left(\bigwedge_{\mathbf{v} \in \mathrm{vertices}(\mathfrak{B} \wedge r_y < 0 \wedge 0 < r_x)} \frac{-v_y}{-v_x}(-4+x) \leq (-2+y) \right).$$

Notice that this is not well defined, for the case, where $\mathbf{v_x} = 0$. However, we will handle such cases in a pre-computation, that we give as a further optimization step later. Therefore, for now, assume that $\mathbf{v_x} \neq 0$.

Computing $vertices(\mathfrak{B}(\mathbf{r}) \wedge r_y < 0 \wedge 0 < r_x)$ gives us the following vertices to substitute: $\{(1,0),(1,-2),(2,0),(2,-2)\}$. After substituting, we obtain the following set of constraints.

$$\frac{0}{-1}(-4+x) \leq -2+y$$
$$\frac{2}{-1}(-4+x) \leq -2+y$$
$$\frac{0}{-2}(-4+x) \leq -2+y$$
$$\frac{2}{-2}(-4+x) \leq -2+y$$

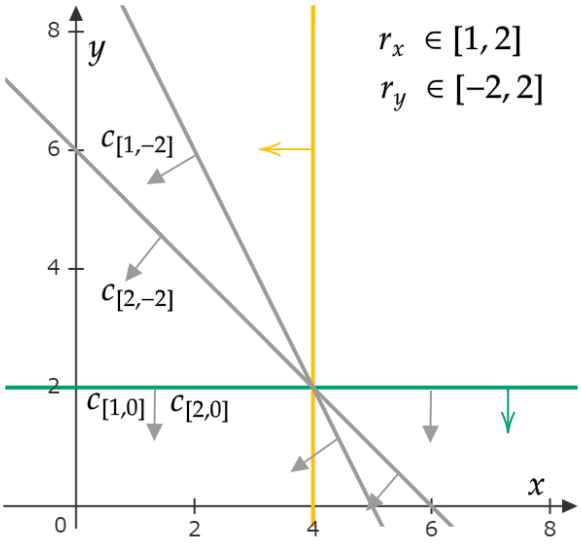


Figure 3.4: Extension of Figure 3.3 with the added constraints of the $Combine_{\mathfrak{B}}$ operation. The added constraints are denoted by $C_{[r_x,r_y]}$, where (r_x,r_y) is the corresponding vertex that generated the constraint.

After simplifying, we obtain the following:

$$\begin{aligned} 2 &\leq y \\ 10 &\leq 2x + y \\ 2 &\leq y \\ 6 &\leq x + y \end{aligned}$$

When closely inspecting the generated constraints (see Figure 3.4), we can notice an interesting fact. In the region of x,y, where $x \leq 4$ and $2 \leq y$, the constraint $10 \leq 2x + y$ implies all other constraints, i.e., we would only require this constraint in our solution - all other constraints are redundant. On the other hand, in the $4 \leq x, y \leq 2$ region, the $10 \leq 2x + y$ constraint is implied by all others, and one of the $2 \leq y$ constraints dominates the rest. Finally, we can make two more observations about the other two regions. In the $x \leq 4, y \leq 2$ region, every point is excluded by some constraint, while in the $4 \leq x, 2 \leq y$ region, every point is included in our solution (see Figure 3.5). However, when we also take into account the $0 \leq t$ constraint and combine it with $t \leq \frac{-2+y}{-r_y}$ from $r_y < 0$ we obtain $2 \leq y$, which is the dominating constraint in the $4 \leq x, 2 \leq y$ region. Therefore, we argue that the $Combine_{\mathfrak{B}}$ operation should only require to return one constraint - in our case, $10 \leq 2x + y$.

For the formal proofs, we will consider the slightly adapted $\text{Combine}_{\mathfrak{B}}$ operator.

$$Combine_{\mathfrak{B}}(\mathbf{A_i},\mathbf{A_j},c_i,c_j,\mathbf{x},\mathbf{r}) = \text{Sol}_{\mathfrak{B}(\mathbf{r})\wedge\mathbf{A_i r}<0\wedge 0<\mathbf{A_j r}}\left(\frac{\mathbf{A_i r}}{\mathbf{A_j r}}(c_j - \mathbf{A_j x}) \leq (c_i - \mathbf{A_i x}), \mathbf{r}\right)$$

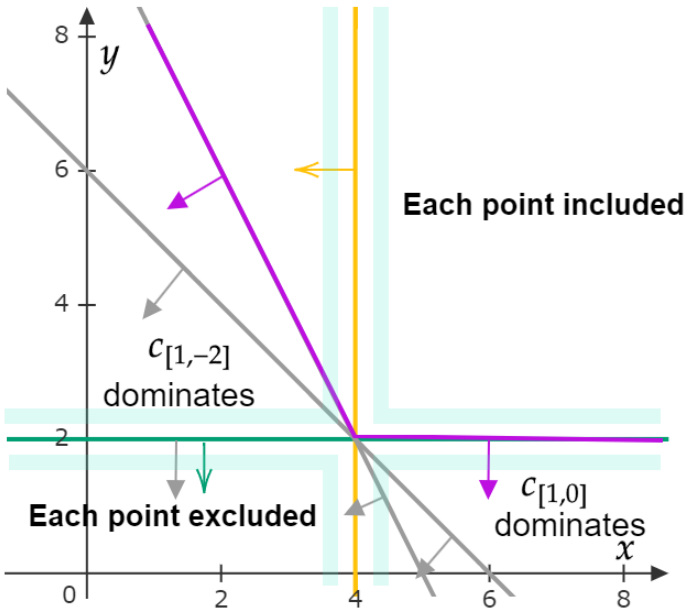


Figure 3.5: Visual representation of the $r_x \in [1,2], r_y \in [-2,2], 4 \leq x, 2 \leq y$ problem, highlighting the four specific regions. The dominating constraints are marked.

The following lemmas cover our observations about the four possible regions. Starting with the region where every point gets excluded.

Lemma 7. *Let* $\mathbf{x} \in \mathbb{R}^n$ *with* $c_i - \mathbf{A_i x} < 0 \wedge c_j - \mathbf{A_j x} < 0$.
Then $\mathbf{x} \notin Combine_{\mathfrak{B}}(\mathbf{A_i}, \mathbf{A_j}, c_i, c_j, \mathbf{x}, \mathbf{r})$

Proof. We consider the signs of individual terms in the constraint:

$$\underbrace{\frac{\overbrace{\mathbf{A_i r}}^{<0}}{\mathbf{A_j r}}}_{>0} \underbrace{(c_j - \mathbf{A_j x})}_{<0} \leq \underbrace{(c_i - \mathbf{A_i x})}_{<0}$$

Hence, the left side is positive, while the right is negative. Therefore, the formula is unsatisfiable - proving that $\mathbf{x}$ can not be a model of it. □

Next, we consider the region where every point is included.

Lemma 8. *Let* $\mathbf{x} \in \mathbb{R}^n$ *with* $c_i - \mathbf{A_i x} \geq 0 \wedge c_j - \mathbf{A_j x} \geq 0$.
Then $\mathbf{x} \in Combine_{\mathfrak{B}}(\mathbf{A_i}, \mathbf{A_j}, c_i, c_j, \mathbf{x}, \mathbf{r})$

Proof. Again, considering the signs of the factors:

$$\underbrace{\frac{\overbrace{\mathbf{A_i r}}^{<0}}{\mathbf{A_j r}}}_{>0} \underbrace{(c_j - \mathbf{A_j x})}_{\geq 0} \leq \underbrace{(c_i - \mathbf{A_i x})}_{\geq 0}$$

The left side is non-positive, while the right is non-negative. Hence, the formula is a tautology - proving that $\mathbf{x}$ is a model of it. □

Next, we will deal with the region where the constraint that gives an upper bound on t is already violated.

Lemma 9. *Let* $\mathbf{x} \in \mathbb{R}^n$ *with* $c_j - \mathbf{A_j x} < 0$ *and let* $\mathfrak{B}(\mathbf{r}) \wedge 0 \leq \mathbf{A_j r}$ *be satisfiable. Then* $\mathbf{x} \notin Sol_{\mathfrak{B}}(\mathfrak{A})$

Proof. We use a proof-by-contradiction. Assume $\mathbf{x} \in \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A})$ then also, using Theorem 2, $\mathbf{x} \in \bigcap_{i,j} \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[j,j])$ and $\mathbf{x} \in \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[j])$ and therefore, using Lemma 6, $\mathbf{x} \in Sol(\mathbf{A_j x} \leq c_j)$ since $\mathfrak{B}(\mathbf{r}) \wedge 0 \leq \mathbf{A_j r}$ is satisfiable. This contradicts $c_j - \mathbf{A_j x} < 0$. □

Finally, we prove that only one constraint is required. This constraint comes from the region where the lower bound on t is not satisfied, and the upper bound on t is satisfied. The generated constraint now represents the fact that we need to satisfy the lower bound before we no longer satisfy the upper bound. We show that one constraint is sufficient to cover this restriction.

Theorem 3. *Let* $i,j \in \{1,\ldots,m\}$ *with* $\mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} \leq 0$ *satisfiable then*

$$\begin{aligned} &Combine_{\mathfrak{B}}(\mathbf{A_i},\mathbf{A_j},c_i,c_j,\mathbf{x},\mathbf{r}) \\ &= Sol\left(\min_{\mathfrak{B}(\mathbf{v}) \wedge \mathbf{A_i v} < 0 \wedge 0 < \mathbf{A_j v})} \left[\frac{\mathbf{A_i v}}{\mathbf{A_j v}}\right](c_j - \mathbf{A_j x}) \leq (c_i - \mathbf{A_i x})\right) \cap Sol(\mathbf{A_j x} \leq c_j) \end{aligned}$$

Proof. We consider the signs of each of the terms in

$$Sol\left(\bigwedge_{\mathbf{v} \in \text{vertices}(\mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j})} (c_j - \mathbf{A_j x})\mathbf{A_i r} \leq (c_i - \mathbf{A_i x})\mathbf{A_j r}\right)$$

$$\underbrace{\frac{\overbrace{\mathbf{A_j r}}^{>0}}{\mathbf{A_i r}}}_{<0}(c_j - \mathbf{A_j x}) \leq (c_i - \mathbf{A_i x})$$

Lemma 9 states that we can exclude any points $\mathbf{x}$ where $c_j - \mathbf{A_j x} < 0$, since there, $\mathbf{x} \notin \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A})$.

$$\underbrace{\frac{\overbrace{\mathbf{A_i r}}^{<0}}{\mathbf{A_j r}}}_{>0} \underbrace{(c_j - \mathbf{A_j x})}_{\geq 0} \leq c_i - \mathbf{A_i x}$$

Since the left side of the inequality is now negative when $c_i - \mathbf{A_i x} \geq 0$ then the inequality becomes true for any $\mathbf{x}$. Hence, we can ignore this case, as the statement holds trivially for any such $\mathbf{x}$. Therefore, the only restrictive case is

$$\varphi(\mathbf{r}) := \underbrace{\frac{\overbrace{\mathbf{A_i r}}^{<0}}{\mathbf{A_j r}}}_{>0} \underbrace{(c_j - \mathbf{A_j x})}_{\geq 0} \leq \underbrace{(c_i - \mathbf{A_i x})}_{\leq 0}$$

Finally, since $\mathbf{r}$ is the only free variable, we can only influence the fraction $\frac{\mathbf{A_i r}}{\mathbf{A_j r}}$. For any $\mathbf{r_1}, \mathbf{r_2}$ $\frac{\mathbf{A_i r_1}}{\mathbf{A_j r_1}} \geq \frac{\mathbf{A_i r_2}}{\mathbf{A_j r_2}}$ implies that $\varphi(\mathbf{r_1}) \implies \varphi(\mathbf{r_2})$ (monotonicity). Since our domain $\mathfrak{B} \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r}$ is convex and bounded, the $\leq$ relation is well-founded hence the maximal element exists

$$\alpha_{i,j} := \max \left\{ \frac{\mathbf{A_i v}}{\mathbf{A_j v}} \mid \mathbf{v} \in \mathbb{R}^n \text{ with } \mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i v} < 0 \wedge 0 < \mathbf{A_j v} \right\}$$

and therefore

$$\alpha_{i,j}(c_j - \mathbf{A_j x}) \leq (c_i - \mathbf{A_i x})$$

is the least element of the implication chain.

Combining this with our previous assumption of $c_j - \mathbf{A_j x} \geq 0$ we obtain

$$\begin{aligned} &\mathrm{Sol}\left(\alpha_{i,j}(c_j - \mathbf{A_j x}) \leq (c_i - \mathbf{A_i x}) \wedge (\mathbf{A_j x} \leq c_j)\right) \\ &= \mathrm{Sol}\left(\alpha_{i,j}(c_j - \mathbf{A_j x}) \leq (c_i - \mathbf{A_i x})\right) \cap \mathrm{Sol}(\mathbf{A_j x} \leq c_j) \end{aligned}$$

□

This is a great result, as it allows us to reduce the number of constraints generated for each pair down to one instead of generating one for each of the vertices of $(\mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} < 0 \wedge 0 < \mathbf{A_j r})$. However, it is not clear how to actually compute $\max \left\{ \frac{\mathbf{A_i v}}{\mathbf{A_j v}} \mid \mathbf{v} \in \mathbb{R}^n \text{ with } \mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i v} < 0 \wedge 0 < \mathbf{A_j v} \right\}$. We can still enumerate all vertices but only add a constraint, where $\frac{\mathbf{A_i v}}{\mathbf{A_j v}}$ is maximized. While this does compute the correct result, the enumeration of all vertices, especially in larger dimensions, becomes inefficient.

This enumeration of vertices and finding a point that minimizes some objective function is eagerly similar to the use case of linear programming. However, our objective function can not be represented by a linear combination and is instead given as a fraction. If we could somehow transform it into a linear programming instance, we could use the powerful solvers of this field to find our solution quicker while possibly only visiting a fraction of the vertices. The Charnes-Cooper transformation we will present next gives such a transformation of fractional linear programs to ones with a linear objective function and allows us to use the linear solvers without much overhead.

3.4.1 Charnes-Cooper transformation

The Charnes-Cooper [CC62] transformation receives as input the following linear program.

$$\begin{aligned} \text{maximize} \quad & \frac{\mathbf{g}^T \mathbf{x} + \alpha}{\mathbf{h^T x} + \beta} \\ \text{subject to} \quad & \mathbf{D x} \leq \mathbf{d} \end{aligned}$$

Where $\mathbf{x} \in \mathbb{R}^n$, $\mathbf{g}$,$\mathbf{h} \in \mathbb{R}^n$, $\mathbf{D} \in \mathbb{R}^{m \times n}$, $\mathbf{d} \in \mathbb{R}^m$, α,$\beta \in \mathbb{R}$. Additionally, $\mathbf{Dx} \leq \mathbf{d}$ has to ensure that $\mathbf{h^T x} + \beta > 0$.

The transformation now gives us an equivalent linear program as follows:

$$\begin{aligned} \text{maximize} \quad & \mathbf{g}^T \mathbf{y} + \alpha z \\ \text{subject to} \quad & \mathbf{D y} \leq \mathbf{d} z \\ & \mathbf{h^T y} + \beta z = 1 \\ & 0 \leq z \end{aligned}$$

The main idea is to scale the denominator by some scalar factor until it becomes one. Then we can maximize the numerator. We can obtain back a solution for the original problem in $\mathbf{x}$ via the following equation $\mathbf{y} = \frac{\mathbf{x}}{\mathbf{h^T x}+\beta}$ and $z = \frac{1}{\mathbf{h^T x}+\beta}$. Reverting the scalar scaling.

Remember, the input for our problem is

$$\max \left\{ \frac{\mathbf{A_i v}}{\mathbf{A_j v}} \mid \mathbf{v} \in \mathbb{R}^n \text{ with } \mathfrak{B}(\mathbf{v}) \wedge \mathbf{A_i v} < 0 \wedge 0 < \mathbf{A_j v} \right\}. \tag{3.6}$$

This can be stated as the following linear-fractional program.

$$\begin{aligned} \text{maximize} \quad & \frac{\mathbf{A_i v}}{\mathbf{A_j v}} \\ \text{subject to} \quad & \mathbf{B v} \leq \mathbf{b} \\ & \mathbf{A_i v} < 0 \\ & \mathbf{A_j v} > 0 \end{aligned}$$

We can now apply the transformation to obtain

$$\begin{aligned} \text{maximize} \quad & \mathbf{A_i y} \\ \text{subject to} \quad & \mathbf{B y} \leq \mathbf{b} z \\ & \mathbf{A_j y} = 1 \\ & 0 \leq z \\ & \mathbf{A_i y} < 0 \\ & \mathbf{A_j y} > 0 \end{aligned}$$

Therefore, we have obtained a linear problem that returns the optimal solution. Since, in the worst case, the Simplex algorithm visits every vertex of the input problem, we will never enumerate more vertices than before. However, with modern optimization and heuristics, it is reasonable to assume that this worst case will not be reached, and instead, fewer steps are required. Since our original algorithm still evaluates every vertex, we can reasonably expect an increase in performance. This claim, however, still needs to be evaluated with respect to actual benchmarks.

3.4.2 Reducing the number of combinations

So far, our efforts have focused solely on reducing the number of generated constraints for each combination of a pair of constraints. Since we have reduced this number to exactly one, we have achieved the optimum regarding the number of constraints generated. However, there is still an improvement in the number of generated constraints to be made. A rather obvious observation is that $\mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[\mathrm{i,j}]) = \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[\mathrm{j,i}])$. Hence, when we compute $\mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}) = \bigcap_{i,j} \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[i,j])$, we only have to compute $\bigcap_{i \leq j} \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[i,j])$. However, the main way to reduce the number of combinations is to consider the cases where we obtain back the original goal constraint. If we can somehow show that we have already obtained both goal constraints for a given pair to be combined, then we do not need to combine them. We formulate this explicitly in the following lemma.

Lemma 10. *If* $\mathfrak{B}(\mathbf{r}) \wedge 0 \leq \mathbf{A_i r} \wedge 0 \leq \mathbf{A_j r}$ *is satisfiable, then*

$$Sol_{\mathfrak{B}}(\mathfrak{A}[i,j]) = Sol(\mathbf{A_i x} \leq c_i \wedge \mathbf{A_j x} \leq c_j)$$

Proof. Recall Theorem 1 and Lemma 6:

$$\mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A})[i,j] = \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[i]) \cap \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[j]) \cap \mathrm{Combine}_{\mathfrak{B},i,j} \cap \mathrm{Combine}_{\mathfrak{B},j,i}$$

$$\mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[i]) = \begin{cases} \mathrm{Sol}(\mathbf{A_i x} \leq c_i) & \text{if } \mathfrak{B}(\mathbf{r}) \wedge 0 \leq \mathbf{A_i r} \text{ satisfiable} \\ \mathbb{R}^n & \text{otherwise} \end{cases}$$

Since $\mathfrak{B}(\mathbf{r}) \wedge 0 \leq \mathbf{A_i r} \wedge 0 \leq \mathbf{A_j r}$ is satisfiable

$$\mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[i,j]) = \mathrm{Sol}(\mathbf{A_i x} \leq c_i) \cap \mathrm{Sol}(\mathbf{A_j x} \leq c_j) \cap \mathrm{Combine}_{\mathfrak{B},i,j} \cap \mathrm{Combine}_{\mathfrak{B},j,i}$$

It remains to be shown that the last two intersections do not restrict the result further. So far, $\mathbf{x} \in \mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[i,j]) \implies \mathbf{x} \in \mathrm{Sol}(\mathbf{A_i x} \leq c_i) \cap \mathrm{Sol}(\mathbf{A_j x} \leq c_j)$. Therefore, $c_i - \mathbf{A_i x} \geq 0 \wedge c_j - \mathbf{A_j x} \geq 0$. Applying Lemma 8 immediately yields $\mathbf{x} \in \mathrm{Combine}_{\mathfrak{B}}(\mathbf{A_j}, \mathbf{A_i}, c_j, c_i, \mathbf{x}, \mathbf{r})$. Due to symmetry reasons, this same argument holds for the second intersection. Hence proving the statement. □

This is actually a pretty decisive result, allowing us to reduce the number of combination computations drastically. The following table now shows when a computation of combine is actually required.

Combine	$\mathbf{A_j r} < 0$	$\mathbf{A_j r} \leq 0$	$\mathbf{A_j r} = 0$	$\mathbf{A_j r} \geq 0$	$\mathbf{A_j r} > 0$	no restriction
$\mathbf{A_i r} < 0$				✓ ×	✓ ×	✓×
$\mathbf{A_i r} \leq 0$				×	×	×
$\mathbf{A_i r} = 0$						×
$\mathbf{A_i r} \geq 0$	✓ ×	×				×
$\mathbf{A_i r} > 0$	✓ ×	×				×
no restriction	✓×	×	×	×	×	×

Table 3.1: The possible computations that induced a Combine-computation. × indicate pairs that previously required a computation, ✓ indicate the pairs that are actually required to be computed

We can define the i,j that encode the required combinations defined in Table 3.1 to be the valid solutions of

$$\phi(i,j) := (\forall \mathbf{r}.\mathfrak{B}(\mathbf{r}) \implies \mathbf{A_i r} < 0) \wedge (\exists \mathbf{r}'.\mathfrak{B}(\mathbf{r}') \wedge \mathbf{A_j r}' \geq 0).$$

To fully update our computation of $\mathrm{Sol}_{\mathfrak{B}}(\mathfrak{A}[i,j])$ we need to consider the computations of $\mathrm{Sol}(\mathbf{A_i x} \leq c_i)$ and $\mathrm{Sol}(\mathbf{A_j x} \leq c_j)$ too. However, since these values do not depend on each other, we can compute them once, in the beginning, to finally obtain.

3.4.3 Resulting improved algorithm

We will now summarize our previous improvements to the algorithm outlined in Section 3.2 into our final algorithm. We begin by iterating over the rows $\mathbf{A_i}$ of our goal constraint factors $\mathbf{A}$ and divide them into the three possibilities. We add $\mathbf{A_i}$ to the "lower" bounds, if for all $\mathbf{r}$ such that $\mathfrak{B}(\mathbf{r})$, $\mathbf{A_i r} < 0$ holds. If instead $\mathbf{A_i r} \geq 0$ for all such $\mathbf{r}$, then we add it to the "upper" bounds. Finally, we add it to the "both" bounds in all other cases. We then add for any $\mathbf{A_k}$ in the "upper" and "both" lists, the

constraint $\mathbf{A_k x} \leq c_k$ to our solution. Next, for any pair of $\mathbf{A_i}$ in the "lower" bounds and $\mathbf{A_j}$ in the "upper" or "both" bounds, we compute the result of the following LP.

$$\begin{aligned} \text{maximize} \quad & \mathbf{A_i y} \\ \text{subject to} \quad & \mathbf{B y} \leq \mathbf{b} z \\ & \mathbf{A_j y} = 1 \\ & 0 \leq z \\ & \mathbf{A_i y} < 0 \end{aligned}$$

It can be the case that this LP does not have any solution. This only happens, when $\mathbf{A_i y} < 0 \wedge \mathbf{A_j y} > 0$ is not satisfiable under $\mathbf{B y} \leq \mathbf{b} z$. Concretely this occurs, when $\mathbf{A_i r} < 0$ and $\mathbf{A_j r} > 0$ are individually satisfiable, but no $\mathbf{r}$ exists which satisfies both at the same time.

If the LP is unsatisfiable, we do nothing; otherwise, let α be the obtained optimum. We add the constraint $\alpha(c_j - \mathbf{A_j x}) \leq c_i - \mathbf{A_i x}$ to our solution. We can also rewrite this constraint in standard form to be

$$(-\alpha \mathbf{A_j} + \mathbf{A_i})\mathbf{x} \leq c_i - \alpha c_j. \tag{3.7}$$

Before adding this constraint, we can quickly check if $(-\alpha \mathbf{A_j} + \mathbf{A_i}) = 0$, since then, the constraint is trivially true. This is because we can not generate constraints that are always false due to *Lemma* 4. Finally, we return the conjunction of all the generated constraints as our solution.

Algorithm 2 Improved Version of our Algorithm (Solve LP Variant)

```
procedure SOLVE(B,b,A,c)
    lower,upper,both ← partitionRowsAccordingToSign(A,B,b)
    result = True
    for j ∈ upper ∪ both do
        result ← result ∧ A_j x ≤ c_j
    end for
    for i ∈ lower do
        for j ∈ upper ∪ both do
            maximum ← charnesCooper(A_i r / A_j r, Br ≤ b ∧ A_i r < 0 ∧ A_j r > 0)
            result ← result ∧ (−maximum A_j + A_i)x ≤ c_i − maximum c_j
        end for
    end for
    return result
end procedure
```

We can also only implement the reduction in number of combinations without the Charnes-Cooper transformation to update our vertex enumeration approach.

Algorithm 3 Improved Version of our Algorithm (Enumerate Vertices Variant)

procedure SOLVE($\mathbf{B}$,$\mathbf{b}$,$\mathbf{A}$,$\mathbf{c}$)
  $lower$,$upper$,$both \leftarrow partitionRowsAccordingToSign(\mathbf{A},\mathbf{B},\mathbf{b})$
  $result =$ **True**
  for $j \in upper \cup both$ **do**
    $result \leftarrow result \wedge \mathbf{A_j x} \leq c_j$
  end for
  for $i \in lower$ **do**
    for $j \in upper \cup both$ **do**
      $maximum \leftarrow \max(\left\{ \frac{\mathbf{A_i r}}{\mathbf{A_j r}} \mid \mathbf{r} \in \text{vertices}(\mathbf{Br} \leq \mathbf{b} \wedge \mathbf{A_i r} < 0 \wedge \mathbf{A_j r} > 0) \right\})$
      $result \leftarrow result \wedge (-maximum\ \mathbf{A_j} + \mathbf{A_i})\mathbf{x} \leq c_i - maximum\ c_j$
    end for
  end for
  return $result$
end procedure

lower	upper	both
$\mathbf{A_1}$	$\mathbf{A_4}$	$\mathbf{A_2}$
	$\mathbf{A_5}$	$\mathbf{A_3}$

Table 3.2: Partition the rows of our goal into the respective "lower," "upper," and "both" cases as defined earlier.

3.5 Example Computation

For our exemplary computation, we will use the example from the introduction. The problem is formally stated as

$$\psi = \forall \mathbf{r}.(\begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \mathbf{r} \leq \begin{bmatrix} 2 \\ -1 \\ 2 \\ 2 \end{bmatrix} \implies \exists t.0 \leq t \wedge \begin{bmatrix} -1 & 0 \\ 0 & 1 \\ -1 & 1 \\ 1 & 0 \\ 4 & -1 \end{bmatrix} (\mathbf{x} + t\mathbf{r}) \leq \begin{bmatrix} -4 \\ -2 \\ 6 \\ 6 \\ 18 \end{bmatrix}).$$

Since our rate restrictions are given as intervals, we can make use of interval arithmetic to divide our constraints into the respective categories. For this, consider the first row of the goal. We compute $-1r_x + 0r_y$ and can now substitute the intervals to obtain $-1[1,2]+0[-2,2] = [-2,-1]$. Since the interval contains only negative values $\mathbf{A_1 r} < 0$ holds. We, therefore, label this constraint as "lower." Repeating this for the other rows gives us Table 3.2.

According to our algorithm, we now add the original constraint for each of the rows from the "upper" and "both" categories to our solution. Therefore, our initial result is

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \\ 1 & 0 \\ 4 & -1 \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} -2 \\ 6 \\ 6 \\ 18 \end{bmatrix}.$$

As the last step, we need to combine each "lower" row with all of the "upper" and "both" rows. We will give the computations for the $\mathbf{A_1}$,$\mathbf{A_4}$ combination. We need to

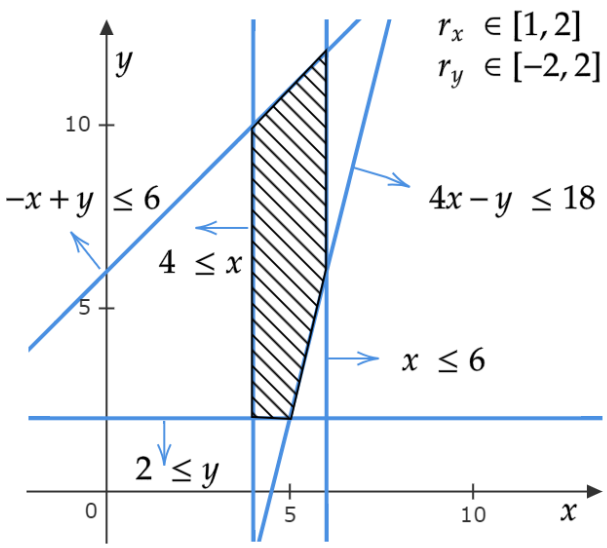


Figure 3.6: Visualization of the example problem. The half-spaces are colored, and their normal vector is shown. Additionally, the resulting goal is depicted as the shaded polytope. The dimensions are annotated with their corresponding rectangular rate intervals.

solve the following LP.

$$\begin{aligned} \text{maximize} \quad & -y_1 \\ \text{subject to} \quad & \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \mathbf{y} \leq \begin{bmatrix} 2 \\ -1 \\ 2 \\ 2 \end{bmatrix} z \\ & y_1 = 1 \\ & 0 \leq z \end{aligned}$$

Which gives the optimum $\alpha = -1$. We then substitute this value back to obtain the constraint $(-(-1)\begin{bmatrix}-1 & 0\end{bmatrix} + \begin{bmatrix}1 & 0\end{bmatrix})\mathbf{x} \leq 6 - (-1)(-4)$ which simplifies to be $0 \leq 2$. As outlined before, we do not add this constraint to our solution, as it is trivially true.

Next, we will combine $\mathbf{A_1}, \mathbf{A_5}$. We again solve an LP

$$\begin{aligned} \text{maximize} \quad & -y_1 \\ \text{subject to} \quad & \begin{bmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \end{bmatrix} \mathbf{y} \leq \begin{bmatrix} 2 \\ -1 \\ 2 \\ 2 \end{bmatrix} z \\ & 4y_1 - y_2 = 1 \\ & 0 \leq z \\ & -y_1 \leq 0 \\ & 4y_1 - y_2 \geq 0 \end{aligned}$$

Which gives the optimum $\alpha = \frac{-1}{6}$. Again substituting into Equation 3.7 gives us $(\frac{1}{6}(4x_1 - x_2) - x_1 \leq -4 + \frac{1}{6}18$ or simplified $\frac{-1}{3}x_1 + \frac{1}{6}x_2 \leq -1$. Finally, we also combine this lower bound with every "both" bound. In the end, we obtain the following result.

$$\begin{bmatrix} 0 & 1 \\ -1 & 1 \\ 1 & 0 \\ 4 & -1 \\ \frac{-1}{3} & \frac{-1}{6} \\ -1 & \frac{-1}{2} \\ -2 & 1 \end{bmatrix} \mathbf{x} \leq \begin{bmatrix} -2 \\ 6 \\ 6 \\ 18 \\ -1 \\ -5 \\ 2 \end{bmatrix}.$$

As we can see in Figure 3.7, most of these constraints are actually required. We only generate one redundant constraint. However, we expect the amount of redundant constraints to increase for more complex goals. Finally, Lemma 4 states that each point of the goal will also be a valid solution to our result. We can see this visually, as the blue-shaded area covers the striped goal.

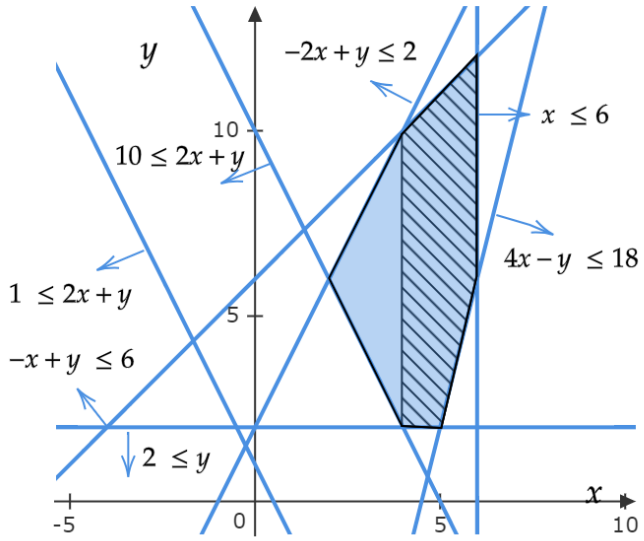


Figure 3.7: Visualization of the solution to the example problem. The goal is shaded in a striped pattern, while the resulting solution is shaded in blue. The constraints for the solution are given and labeled with the corresponding halfspace equation.

Chapter 4

Experimental Evaluation

4.1 Methodology

In this chapter, we will discuss how our algorithm was evaluated. We will demonstrate which benchmarks were used, and their advantages and disadvantages will be analyzed. Ideally, we would like to investigate the performance of our algorithm on a suite of standardized benchmarks and case studies. However, since this field is still developing, no suitable benchmarks exist for our fragment. Nevertheless, we must still rely on practical test cases to substantiate our claims about the algorithm's performance, capabilities, and limitations. Therefore, we developed an algorithm to generate benchmarks.

4.1.1 Benchmark generation

To generate our benchmarks, we will first define the dimension of the benchmark that we are interested in. This way, we can test how more complex problems affect the number of constraints generated and the algorithm's runtime. Since we want to test how more complicated goal representations affect these metrics, we would also like to influence the number of goal constraints. Our generation algorithm then works as follows. Given some dimension d and number of points n

1. Sample d rate-intervals, ensuring that at least one of these intervals is either strictly positive or negative.
2. Sample d points uniformly from $[0,100]^d$
3. Add up to $5 + 2d$ additional points from $[0,100]^d$
4. Compute the convex hull of the sampled points as an LCS

We hope to capture the whole range of possible problem inputs in this way. However, there are, of course, limitations to this approach. The advantages of this approach are:

- Fast generation
- Diversity in number of goal constraints

- Few assumptions about the problem input
- Reduction in the number of trivial cases
- Diverse mix of lower and upper bounds in the goal

The downsides are:

- No representation of highly complex goals
- No guarantee that the problem has a solution that differs from the goal
- Not founded in real-world problems - there might be aspects to actual case studies that are not represented in this approach

4.1.2 Benchmark Analysis

To ensure that our benchmarks are relevant to the problem, we will visualize some sampled problems and make statements about the whole benchmark suite. As can be seen in Figure 4.1, our benchmarks cover a broad range of goal constraints. Our benchmarks consist of many problems with few constraints, mainly in the lower dimensional problems. These help to detect any overheads in the algorithm. Additionally, more complex goals in higher dimensions allow for analysis of the algorithm's asymptotic behavior.

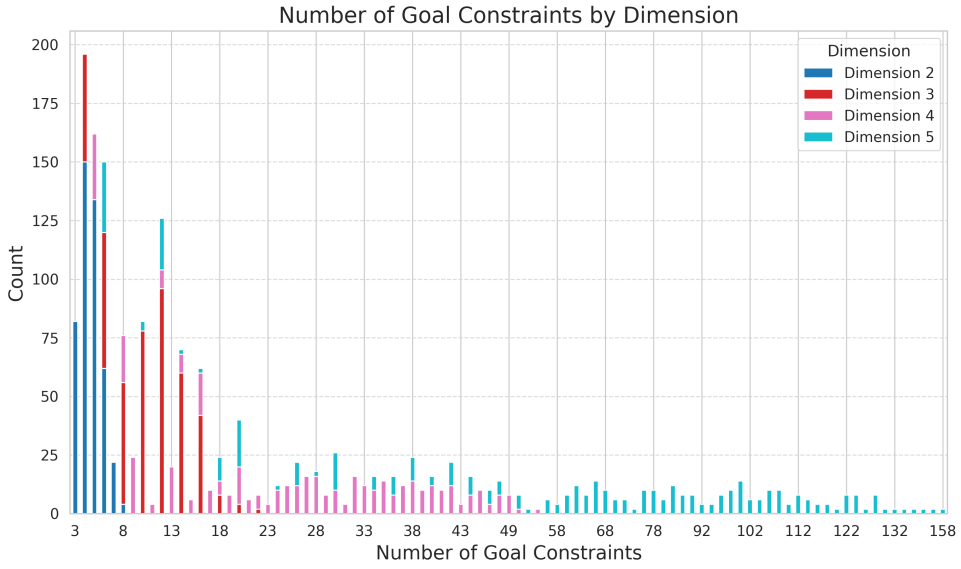
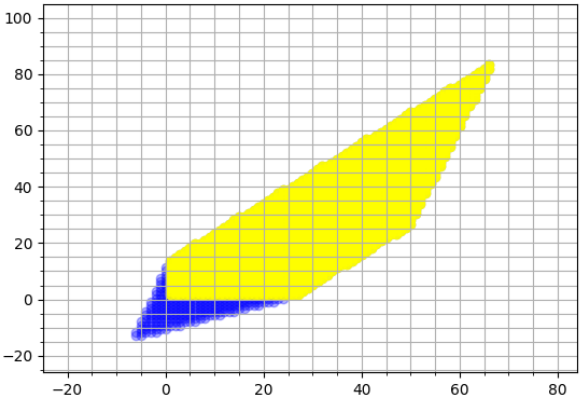


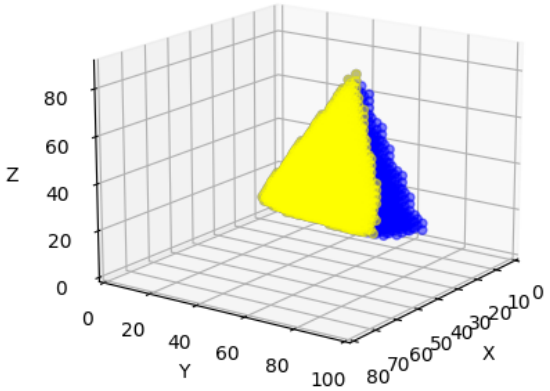
Figure 4.1: Stacked bar plot displaying the number of benchmarks with a given amount of goal constraints. Colored by the dimension of the problem.

The generated problems are non-trivial and instead pose interesting structures as for example shown in Figure 4.2.

Our set of benchmarks was generated for up to 5 dimensions, with a set of 250 problems generated for each dimension. All 1250 benchmarks were run sequentially on an Intel(R) Core(TM) i7-8565U CPU @ 1.80 GHz on our Python implementation. Additionally, no timeout was used. Hence, all benchmarks successfully terminated.



(a)



(b)

Figure 4.2: Presented are two problem instances selected from our set of benchmarks. The goal is highlighted in yellow, while the corresponding solution is depicted in blue. Panel a) presents a two-dimensional problem instance, whereas panel b) showcases a three-dimensional one. The actual constraints and rates required for computation are not shown.

4.2 Results

4.2.1 Presentation of Results

The key metrics that we recorded for each benchmark are

- Dimension of the problem
- Number of goal constraints
- Number of resulting constraints
- Runtime (s)

These metrics were chosen for their relevance to actual problem instances. Additional metrics such as peak memory usage, number of redundant constraints, and memory size of the result were considered but ultimately proved to be beyond the scope of this thesis.

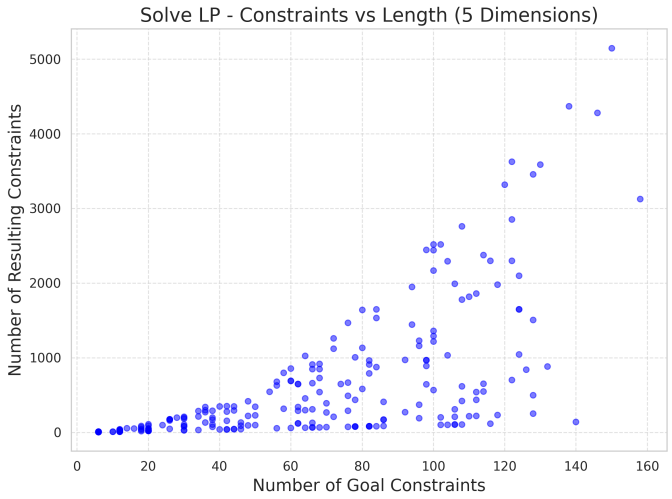


Figure 4.3: Scatter plot of the number of resulting constraints in the solution vs. the number of goal constraints of the problem for the 5-dimensional benchmarks. The plots for the lower dimension can be found in Figure A.1.

4.3 Discussion

4.3.1 Interpretation of Results

When interpreting the data of the number of generated constraints against the number of goal constraints (see Figure A.1), we notice that each previous dimension graph seems contained in the higher dimensional graphs. We therefore only consider the 5-dimensional case in this section. When considering Figure 4.3 we notice an explicit upper bound of roughly

$$\text{Number of Resulting Constraints} \lessapprox \frac{\text{Number of Goal Constraints}^2}{2}.$$

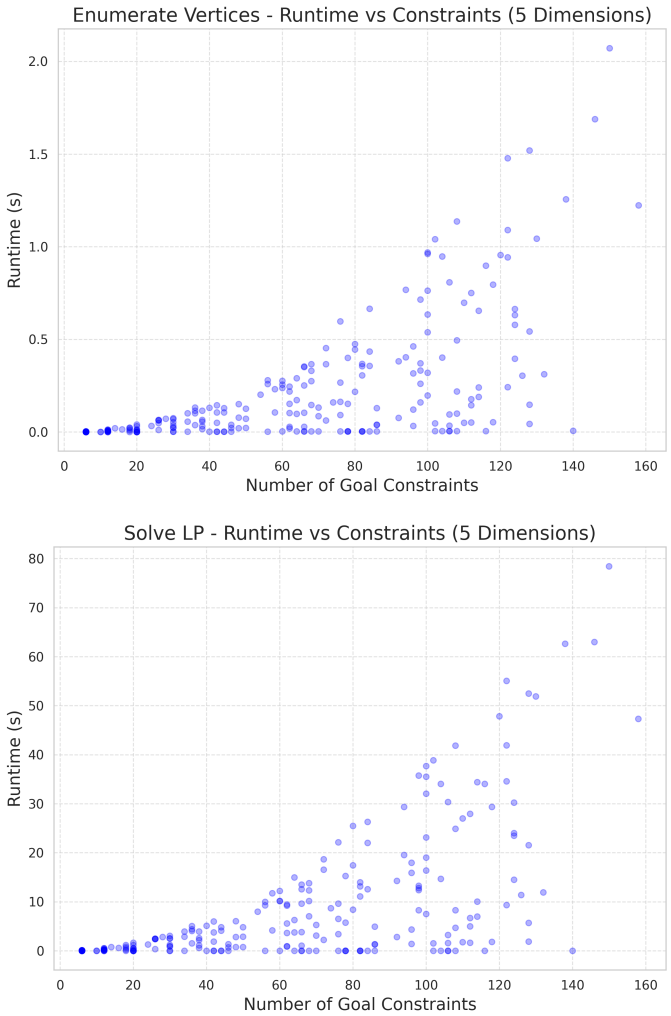


Figure 4.4: Comparison of the runtime of both the enumerate vertices and solve LP variants of our algorithm against the number of goal constraints in the 5-dimensional benchmarks. Apparent are the similar shapes of the graphs. However, the runtimes of the enumerate vertices variant seem to be lower by a considerable margin. The graphs for the lower dimension can be found in Figure A.2 and Figure A.3 for the enumerate vertices and solve LP variant respectively.

Additionally, we observe a distinct lower bound

$$\text{Number of Resulting Constraints} \gtrapprox \text{Number of Goal Constraints.}$$

Rather than observing different behavior for higher dimensions, our results suggest that the number of resulting constraints depends not on the problem's dimensionality but on the number of goal constraints. The graph of the runtime against the number of goal constraints appears similar to the previous findings for both versions of the algorithm. This suggests that the runtime and number of resulting constraints are tightly related. Moreover, the problem's dimensionality does not considerably impact the runtime. However, the runtime of the vertex enumeration seems to be much faster as compared to the variant where we solve the LP.

4.3.2 Analysis of Findings

To support the findings of the strict upper and lower bound, we proposed two polynomials $f(x) = x$, $g(x) = (\frac{x}{2})^2$ as the respective bounds. We can use a log-log plot to verify these claims and fit the respective functions. Since $log(f(x)) = log(x)$ for the lower bound, we expect a linear function of the form $f'(x') = x'$ in the log-log plot. For the upper bound $\frac{x}{2}^2$ we expect $log(g(x)) = log(\frac{x}{2}^2) = log(x^2) + log(1/4) = 2log(x) - log(4)$. Hence, we expect a linear function of the form $g'(x') = 2x - log(4)$ in the log-log plot. These results are depicted in Figure 4.5. We only provide one such graph, as both algorithm variants compute the same result. In the graph, the lower bound is a perfect fit; the upper bound, on the other hand, only approximates the real upper bound for a higher number of goal constraints. We can relate these findings to our algorithm's complexity as follows. Reminder: the result we are computing is given as.

$$\begin{aligned}\text{Sol}_{\mathfrak{B}}(\mathfrak{A}) = &\bigcap_{i \text{ s.t. } \exists \mathbf{r}.\mathfrak{B}(\mathbf{r}) \wedge \mathbf{A_i r} \geq 0} \text{Sol}(\mathbf{A_i x} \leq c_i) \quad \cap \\ &\bigcap_{i,j \text{ s.t. } \phi(i,j)} \text{Sol}\left(\alpha_{i,j}(c_j - \mathbf{A_j x}) \leq (c_i - \mathbf{A_i x})\right)\end{aligned}$$

Closely inspecting the formula, we can observe that we obtain precisely n_u constraints for the first intersection, where u is the number of upper bounds on t that we can obtain. We will compute n_{l*u} constraints for the second intersection, where l is the number of lower bounds on t. Since every constraint is either a lower bound, an upper bound, or both a lower and upper bound on t, we will obtain at least $u+l$ constraints.

For the upper bound, we can give a similar argument. We want to maximize $n_u + n_{l*u}$ with $l + u = g$ where g is the number of goal constraints. Therefore, we maximize $u(l + 1)$ with respect to $l + u = g$. This maximum is obtained for $l = \lfloor \frac{g-1}{2} \rfloor, u = l+1$ leading to the maximum of $\lfloor \frac{g-1}{2} \rfloor (\lfloor \frac{g-1}{2} \rfloor + 1)$ which approximates $(\frac{g}{2})^2$ for larger g. This explains why our linear bound for the maximum only approximates the actual maximum for goals with more constraints. This approximation is suitable since we are primarily concerned with the algorithm's asymptotic behavior.

When investigating the runtimes of our two variants, we can observe that they behave similarly. However, the runtime of the vertex enumeration seems to be significantly shorter compared to the linear programming variant.

We can see this trend when compared to the dimension in Figure 4.6 and when plotting against the number of resulting constraints in Figure 4.7. This is a surprising

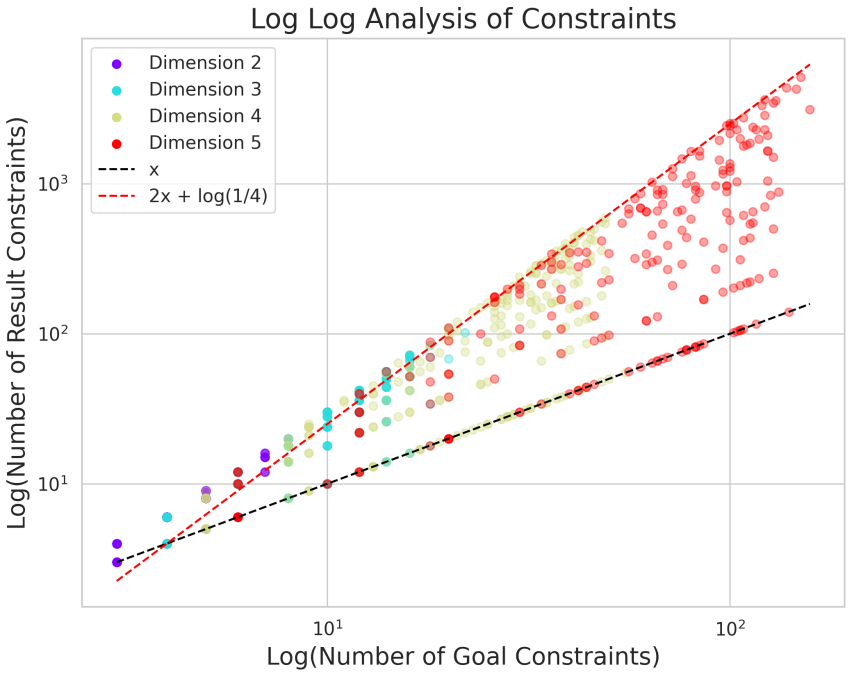


Figure 4.5: Log-log plot of the resulting constraints of goal constraints against the number of goal constraints. Additionally, our two hypotheses of the upper and lower bound as their respective transformed linear functions are plotted. The samples are colored by the dimensionality of the benchmark that the sample was computed from.

result, as we expected that solving the LP is quicker than iterating over all vertices. Since, for the vertex iteration, we first need to compute all the vertices in the first place. We believe the LP approach provides a significant overhead, slowing down the computations. Additionally, the vertex enumeration might be more optimized for parallel computing. Further research is needed to verify if this is an anomaly for our benchmarks or if the LP transformation is too costly to provide any benefit.

Lastly, we want to substantiate the claim that the runtime mostly depends on the number of resulting constraints and not on the problem's dimensionality. We will only investigate the quicker algorithm using the vertex enumeration. We can plot the runtime against the number of resulting constraints for this. We expect a mostly linear correlation between these points. We can see a clear linear relationship between the two measurements in Figure 4.8. Based on this, we can conclude that the vertex enumeration, while costly in higher dimensions, does not influence the runtime as drastically as the number of goal constraints does. As a result, the runtime of our algorithm is dominated by the number of resulting constraints, which is determined solely by the number of goal constraints. We investigated the apparent skewed "V" shape but could not find any particular reason for it. The shape could either be caused by random noise in the data, a lack of representation of certain types of problems (bias) in the benchmarks, or a lack of data for the highly complex problems.

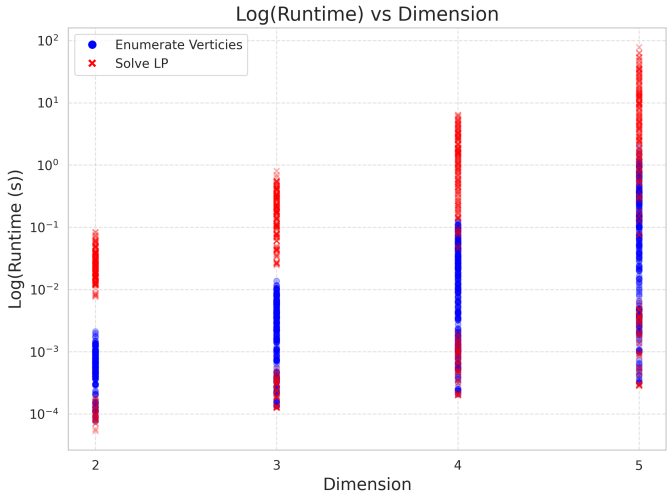


Figure 4.6: Log runtime of both variants plotted against the dimension of the benchmarks. The solve LP variant consistently displays are slower runtime. Only for some small subset of the benchmarks do both variants perform similarly and terminate after a fraction of a second

4.3.3 Limitations and Challenges

We are currently limited to a relatively small dimension of the problems. This is, however, somewhat an indirect consequence of the fact that goals of higher dimensions tend to require more constraints. Therefore, if a smaller set of constraints still describes the problem, we should be able to handle large domains. This is particularly of interest when considering hybrid automata. The problems there are usually highly dimensional because of the introduction of random clocks. Each clock introduces a new dimension to the problem. However, since these clocks are either running or stopped, they tend not to influence the goal by adding new constraints but rather by adapting them. Therefore, we should be able to handle many clocks if the goals do not become too complex. A particular challenge of this research was finding suitable benchmarks to test all these claims on. Without substantiating these claims, we can only speculate on the performance in the use-case of actual hybrid stochastic automata. Indeed, the only claims we can definitely make are about the upper bound on the constraints generated and the correlation between the number of resulting constraints and the runtime.

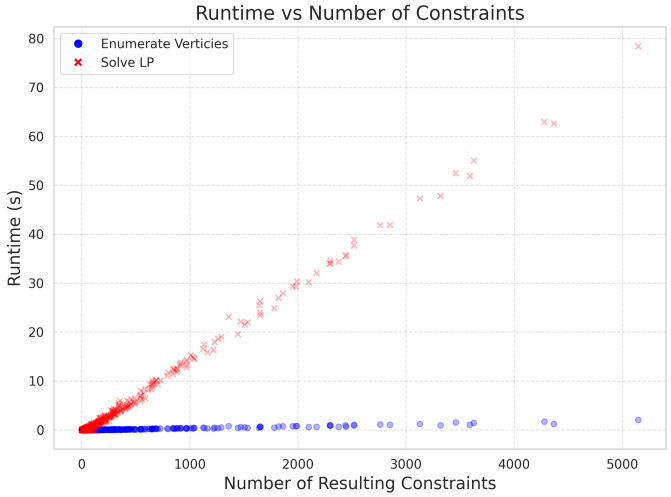


Figure 4.7: Runtime in seconds against the number of resulting constraints for both variants of our algorithm. Both variants appear to scale linearly with the number of resulting constraints. Tthe runtime of the solve LP approach indicates a slower performance.

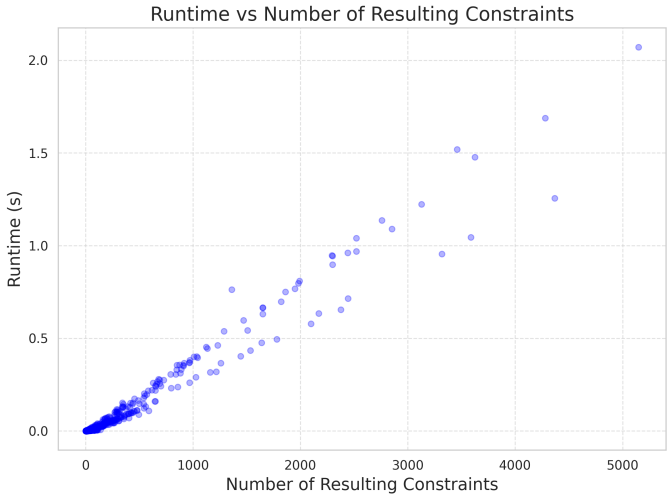


Figure 4.8: Plot of the runtime of each benchmark against the number of resulting constraints for the vertex enumeration variant. We can observe a linear trend that appears to become more noisy for a larger number of resulting constraints. Interesting to note is the apparent lack of points between two mostly linear bands.

Chapter 5

Conclusion

5.1 Summary of Contributions

We have developed a sound quantifier elimination algorithm to solve the fragment we are interested in. Additionally, we proved a lower and upper bound on the number of constraints generated. These serve as a baseline to further improve the algorithm or develop new approaches. To evaluate the performance of our algorithm, we have provided a basic set of benchmarks and outlined their strengths and weaknesses. Finally, our source code is integrated into the HyPro tool [S22] and evaluated on the benchmarks.

5.2 Implications of Findings

Our research has yielded significant findings in the field of rectangular stochastic automata and their probabilistic minimization. Developing an algorithm to solve this sub-problem has laid the necessary groundwork for further analysis and exploration of minimal probabilities. By developing a suite of benchmarks, we are able to make claims about the performance on real applications of the algorithm.

5.3 Suggestions for Future Work

As mentioned, this algorithm is used as a sub-procedure to compute the minimal reachability probabilities in rectangular automata with random clocks. A second algorithm is being developed that will then make use of our algorithm. This idea was developed as the opposite (minimizing) approach to the one presented in "Maximizing Reachability Probabilities in Rectangular Automata with Random Clocks" [DSÁR23]. We suggest the following areas for future work.

Investigate the vertex enumeration vs. solving the LP By providing both implementations of our algorithm, we could test both approaches' performance. Surprising to us, the vertex enumeration was the faster method. Future work should investigate whether this holds in general or whether the set of benchmarks, the low dimensionality, or our implementation led to this result. Additionally, incremental SMT solvers could limit the overhead of solving the LP.

Since our problem will always contain the base rate constraints $\mathfrak{B}$ with two additional constraints, we might be able to leverage the incremental approach to keep the original constraints while only adding two new constraints and an adapted optimization function. This way, we might be able to avoid the overhead of adding the individual rate constraints for each combination.

Develop case studies to evaluate the performance Since no standardized benchmarks or case studies exist that are suitable for testing the performance of our algorithm, we had to resort to developing our own suite of benchmarks. This comes with a disadvantage, as we have to make assumptions about the structure of real-world applications. Once the minimizing algorithm is finished, we can adapt the case studies from the maximizing algorithm [DSÁR23] to generate a case study for our algorithm. Of course, developing a new case study directly applicable to the minimization would be desirable. Since our algorithm is only used as a sub-routine, we require the whole minimization algorithm in order to adapt the case study. To generate benchmarks from the case study, we will then record each call to our algorithm as a single benchmark.

Reduction in number of generated constraints We currently generate a significant amount of redundant constraints. As we have demonstrated, more complex goals tend to lead to more complex results. We would prefer a minimal representation since we want to apply our algorithm successively, i.e., use the previous result as a new goal. Our current approach for this is to apply a second algorithm that can eliminate redundant constraints to our result. However, it might be possible to use insights from such algorithms to adapt our algorithm to further reduce the number of generated constraints in the first place.

Possibly eliminate the linear optimization problem During our research, we went from enumerating all constraints to enumerating all vertices and generating one constraint to finally solving a small linear optimization problem to find the correct constraint. While we have shown for lower dimensions that the linear optimization problems can indeed be solved efficiently and tend not to have a major influence on the runtime, for larger problems, this might no longer hold. It might be possible to eliminate the linear optimization problem as a whole and instead find some closed-form solution for the minimal values we are interested in. The main motivation why we believe this to be possible is that each combination of two constraints, no matter the number of dimensions, can be projected onto a simple two-dimensional problem, where a closed-form solution might be more simple to find.

Bibliography

[ACH+95] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[CÁ11] Florian Corzilius and Erika Ábrahám. Virtual substitution for SMT-solving. In *Fundamentals of Computation Theory*, pages 360–371. Springer Berlin Heidelberg, 2011.

[CABls] G A Pérez Castañeda, J-F Aubry, and N Brinzei. Stochastic hybrid automata model for dynamic reliability assessment. *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, 225(1):28–41, 2011. Sage Journals.

[CC62] A. Charnes and W. W. Cooper. Programming with linear fractional functionals. *Naval Research Logistics Quarterly*, 9(3-4):181–186, 1962.

[Dan90] George B. Dantzig. Origins of the simplex method. In *A history of scientific computing*, pages 141–151. Association for Computing Machinery, 1990.

[DSÁR23] Joanna Delicaris, Stefan Schupp, Erika Ábrahám, and Anne Remke. Maximizing reachability probabilities in rectangular automata with random clocks. In *Theoretical Aspects of Software Engineering*, pages 164–182, Cham, 2023. Springer Nature Switzerland.

[Fou27] Joseph Fourier. Analyse des travaux de l'académie royale des sciences pendant l'année 1824, partie mathématique, 1827. engl. transl. (partially) in: D.A. Kohler, translation of a report by Fourier on his work on linear inequalities. 1827.

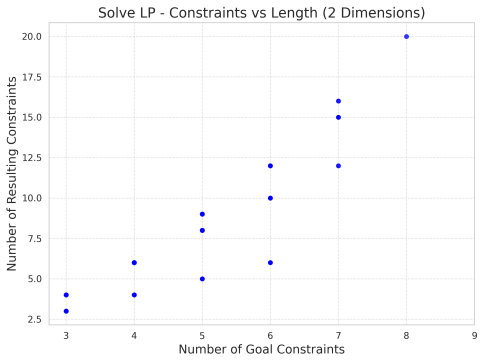
[HKD98] George Hassapis, Isabella Kotini, and Zoe Doulgeri. Validation of a SFC software specification by using hybrid automata. *IFAC Proceedings Volumes*, 31(15):107–112, 1998. Proceedings of the 9th IFAC Symposium on Information Control in Manufacturing (INCOM '98) 1998. Elsevier Science.

[PSR21] Carina Pilch, Stefan Schupp, and Anne Remke. Optimizing reachability probabilities for a restricted class of stochastic hybrid automata via flowpipe-construction. In *Quantitative Evaluation of Systems*, pages 435–456. Springer International Publishing, 2021.

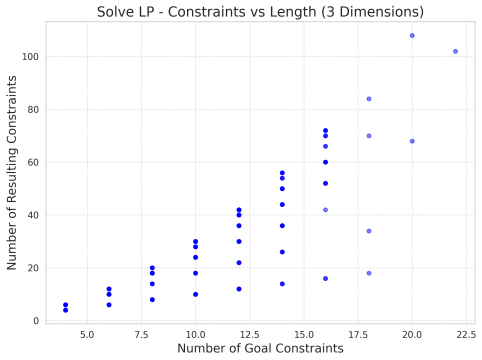
[S22] Stefan Schupp, Erika Ábrahám, and Tristan Ebert. Recent developments in theory and tool support for hybrid systems verification with HyPro. *Information and Computation*, 289:104945, 2022.

Appendix A

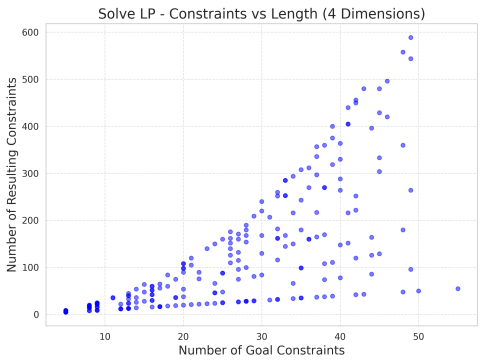
Graphs



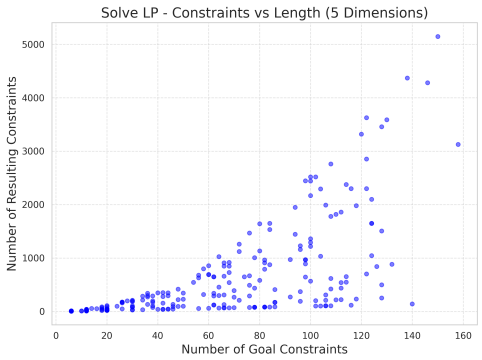
(a)



(b)

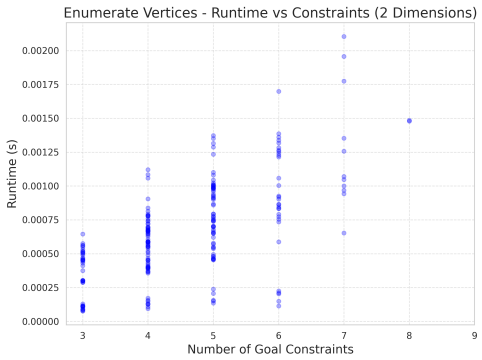


(c)

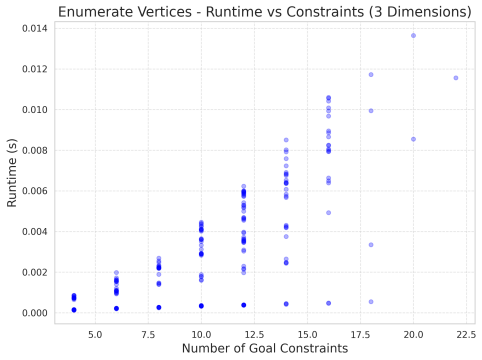


(d)

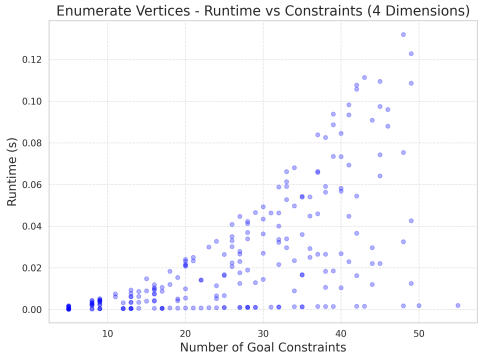
Figure A.1: The graphs show the resulting constraints for a given number of goal constraints. They illustrate the 2-dimensional problems (a), 3-dimensional problems (b), 4-dimensional problems (c), and 5-dimensional problems (d).



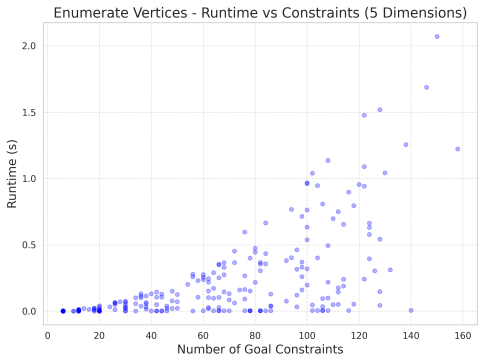
(a)



(b)

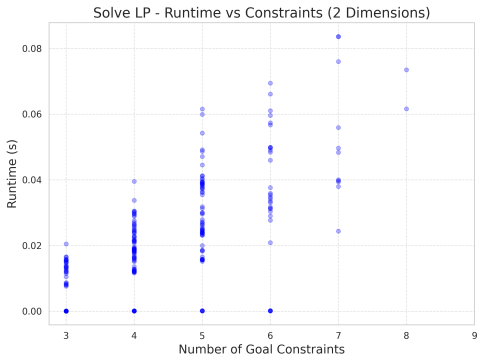


(c)

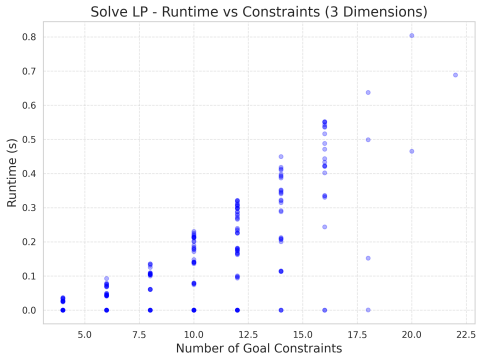


(d)

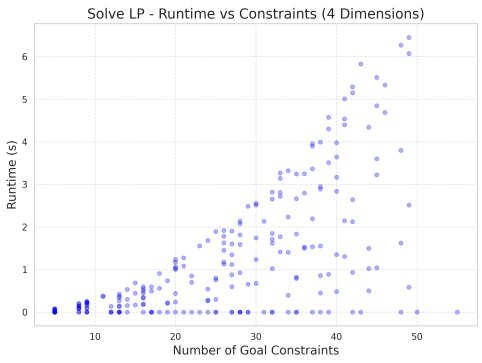
Figure A.2: The graphs show the runtime for a given number of goal constraints in the enumerate vertices approach. They illustrate the 2-dimensional problems (a), 3-dimensional problems (b), 4-dimensional problems (c), and 5-dimensional problems (d).



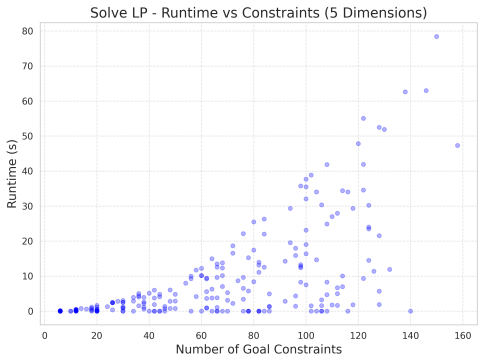
(a)



(b)



(c)



(d)

Figure A.3: The graphs show the runtime for a given number of goal constraints when applying the LP-solving approach. They illustrate the 2-dimensional problems (a), 3-dimensional problems (b), 4-dimensional problems (c), and 5-dimensional problems (d).