Bachelor thesis
Computer Science

Radboud University

Reachability properties for uncertain MDPs

Author:
Marnix Suilen
s4443772

First supervisor/assessor:
Dr. Nils Jansen
`n.jansen@science.ru.nl`

Second assessor:
Prof. Dr. Mariëlle Stoelinga
`m.stoelinga@cs.ru.nl`

April 17, 2018

Abstract

We study uncertain Markov decision processes with interval uncertainty. The problem is to find a scheduler that satisfies both a reachability property and an expected cost property. We a propose robust geometric program (robust GP) to find this scheduler. Solutions to this robust geometric program are proven to be correct but not complete: if the solution to the robust GP defines a valid scheduler, it is a solution to the problem, but getting a valid scheduler is not guaranteed. Because robust geometric programming is PSPACE-hard, we employ an approximation method that results in a robust linear program (robust LP). A solution to this robust LP is also a solution to the robust GP. Robust LPs can be solved efficiently depending on the uncertainty set that is used. What specific kind the uncertainty set in the robust GP (and as a consequence in the robust LP) is, remains open.

Contents

1 Introduction **3**

1.1 Contributions 4

1.2 Structure of the thesis 5

2 Related work **6**

3 Preliminaries **7**

4 Problem Statement **11**

4.1 A robust nonlinear programming encoding of uncertain Markov decision processes 11

4.2 Robust geometric programming 13

4.3 Correctness of the robust GP 16

4.4 Counter example to the completeness of the robust GP 20

4.5 Solving the robust GP 23

5 Convex reformulation of the robust GP **24**

5.1 General transformation method 24

5.2 Applying the transformation method 27

5.2.1 Resulting convex form robust GP 33

6 Approximating the robust GP by a robust LP **35**

6.1 Simplifications to the robust GP 35

6.2 Rewriting a k-term lse-function as k-1 2-term lse-functions . . 36

6.3 Approximating a 2-term lse-function by a piecewise-linear function 41

6.3.1 Python implementation of the pwl-approximation method 41

6.4 Choosing between the upper and lower approximation 42

6.5 Robust linear programming 44

6.6 Open problem: the uncertainty set $\mathcal{P}$ 45

7 Conclusions **47**

7.1 Future work 47

A Details on the counter example **51**

B Details on the robust GP reformulation **52**

C Details on the lse-function approximation **54**

Chapter 1

Introduction

Imagine a robot in a room with other moving objects. We want it to reach the other end of the room without bumping into anything, or more realistically: with the probability of bumping into something smaller than a percentage of our choice. Our robot is, of course, smart. We don't tell it how the other objects move. Nor do we tell it how it should move itself through the room. We want it to find its own way, within the limits of its the battery power. The general idea behind this is illustrated in figure 1.[1]

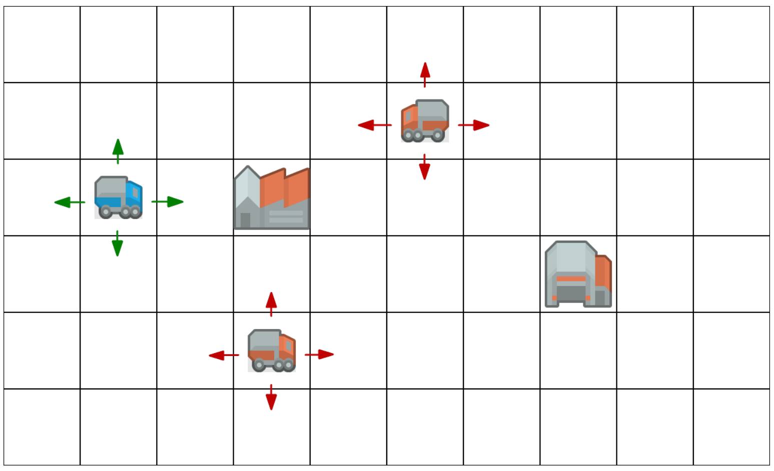


Figure 1: Illustration of the problem

The blue truck is our robot. The red ones are other moving objects. The buildings don't move of course, but our robot doesn't know that. With some form of machine learning, it can learn about the movements of other objects, and translate those into probabilities and confidence bounds that tell whether a position is free or not.

[1]Images from `https://kenney.nl/assets/sci-fi-rts` under CC0 1.0 license, `https://creativecommons.org/publicdomain/zero/1.0/`

Using reinforcement learning, an optimal strategy for this scenario and the underlying model can be determined [23]. It can be modeled using an *uncertain Markov Decision Process* (uMDP), where the uncertainty is given by intervals, and a *reachability property* for which we want to find a *scheduler*. This is a realistic model: as our robot is self-learning, it will determine the likelihood with which an event occurs together with a confidence bound. This can be translated to intervals. At first, the intervals will be large, as its confidence is low. The longer the robot spends learning about its environment, the higher the confidence becomes and as a result the smaller the intervals become.

Verifying whether a reachability property holds can be done with *model checking* [3]. Model checking can be used to formally verify that in a model the probability of reaching a certain state is less than a predefined percentage. That way we can verify whether the model behaves as intended and, if the model properly models the object, we can conclude whether the object behaves as intended. In our case, the problem of verifying a reachability property lies in the uncertainty of the uMDP.

To that end, we want to find a valid scheduler such that the probability of reaching the *target set* (a set of 'bad' states we want to avoid) is smaller than a given value, and that the *expected cost* of reaching the *goal set* is as small as possible. This problem can be formulated as a *robust nonlinear program*. Nonlinear programming in general is hard to solve. Even small NLPs can be extremely challenging [6]. However, a subset of optimization problems, where all functions in the optimization problem are *convex functions*, like in *linear programming* or after a transformation in *geometric programming* can be solved efficiently, [8, 11, 1, 13]. These problems are called *convex optimization problems*.

Robust optimization problems are optimization problems that account for *uncertainty* in the problem data. This is typically done through an uncertainty set. Depending on the optimization problem and the uncertainty set, a robust optimization problem can be solved efficiently, like robust linear programming with box or polyhedral uncertainty [4].

We reduce the robust nonlinear programming formulation of our problem to a robust geometric program to make it feasible. A robust geometric program can be approximated by a robust linear program [20], which can then be solved efficiently depending on which type of uncertainty set is used [4, 26].

1.1 Contributions

This thesis provides the following key results:

- a robust geometric programming encoding of the problem and a correctness proof,

- a counter example that shows that the robust GP encoding is not complete,
- a convex reformulation of the robust GP,
- a Python implementation of the algorithm that calculates both the upper and lower r-term piecewise-linear approximation of the 2-term lse-function,
- a robust LP that approximates the robust GP.

1.2 Structure of the thesis

In chapter 4 we give a formal problem statement and propose a solution through *robust geometric programming.* We show that this method gives us solutions that are *correct* (section 4.3), and show by counter example that this method is not *complete* (section 4.4). Then, in chapter 5, we show how to transform the *posynomial form* robust geometric program into a *convex form* robust geometric program. In chapter 6 we solve the robust GP by transforming it into a robust linear program. In order to solve the robust LP, we need to determine what kind of uncertainty set is used. This remains an open problem. A solution to this robust LP can then be transformed back into a solution satisfying the robust GP, and satisfies for *all* possible instantiations of the uncertainty in the uMDP. We present our conclusions in chapter 7.

Chapter 2

Related work

Verifying reachability is not new. This and similar problems have been examined for various kinds of Markov Decision Processes, like bounded-parameter MDPs [25, 17] or parametric MDPs [16, 10]. Different approaches to solving these problems exist, like the use of machine learning [7], or statistical methods [18]. Convex optimization has proven successful in the verification of parametric MDPs [10]. Dedicated model checkers that can verify these properties for certain types of MDPs exist [15, 21].

For uncertain MDPs, this problem is as of yet unsolved, though for the specific case where the uncertainty is given by intervals, the problem has been analyzed before [14]. Different problems for uMDPs have been examined and solved before. A method for finding a robust control policy for uMDPs with temporal logic specifications exists [24]. And for uncertain MDPs where the uncertainty is given by convex sets, verification of PCTL properties can be done in polynomial time [22].

Chapter 3

Preliminaries

We define $\mathbb{R}_+ = \{r \in \mathbb{R} \mid r \geq 0\}$ and $\mathbb{R}_{++} = \{r \in \mathbb{R} \mid r > 0\}$. For a finite set A we write $|A|$ for the number of elements in A. A probability distribution over a finite set A is a function $\mu\colon A \to \mathbb{R}$ with $\sum_{a \in A} \mu(a) = 1$. $\mathit{Distr}(A)$ is the set of all probability distributions over A. Vectors and matrices are written in bold, like $\mathbf{x} \in \mathbb{R}^n$, and for a vector $\mathbf{x}$ we write x_i for the i-th element in the vector.

Definition 1 (Monomial)**.** *A monomial is a function $f\colon \mathbb{R}^n_{++} \to \mathbb{R}$ as*

$$f(\mathbf{x}) = d \prod_{j=1}^{n} x_j^{a_j}$$

where $d \geq 0$ and $a_j \in \mathbb{R}$.

Definition 2 (Posynomial)**.** *A posynomial is the sum of a number of monomials:*

$$f(\mathbf{x}) = \sum_{k=1}^{K} d_k \prod_{j=1}^{n} x_j^{a_j k}.$$

Definition 3 (Markov decision process (MDP))**.** *Let $\mathcal{M} = (S, \mathit{Act}, s_I, \mathcal{P})$ be a tuple where S is a finite set of states, Act a finite set of actions, s_I an initial state ($s_I \in S$), and $\mathcal{P}$ a transition function defined as $\mathcal{P}\colon S \times \mathit{Act} \times S \to [0,1]$. $\mathcal{M}$ is a Markov decision process (MDP) if the transition function $\mathcal{P}$ is a valid probability distribution: $\sum_{s' \in S} \mathcal{P}(s, \alpha, s') = 1$ for all $s \in S$ and $\alpha \in \mathit{Act}$.*

For a MDP, we can also define a cost function $c\colon S \times \mathit{Act} \to \mathbb{R}$ that assigns a value in $\mathbb{R}$ (cost) to each act going out of state s.

Definition 4 (Scheduler)**.** *A scheduler for an MDP $\mathcal{M}$ is a function $\sigma\colon S \to \mathit{Distr}(\mathit{Act})$ such that $\sigma(s)(\alpha) > 0$ implies $\alpha \in \mathit{Act}(s)$. $\mathit{Sched}^{\mathcal{M}}$ is the set of all schedulers over $\mathcal{M}$.*

So a scheduler assigns probabilities to each action going out of a state s. That way, after applying a scheduler, the full probability for each transition (from a state s over an action α) is known. Applying a scheduler to a MDP yields an induced Markov chain:

Definition 5 (Induced Markov chain)**.** *Given MDP $\mathcal{M} = (S, Act, s_I, \mathcal{P})$ and scheduler $\sigma \in Sched^{\mathcal{M}}$, the Markov chain induced by $\mathcal{M}$ and σ is $\mathcal{M}^\sigma = (S, Act, s_I, \mathcal{P}^\sigma)$, where for all $s, s' \in S$*

$$\mathcal{P}^\sigma(s, s') = \sum_{\alpha \in Act} \sigma(s)(\alpha) \cdot \mathcal{P}(s, \alpha, s').$$

We write $\Pr(\mathcal{M}^\sigma, \lozenge B, s)$ for the probability of reaching a state in the set $B \subseteq S$ from state s in the MDP $\mathcal{M}$ under scheduler σ. We write $\mathrm{EC}(\mathcal{M}^\sigma, \lozenge B, s)$ for the expected cost of reaching a state in the set $B \subseteq S$ from state s in the MDP $\mathcal{M}$ under scheduler σ. In case the state s is not specified in both Pr and EC, it is assumed to be the initial state s_I.

Definition 6 (Reachabiltiy Property)**.** *We write $\mathbb{P}_{\leq\lambda}(\lozenge B)$ for the reachability property where the probability of eventually reaching B is smaller than λ.*

Definition 7 (Expected cost property)**.** *We write $EC_{\leq\kappa}(\lozenge B)$ for the expected cost property where the expected cost of eventually reaching B is smaller than κ.*

A MDP can be extended into an *uncertain Markov decision process.*

Definition 8 (Uncertain MDP (uMDP))**.** *An uncertain Markov decision process (uMDP) is a tuple $\mathcal{M} = (S, Act, s_I, \mathbb{I}, \mathcal{P})$, where S is a finite set of states, Act a finite set of actions, s_I an initial state ($s_I \in S$), and $\mathbb{I}$ a set of intervals between 0 and 1: $\mathbb{I} = \{[a, b] \mid a, b \in [0, 1]$ and $a \leq b\}$. The uncertain transition probabilities are given by a function $\mathcal{P}\colon S \times Act \times S \to I$, for a certain $I \in \mathbb{I}$.*

For a function $P\colon S \times Act \times S \to Distr(S)$, we write $P \in \mathcal{P}$ if for each $s, s' \in S$, $\alpha \in Act$, $P(s, \alpha, s') \in \mathcal{P}(s, a, s')$ holds. That is, a transition between states s and s' by choosing action α is a value in the interval $\mathcal{P}(s, \alpha, s')$, and the sum over the values of all transitions out of s is 1. A uMDP $\mathcal{M}$ can be *instantiated* by a $P \in \mathcal{P}$, and that *instantiation* is given by the MDP $\mathcal{M}[P] = (S, Act, s_I, P)$.

Definition 9 (Nonlinear program (NLP))**.** *A nonlinear program is an optimization problem of the form:*

$$\begin{aligned} &\textit{minimize} && f_0(x), && \\ &\textit{subject to} && f_i(x) \leq 0, && i = 1, \ldots, m, \\ & && g_j(x) = 0, && j = 1, \ldots, p, \end{aligned}$$

where the functions f_i or g_j are allowed to be nonlinear.

Definition 10 (Geometric program (GP) in posynomial form). *A geometric program in posynomial (or standard) form is an optimization problem of the form:*

$$\begin{aligned} &\textit{minimize} && f_0(x), && \\ &\textit{subject to} && f_i(x) \leq 1, && i = 1, \dots, m, \\ & && g_j(x) = 1, && j = 1, \dots, p, \end{aligned}$$

where the objective function f_0 is a posynomial, all other f_i are posynomial inequality constraints, and all g_j are monomial equality constraints.

A GP in posynomial form is not a convex optimization problem, but it can be transformed into one.

Definition 11 (Log-sum-exp (lse-) function). *We define the convex function lse: $\mathbb{R}^k \to \mathbb{R}$ by*

$$\mathit{lse}(z_1, ..., z_k) = \log(e^{z_1} + \cdots + e^{z_k}), \quad z_1, ..., z_k \in \mathbb{R},$$

and call this the (k-term) log-sum-exp *function, or* lse-*function.*

Definition 12 (Geometric program (GP) in convex form). *A geometric program in convex form is a* convex *optimization problem of the form:*

$$\begin{aligned} &\textit{minimize} && \mathbf{c}^T\mathbf{y}, && \\ &\textit{subject to} && \mathit{lse}(\mathbf{A}_i\mathbf{y} + b_i) \leq 0, && i = 1, \dots, m, \\ & && \mathbf{G}\mathbf{y} + \mathbf{h} = 0, && j = 1, \dots, p, \end{aligned}$$

where $\mathbf{y} \in \mathbb{R}^n$ is a vector of n optimization variables. The problem data is given by $\mathbf{c}, veh \in \mathbb{R}^n$, $\mathbf{A} \in \mathbb{R}^{k \times n}$, with k the number of terms in the lse-function, and $\mathbf{G} \in \mathbb{R}^{l \times n}$, with l the number of monomial equality constraints in the corresponding posynomial form GP.

Definition 13 (Linear program (LP)). *A linear program is an optimization problem of the form:*

$$\begin{aligned} &\textit{minimize} && f_0(x), && \\ &\textit{subject to} && f_i(x) \leq 1, && i = 1, \dots, m, \\ & && g_j(x) = 1, && j = 1, \dots, p, \end{aligned}$$

where all f_i and g_j are linear functions.

We say that a NLP, GP (in posynomial and in convex form) or LP is *feasible* if a solution that satisfies all the constraints in the problem exists and the objective function takes a value in $\mathbb{R}$. If such an optimization problem is not feasible, it is either *unbounded* (if the objective function takes a value in

$\{-\infty, \infty\}$ when minimizing or maximizing respectively), or it is *infeasible*: a solution satisfying *all* constraints does not exist.

The optimization problems defined in definitions 9 10, 12 and 13 can be extended to account for *uncertainty* in their problem data. The general idea behind this is that one or more coefficients in one or more of the functions that form the constraints is not deterministic, but can be any value in a certain *uncertainty set.*

Definition 14 (Robust LP)**.** *A robust linear program (written in its vector form) with uncertainty set* $\mathcal{U}$ *is written as*

$$\begin{aligned} \textit{minimize} \quad & \mathbf{c}^T\mathbf{x}, \\ \textit{subject to} \quad & \mathbf{A}\mathbf{x} + \mathbf{b} \leq 0, \\ & \mathbf{B}\mathbf{x} + \mathbf{d} = 0, \end{aligned}$$

where $(\mathbf{A}, \mathbf{b}, \mathbf{B}, \mathbf{d}) \in \mathcal{U}$*, for matrices* $\mathbf{A}$ *and* $\mathbf{B}$*, and vectors* $\mathbf{b}$ *and* $\mathbf{d}$. $\mathbf{x}$ *is the vector of optimization variables.*

Of course not all coefficients have to be in the uncertainty set, for example if only $\mathbf{b} \in \mathcal{U}$, it would still be a robust LP. Only if *none* of the coefficients lie in the uncertainty set, the LP above is *not* robust. Robust GPs and robust NLPs are defined analogously.

All of our definitions are consistent with [5], [10], [20], [6] and [4].

Chapter 4

Problem Statement

Problem 1

Given an uMDP $\mathcal{M}$, a reachability property $\mathbb{P}_{\leq\lambda}(\lozenge T)$, and an expected cost property $EC_{\leq\kappa}(\lozenge G)$, we want to find a scheduler $\sigma \in Sched^{\mathcal{M}}$ such that

$$\forall P \in \mathcal{P}. \quad \Pr(\mathcal{M}^{\sigma}[P], \lozenge T) \leq \lambda, \text{ and } \mathrm{EC}(\mathcal{M}^{\sigma}[P], \lozenge G) \leq \kappa.$$

In other words, a scheduler σ is a solution to this problem if the probability of reaching the target set $T \subseteq S$ is less than λ, and the expected cost of reaching the goal set $G \subseteq S$ is less than κ.

4.1 A robust nonlinear programming encoding of uncertain Markov decision processes

We can express problem 1 as a *nonlinear program*, using the following optimization variables:

- $\{\sigma^{s,\alpha} \mid s \in S, \alpha \in Act(s)\}$, which define the randomized schedulers σ given by $\sigma(s)(\alpha) = \sigma^{s,\alpha}$,
- $\{p_s \mid s \in S\}$, where p_s is the probability of reaching the target set $T \subseteq S$ from state s under scheduler σ,
- $\{c_s \mid s \in S\}$, where c_s is the expected cost to reach $G \subseteq S$ from state s under scheduler σ.

We define the robust nonlinear program as follows and will then explain how this encodes problem 1.

$$\text{minimize} \quad c_{s_I}, \tag{NLP_{obj}}$$

$$\text{subject to} \quad p_{s_I} \leq \lambda, \tag{NLP_1}$$

$$c_{s_I} \leq \kappa, \tag{NLP_2}$$

$$\forall s \in S. \quad \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} = 1, \tag{NLP_3}$$

$$\forall s \in T. \quad p_s = 1, \tag{NLP_4}$$

$$\begin{matrix} \forall s \in S \setminus T. \\ \forall P \in \mathcal{P}. \end{matrix} \quad p_s = \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot p_{s'}, \tag{NLP_5}$$

$$\forall s \in G. \quad c_s = 0 \tag{NLP_6}$$

$$\begin{matrix} \forall s \in S \setminus G. \\ \forall P \in \mathcal{P}. \end{matrix} \quad c_s = \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \Big(c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c_{s'} \Big). \tag{NLP_7}$$

The objective function (NLP_{obj}) can be any function f over the optimization variables. We choose to minimize the expected cost for the initial state s_I.

The first constraint, (NLP_1), expresses that the probability of reaching the target set T from start state s_I is less than or equal to λ. This is exactly our reachability property.

The second constraint, (NLP_2), expresses the expected cost property in a similar way the reachability property is expressed: the expected cost of reaching the goal set G from s_I is less than or equal to κ.

The third constraint, (NLP_3), ensures that the values we find for σ when solving this NLP sum up to 1 for every state s, thus forming a valid scheduler.

The fourth constraint, (NLP_4), ensures that for all states s in the set of target states T, the probability of reaching the target set is 1.

The fifth constraint, (NLP_5), encodes how the reachability properties depend on each other. For states in the target set T, we have a reachability probability of 1 by constraint (NLP_3). States not in the target set have probabilities that depend on the probability of the successor state s', the probability of the transition function, and the probability of the scheduler.

The sixth constraint, (NLP_6), is similar to the fourth. We want to ensure that the cost to reach any of the goal states in G is 0 for any state that is in the set of goal states.

The seventh constraint, (NLP_7), is similar to the fifth. For all goal states c_s is defined 0 by constraint (NLP_6). For all non-goal states, the cost c_s is given by the cost function $c(s, \alpha)$, the cost of reaching the goal set from successor state s' and the probability of reaching that s'.

Theorem 1. *The NLP-encoding given by* (NLP_{obj})—(NLP_7) *of problem 1 is correct and complete by construction.*

By *correctness*, we mean that a solution to the NLP gives us values for the optimization variables in $\mathbb{R}$ that satisfy the constraints, and by construction form a valid scheduler that solves problem 1.

Completeness means that all possible solutions to problem 1 can be encoded in this NLP. The NLP then has either a solution, which is a correct solution to problem 1, or is *infeasable*: no solution to the NLP exists, meaning there is no solution to problem 1 either.

Unfortunately, nonlinear programming in general is hard to solve [6]. However, with some relatively small changes, we can transform the robust NLP given by (NLP_{obj})—(NLP_7) into a *robust geometric program.*

4.2 Robust geometric programming

In this section, we define a *robust geometric program* (robust GP) and show how this robust GP follows from the robust NLP in (NLP_{obj})—(NLP_7). We use the same optimization variables $\sigma^{s,\alpha}$ and p_s as in the NLP encoding. For the expected cost we introduce new optimization variables $\{c'_s \mid s \in S, c'_s = c_s + 1\}$. With these optimization variables we can transform the robust NLP into the following robust GP:

$$\text{minimize} \quad c'_{s_I} + \sum_{s\in S,\ \alpha\in Act} \frac{1}{\sigma^{s,\alpha}}, \tag{RGP_{obj}}$$

$$\text{subject to} \quad \frac{p_{s_I}}{\lambda} \le 1, \tag{RGP_1}$$

$$\frac{c'_{s_I}}{\kappa+1} \le 1, \tag{RGP_2}$$

$$\forall s \in S. \quad \sum_{\alpha\in Act(s)} \sigma^{s,\alpha} \le 1, \tag{RGP_3}$$

$$\forall s \in T. \quad p_s = 1, \tag{RGP_4}$$

$$\begin{array}{r} \forall s \in S \setminus T. \\ \forall P \in \mathcal{P}. \end{array} \quad \frac{\sum_{\alpha\in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s'\in S} P(s,\alpha,s') \cdot p_{s'}}{p_s} \le 1, \tag{RGP_5}$$

$$\forall s \in G. \quad c'_s = 1, \tag{RGP_6}$$

$$\begin{array}{r} \forall s \in S \setminus G. \\ \forall P \in \mathcal{P}. \end{array} \quad \frac{\sum_{\alpha\in Act(s)} \sigma^{s,\alpha} \cdot \Big(c(s,\alpha) + \sum_{s'\in S} P(s,\alpha,s') \cdot c'_{s'} \Big)}{c'_s} \le 1. \tag{RGP_7}$$

By definitions 10 and 14 a robust GP is only allowed to have posynomial inequality constraints, and monomial equality constraints. The objective function has to be a posynomial or a monomial as well. We will now show how each constraint in (NLP_{obj})—(NLP_7) is transformed to get (RGP_{obj})—(RGP_7).

For first constraint, (NLP_1), we divide both sides by λ. This gives us the inequality (RGP_1), which is a valid inequality constraint.

For the second constraint, (NLP_2), we first replace c_{s_I} with c'_{s_I} and add 1 to the other side of the inequality. Now we divide both sides by $\kappa + 1$, such that we have the valid inequality constraint (RGP_2).

The third constraint, (NLP_3), gives us a sum over optimization variables, which is a posynomial by definition 2. The constraint, however, is a equality, which is not allowed for posynomial constraints in a GP. So we *relax* this constraint to an inequality. This immediately results in (RGP_3) which is a posynomial inequality constraint, which is allowed in a GP.

The fourth constraint, (NLP_4), is a monomial equality constraint and can be copied over directly. This gives us (RGP_4).

The fifth constraint, (NLP_5), can be transformed into a posynomial inequality constraint. We do this by first dividing both sides of the equation by

p_s and swapping the left-hand side and the right-hand side of the equation, giving us

$$\frac{\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s, \alpha, s') \cdot p_{s'}}{p_s} = 1. \tag{1}$$

Now we relax the equality, giving us a valid posynomial inequality constraint in (RGP_5).

For the sixth constraint, (NLP_6), we use our newly introduced variables c'_s to ensure we have a monomial that is equal to 1. In (NLP_6) we have $c_s = 0$, which we can rewrite as $c_s + 1 = 1$. Because this is not a monomial but a posynomial, we are not allowed to use equality. To amend this, we use the new optimization variables c'_s, which in this GP are considered monomials, and thus allowed to be equal to 1.

The seventh constraint, (NLP_7), is similar to the fifth. First we replace c_s with c'_s, the we divide both sides by c'_s, giving us a posynomial equality constraint, which we then relax to get (RGP_7).

For a GP, the objective function also has to be a monomial or posynomial that is minimized. The changes we made to the constraints have an impact on the correctness of the encoding, which we will discuss in detail in section 4.4. The objective function we choose in (NLP_{obj}) is a monomial that is minimized, and thus can be copied over.

However, because we had to relax constraint (RGP_3), we are no longer guaranteed to get a valid probability distribution over the actions enabled at each state. In general, we can fix this by defining the following posynomial as our objective function:

$$\text{maximize} \quad \sum_{s \in S, \alpha \in Act} \sigma^{s,\alpha}. \tag{2}$$

Maximizing this sum, constrained by (RGP_3), ensures that if a valid probability distribution for the σ that is feasible to the entire robust GP exists, we will find it. In general, maximizing x is equivalent to minimizing x^{-1}. So we can rewrite (2) to

$$\text{minimize} \quad \frac{1}{\sum_{s \in S, \alpha \in Act} \sigma^{s,\alpha}}. \tag{3}$$

But this is no longer a posynomial. So instead, we use

$$\text{minimize} \quad \sum_{s \in S, \alpha \in Act} \frac{1}{\sigma^{s,\alpha}}, \tag{4}$$

and call this a *regularization function.* While (4) is not equivalent to (2), it is a posynomial that is minimized, and thus allowed as an objective function. It also expresses a preference for valid schedulers, as invalid schedulers don't sum up to 1, which means that (4) will be larger than for a valid scheduler. We can combine this with the objective function we used in the NLP. This gives us our new objective function (RGP_{obj}).

4.3 Correctness of the robust GP

In this section, we formally define when the robust GP is considered correct, and prove that is indeed the case.

Theorem 2 (Correctness of the robust GP)**.** *Let $(c'_s)_{s\in S}$, $(p_s)_{s\in S}$ and $(\sigma^{s,\alpha})_{s\in S,\alpha\in Act}$ be a solution to the robust GP (RGP_{obj})—(RGP_7) such that $(\sigma^{s,\alpha})_{s\in S,\alpha\in Act}$ defines a valid scheduler $\sigma \in Sched^{\mathcal{M}}$. Then the scheduler σ is a correct solution to problem 1, if for all $P \in \mathcal{P}$ we have $Pr(\mathcal{M}^\sigma[P], \lozenge T) \leq \lambda$, and $EC(\mathcal{M}^\sigma[P], \lozenge G) \leq \kappa$.*

In order to prove this correctness theorem, we will first introduce two lemmas to help us. Once these lemmas are proven, we can easily prove theorem 2.

Lemma 1 (Reachability lemma)**.** *Let $(p_s)_{s\in S}$ and $(\sigma^{s,\alpha})_{s\in S,\,\alpha\in Act}$ be a solution satisfying constraints (RGP_3), (RGP_4) and (RGP_5), such that $(\sigma^{s,\alpha})_{s\in S,\,\alpha\in Act}$ defines a valid scheduler $\sigma \in Sched^{\mathcal{M}}$. Then, for every $P \in \mathcal{P}$ and every $s \in S$, the following holds:*

$$Pr(\mathcal{M}^\sigma[P], \lozenge T, s) \leq p_s. \tag{5}$$

Proof. Fix an arbitrary $P \in \mathcal{P}$. We will show that for every $s \in S$ we have $\Pr(\mathcal{M}^\sigma[P], \lozenge T, s) \leq p_s$.

For every $s \in S$ we define $q_s = \Pr(\mathcal{M}^\sigma[P], \lozenge T, s)$, $x_s = q_s - p_s$, and the set $S_< = \{s \in S \mid p_s < q_s\}$. For states $s \in T$ we have $p_s = q_s = 1$ because of (RGP_4), so $s \in T$ means that $s \notin S_<$. For states s with $q_s = 0$ we have $p_s \geq q_s$, which implies $s \notin S_<$. So for every state $s \in S_<$, p_s satisfies inequality (RGP_5), and T is reachable with positive probability. We will now prove this lemma by contradiction. Assume $S_< \neq \emptyset$, and define

$$x_{max} = \max\{x_s \mid s \in S\},$$
$$S_{max} = \{s \in S \mid x_s = x_{max}\}.$$

So x_{max} is the largest difference between q_s and p_s out of all states, and S_{max} is a set of all states for which this difference is maximal. Because we assume that $S_< \neq \emptyset$, we know there is a $x_s > 0$, so $x_{max} > 0$, and thus a state $s \in S$ exists for which $x_s = x_{max}$. Then this means that $s \in S_<$, and for this state s we have

$$p_s \geq \sum_{\alpha\in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s'\in S} P(s,\alpha,s') \cdot p_{s'}. \tag{6}$$

And for q_s we have

$$q_s = \Pr(\mathcal{M}^\sigma[P], \lozenge T, s) = \sum_{\alpha\in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s'\in S} P(s,\alpha,s') \cdot q_{s'}. \tag{7}$$

Thus we have

$$q_s - p_s \leq \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot (q_{s'} - p_{s'}), \tag{8}$$

which simplifies to

$$x_s \leq \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot x_{s'}. \tag{9}$$

By definition we know that for all $\alpha \in Act$ and $s' \in S$ we have that $\sigma^{s,\alpha} \cdot P(s,\alpha,s') \geq 0$ and that $\sum_{\alpha \in Act(s)} \sum_{s' \in S} \sigma^{s,\alpha} \cdot P(s,\alpha,s') = 1$. Using these facts, we establish

$$\begin{aligned} x_{max} = x_s &\leq \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot x_{s'} \\ &\leq \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot x_{max} \\ &\leq 1 \cdot x_{max} = x_{max}. \end{aligned} \tag{10}$$

So the inequalities are equalities, giving us

$$\begin{aligned} x_{max} = x_s &= \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot x_{s'} \\ &= x_{max} \cdot \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s'). \end{aligned} \tag{11}$$

By (11) we have $x_{s'} = x_{max} > 0$ for every successor state s' of s in $\mathcal{M}^\sigma[P]$. This equality means that for every state $s \in S_{max}$, all successors of s are also in S_{max}. Earlier we showed that $S_< \cap T = \emptyset$, which means that T is *not* reachable (with positive probability) from any state $s \in S_{max}$. This is a contradiction with the fact that T is reachable from every state in $S_<$.

So $S_< = \emptyset$, which means there are no states where $p_s < q_s$, thus $\Pr(\mathcal{M}^\sigma[P], \lozenge T, s) \leq p_s$ for all $P \in \mathcal{P}$ and all $s \in S$. □

Lemma 2 (Expected cost lemma). *Let $(c'_s)_{s \in S}$ and $(\sigma^{s,\alpha})_{s \in S, \alpha \in Act}$ be a solution satisfying constraints (RGP_3), (RGP_6) and (RGP_7), such that $(\sigma^{s,\alpha})_{s \in S, \alpha \in Act}$ defines a valid scheduler $\sigma \in Sched^{\mathcal{M}}$. Then, by definition we have $c_s = c'_s - 1$ for all states s, and for every $P \in \mathcal{P}$ and every $s \in S$, the following holds:*

$$EC(\mathcal{M}^\sigma[P], \lozenge G, s) \leq c_s. \tag{12}$$

Proof. This proof is very similar to that of the reachability lemma. Fix an arbitrary $P \in \mathcal{P}$. We will show that for every $s \in S$ $\mathrm{EC}(\mathcal{M}^\sigma[P], \Diamond G, s) \leq c_s$ holds.

For every $s \in S$ we define $q_s = \mathrm{EC}(\mathcal{M}^\sigma[P], \Diamond G, s)$, $x_s = q_s - c_s$, and the set $S_< = \{s \in S \mid c_s < q_s\}$. For states $s \in G$ we have $c_s = q_s = 0$ because of (RGP_6), so $s \in G$ means that $s \notin S_<$. For states s with $q_s = 1$ we have $c_s \geq q_s$, which implies $s \notin S_<$. So for every state $s \in S_<$, c_s satisfies inequality (RGP_7), and G has expected cost zero. We will now prove this lemma by contradiction. Assume $S_< \neq \emptyset$, and define

$$\begin{aligned} x_{max} &= \max\{x_s \mid s \in S\}, \\ S_{max} &= \{s \in S \mid x_s = x_{max}\}. \end{aligned}$$

So x_{max} is the largest difference between q_s and c_s out of all states, and S_{max} is a set of all states for which this difference is maximal. Because we assume that $S_< \neq \emptyset$, we know there is a $x_s > 0$, so $x_{max} > 0$, and thus a state $s \in S$ exists for which $x_s = x_{max}$. Then this means that $s \in S_<$, and for this state s we have, by (RGP_7) and $c'_s = c_s + 1$,

$$\begin{aligned} c'_s \geq &\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \Big(c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c'_{s'} \Big) \\ = &\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \Big(c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c_{s'} \Big) \\ &+ \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \\ = &\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \Big(c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c_{s'} \Big) + 1. \end{aligned} \tag{13}$$

Using $c'_s = c_s + 1$ we conclude

$$c_s \geq \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \Big(c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c_{s'} \Big). \tag{14}$$

For q_s we have

$$\begin{aligned} q_s &= \mathrm{EC}(\mathcal{M}^\sigma[P], \Diamond G, s) \\ &= \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \Big(c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot q_{s'} \Big). \end{aligned} \tag{15}$$

Thus we have

$$q_s - c_s \leq \Big(\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot (q_{s'}) \Big). \tag{16}$$

Which simplifies to

$$x_s \leq \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot x_{s'}. \tag{17}$$

By definition we know that for all $\alpha \in Act$ and $s' \in S$ we have that $\sigma^{s,\alpha} \cdot P(s,\alpha,s') \geq 0$ and that $\sum_{\alpha \in Act(s)} \sum_{s' \in S} \sigma^{s,\alpha} \cdot P(s,\alpha,s') = 1$. Using these facts, we establish

$$\begin{aligned} x_{max} = x_s &\leq \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot x_{s'} \\ &\leq \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot x_{max} \\ &\leq 1 \cdot x_{max} = x_{max}. \end{aligned} \tag{18}$$

So the inequalities are equalities, giving us

$$\begin{aligned} x_{max} = x_s &= \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot x_{s'} \\ &= x_{max} \cdot \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s'). \end{aligned} \tag{19}$$

By (19) we have $x_{s'} = x_{max} > 0$ for every successor state s' of s in $\mathcal{M}^\sigma[P]$. This equality means that for every state $s \in S_{max}$, all successors of s are also in S_{max}. Earlier we showed that $S_< \cap G = \emptyset$, which means that G is *not* reachable (with smaller expected cost) from any state $s \in S_{max}$. This is a contradiction with the fact that G is reachable from every state in $S_<$.

So $S_< = \emptyset$, which means there are no states where $c_s < q_s$, thus $\mathrm{EC}(\mathcal{M}^\sigma[P], \lozenge G, s) \leq c_s$ for all $P \in \mathcal{P}$ and all $s \in S$. □

Now we can prove theorem 2.

Proof theorem 2. We have $(c'_s)_{s \in S}$, $(p_s)_{s \in S}$ and $(\sigma^{s,\alpha})_{s \in S, \alpha \in Act}$ satisfying constraints (RGP_1)—(RGP_7), for all $s \in S$ we have $c'_s = c_s + 1$, and $(\sigma^{s,\alpha})_{s \in S, \alpha \in Act}$ defines a valid scheduler.

We can apply lemma 1. This gives us

$$\forall P \in \mathcal{P}. \forall s \in S.\ \Pr(\mathcal{M}^\sigma[P], \lozenge T, s) \leq p_s. \tag{20}$$

So in particular

$$\forall P \in \mathcal{P}.\ \Pr(\mathcal{M}^\sigma[P], \lozenge T, s_I) \leq p_{s_I}. \tag{21}$$

And because all constraints are satisfied, we know that $p_{s_I} \leq \lambda$ for all $P \in \mathcal{P}$. So we can rewrite (21) to

$$\forall P \in \mathcal{P}.\ \Pr(\mathcal{M}^\sigma[P], \lozenge T) \leq \lambda. \tag{22}$$

This proves the first part of theorem 2.

For the second part, we apply lemma 2. This gives us

$$\forall P \in \mathcal{P}. \forall s \in S.\ \mathrm{EC}(\mathcal{M}^\sigma[P], \Diamond G, s) \leq c_s. \tag{23}$$

So we also have

$$\forall P \in \mathcal{P}.\ \mathrm{EC}(\mathcal{M}^\sigma[P], \Diamond G, s_I) \leq c_{s_I}. \tag{24}$$

Using that c'_{s_I} satisfies (RGP_2), and $c'_s = c_s + 1$, we know that $c_{s_I} \leq \kappa$. Then (24) can be rewritten as

$$\forall P \in \mathcal{P}.\ \mathrm{EC}(\mathcal{M}^\sigma[P], \Diamond G, s_I) \leq \kappa. \tag{25}$$

This proves the second part of theorem 2. So we conclude that if a solution to the robust GP (RGP_{obj})—(RGP_7) defines a valid scheduler, that scheduler is a solution to problem 1, and thus the robust GP is correct with regards to problem 1. □

4.4 Counter example to the completeness of the robust GP

We have shown our robust GP to be *correct*, but it is not *complete*. It is possible a solution to a GP gives us a invalid scheduler.

Theorem 3 (Robust GP is not complete)**.** *The robust GP* (RGP_{obj})*—* (RGP_7) *is not complete with regards to problem 1.*

Proof by counter example. We define our uMDP $\mathcal{M} = (S, Act, s_I, \mathbb{I}, \mathcal{P})$ where $S = \{s_1, s_2, s_3\}$, $Act = \{a, b\}$, and initial state $s_I = s_1$. We define the transitions by the diagram in figure 2.

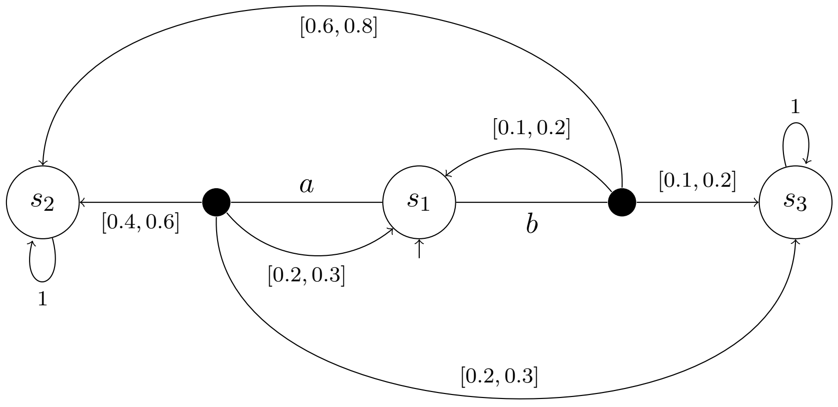
For simplicity, we will only consider the reachability property. This is sufficient for a counter example, as checking an expected cost property too would add extra constraints to the GP, thus further limiting the solution space.

We are interested in reaching state s_3, so we define out target set $T = \{s_3\}$. Let p_s denote the reachability probability from state s to state s_3. Then $p_2 = 0$, since state s_3 cannot be reached from s_2, and $p_3 = 1$ since s_3 can never be left once reached.

For s_1, the probability of reaching s_3 is the addition of the probability of reaching s_3, and the probability of looping back to s_1 times the probability of reaching s_3. Let $P(s, s')$ denote the probability of going from state s to state s'. For p_1 we have

$$p_1 = P(s_1, s_3) + P(s_1, s_1) \cdot p_1. \tag{26}$$

Figure 2: Counter example uMDP



Self loops for states s_2 and s_3 of probability 1 are over both acts.

Equation (26) can be written in closed form as

$$p_1 = \frac{P(s_1, s_3)}{1 - P(s_1, s_1)}. \tag{27}$$

We now need to take the actions a and b into account. A probability $P(s, s')$ can be rewritten as $\sigma^{s,a}P(s, a, s') + \sigma^{s,b}P(s, b, s')$. If $(\sigma^{s,\alpha})_{s\in S,\alpha\in Act}$ defines a valid scheduler, then $P(s, \alpha, s') \in \mathcal{P}(s, \alpha, s')$. We substitute

$$p_1 = \sigma^{1,a}P(1, a, 3) + \sigma^{1,b}P(1, b, 3) + (\sigma^{1,a}P(1, a, 1) + \sigma^{1,b}P(1, b, 1))p_1$$

Because reaching state s_3 from state s_1 is dependent on reaching state s_3 directly, or looping back to state s_1 and then going to state s_3. (Formally, we would need to add $P(1, 2)p_2$ to the equation too, but since $p_2 = 0$ we can leave this out).

Equation (27) shows us that p_1 is a function of a scheduler σ. From figure 2 it is clear that the probability of reaching s_3 is highest when the scheduler always takes action a, and lowest when the scheduler always takes action b. So the upper bound of p_1 is given by always choosing action a and the upper bound of the intervals. The lower bound of p_1 is given by choosing action b and taking the upper bounds. This gives us upper bound $\bar{p}_1$ and lower bound $\underline{p}_1$ with the following values:

$$\bar{p}_1 = \frac{0.3}{1 - 0.3} \approx 0.429, \qquad \underline{p}_1 = \frac{0.2}{1 - 0.2} = 0.25. \tag{28}$$

For $\delta \in [0,1]$, we know there exists a scheduler σ with $p_1(\sigma) = \delta\bar{p}_1 + (1-\delta)\underline{p}_1$. So we choose this value as our λ in the following GP:

$$\text{minimize} \quad \sum_{s\in S,\ \alpha\in Act} \frac{1}{\sigma^{s,\alpha}}, \qquad (CEX_{obj})$$

$$\text{subject to} \quad \frac{p_{s_1}}{0.429\cdot\delta + 0.25\cdot(1-\delta)} \leq 1, \qquad (CEX_1)$$

$$\forall s \in \{s_1, s_2, s_3\}. \quad \sum_{\alpha\in Act(s)} \sigma^{s,\alpha} \leq 1, \qquad (CEX_3)$$

$$p_3 = 1, \qquad (CEX_4)$$

$$\forall s \in \{s_1\}. \quad \frac{\sum\limits_{\alpha\in Act(s)} \sigma^{s,\alpha} \cdot \sum\limits_{s'\in S} P(s,\alpha,s')\cdot p_{s'}}{p_s} \leq 1. \qquad (CEX_5)$$

We can further simplify this counter example GP by filling in the values we choose for P and some rewriting to make it easy to implement. The following GP is implemented in GPkit [8] and used to calculate the results:

$$\text{minimize} \quad \frac{1}{\sigma^{s_1,a}} + \frac{1}{\sigma^{s_1,b}} + \frac{1}{\sigma^{s_2,a}} + \frac{1}{\sigma^{s_2,b}} + \frac{1}{\sigma^{s_3,a}} + \frac{1}{\sigma^{s_3,b}}, \qquad (CEX'_{obj})$$

$$\text{subject to} \quad p_{s_1} \leq 0.429\cdot\delta + 0.25\cdot(1-\delta), \qquad (CEX'_1)$$

$$\begin{aligned} &\sigma^{s_1,a} + \sigma^{s_1,b} \leq 1,\\ &\sigma^{s_2,a} + \sigma^{s_2,b} \leq 1,\\ &\sigma^{s_3,a} + \sigma^{s_3,b} \leq 1, \end{aligned} \qquad (CEX'_3)$$

$$p_3 = 1, \qquad (CEX'_4)$$

$$\frac{\sigma^{s_1,a}\cdot 0.3 + \sigma^{s_1,b}\cdot 0.2 + (\sigma^{s_1,a}\cdot 0.3 + \sigma^{s_1,b}\cdot 0.2)\cdot p_1}{p_1} \leq 1. \qquad (CEX'_5)$$

GPkit offers a special function to tighten inequalities[1]. With this function we can set a tolerance, and GPkit will warn us if the difference between both sides of inequalities is larger than the allowed tolerance. We use this on the constraints in (CEX'_3). These are our scheduler variables, and we want them to be equal to 1. We calculate the solution to (CEX'_{obj})—(CEX'_5) for a number of different choices for δ. The results are in table 1.

[1]`http://gpkit.readthedocs.io/en/latest/advancedcommands.html#tight-constraintsets`

Table 1: Solutions of the counter example GP

δ	λ	$\sigma^{1,a}$	$\sigma^{1,b}$	$\sigma^{1,a} + \sigma^{1,b}$	Error	p_1
1	0.429	0.5	0.5	1	0%	0.3403
0.5	0.3395	0.5	0.5	1	0%	0.3337
0.4	0.3216	0.4465	0.5469	0.9934	0.66%	0.3216
0.3	0.3037	0.4275	0.5235	0.9510	4.9%	0.3037
0.2	0.2858	0.4079	0.4995	0.9074	9.3%	0.2858

As we can see in table 1, depending on the choice of δ, and thus λ, the GP can produce schedulers that are valid (like for $\delta = 1$ or $\delta = 0.5$), or invalid schedulers (like when $\delta = 0.4$, $\delta = 0.3$, or $\delta = 0.2$), while we know that by construction of this uMDP a valid scheduler exists for all δ. Details on how to reproduce these results can be found in appendix A. $\square$

4.5 Solving the robust GP

Now that we know we can get correct solutions to problem 1 by solving the robust GP (RGP_{obj})—(RGP_7), we need to find a way to solve it. Robust geometric programming is PSPACE-hard [9]. However, an efficient approximation method exists [20]. With this method, we can approximate the robust GP by a robust linear program, which can be solved efficiently for certain kinds of uncertainty sets [26].

The approximation method works by reformulating the posynomial form robust GP as a convex form robust GP. The convex form robust GP consists of lse-functions of k-terms, and each of those can be split into a number of 2-term lse-functions. The 2-term lse-function can be approximated by a piecewise-linear function, which gives us an approximate robust LP for our robust GP.

Chapter 5

Convex reformulation of the robust GP

The robust GP (RGP_{obj})—(RGP_7) is currently defined in *posynomial form*, while the method for solving robust GPs in [20] requires a robust GP in *convex form*. Appendix A in [20] describes how to transform such a GP. We will apply this method onto our GP. The method in the appendix describes a transformation of a GP consisting of posynomial and monomial functions of n optimization variables for each term of the posynomial. We will first describe the transformation method in general, and then apply it to our GP.

5.1 General transformation method

Recall definitions 1 and 2. We define a *monomial* as a function $f\colon \mathbb{R}^n_{++} \to \mathbb{R}$, where $\mathbb{R}^n_{++}$ is the set of real n-vectors with positive components, as

$$f(\mathbf{x}) = d \prod_{j=1}^{n} x_j^{a_j} \tag{mon}$$

with $d \geq 0$ and $a_j \in \mathbb{R}$. A *posynomial* is the sum of a number of monomials:

$$f(\mathbf{x}) = \sum_{k=1}^{K} d_k \prod_{j=1}^{n} x_j^{a_{jk}}. \tag{pos}$$

By definition 10, a *geometric program in posynomial form* is defined as

$$\begin{array}{rlr} \text{minimize} & f_0(\mathbf{x}), & \\ \text{subject to} & f_i(\mathbf{x}) \leq 1, & i = 1, \ldots, m, \\ & h_i(\mathbf{x}) = 1, & i = 1, \ldots, l, \end{array} \tag{GP$_1$}$$

where $f_1, \ldots, f_m$ are posynomials, $h_1, \ldots, h_l$ are monomials, and $\mathbf{x}$ is a vector of n optimization variables. We write x_i for the i-th optimization variable in the vector $\mathbf{x}$. The objective function f_0 is assumed to be a monomial. If this is not the case, and f_0 is a posynomial, the following transformation can be applied:

$$\begin{array}{lll} \text{minimize} & t, & \\ \text{subject to} & f_0(\mathbf{x})t^{-1} \leq 1, & \\ & f_i(\mathbf{x}) \leq 1, & i = 1, \ldots, m, \\ & h_i(\mathbf{x}) = 1, & i = 1, \ldots, l. \end{array} \tag{GP$_2$}$$

Where t is a new optimization variable alongside $\mathbf{x}$, and $f_0(\mathbf{x})t^{-1} \leq 1$ is a new posynomial constraint.

Proposition 1. *The geometric programs* (GP$_1$) *and* (GP$_2$) *are equivalent [20].*

Equivalence of two optimization problems is very intuitive: a solution to the first is also a solution to the second, and in reverse.

If we have a GP of the posynomial form above, a transformation to a GP in convex form can be applied. This involves a change of variables and transforming the objective function and each individual constraint. For every optimization variable x_i we define $y_i = \log x_i$ (so that $x_i = e^{y_i}$).

First, we transform the objective function. We assume this to be a monomial (otherwise, apply the transformation to (GP$_2$)) of the form

$$f_0(x) = \prod_{j=1}^{n} x_j^{c_j}$$

where x_j is the j-th optimization variable in the vector $\mathbf{x} = (x_1, \ldots, x_n)$ and we write $\mathbf{c} = (c_1, \ldots, c_n)$ for the vector of exponents. The new objective function becomes $\mathbf{c}^T\mathbf{y}$. In case the original objective function was posynomial and replaced by t, the new objective function becomes $\log t$ as the exponents of all other variables x_i are 0.

Now that we have transformed our objective function, we move on to the posynomial constraints. If we take the function f of $\mathbf{y} = (y_1, \ldots, y_n)$ and then take the logarithm, we get

$$\tilde{f}(\mathbf{y}) = \log(f(e^{y_1}, \ldots, e^{y_n})) = \log(\sum_{i=1}^{K} e^{\mathbf{a}_k^T\mathbf{y}+b_k}). \tag{29}$$

And by definition 11, (29) is equivalent to

$$\operatorname{lse}(\mathbf{a}_1^T\mathbf{y} + b_1, \ldots, \mathbf{a}_K^T\mathbf{y} + b_K) \tag{30}$$

Where $\mathbf{a}_k = (a_{1,k}, \ldots, a_{n,k})$ are the exponents of the k-th term of the posynomial, and $b_k = \log d_k$ is the logarithm of the constant d_k for the k-th term of the posynomial.

When multiplying $\mathbf{a}_i^T \mathbf{y}$ we get a scalar that is a sum of the i-th exponent times the i-th optimization variable of the posynomial. Optimization variables that do not occur in the posynomial have exponent 0, so they will also not occur in this sum.

Then we replace every posynomial constraint f_i by the lse-function

$$\mathrm{lse}(\mathbf{a}_{i,1}^T \mathbf{y} + b_{i,1}, \ldots, \mathbf{a}_{i,K_i}^T \mathbf{y} + b_{i,K_i}) \leq 0, \quad i = 1, \ldots, m. \tag{lse-constr.}$$

With $\mathbf{a}_{i,j}$ the j-th exponent of the i-th monomial, and $b_{i,j}$ the logarithm of the j-th constant in front of the of the i-th monomial.

Now, all that remains is transforming the monomial constraints. We replace every monomial constraint h_i, $i = 1, \ldots, l$, by

$$\mathbf{g}_i^T \mathbf{y} + \log d_i = 0, \quad i = 1, \ldots, l, \tag{mon-constr.}$$

where $\mathbf{g}_i$ is the vector of exponents of the i-th monomial, and d_i is the constant in front of the product of the original monomial. Now we can rewrite the GP in posynomial form into a GP in convex form:

$$\begin{array}{lll} \text{minimize} & \mathbf{c}^T \mathbf{y}, & \\ \text{subject to} & \mathrm{lse}(\mathbf{a}_{i,1}^T \mathbf{y} + b_{i,1}, \ldots, \mathbf{a}_{i,K_i}^T \mathbf{y} + b_{iK_i}) \leq 0, & i = 1, \ldots, m, \\ & \mathbf{g}_i^T \mathbf{y} + \log d_i = 0, & i = 1, \ldots, l. \end{array} \tag{GP$_3$}$$

Proposition 2. *The geometric program in convex form* (GP$_3$) *is equivalent to the geometric program in posynomial form* (GP$_2$)*, and by proposition 1 also equivalent to* (GP$_1$) *[20].*

We can summarize this convex reformulation of a posynomial form GP by the following step-by-step guide:

1. Check if the objective function $f_0(\mathbf{x})$ in (GP$_1$) is a monomial. If this is the case, go to step 3, else continue with step 2.
2. Change the objective function to a new optimization variable t and add a new constraint such that the GP is now of the form (GP$_2$).
3. Define a new vector of optimization variables $\mathbf{y}$ such that $y_i = \log x_i$.
4. For each constraint: rewrite it to its simplest form: a sum of products and determine whether the constraint is a monomial or a posynomial.
5. If the constraint is a posynomial, transform it into a lse-function as in (lse-constr.), else transform it into a linear constraint as in (mon-constr.).
6. Repeat until all constraints are transformed, and form the new GP as in (GP$_3$).

Proposition 3. *The transformation method for rewriting a posynomial form GP into a convex form GP can also be used to rewrite a robust GP in posynomial form to a robust GP in convex form [20].*

This is also a logical consequence of how robust optimization problems are defined (see definition 14). The coefficients in the uncertainty set are constants in the monomials and posynomials, and by definition 10 are assumed to be positive. So for a coefficient $a \in \mathcal{U}$ we have a as a coefficient in some constraint in the posynomial form GP, and $\log a$ as some $b_{i,j}$ or $\log d_i$ in the convex form GP (GP$_3$).

5.2 Applying the transformation method

We will now apply the transformation method discussed in section 5.1 to the posynomial form robust GP we constructed in section 4.2. To recap: the robust GP is given by:

$$\text{minimize} \quad c'_{s_I} + \sum_{s \in S,\ \alpha \in Act} \frac{1}{\sigma^{s,\alpha}}, \tag{RGP_{obj}}$$

$$\text{subject to} \quad \frac{p_{s_I}}{\lambda} \leq 1, \tag{RGP_1}$$

$$\frac{c'_{s_I}}{\kappa + 1} \leq 1, \tag{RGP_2}$$

$$\forall s \in S. \quad \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \leq 1, \tag{RGP_3}$$

$$\forall s \in T. \quad p_s = 1, \tag{RGP_4}$$

$$\begin{array}{l}\forall s \in S \setminus T. \\ \forall P \in \mathcal{P}.\end{array} \quad \frac{\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot p_{s'}}{p_s} \leq 1, \tag{RGP_5}$$

$$\forall s \in G. \quad c'_s = 1, \tag{RGP_6}$$

$$\begin{array}{l}\forall s \in S \setminus G. \\ \forall P \in \mathcal{P}.\end{array} \quad \frac{\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot (c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c'_{s'})}{c'_s} \leq 1. \tag{RGP_7}$$

In this robust GP, we have optimization variables p_s, c'_s and $\sigma^{s,\alpha}$ for every state $s \in S$ and every action $\alpha \in Act$. We define $\tilde{c}'_s = \log c'_s$, $\tilde{p}_s = \log p_s$, and $\tilde{\sigma}^{s,\alpha} = \log \sigma^{s,\alpha}$ as our new optimization variables. Additionally, the transition probability $P(s,\alpha,s')$ and the cost of a transition $c(s,\alpha)$ are constants, for which we also define $\tilde{P}(s,\alpha,s') = \log P(s,\alpha,s')$ and $\tilde{c}(s,\alpha) = \log c(s,\alpha)$. We will now transform the (RGP_{obj})—(RGP_7) step-by-step as discussed in section 5.1.

The objective function

We start with the objective function (RGP_{obj}):

$$c_{s_I} + \sum_{s\in S,\alpha\in Act} \frac{1}{\sigma^{s,\alpha}}. \tag{31}$$

This is a posynomial (step 1), so we change this to a new monomial objective function t and add the following constraint given by step 2:

$$(c_{s_I} + \sum_{s\in S,\alpha\in Act} \frac{1}{\sigma^{s,\alpha}})t^{-1} \leq 1. \tag{32}$$

We add $\tilde{t} = \log t$ to our optimization variables (step 3). Now we need to transform the newly added constraint (32). We start by simplifying this posynomial to a sum of products (step 4).

$$\begin{aligned}
& (c_{s_I} + \sum_{s\in S,\, \alpha\in Act} \frac{1}{\sigma^{s,\alpha}})t^{-1} \\
= \; & c_{s_I} \cdot t^{-1} + \sum_{s\in S,\, \alpha\in Act} \frac{1}{t\cdot \sigma^{s,\alpha}} \\
= \; & c_{s_I} \cdot t^{-1} + \sum_{s\in S,\, \alpha\in Act} t^{-1} \cdot (\sigma^{s,\alpha})^{-1} \\
= \; & c_{s_I} \cdot t^{-1} + t^{-1} \cdot (\sigma^{s_1,\alpha_1})^{-1} + \cdots + t^{-1} \cdot (\sigma^{s_{|S|},\alpha_{|Act|}})^{-1} \\
\leq \; & 1
\end{aligned}$$

Now we have a function that is the sum of a product of optimization variables. (Like in the definition of a posynomial in (pos)). We have optimization variables t, with exponent -1, c_{s_I} with exponent 1 and all $\sigma^{s,\alpha}$, each with exponent -1. The exponents for all other decision variables are 0. So the convex constraint is given by step 5:

$$\text{lse}(\tilde{c}_{s_I} - \tilde{t}, -\tilde{t} - \tilde{\sigma}^{s_1,\alpha_1}, \ldots, -\tilde{t} - \tilde{\sigma}^{s_{|S|},\alpha_{|Act|}}) \leq 0 \tag{33}$$

Constraint 1

Next, the first constraint (RGP_1):

$$\frac{p_{s_I}}{\lambda} \leq 1. \tag{34}$$

Note that this is a monomial, but should be treated as a posynomial constraint because of the inequality. λ is a constant, so λ^{-1} is a constant too. We rewrite (34) as

$$\text{lse}(\tilde{p}_{s_I} + \log(\lambda^{-1})) = \tilde{p}_{s_I} + \log(\lambda^{-1}) \leq 0 \tag{35}$$

since the lse-function of 1 parameter is the identity (step 5).

Constraint 2

The second constraint (RGP_2):

$$\frac{c'_{s_I}}{\kappa + 1} \leq 1, \tag{36}$$

is transformed similarly to the first constraint. We rewrite (36) to the convex constraint

$$\text{lse}(\tilde{c}_{s_I} + \log((\kappa+1)^{-1})) = \tilde{c}_{s_I} + \log((\kappa+1)^{-1}) \leq 0. \tag{37}$$

Constraint 3

Next, the third constraint (RGP_3):

$$\forall s \in S. \quad \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \leq 1. \tag{38}$$

This is a posynomial. We use step 4 and write the sum out.

$$\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} = \sigma^{s,\alpha_1} + \cdots + \sigma^{s,\alpha_{|Act(s)|}}. \tag{39}$$

Note that the exponents for all σ are 1, and for all other decision variables are 0. There are no constants, so the resulting convex constraint is given by step 5:

$$\forall s \in S. \quad \text{lse}(\tilde{\sigma}^{s,\alpha_1}, \ldots, \tilde{\sigma}^{s,\alpha_{|Act(s)|}}) \leq 0. \tag{40}$$

Constraint 4

The fourth constraint (RGP_4) is next.

$$\forall s \in T. \quad p_s = 1. \tag{41}$$

We take step 4 and see this monomial already is in its simplest form, we don't need to rewrite it and can immediately take step 5, resulting in the following convex constraint:

$$\forall s \in T. \quad \tilde{p}_s = 0. \tag{42}$$

Constraint 5

The fifth constraint is given by (RGP_5):

$$\forall s \in S \setminus T. \forall P \in \mathcal{P}. \quad \frac{\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot p_{s'}}{p_s} \leq 1. \tag{43}$$

First, we take step 4 and simplify:

$$\begin{aligned}
& \frac{\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum_{s' \in S} P(s,\alpha,s') \cdot p_{s'}}{p_s} \\
= \; & (p_s)^{-1} \cdot (\sigma^{s,\alpha_1} \cdot (P(s,\alpha_1,s'_1) \cdot p_{s'_1} + \cdots + P(s,\alpha_1,s'_{|S|}) \cdot p_{s'_{|S|}}) \\
& + \\
& \vdots \\
& + \sigma^{s,\alpha_{|Act(s)|}} \cdot (P(s,\alpha_{|Act(s)|},s'_1) \cdot p_{s'_1} + \cdots + P(s,\alpha_{k_Act(s)},s'_{|S|}) \cdot p_{s'_{|S|}}) \\
= \; & p_s^{-1}\sigma^{s,\alpha_1}P(s,\alpha_1,s'_1)p_{s'_1} + \cdots + p_s^{-1}\sigma^{s,\alpha_1}P(s,\alpha_1,s'_{|S|})p_{s'_{|S|}} \\
+ \; & p_s^{-1}\sigma^{s,\alpha_2}P(s,\alpha_2,s'_1)p_{s'_1} + \cdots + p_s^{-1}\sigma^{s,\alpha_2}P(s,\alpha_2,s'_{|S|})p_{s'_{|S|}} \\
+ \; & \cdots \\
+ \; & p_s^{-1}\sigma^{s,\alpha_{|Act(s)|}}P(s,\alpha_{|Act(s)|},s'_1)p_{s'_1} + \\
& \quad \cdots \\
& \qquad + p_s^{-1}\sigma^{s,\alpha_{|Act(s)|}}P(s,\alpha_{|Act(s)|},s'_{|S|})p_{s'_{|S|}}.
\end{aligned}$$

Note that in case the transition $P(s,\alpha,s')$ doesn't exist (or equivalently has an exact probability of 0) the entire term becomes 0 and can be removed from the posynomial. Now we transform this posynomial into a lse-function and write it as the following constraint (step 5):

$\forall s \in S \setminus T. \forall P \in \mathcal{P}.$

$$\begin{aligned}
\mathrm{lse}(&-\tilde{p}_s + \tilde{\sigma}^{s,\alpha_1} + \tilde{p}_{s'_1} + \tilde{P}(s, \alpha_1, s'_1),\\
&\vdots\\
&-\tilde{p}_s + \tilde{\sigma}^{s,\alpha_1} + \tilde{p}_{s'_{|S|}} + \tilde{P}(s, \alpha_1, s'_{|S|}),\\
&-\tilde{p}_s + \tilde{\sigma}^{s,\alpha_2} + \tilde{p}_{s'_1} + \tilde{P}(s, \alpha_2, s'_1),\\
&\vdots\\
&-\tilde{p}_s + \tilde{\sigma}^{s,\alpha_2} + \tilde{p}_{s'_{|S|}} + \tilde{P}(s, \alpha_2, s'_{|S|}),\\
&\vdots\\
&-\tilde{p}_s + \tilde{\sigma}^{s,\alpha_{|Act(s)|}} + \tilde{p}_{s'_1} + \tilde{P}(s, \alpha_{|Act(s)|}, s'_1),\\
&\vdots\\
&-\tilde{p}_s + \tilde{\sigma}^{s,\alpha_{|Act(s)|}} + \tilde{p}_{s'_{|S|}} + \tilde{P}(s, \alpha_{|Act(s)|}, s'_{|S|})\\
&) \leq 0.
\end{aligned} \tag{44}$$

Constraint 6

Next the monomial constraint (RGP_6):

$$\forall s \in G. \quad c'_s = 1, \tag{45}$$

turns into

$$\forall s \in G. \quad \tilde{c}'_s = 0. \tag{46}$$

Constraint 7

And the last constraint (RGP_7):

$$\forall s \in S \setminus G. \forall P \in \mathcal{P}. \; \frac{\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot (c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c_{s'})}{c_s} \leq 1. \tag{47}$$

Again, we start by simplifying this posynomial (step 4):

$$\begin{aligned}
& \frac{\sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot (c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c_{s'})}{c_s} \\
= \; & (c_s)^{-1} \sum_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot (c(s,\alpha) + \sum_{s' \in S} P(s,\alpha,s') \cdot c_{s'}) \\
= \; & (c_s)^{-1} \sigma^{s,\alpha_1} c(s,\alpha_1) \\
+ \; & (c_s)^{-1} \sigma^{s,\alpha_1} P(s,\alpha_1,s'_1) c_{s'_1} \\
& \quad + \cdots + \\
& \quad\; (c_s)^{-1} \sigma^{s,\alpha_1} P(s,\alpha_1,s'_{|S|}) c_{s'_{|S|}} \\
+ \; & (c_s)^{-1} \sigma^{s,\alpha_2} c(s,\alpha_2) \\
+ \; & (c_s)^{-1} \sigma^{s,\alpha_2} P(s,\alpha_2,s'_1) c_{s'_1} \\
& \quad + \cdots + \\
& \quad\; (c_s)^{-1} \sigma^{s,\alpha_2} P(s,\alpha_2,s'_{|S|}) c_{s'_{|S|}} \\
\vdots & \\
+ \; & (c_s)^{-1} \sigma^{s,\alpha_{|Act(s)|}} c(s,\alpha_{|Act(s)|}) \\
+ \; & (c_s)^{-1} \sigma^{s,\alpha_{|Act(s)|}} P(s,\alpha_{|Act(s)|},s'_1) c_{s'_1} \\
& \quad + \cdots + \\
& \quad\; (c_s)^{-1} \sigma^{s,\alpha_{|Act(s)|}} P(s,\alpha_{|Act(s)|},s'_{|S|}) c_{s'_{|S|}}
\end{aligned} \tag{48}$$

And now we turn this into a convex constraint (step 5):

$\forall s \in S \setminus G. \quad \forall P \in \mathcal{P}.$

$$
\begin{aligned}
\operatorname{lse}(\; & -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_1} + \tilde{c}(s,\alpha_1), \\
& -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_1} + \tilde{c}_{s'_1} + \tilde{P}(s,\alpha_1,s'_1), \\
& \vdots \\
& -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_1} + \tilde{c}_{s'_{|S|}} + \tilde{P}(s,\alpha_1,s'_{|S|}), \\
& -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_2} + \tilde{c}(s,\alpha_2), \\
& -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_2} + \tilde{c}_{s'_1} + \tilde{P}(s,\alpha_2,s'_1), \\
& \vdots \\
& -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_2} + \tilde{c}_{s'_{|S|}} + \tilde{P}(s,\alpha_2,s'_{|S|}), \qquad (49) \\
& \vdots \\
& -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_{|Act(s)|}} + \tilde{c}(s,\alpha_{|Act(s)|}), \\
& -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_{|Act(s)|}} + \tilde{c}_{s'_1} + \tilde{P}(s,\alpha_{|Act(s)|},s'_1), \\
& \vdots \\
& -\tilde{c}_s + \tilde{\sigma}^{s,\alpha_{|Act(s)|}} + \tilde{c}_{s'_{|S|}} + \tilde{P}(s,\alpha_{|Act(s)|},s'_{|S|}), \\
) \quad & \leq 0.
\end{aligned}
$$

5.2.1 Resulting convex form robust GP

Now we can rewrite the posynomial form robust GP (RGP_{obj})—(RGP_7) as a convex form robust GP with new optimization variables, a new objective function $\tilde{t}$ and the transformed constraints (step 6). We will write the resulting convex form robust GP in vector form for the large posynomial constraints below.

$$
\begin{aligned}
&\text{minimize} && \tilde{t}, && (C_{obj})\\
&\text{subject to} && \operatorname{lse}(\mathbf{a}_{1,c_0}^T\mathbf{y} + b_{1,c_0}, \ldots, \mathbf{a}_{K_0,c_0}^T\mathbf{y} + b_{K_0,c_0}) \leq 0, && (C_0)\\
& && \tilde{p}_{s_I} + \log(\lambda^{-1}) \leq 0, && (C_1)\\
& && \tilde{c}'_{s_I} + \log((\kappa+1)^{-1}) \leq 0, && (C_2)\\
&\forall s \in S. && \operatorname{lse}(\mathbf{a}_{1,c_2}^T\mathbf{y} + b_{1,c_2}, \ldots, \mathbf{a}_{K_2,c_2}^T\mathbf{y} + b_{K_2,c_2}) \leq 0, && (C_3)\\
&\forall s \in T. && \tilde{p}_s = 0, && (C_4)\\
&\begin{matrix}\forall s \in S \setminus T.\\ \forall P \in \mathcal{P}.\end{matrix} && \operatorname{lse}(\mathbf{a}_{1,c_4}^T\mathbf{y} + b_{1,c_4}, \ldots, \mathbf{a}_{K_4,c_4}^T\mathbf{y} + b_{K_4,c_4}) \leq 0, && (C_5)\\
&\forall s \in G. && \tilde{c}'_s = 0, && (C_6)\\
&\begin{matrix}\forall s \in S \setminus G.\\ \forall P \in \mathcal{P}.\end{matrix} && \operatorname{lse}(\mathbf{a}_{1,c_6}^T\mathbf{y} + b_{1,c_6}, \ldots, \mathbf{a}_{K_6,c_6}^T\mathbf{y} + b_{K_6,c_6}) \leq 0. && (C_7)
\end{aligned}
$$

Where $\mathbf{a}_{i,c_j}$ is the vector of exponents of the optimization variables for the i-th product in the posynomial of the j-th constraint, $\mathbf{y}$ is the vector of (new) optimization variables, and b_{i,c_j} is the logarithm of the constant in front of the i-th product of the j-th constraint.

Proposition 4. *By proposition* (2)*, this convex form robust GP is equivalent to the original posynomial form robust GP.*

Chapter 6

Approximating the robust GP by a robust LP

6.1 Simplifications to the robust GP

The robust GP in convex form given by (C_{obj})—(C_7) can be simplified by defining a matrix $\mathbf{A}$ such that $(\mathbf{A}\mathbf{y}+\mathbf{b}) = (\mathbf{a}_1^T\mathbf{y}+b_1,\dots,\mathbf{a}_n^T\mathbf{y}+b_n)$ for each of the posynomial constraints.

So we rewrite $\text{lse}(\mathbf{a}_{1,c_j}^T\mathbf{y}+b_{1,c_j},\dots,\mathbf{a}_{K_j,c_j}^T\mathbf{y}+b_{K_j,c_j})$ as $\text{lse}(\mathbf{A}_{c_j}\mathbf{y}+\mathbf{b}_{c_j})$, where $\mathbf{A}_{c_j}\in\mathbb{R}^{K_j\times n}$ is the matrix formed by filling the i-th row of the matrix with the vector $\mathbf{a}_{i,c_j}^T$ for the j-th constraint denoted by c_j, and $\mathbf{b}_{c_j}\in\mathbb{R}^{K_j}$ is the vector formed by the constants b_{i,c_j}. K_j is the number of terms in the j-th lse-constraint.

Alongside the posynomial constraints, we can also simplify the monomial constraints. The monomial inequality constraint (C_1), $\tilde{p}_{s_I}+\log(\lambda^{-1})$, can be rewritten as $\mathbf{a}_{c_1}^T\mathbf{y}+\mathbf{b}_{c_1}$. The same can be done for (C_2), resulting in $\mathbf{a}_{c_2}^T\mathbf{y}+\mathbf{b}_{c_2}$.

As for the monomial equality constraints, we can group these together into one monomial constraint. We do this by defining the matrix $\mathbf{A}_{c_{mon}}$ such that i-th row of the matrix contains the vector $\mathbf{g}_i^T$, the vector of coefficients in front of each optimization variable. We also define the vector $\mathbf{b}_{c_{mon}}$ as the vector of the constants b_i for each i-th constraint, giving us

$$\mathbf{A}_{c_{mon}} = \begin{bmatrix}\mathbf{g}_1^T\\ \vdots \\ \mathbf{g}_l^T\end{bmatrix}\in\mathbb{R}^{(|T|+|G|)\times n},\qquad \mathbf{b}_{c_{mon}} = \begin{bmatrix}b_1\\ \vdots \\ b_l\end{bmatrix}\in\mathbb{R}^{|T|+|G|}. \tag{50}$$

Where $l=|T|+|G|$ is the number monomial constraints because (C_4) and (C_6) are defined for all states in their respective sets T and G. The simplified convex form robust GP now looks like this:

$$\begin{aligned}
&\text{minimize} && \tilde{t}, && (\mathcal{C}_{obj})\\
&\text{subject to} && \text{lse}(\mathbf{A}_{c_0}\mathbf{y}+\mathbf{b}_{c_0}) \leq 0, && (\mathcal{C}_0)\\
& && \mathbf{a}_{c_1}^T\mathbf{y}+\mathbf{b}_{c_1} \leq 0, && (\mathcal{C}_1)\\
& && \mathbf{a}_{c_2}^T\mathbf{y}+\mathbf{b}_{c_2} \leq 0, && (\mathcal{C}_2)\\
&\forall s \in S. && \text{lse}(\mathbf{A}_{c_3}\mathbf{y}+\mathbf{b}_{c_3}) \leq 0, && (\mathcal{C}_3)\\
&\begin{matrix}\forall s \in S \setminus T.\\ \forall P \in \mathcal{P}.\end{matrix} && \text{lse}(\mathbf{A}_{c_5}\mathbf{y}+\mathbf{b}_{c_5}) \leq 0, && (\mathcal{C}_5)\\
&\begin{matrix}\forall s \in S \setminus G.\\ \forall P \in \mathcal{P}.\end{matrix} && \text{lse}(\mathbf{A}_{c_7}\mathbf{y}+\mathbf{b}_{c_7}) \leq 0, && (\mathcal{C}_7)\\
& && \mathbf{A}_{c_{mon}}\mathbf{y}+\mathbf{b}_{c_{mon}} = \mathbf{0}. && (\mathcal{C}_{mon})
\end{aligned}$$

Proposition 5. *The robust GP given by* $(\mathcal{C}_{obj})$*—*$(\mathcal{C}_{mon})$ *is equivalent to the robust GP given by* (C_{obj})*—*(C_7) *[20].*

So this simplification does not affect the correctness or incompleteness in any way.

Theorem 4. *In the robust GP* $(\mathcal{C}_{obj})$*—*$(\mathcal{C}_{mon})$*, the objective function* $(\mathcal{C}_{obj})$ *and constraints* $(\mathcal{C}_1)$*,* $(\mathcal{C}_2)$ *and* $(\mathcal{C}_{mon})$ *are linear functions, and by definition 13 are allowed in a (robust) linear program.*

This is easy to see. If you perform the vector or matrix multiplications in these functions, you get a linear combination of optimization variables, thus a linear function.

6.2 Rewriting a k-term lse-function as k-1 2-term lse-functions

In the RGP $(\mathcal{C}_{obj})$—$(\mathcal{C}_{mon})$, a function is either linear, or a lse-function. Paragraph 2.2 of [20] describes how to rewrite a k-term lse-function as a number of 2-term lse-functions, which can then be approximated by a piecewise-linear function. Before we can do this, we have to reformulate the convex form robust GP into another convex form robust GP.

We add a new optimization variable τ at the end of our vector of optimization variables $\mathbf{y}$ such that

$$\eta = \begin{bmatrix} \mathbf{y} \\ \tau \end{bmatrix} \in \mathbb{R}^{n+1}. \tag{51}$$

We define $\bar{\mathbf{c}}$ as a vector of $\mathbf{c}$, the vector of the exponents of our monomial objective function, and add a 0 to it, such that

$$\bar{\mathbf{c}}^T = (\mathbf{c}, 0). \tag{52}$$

Note that because of the transformation we performed in section 5.2, where we replaced the posynomial objective function by a monomial objective function t, we have $\mathbf{c} = (0, \dots, 0, 1)$ such that $\mathbf{c}^T\mathbf{y} = \tilde{t}$. So $\bar{\mathbf{c}}^T\eta = \tilde{t}$. So the addition of τ to the optimization variables does not change the objective function.

We define $\bar{\mathbf{A}}_{c_{mon}}$ as the matrix

$$\bar{\mathbf{A}}_{c_{mon}} = \begin{bmatrix} \mathbf{A}_{c_{mon}} & 0 \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{(l+1)\times(n+1)}, \tag{53}$$

and the vector $\bar{\mathbf{b}}_{mon}$ as

$$\bar{\mathbf{b}}_{mon} = \begin{bmatrix} \mathbf{b}_{mon} \\ -1 \end{bmatrix} \in \mathbb{R}^{l+1}. \tag{54}$$

For every lse-constraint we add the vector $\mathbf{b}_{c_i}$ as a column to the end of $\mathbf{A}_{c_i}$, such that

$$\bar{\mathbf{A}}_{c_i} = \begin{bmatrix} \mathbf{A}_{c_i} & \mathbf{b}_{c_i} \end{bmatrix} \in \mathbb{R}^{k_i \times n+1}. \tag{55}$$

Now define the ω-th row of $\bar{\mathbf{A}}_{c_i}$ as $\bar{\mathbf{a}}_{c_i,\omega}^T$ for $\omega = 1, \dots, K_{c_i}$. So $\bar{\mathbf{a}}_{c_i,\omega}^T$ is a row vector in $\mathbb{R}^{1\times(n+1)}$, such that

$$\bar{\mathbf{A}}_{c_i} = \begin{bmatrix} \bar{\mathbf{a}}_{c_i,1}^T \\ \vdots \\ \bar{\mathbf{a}}_{c_i,K_{c_i}}^T \end{bmatrix}. \tag{56}$$

With these new definitions, we can rewrite the convex form robust GP $(\mathcal{C}_{obj})$—$(\mathcal{C}_{mon})$ into the following convex form robust GP:

$$
\begin{aligned}
&\text{minimize} && \tilde{t}, && (\mathcal{C}'_{obj})\\
&\text{subject to} && \operatorname{lse}(\bar{\mathbf{a}}_{c_0,1}^T\eta,\ldots,\bar{\mathbf{a}}_{c_0,K_{c_0}}^T\eta) \le 0, && (\mathcal{C}'_0)\\
& && \bar{\mathbf{a}}_{c_1}^T\eta \le 0, && (\mathcal{C}'_1)\\
& && \bar{\mathbf{a}}_{c_2}^T\eta \le 0, && (\mathcal{C}'_2)\\
&\forall s \in S. && \operatorname{lse}(\bar{\mathbf{a}}_{c_3,1}^T\eta,\ldots,\bar{\mathbf{a}}_{c_3,K_{c_3}}^T\eta) \le 0, && (\mathcal{C}'_3)\\
&\begin{array}{r}\forall s \in S \setminus T.\\ \forall P \in \mathcal{P}.\end{array} && \operatorname{lse}(\bar{\mathbf{a}}_{c_5,1}^T\eta,\ldots,\bar{\mathbf{a}}_{c_5,K_{c_5}}^T\eta) \le 0, && (\mathcal{C}'_5)\\
&\begin{array}{r}\forall s \in S \setminus G.\\ \forall P \in \mathcal{P}.\end{array} && \operatorname{lse}(\bar{\mathbf{a}}_{c_7,1}^T\eta,\ldots,\bar{\mathbf{a}}_{c_7,K_{c_7}}^T\eta) \le 0, && (\mathcal{C}'_7)\\
& && \bar{\mathbf{A}}_{c_{mon}}\eta + \bar{\mathbf{b}}_{c_{mon}} = \mathbf{0}. && (\mathcal{C}'_{mon})
\end{aligned}
$$

Proposition 6. *The RGPs given by $(\mathcal{C}_{obj})$—$(\mathcal{C}_{mon})$ and $(\mathcal{C}'_{obj})$—$(\mathcal{C}'_{mon})$ are equivalent: $y^* \in \mathbb{R}^n$ is feasible to the first if and only if $(y^*, \tau^*) \in \mathbb{R}^{n+1}$ is feasible to the second for some $\tau^* \in \mathbb{R}$ [20].*

So this non-trivial change to the robust GP does not affect our problem in any way. The correctness still holds and the incompleteness is unchanged. For readability, we have chosen a slightly different notation then in [20], but this change of notation does not affect any of the results derived form [20]. See appendix B for the details.

A k-term lse-function can be replaced by a number of 2-term lse-functions by the *approximate reduction procedure* described in paragraph 2.2 of [20]. In general, we replace a single k-term lse constraint

$$\operatorname{lse}(\bar{\mathbf{a}}_1^T\eta,\ldots,\bar{\mathbf{a}}_k^T\eta) \le 0 \tag{57}$$

by $k-1$ 2-term lse-constraints, given by

$$
\begin{aligned}
\operatorname{lse}(\bar{\mathbf{a}}_1^T\eta, z_1) &\le 0,\\
\operatorname{lse}(\bar{\mathbf{a}}_{w+1}^T\eta - z_w, z_{w+1} - z_w) &\le 0, \quad w = 1,\ldots,k-3,\\
\operatorname{lse}(\bar{\mathbf{a}}_{k-1}^T\eta - z_{k-2}, \bar{\mathbf{a}}_k^T\eta - z_{k-2}) &\le 0,
\end{aligned}
\tag{58}
$$

where $\mathbf{z} = (z_1,\ldots,z_{k-2}) \in \mathbb{R}^{k-2}$ are $k-2$ new optimization variables.

Proposition 7. *If* $(\eta^*, \mathbf{z}^*) \in \mathbb{R}^{n+1+k-2}$ *satisfies the set of constraints in* (58)*, then* η^* *satisfies* (57) *[20].*

A solution that satisfies the set of 2-term lse-constraints will also satisfy the k-term lse-constraint, but not necessarily the other way around. That's why it is called an approximate reduction procedure.

Each 2-term lse-function can then be simplified to

$$\text{lse}((\hat{\mathbf{a}}^1_\omega)^T\hat{\eta}, (\hat{\mathbf{a}}^2_\omega)^T\hat{\eta}) \le 0, \quad \omega = 1, \ldots, K_{c_i} - 1 \tag{59}$$

by extending η to include $\mathbf{z}$, a vector with $K_{c_i} - 2$ new variables *for each* k-term lse-constraint, and extending each vector $\bar{\mathbf{a}}$ to include coefficients for each variable in the vector $\mathbf{z}$ we just added. For every constraint we want to transform, we have to introduce new variables $\mathbf{z}$.

Using the above transformation, we can rewrite the k-term convex form robust GP $(\mathcal{C}'_{obj})$—$(\mathcal{C}'_{mon})$ as a 2-term convex form robust GP:

$$\begin{aligned}
&\text{minimize} && \hat{\mathbf{c}}^T\hat{\eta} = \tilde{t}, && (\mathcal{T}_{obj})\\
&\text{subject to} && && \\
&\omega = 1, \ldots, K_{c_0} - 1 && \text{lse}((\hat{\mathbf{a}}^1_{c0,\omega})^T\hat{\eta}, (\hat{\mathbf{a}}^2_{c0,\omega})^T\hat{\eta}) \le 0, && (\mathcal{T}_0)\\
& && \hat{\mathbf{a}}^T_{c_1}\hat{\eta} \le 0, && (\mathcal{T}_1)\\
& && \hat{\mathbf{a}}^T_{c_2}\hat{\eta} \le 0, && (\mathcal{T}_2)\\
&\forall s \in S.\ \ \omega = 1, \ldots, K_{c_3} - 1 && \text{lse}((\hat{\mathbf{a}}^1_{c_3,\omega})^T\hat{\eta}, (\hat{\mathbf{a}}^2_{c_3,\omega})^T\hat{\eta}) \le 0, && (\mathcal{T}_3)\\
&\forall s \in S \setminus T.\ \ \forall P \in \mathcal{P}.\ \ \omega = 1, \ldots, K_{c_5} - 1 && \text{lse}((\hat{\mathbf{a}}^1_{c_5,\omega})^T\hat{\eta}, (\hat{\mathbf{a}}^2_{c_5,\omega})^T\hat{\eta}) \le 0, && (\mathcal{T}_5)\\
&\forall s \in S \setminus G.\ \ \forall P \in \mathcal{P}.\ \ \omega = 1, \ldots, K_{c_7} - 1 && \text{lse}((\hat{\mathbf{a}}^1_{c_7,\omega})^T\hat{\eta}, (\hat{\mathbf{a}}^2_{c_7,\omega})^T\hat{\eta}) \le 0, && (\mathcal{T}_7)\\
& && \hat{\mathbf{A}}_{c_{mon}}\hat{\eta} + \hat{\mathbf{b}}_{c_{mon}} = \mathbf{0}. && (\mathcal{T}_{mon})
\end{aligned}$$

The optimization variables are $\hat{\eta} = (\mathbf{y}, \tau, \mathbf{z}) \in \mathbb{R}^{n+1+K_v}$, and the problem data is given by:

$$K_{c_0} = 1 + |S| \cdot |Act|,$$

$$K_{c_3,s} = |Act(s)|,$$

$$K_{c_5,s} = |S| \cdot |Act(s)|,$$

$$K_{c_7,s} = (1 + |S|) \cdot |Act(s)|,$$

$$K_v = (K_{c_0} - 2) + \sum_{s \in S}(K_{c_3,s} - 2) + \sum_{s \in S \setminus T}(K_{c_5,s} - 2) + \sum_{s \in S \setminus G}(K_{c_7,s} - 2),$$

$$\hat{\mathbf{a}}^1_{c_i,\omega} \quad \text{and} \quad \hat{\mathbf{a}}^2_{c_i,\omega} \in \mathbb{R}^{n+1+K_v}, \quad \omega = 1, \dots, K_{c_i} - 1, \quad i = 0, 1, 2, 4, 6,$$

$$\hat{\mathbf{c}} = (\bar{\mathbf{c}}, \mathbf{0}) \in \mathbb{R}^{n+1+K_v},$$

$$\hat{\mathbf{A}}_{c_{mon}} = \begin{bmatrix} \bar{\mathbf{A}}_{c_{mon}} & \mathbf{0} \end{bmatrix} \in \mathbb{R}^{(l+1)\times(n+1+K_v)},$$

$$\hat{\mathbf{b}}_{c_{mon}} = \bar{\mathbf{b}}_{c_{mon}} \in \mathbb{R}^{l+1}.$$

Where $l = |T| + |G|$, the number of monomial equality constraints, and $n = 2 \cdot |S| + |S||Act| + 1$, the number of optimization variables in the first convex robust GP (C_{obj})—(C_7).

Theorem 5. *The robust GP given by* $(\mathcal{T}_{obj})$—$(\mathcal{T}_{mon})$ *is a conservative approximation of the convex form RGP* (C_{obj})—(C_7)*. This means that if a vector of values is feasible to* $(\mathcal{T}_{obj})$—$(\mathcal{T}_{mon})$*, then it is also feasible to* (C_{obj})—(C_7)*, and the value of the objective function* $(\mathcal{T}_{obj})$ *is an upper bound for the objective function* (C_{obj}) *[20].*

This means that a solution to robust GP consisting of 2-term lse-constraints is also a solution to the original robust GP, but the objective function is no longer minimized.

Depending on the exact formulation of the problem, this approximation may affect the problem. In this specific case, where both the reachability λ and the expected cost κ are explicitly defined, the value of the objective function doesn't matter. If, for example, you don't specify κ, but want the robust GP to calculate an upper bound, you need to objective function be minimized, and thus will be affected by this approximate reduction procedure.

6.3 Approximating a 2-term lse-function by a piecewise-linear function

We can now approximate the 2-term robust GP by a robust Linear Program (robust LP). This is done by replacing every 2-term lse-function in the 2-term robust GP ($\mathcal{T}_{obj}$)—($\mathcal{T}_{mon}$) by a piecewise-linear (pwl-)function. All other constraints are already linear (see theorem 4) and can be copied over to the robust LP. The approximation of a 2-term lse-function is done by the algorithm described in chapter 3 of [20].

A r-term pwl-approximation is given by a matrix $\mathbf{A}_{pwl} \in \mathbb{R}^{r\times 2}$, a vector $\mathbf{b}_{pwl} \in \mathbb{R}^r$ and a approximation error $\epsilon \in \mathbb{R}$. Such that the pwl-function to replace the lse-function $\operatorname{lse}(x, y)$ is given by

$$\begin{aligned} \max\{ \quad & a_{1,1}x + a_{1,2}y + b_1, \\ & \vdots \\ & a_{r,1}x + a_{r,2}y + b_r \quad \}. \end{aligned} \tag{60}$$

The higher the number of terms in the piecewise-linear approximation, the smaller the error becomes. The approximation of a 2-term lse function is independent of its variables.

Theorem 6. *If we have r-term pwl-upper approximation $upper(x, y)$ and r-term lower approximation $lower(x, y)$ then they approximate every 2-term lse-function by*

$$lower(x, y) \leq lse(x, y) \leq upper(x, y), \tag{61}$$

independent of the variables x and y [20].

Theorem 6 has an important corollary: we only have to calculate the approximation for the number of pieces r once, then all lse-functions can be approximated. The upper and lower approximation are also related: the upper approximation is given by adding the approximation error ϵ to the lower approximation. Suppose that (60) is a r-term lower approximation, then the r-term upper approximation is given by

$$\begin{aligned} \max\{ \quad & (a_{1,1} + \epsilon)x + (a_{1,2} + \epsilon)y + (b_1 + \epsilon), \\ & \vdots \\ & (a_{r,1} + \epsilon)x + (a_{r,2} + \epsilon)y + (b_r + \epsilon) \quad \}. \end{aligned} \tag{62}$$

6.3.1 Python implementation of the pwl-approximation method

A Matlab implementation of the algorithm that approximates the lse-function by its lower pwl-approximation by the same authors of [20] exists.[1] We

[1] `http://web.stanford.edu/~boyd/papers/rgp.html`

translated this Matlab implementation to Python. We have extended the implementation so that both the upper and lower piecewise-linear approximations can easily be calculated. Details on how to use this implementation can be found in appendix C.

6.4 Choosing between the upper and lower approximation

Now we have to decide if we replace our lse-constraints with the upper or lower pwl-approximations. Figure 3 shows a top-down plot of the lse-function and the upper and lower 3-term approximations. The areas below the lines are the points (x, y) that are feasible to the ≤ 0 constraints. As the upper approximation lies *above* the lse-function in three dimensions, the feasible area lies *below* the lse-function in this plot. In other words: the upper approximation will pass through the $z = 0$ plane before the lse-function. The lower approximation lies *below* the lse-function in three dimensions, so the feasible area lies *above* the lse-function.

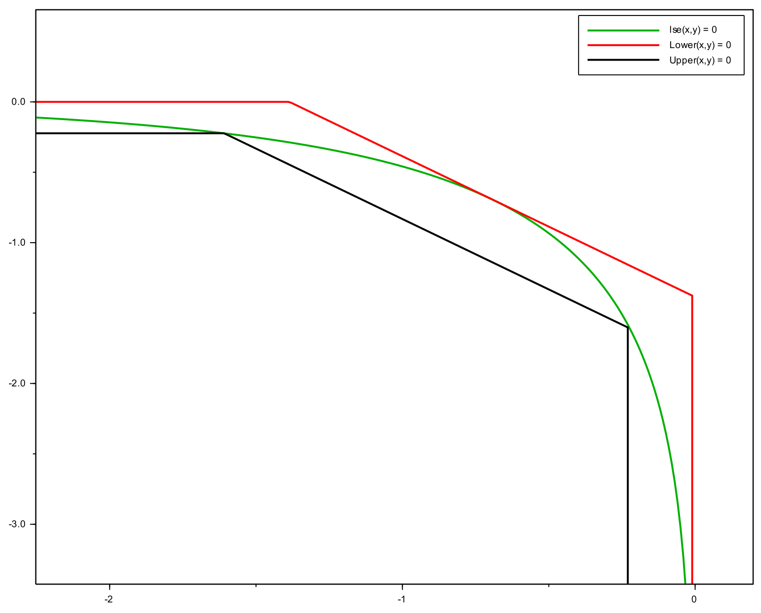


Figure 3: Plot of points satisfying equal-to-0 constraints for the lse-function and the two 3-term approximations.

The choice for upper or lower approximation has implications for the correctness and completeness of our robust GP. If we take the upper approximation, only points below the black line in figure 3 will satisfy the pwl-constraint. Points between the black and green line will not be excluded from the solution space, which means that if the only solutions to the robust GP we approximate lie in that area, we won't find them. So choosing the upper approximation impacts completeness of the robust GP.

On the other hand, if we choose the lower approximation, points below the red line will satisfy the constraint. So we might find a solution that satisfies the lower approximation but not the lse-function, which means this solution is not correct. Of course, the higher the degree of approximation we choose, the smaller the areas between the approximations and the lse-function become.

Theorem 7. *Choosing the upper approximation impacts completeness. Choosing the lower approximation impacts correctness.*

Corollary 7.1. *When choosing the lower approximation, we have to check if the solution we found also satisfies the lse-constraints. For all non-robust constraints, this is trivial: fill in the values of the variables and see if the (in)equalities hold.*

For the robust constraints we have the following theorem, that says that if we have a solution, we only have to verify if that solution holds for $\sup \mathcal{P}$.

Theorem 8. *A solution to the lower approximation satisfies the robust lse-constraints $\forall P \in \mathcal{P}$ if it satisfies the lse-constraint instantiated with* $\sup \mathcal{P}$.

Proof. By proposition (2) we know that satisfying the lse-constraint is equivalent to satisfying the corresponding constraint in posynomial form. We want to verify that if we have a solution $\hat{\eta}^* = (\mathbf{y}^*, \tau^*, \mathbf{z}^*)$ that $\mathbf{x}$ given by $x_i = e^{y_i^*}$ satisfies the robust constraints given by either (RGP_5) or (RGP_7). We have

$$P(s,\alpha,s') \leq \sup \mathcal{P}(s,\alpha,s') \quad \forall P \in \mathcal{P}. \forall s,s' \in S. \forall \alpha \in Act. \tag{63}$$

We also know that all variables $x_i^* \in \mathbb{R}_{++}$, and thus for (RGP_5) we have

$$\frac{\sum\limits_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum\limits_{s' \in S} P(s,\alpha,s') \cdot p_{s'}}{p_s} \tag{64}$$

$$\leq \frac{\sum\limits_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum\limits_{s' \in S} \sup \mathcal{P}(s,\alpha,s') \cdot p_{s'}}{p_s} \tag{65}$$

for all instantiations P. So if

$$\frac{\sum\limits_{\alpha \in Act(s)} \sigma^{s,\alpha} \cdot \sum\limits_{s' \in S} \sup \mathcal{P}(s,\alpha,s') \cdot p_{s'}}{p_s} \leq 1 \tag{66}$$

holds for the values of $\mathbf{x}^*$ we found, then the inequality holds for all $P \in \mathcal{P}$, and thus the constraint (RGP_5) is satisfied. The proof for satisfaction of constraint (RGP_7) is completely analogous to the proof above for (RGP_5). $\square$

6.5 Robust linear programming

We can now form a robust linear program by changing the lse-constraints in ($\mathcal{T}_{obj}$)—($\mathcal{T}_{mon}$) by a piecewise-linear function. For simplicity, we write pwl_r for a pwl-function that approximates the lse-function. The robust linear program is then given by:

$$\begin{aligned}
&\text{minimize} && \hat{\mathbf{c}}^T\hat{\eta} = \tilde{t}, && (\mathcal{L}_{obj})\\
&\text{subject to} && && \\
&\omega = 1,\dots,K_{c_0}-1 && pwl_r((\hat{\mathbf{a}}^1_{c0,\omega})^T\hat{\eta}, (\hat{\mathbf{a}}^2_{c0,\omega})^T\hat{\eta}) \leq 0, && (\mathcal{L}_0)\\
& && \hat{\mathbf{a}}^T_{c_1}\hat{\eta} \leq 0, && (\mathcal{L}_1)\\
& && \hat{\mathbf{a}}^T_{c_2}\hat{\eta} \leq 0, && (\mathcal{L}_2)\\
&\forall s \in S.\ \omega = 1,\dots,K_{c_3}-1 && pwl_r((\hat{\mathbf{a}}^1_{c3,\omega})^T\hat{\eta}, (\hat{\mathbf{a}}^2_{c3,\omega})^T\hat{\eta}) \leq 0, && (\mathcal{L}_3)\\
&\forall s \in S \setminus T.\ \forall P \in \mathcal{P}.\ \omega = 1,\dots,K_{c_5}-1 && pwl_r((\hat{\mathbf{a}}^1_{c5,\omega})^T\hat{\eta}, (\hat{\mathbf{a}}^2_{c5,\omega})^T\hat{\eta}) \leq 0, && (\mathcal{L}_5)\\
&\forall s \in S \setminus G.\ \forall P \in \mathcal{P}.\ \omega = 1,\dots,K_{c_7}-1 && pwl_r((\hat{\mathbf{a}}^1_{c7,\omega})^T\hat{\eta}, (\hat{\mathbf{a}}^2_{c7,\omega})^T\hat{\eta}) \leq 0, && (\mathcal{L}_7)\\
& && \hat{\mathbf{A}}_{c_{mon}}\hat{\eta} + \hat{\mathbf{b}}_{c_{mon}} = \mathbf{0}. && (\mathcal{L}_{mon})
\end{aligned}$$

Robust linear programs can be solved efficiently by existing methods depending on the specific kind of uncertainty set used [4]. Depending on the choice of piecewise-linear approximation, we may need to check whether the solution also holds for the lse-constraints. See theorem 7 and theorem 8. Once solved, we can retrieve the solution to our original robust GP (RGP_{obj})—(RGP_7).

Theorem 9. *If $\hat{\eta}^* = (\mathbf{y}^*, \tau^*, \mathbf{z}^*)$ is a solution to ($\mathcal{L}_{obj}$)—($\mathcal{L}_{mon}$), and also holds for the lse-constraints, then it is an approximate solution to*

$(\mathcal{T}_{obj})$—$(\mathcal{T}_{mon})$. The vector $\mathbf{y}^*$ in this solution satisfies our original convex robust GP given by (C_{obj})—(C_7). Each variable y_i^* in this vector can then be transformed into the variable x_i^* by $x_i^* = e^{y_i^*}$ since we defined $y_i = \log x_i$. The vector $\mathbf{x}^*$ then satisfies all constraints of the posynomial form robust GP, and thus is a solution to (RGP_{obj})—(RGP_7).

What this theorem says, intuitively, is that a solution to the robust LP contains a solution to the convex form robust GP. The variables that form the solution to the convex form robust GP can then be translated back to a solution for the posynomial form robust GP.

A solution to the posynomial form robust GP gives us, by the correctness theorem, a solution to problem 1, if it defines a valid scheduler. This is formalized in the following theorem.

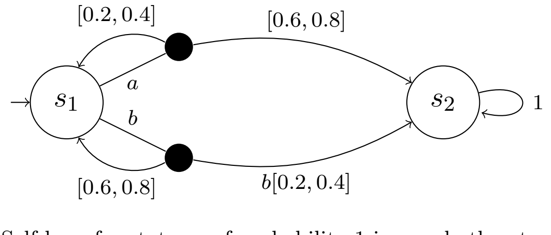
Theorem 10. *If $\mathbf{x}^*$ is a solution to the robust GP (RGP_{obj})—(RGP_7), then we have values for the variables $(c'_s)_{s\in S}$, $(p_s)_{s\in S}$ and $(\sigma^{s,\alpha})_{s\in S,\alpha\in Act}$. If $(\sigma^{s,\alpha})_{s\in S,\alpha\in Act}$ defines a valid scheduler $\sigma \in Sched^{\mathcal{M}}$, then by the correctness theorem (theorem 2) we have a valid scheduler that satisfies the reachability property and expected cost property in problem 1.*

And with this final theorem, we can conclude that with the robust LP $(\mathcal{L}_{obj})$—$(\mathcal{L}_{mon})$, it's possible to find a solution to problem 1, as soon as we know how to solve this robust LP.

6.6 Open problem: the uncertainty set $\mathcal{P}$

All that remains now, is to determine the uncertainty set $\mathcal{P}$ to solve the robust LP. The fact that we use intervals for our uncertain transition function does not mean that we want to solve the $(\mathcal{L}_{obj})$—$(\mathcal{L}_{mon})$ for all possible values in those intervals. Recall definition 8 from the preliminaries: $P \in \mathcal{P}$ if $P(s,\alpha,s') \in \mathcal{P}(s,\alpha,s')$ *and* P forms a valid probability distribution: $\sum_{s'\in S} P(s,\alpha,s') = 1$. Consider the example uMDP in figure 4.

Figure 4: Small example uMDP



Self loop for state s_2 of probability 1 is over both acts.

There are values in the intervals that do not form a valid probability distribution. For example the upper bounds: 0.8 and 0.4 certainly lie in their respective intervals, but sum up to 1.2, thus not forming a valid probability distribution. For the function P with $P(s_1, a, s_1) = 0.4, P(s_1, a, s_2) = 0.8$, we have that P is not a valid transition function, thus $P \notin \mathcal{P}$. So the uncertainty in the robust LP given by $P \in \mathcal{P}$ is not interval uncertainty, which means that interval linear programming [2, 12] cannot be used to solve this robust LP without breaking the correctness proof. What specific type of uncertainty this is, and consequently how to solve the robust LP, remains open.

Chapter 7

Conclusions

We have shown how a robust geometric program can be used to solve multi-objective verification for uncertain Markov decision processes where the uncertainty is given by intervals. This approach is correct, but not complete. Since robust GPs cannot be solved efficiently, we use the approximation method proposed by Hsiung, Kim and Boyd. This results in a robust linear program, which for certain types of uncertainty can be solved efficiently. However, we have not identified the type of uncertainty used in our robust LP. The approximation method used is independent of the uncertainty used, so our approach can be applied to other kinds of uMDPs.

7.1 Future work

We have shown how problem 1 can be reduced to solving a robust LP. If (and how) this robust LP can be solved remains an open problem until the uncertainty set $\mathcal{P}$ is identified as a known uncertainty set like polyhedral or ellipsoidal uncertainty. In case the robust LPs with that uncertainty set cannot be solved, future research can consider different kinds of uncertainty in the uMDP, which would result into a different uncertainty set for the robust LP.

Bibliography

[1] MOSEK ApS. *The MOSEK optimization toolbox for MATLAB manual. Version 8.1.*, 2017.

[2] H. A. Ashayerinasab, H. M. Nehi, and M. Allahdadi. Overview of solution methods for solving interval linear programming and new method. In *2015 4th Iranian Joint Congress on Fuzzy and Intelligent Systems (CFIS)*, pages 1–5, Sept 2015.

[3] Christel Baier and Joost-Pieter Katoen. *Principles of model checking.* The MIT Press, 2008.

[4] A. Ben-Tal, Laurent El Ghaoui, and Nemirovskiĭ Arkadiĭ Semenovich. *Robust optimization.* Princeton University Press, 2009.

[5] Stephen Boyd, Seung-Jean Kim, Lieven Vandenberghe, and Arash Hassibi. A tutorial on geometric programming. *Optimization and Engineering*, 8(1):67, Apr 2007.

[6] Stephen Boyd and Lieven Vandenberghe. *Convex optimization.* Cambridge University Press, 2015.

[7] Tomáš Brázdil, Krishnendu Chatterjee, Martin Chmelík, Vojtěch Forejt, Jan Křetínský, Marta Kwiatkowska, David Parker, and Mateusz Ujma. Verification of markov decision processes using learning algorithms. In Franck Cassez and Jean-François Raskin, editors, *Automated Technology for Verification and Analysis*, pages 98–114, Cham, 2014. Springer International Publishing.

[8] Edward Burnell and Warren Hoburg. Gpkit software for geometric programming. `https://github.com/convexengineering/gpkit`, 2017. Version 0.6.0.

[9] André Chassein and Marc Goerigk. Robust geometric programming is co-np hard. 2014.

[10] Murat Cubuktepe, Nils Jansen, Sebastian Junges, Joost-Pieter Katoen, Ivan Papusha, Hasan A. Poonawala, and Ufuk Topcu. *Sequential Con-*

vex Programming for the Efficient Verification of Parametric MDPs, pages 133–150. Springer Berlin Heidelberg, Berlin, Heidelberg, 2017.

[11] Joachim Dahl and Lieven Vandenberghe. CVXOPT: A python package for convex optimization, 2008.

[12] Huiling Duan and Tao Peng. Numerical solution of interval linear programming. *International Mathematical Forum*, 9:1333–1340, 2014.

[13] Inc. Gurobi Optimization. Gurobi optimizer reference manual, 2016.

[14] Ernst Moritz Hahn, Vahid Hashemi, Holger Hermanns, Morteza Lahijanian, and Andrea Turrini. Multi-objective robust strategy synthesis for interval markov decision processes. In *QEST*, volume 10503 of *Lecture Notes in Computer Science*, pages 207–223. Springer, 2017.

[15] Ernst Moritz Hahn, Holger Hermanns, Björn Wachter, and Lijun Zhang. Param: A model checker for parametric markov models. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 660–664, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[16] Ernst Moritz Hahn, Holger Hermanns, and Lijun Zhang. Probabilistic reachability for parametric markov models. *International Journal on Software Tools for Technology Transfer*, 13(1):3–19, Jan 2011.

[17] Vahid Hashemi, Holger Hermanns, and Lei Song. Reward-bounded reachability probability for uncertain weighted mdps. In Barbara Jobstmann and K. Rustan M. Leino, editors, *Verification, Model Checking, and Abstract Interpretation*, pages 351–371, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.

[18] D. Henriques, J. G. Martins, P. Zuliani, A. Platzer, and E. M. Clarke. Statistical model checking for markov decision processes. In *2012 Ninth International Conference on Quantitative Evaluation of Systems*, pages 84–93, Sept 2012.

[19] Kan-Lin Hsiung, Seung-Jean Kim, and Stephen Boyd. A matlab implementation for best approximation of two-term log-sum-exp function, 2006.

[20] Kan-Lin Hsiung, Seung-Jean Kim, and Stephen Boyd. Tractable approximate robust geometric programming. *Optimization and Engineering*, 9(2):95–118, Jun 2008.

[21] Marta Kwiatkowska, Gethin Norman, and David Parker. Prism: Probabilistic symbolic model checker. In Tony Field, Peter G. Harrison,

Jeremy Bradley, and Uli Harder, editors, *Computer Performance Evaluation: Modelling Techniques and Tools*, pages 200–204, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[22] Alberto Puggelli, Wenchao Li, Alberto L. Sangiovanni-Vincentelli, and Sanjit A. Seshia. Polynomial-time verification of pctl properties of mdps with convex uncertainties. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, pages 527–542, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: an introduction.* The MIT Press, 2012.

[24] E. M. Wolff, U. Topcu, and R. M. Murray. Robust control of uncertain markov decision processes with temporal logic specifications. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 3372–3379, Dec 2012.

[25] Di Wu and Xenofon Koutsoukos. Reachability analysis of uncertain systems using bounded-parameter markov decision processes. *Artificial Intelligence*, 172(8):945 – 954, 2008.

[26] Ihsan Yanikoglu. Robust optimization methods for chance constrained, simulation-based, and bilevel problems. 2014.

Appendix A

Details on the counter example

See the accompanying `counter_example.py` for the code used to get the results in table 1. In order to run, the Python 2.7 is required, using the following packages.

Package	Version used
Numpy	1.14.2
Pint	0.8.1
Scipy	1.0.1
Cvxopt	1.1.9
GPkit	0.7.0.0

To reproduce the results in the table (and for other δ) run the code with a chosen value for δ. `Tight.reltol` controls the tolerance for the `Tight` function.

Appendix B

Details on the robust GP reformulation

Appendix C of [20] states that the standard convex form robust GP

$$\begin{aligned}
\text{minimize} \quad & \mathbf{c}^T\mathbf{y}, \\
\text{subject to} \quad & \sup_{u\in\mathcal{U}} \operatorname{lse}(\tilde{\mathbf{A}}_i(u)\mathbf{y} + \tilde{\mathbf{b}}_i(u)) \leq 0, \quad i = 1,\dots,m, \\
& \mathbf{G}\mathbf{y} + \mathbf{h} = \mathbf{0},
\end{aligned} \tag{67}$$

where $\tilde{\mathbf{A}}_i(u) = \mathbf{A}_i^0 + \sum_{j=1}^L u_j\mathbf{A}_i^j$, and $\tilde{\mathbf{b}}_i(u) = \mathbf{b}_i^0 + \sum_{j=1}^L u_j\mathbf{b}_i^j$ is equivalent to the robust GP

$$\begin{aligned}
\text{minimize} \quad & \bar{\mathbf{c}}^T \begin{bmatrix} \mathbf{y} \\ \tau \end{bmatrix}, \\
\text{subject to} \quad & \sup_{u\in\mathcal{U}} \operatorname{lse}((\tilde{\mathbf{A}}_i^0 + \sum_{j=1}^L u_j\tilde{\mathbf{A}}_i^j) \begin{bmatrix} \mathbf{y} \\ \tau \end{bmatrix}) \leq 0, \\
& \qquad\qquad i = 1,\dots,m, \\
& \bar{\mathbf{G}} \begin{bmatrix} \mathbf{y} \\ \tau \end{bmatrix} + \bar{\mathbf{h}} = \mathbf{0},
\end{aligned} \tag{68}$$

where $\tilde{\mathbf{A}}_i^j = \begin{bmatrix} \mathbf{A}_i^j & \mathbf{b}_i^j \end{bmatrix}$. This can then readily be rewritten as

$$\begin{aligned}
&\text{minimize} && \bar{\mathbf{c}}^T \begin{bmatrix} \mathbf{y} \\ \tau \end{bmatrix}, \\
&\text{subject to} && \sup_{u \in \mathcal{U}} \operatorname{lse}((\bar{\mathbf{a}}_{\mathbf{i1}} + \bar{\mathbf{B}}_{\mathbf{i1}} u)^T \begin{bmatrix} \mathbf{y} \\ \tau \end{bmatrix}, \ldots, (\bar{\mathbf{a}}_{\mathbf{iK_i}} + \bar{\mathbf{B}}_{\mathbf{iK_i}} u)^T \begin{bmatrix} \mathbf{y} \\ \tau \end{bmatrix} \leq 0, \\
& && \qquad i = 1, \ldots, m \qquad (69) \\
& && \bar{\mathbf{G}} \begin{bmatrix} \mathbf{y} \\ \tau \end{bmatrix} + \bar{\mathbf{h}} = \mathbf{0}.
\end{aligned}$$

For details on the variables, see appendix C of [20].

We transform our robust GP from (67) to (68), which is equivalent to (69). In our case, it is easier to transform the robust GP of the form (68) into a two-term robust GP. Since all these robust GPs are equivalent, all theorems of [20] still hold.

Appendix C

Details on the lse-function approximation

Our Python implementation of the algorithm that approximates the lse-function by either a piecewise-linear upper or lower approximation can be found in the file `lseapprox.py`. There are two main functions, `upperapprox(r)` and `lowerapprox(r)`, that provide Numpy ndarrays A and b, and a float $approx_err$ that gives the approximation error. These functions describe the coefficients for the r-term upper and lower pwl-approximation respectively.

The implementation is based on the Matlab function provided with [19] and the algorithm described in [20]. Running our implementation of the algorithm requires Python 3.6 and the Numpy package.