Markov Decision Processes With Uncertain Parameters

DISSERTATION

zur Erlangung des Grades eines

DOKTORS DER NATURWISSENSCHAFTEN

der Technischen Universität Dortmund
an der Fakultät für Informatik
im Rahmen des Graduiertenkollegs 1855 „Diskrete Optimierung technischer Systeme unter Unsicherheit“ der DFG

von
Dimitri Scheftelowitsch

Dortmund
2018

Tag der mündlichen Prüfung: 3. Mai 2018

Dekan: **Prof. Dr.-Ing. Gernot A. Fink**

Gutachter: **Prof. Dr. Peter Buchholz** (Technische Universität Dortmund), **Prof. Dr. Holger Hermanns** (Universität des Saarlandes)

Contents

List of Figures

List of Algorithms

List of Tables

Acknowledgments

Here I would like to mention all the people who have made this work possible. I am particularly thankful to Prof. Dr. Peter Buchholz for his comprehensive support and advice and to Prof. Dr. Holger Hermanns for a very productive month in Saarbrücken. Individual thanks goes also to my mentor who has given useful general hints on research structure and scientific procedures, Prof. Dr. Thomas Schwentick.

I also want to thank Iryna Dohndorf and Jan Kriege for the comments on the various aspects of my thesis, productive discussions, and for being patient and fun officemates.

Furthermore, I would like to thank my friends for their support during the time. Specifically, I want to thank Klaus Aehlig and Moritz von Looz for the opportunities to co-operate in giving courses on games on graphs and Markov chains, respectively. I also would like to mention the various discussions on PhD-related topics, different scientific approaches and academic life in general with Carsten Burgard, Gottfried Herold, and Lucie Plaga.

Last but not least, I want to thank my family for their faith in me, and their advice and support over the whole course of my life and education until now.

Abstract

Markov decision processes model stochastic uncertainty in systems and allow one to construct strategies which optimize the behaviour of a system with respect to some reward function. However, the parameters for this uncertainty, that is, the probabilities inside a Markov decision model, are derived from empirical or expert knowledge and are themselves subject to uncertainties such as measurement errors or limited expertise. This work considers second-order uncertainty models for Markov decision processes and derives theoretical and practical results.

Among other models, this work considers two main forms of uncertainty. One form is a set of discrete *scenarios* with a prior probability distribution and the task to maximize the expected reward under the given probability distribution. Another form of uncertainty is a continuous *uncertainty set* of scenarios and the task to compute a policy that optimizes the rewards in the optimistic and pessimistic cases.

The work provides two kinds of results. First, we establish complexity-theoretic hardness results for the considered optimization problems. Second, we design heuristics for some of the problems and evaluate them empirically. In the first class of results, we show that additional model uncertainty makes the optimization problems harder to solve, as they add an additional party with own optimization goals. In the second class of results, we show that even if the discussed problems are hard to solve in theory, we can come up with efficient heuristics that can solve them adequately well for practical applications.

1 Introduction

No decision is also a decision, and almost always the worst possible one.

— *Folklore*

MOST aspects of everyday life are, in one way or another, connected to *decisions*, i. e., choices between several alternative *actions*. Simplest examples include the decision which book to buy, what to wear, whether to repair a slight malfunction in one's car or not etc. One challenging aspect of these choices is that the direct consequences lie in the future and cannot be foreseen from the time point of the decision. Another challenge lies in the mere fact of different possible outcomes some of which may be more preferential to the decision making agent than others.

In the domain of philosophy and economics, investigation of different aspects of choice is known as *decision theory* [Han95]. From the perspective of decision theory, to solve the second challenge means to quantify all possible outcomes with a *value function* which not only provides an order of preferences, but also numerical values which correspond to *rewards* for each outcome. These rewards can correspond to financial benefits, but they may also represent some other, abstract units of utility. Then, given a function that maps possible actions to their respective outcomes, one can compute the optimal action that maximizes the composition of the outcome and value functions. The unpredictability of the outcomes can be modeled by means of probability theory, designing the outcome as a random variable.

In this setting, different mathematical formalisms can be considered which yield different decision models. The models may vary in the value function, the number of agents or other parameters. For example, one can think of an adversary who may make decisions on her own in order to pursue her own goals which are orthogonal to the goals of the agent whose actions one wants to optimize. A fairly popular mathematical model for one-party decision making under stochastic uncertainty is the *Markov decision process* (MDP) formalism [Put94, Kal16] which, informally, encompasses a notion of a controlled Markov chain with additional rewards which are paid out depending on the current state of the chain. The benefits of this formalism are twofold: First, the formalism itself allows one to model practical applications in a direct manner, and, second, efficient algorithms exist which compute optimal *policies*, i. e., sequences of actions in dependence on the current state.

What we want to do in this work is to consider additional notions of *model uncertainty* in Markov decision processes. Briefly, model uncertainty deals with the case where the parameters of the underlying mathematical model are not known exactly. In the case of decision theory, this means that the outcomes of a specific action are uncertain in the sense that even the probability distribution that defines the outcome is not precisely known. From the mathematical viewpoint, our desire is to formalize this notion of model uncertainty and find algorithms which compute policies that are *robust* against this uncertainty (for a well-defined notion of robustness).

1.1 Motivation

In order to motivate additional model uncertainty in Markov decision models, we provide an example.

Suppose a fictional company, *Electric Till Corporation* (ETC [Ste00]), is employing large computer systems for large-scale parallel computing tasks such as deep learning, computer vision and text recognition. These computer systems consist of a number of similar components which are bought from one supplier, *Forsooth Heavy Industries* (FHI) [vL13]. As the reader may know from her own experience, computer systems may suddenly fail and generally require regular maintenance, and thus ETC needs a maintenance schedule for its computer systems which allocates (limited) maintenance personnel to the machinery, judging from typical failure times and signs of malfunctioning.

Mathematically, up to this point, this problem can be formulated as a Markov decision process which can be solved with existing methods [Put94, Kal16]. However, in our case, the supplier, FHI, cannot provide constant quality, be it for fabrication process features or varying QA standards in the supply chain [WBG06]. In any case, some of the supplied computers may fail early while some others may not require maintenance for a longer time than initially planned. This means that the behaviour, and thus the mathematical model of a single component, is subject to an uncertainty which cannot be resolved a priori by means of the model itself. Considering only the average performance also may not help here, as the costs of missing a failure may be prohibitively high compared to the costs of extra maintenance.

From this problem formulation, the ETC house expert whose task is to devise a maintenance schedule can pursue two different paths. On the one hand, she can consider several archetypal behaviour scenarios of the supplied hardware, such as fast failure, low maintenance, or need for frequent maintenance yet low failure rates if maintenance is regular. The difficulty here lies in possible incomparability of the scenarios: an optimal strategy for one scenario may behave badly in the other scenario. In this setting, she can search for a maintenance policy that accounts for the relative frequency of the possible scenarios.

On the other hand, the decision maker can, from her observations, define a continuous space of possible modes of operation which also can account for possibly unseen yet probable behaviours. The main difference here is that the number of possible scenarios is infinite and the worst and best scenario are not necessarily a priori known. Here, multiple optimization goals are possible. The expert may search for the policy which is best in the pessimistic case, possibly at the cost of smaller maintenance intervals and higher load on the maintenance personnel, she also may search for a policy which is best in the optimistic case, cutting maintenance costs. Or she may search for a compromise between the two and possibly other scenarios in pursuit for a policy which behaves well in all cases without sacrificing too much.

In both cases, modeling is performed by admitting uncertainty at the level of the model; the uncertainty cannot be expressed with the means of the initial modeling formalism. In the language of Markov decision processes as base model, model uncertainty means uncertainty in the transition probabilities between states and the rewards. Especially the uncertainty in the transition probabilities offers a challenge both from the modeling and the algorithmic aspects, as the uncertainty has to be formalized in a suitable mathematical model and the policy optimization problems in these models are different from those known in the MDP literature. Knowing what algorithms are suitable and what their efficiency limits in terms of time complexity are is important, both from the theoretical point of view as well as from that of the ETC's expert.

This thesis aims at solving these challenges from several different directions. First, we discuss mathematical formalisms that capture model uncertainty in Markov decision processes. Second, we derive complexity-theoretic results which establish lower and upper bounds for the respective optimization problems. Third, we design practically applicable algorithms for the two main problems: the multi-scenario optimization problem and the multi-objective Pareto frontier enumeration problem. Fourth, we derive a complex maintenance and repair model and its interpretations in both perspectives on uncertainty and apply our algorithms on it.

1.2 Structure of the thesis

Beside the current chapter which serves as an introduction into the topic and motivates further research, this thesis consists of two significant parts that represent two main research directions.

In the first part, Chapter 2, the theoretical properties of parameter uncertainty on the complexity of Markov decision problems are explored. We consider state-of-the-art work and models such as those defined in [GLD00, FV97, SL73, WED94, DM10] and present own research on the matter. The own research part concentrates around hardness results which show algorithmic limits of finite-horizon and multi-objective optimization approaches and of related uncertainty models. This contribution is based on the conference article [Sch15], the theoretical parts of [SBHH17] and [BS17a], and some previously unpublished remarks.

In the second part, Chapter 3, we present approaches to multi-objective optimization perspectives for uncertain Markov decision models. There, we present algorithms for various formulations of the multi-objective approach, their properties, and their experimental evaluation. This part is based on the *practical* parts of [SBHH17] and [BS17a]. In contrast to the previous chapter, the main results are empirical in their nature; the algorithms that are presented are judged by their performance on random data sets. The reasons for that lie in lack of efficient algorithms for the underlying problems, be it for theoretical hardness or for general lack of theoretical results.

The third part, Chapter 4, contains an application of the algorithms designed in the previous chapter. While in chapter 3, the main emphasis was on performance of the individual algorithms, here, we consider a complete path from an abstract problem formulation to the solution. In detail, we consider a model of composed components that degrade individually but can be repaired. In the model, the number of maintenance workers is limited, which limits the number of components that can be in maintenance mode simultaneously. We design uncertain Markov decision models which capture this behaviour and apply our algorithms to them.

Finally, in Chapter 5, we summarize and discuss the presented results. Specifically, we consider applications and possibilities to extend the methods to other problems and settings; furthermore, we consider possible future work on other problems in the general setting of MDPs under uncertainty.

1.2.1 Personal contribution

One of the three publications that serve as a basis for this work, namely [SBHH17], has been completed in co-operation with other researchers. My individual contribution to this publication is the following.

- Algorithm 11 along with the optimality proof (Lemma 2.4.5, Theorem 2.4.6),
- Algorithm 15,
- Algorithmic optimizations and parallelization of Algorithm 16,
- Evaluation of Algorithm 16.

1.3 Definitions and notation

Prior to discussing formalisms and results, we define the mathematical concepts that will be used in this thesis. The main purpose of this section is two-fold: from the mathematical point of view, we introduce the concepts that we base our results on; from a "technical" point of view, we introduce notation and identifiers. Concerning mathematical notation, we adhere to the following conventions.

- $\mathbb{N}$ is the set of natural numbers, that is $\{1, 2, \ldots\}$. The set of natural numbers with zero is written as $\mathbb{N}_0$.
- $\mathbb{R}$ is the set of real numbers. Non-negative reals are designated with $\mathbb{R}_{\geqslant 0}$.
- For $n \in \mathbb{N}$, we abbreviate the set $\{1, \ldots, n\}$ by $[n]$.
- If X is a set, then $\mathcal{P}(X)$ is the power set of X.
- For sets A, B, the set $A \uplus B$ is the *disjoint union* of A and B, a set with the properties $(A \uplus B) \setminus A = B$ and $(A \uplus B) \setminus B = A$.
- Identifiers like $\vec{v}$, denote vectors; if $\vec{v}$ is an n-dimensional vector, then $\vec{v}(1), \ldots, \vec{v}(n)$ are the entries of $\vec{v}$.
- Matrices are written in bold script, such as $\boldsymbol{M}$. The dimensions of a matrix $\boldsymbol{M}$ are $p \times q$ if $\boldsymbol{M}$ has p rows and q columns.
- For matrices $\boldsymbol{M} \in \mathbb{R}^{p \times q}$ with dimensions $p \times q$ (i. e., with p rows and q columns), we designate the row vectors of $\boldsymbol{M}$ with the notation $\boldsymbol{M}(1\bullet), \ldots, \boldsymbol{M}(p\bullet)$. The column vectors of $\boldsymbol{M}$ are designated by $\boldsymbol{M}(\bullet 1), \ldots, \boldsymbol{M}(\bullet q)$; the individual entries are designated with $\boldsymbol{M}(i, j)$ for $i \in [p], j \in [q]$.
- For square matrices $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, the dimension of $\boldsymbol{M}$ is denoted by $\dim M := n$.
- Special vectors are $\vec{1}$ and $\vec{0}$ which are column vectors of ones resp. zeros and $\vec{e}_i$, column basis vectors with a one in the i-th position and zeros everywhere else. When the dimension is ambiguous, it is stated explicitly.
- The scalar product between two vectors $\vec{u}, \vec{v} \in \mathbb{R}^n$, is designated by $\vec{u} \cdot \vec{v}$ and evaluates to $\sum_{i \in [n]} \vec{u}(i)\vec{v}(i)$.
- By *stochastic* matrices and vectors we designate non-negative matrices $\boldsymbol{A}$ and vectors $\vec{v}$ with unit (row) sum, that is, $\boldsymbol{A}\vec{1} = \vec{1}$ and $\vec{v}\vec{1} = 1$.
- Special matrices are $\boldsymbol{I}$, the identity matrix and $\boldsymbol{0}$, the zero matrix. Where ambiguity may arise, a subscript will denote the dimensions of the matrix.
- The operator $\otimes$ denotes the Kronecker product, that is, for two matrices $\boldsymbol{A} \in \mathbb{R}^{m \times n}, \boldsymbol{B}$, the expression $\boldsymbol{A} \otimes \boldsymbol{B}$ is

$$\begin{pmatrix} \boldsymbol{A}(1,1)\boldsymbol{B} & \ldots & \boldsymbol{A}(1,n)\boldsymbol{B} \\ \vdots & \ddots & \vdots \\ \boldsymbol{A}(m,1)\boldsymbol{B} & \ldots & \boldsymbol{A}(m,n)\boldsymbol{B} \end{pmatrix}.$$

- The Kronecker sum, denoted by $\oplus$, defines, for two square matrices $\boldsymbol{A}, \boldsymbol{B}$ the expression $\boldsymbol{A} \otimes \boldsymbol{I}_{\dim \boldsymbol{B}} + \boldsymbol{I}_{\dim \boldsymbol{A}} \otimes \boldsymbol{B}$.

- For two functions $f, g \colon \mathbb{N} \to \mathbb{N}$, it is $f(n) = \mathcal{O}\left(g(n)\right)$ if there exist $c \in \mathbb{R}_{\geqslant 0}, N \in \mathbb{N}$ such that for all natural numbers n with $n > N$ it is $\frac{f(n)}{g(n)} < c$.
- For a set A, its *Kleene closure* A^* is the set of all finite sequences of values from A, that is, finite tuples $(a_1, a_2, \ldots, a_n)$ with $n \in \mathbb{N}_0$ and $a_i \in A$. This especially includes the empty tuple ε. The set of all non-empty sequences over A is then given by $A^+ = A^* \setminus \{\varepsilon\}$.

1.3.1 Useful concepts of probability theory

As major parts of our contribution will, in one way or another, be dealing with probabilities, we briefly introduce the most important parts of probability theory we use.

Definition 1.3.1 (Probabilities)**.** Let Ω be a set, $\mathcal{A} \subset \mathcal{P}(\Omega)$ and $\mathbb{P} \colon \mathcal{A} \to \mathbb{R}$. The triple $(\Omega, \mathcal{A}, \mathbb{P})$ is a *probability space* if the following conditions are met.

- $\varnothing, \Omega \in \mathcal{A}$.
- If $X \in \mathcal{A}$, then also $\Omega \setminus X \in \mathcal{A}$
- If $X_i \in \mathcal{A}$ holds for each X_i in the sequence $(X_i)_{i \in \mathbb{N}}$, then $\cup_{i \in \mathbb{N}} X_i \in \mathcal{A}$.
- $\mathbb{P}(\Omega) = 1, \mathbb{P}(\varnothing) = 0$.
- If $X_i \in \mathcal{A}$ holds for each X_i in the sequence $(X_i)_{i \in \mathbb{N}}$ and $X_i \cap X_j = \varnothing$ for all $i, j \in \mathbb{N}$, then $\mathbb{P}(\cup_{i \in \mathbb{N}} X_i) = \sum_{i \in \mathbb{N}} \mathbb{P}(X_i)$.

The first three conditions ensure that the set of *events* $\mathcal{A}$ is a *σ-algebra* on Ω, and the last two conditions define $\mathbb{P}$ as a *probability measure* on $\mathcal{A}$.

Definition 1.3.2 (Conditional probability)**.** For a probability space $(\Omega, \mathcal{A}, \mathbb{P})$ and two events A, B, the *conditional probability of A given B*, denoted $\mathbb{P}\left(A \mid B\right)$ is $\frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$.

In the sequel, we shall often adhere to the notation $\Pr[A]$, which designates the probability of some event A, given as a logical expression. This notation is short for $\mathbb{P}$(set of events where A is true) in a suitable probability space.

Definition 1.3.3 (Random variable)**.** A *random variable* is a function $X \colon \Omega \to \mathbb{R}$ where $(\Omega, \mathcal{A}, \mathbb{P})$ is a probability space. Often a random variable is defined by introducing a *probability density function* $f_X \colon B \to \mathbb{R}$ such that $B \subseteq \mathbb{R}$, $\int_B f_X(x)\,\mathrm{d}x = 1$ and $\Pr[X \in M] = \mathbb{P}(M) = \int_M f_X(x)\,\mathrm{d}x$.

In our discussions, we mostly skip the formal foundations and use a more simple notation. If the probability space is obvious from the given context, we write $\Pr[X]$ for the probability of the event X. In some cases, to clarify that a specific probability space $(\Omega, \mathcal{A}, \mathbb{P})$ is to be used, we use the notation $\Pr_{X \in \mathcal{A}}[\cdot]$, or, if the context associates a unique identifier x with a probability space, $\Pr_x[\cdot]$.

For the sake of completeness, we introduce Markov chains as the basis for generalized Markov models. The introduction of Markov chains also introduces some terminology that will be used in this thesis.

Definition 1.3.4 (Discrete-time Markov chain)**.** Let $n \in \mathbb{N}$, $S = [n]$ be a set of states, $\vec{q} = \left(q_1, \ldots, q_n\right) \in \mathbb{R}^n$ a stochastic vector, and $\boldsymbol{P} \in \mathbb{R}^{n \times n}$ a stochastic matrix. Then the sequence $(X_t(S, \boldsymbol{P}, \vec{q}))_{t \in \mathbb{N}}$ of random variables on S is defined as follows.

$$\begin{aligned} \Pr\left[X_1(S, \boldsymbol{P}, \vec{q}) = s\right] &= q_s \\ \Pr\left[X_{i+1}(S, P, \vec{q}) = s \mid X_i(S, \boldsymbol{P}, \vec{q}) = s'\right] &= \boldsymbol{P}(s', s) \end{aligned} \tag{1.1}$$

We call such a sequence a *Markov chain on S with transition probability matrix $\mathbf{P}$ and initial distribution $\vec{q}$.*

The individual states in a Markov chain can be classified with respect to their reachability properties. We call a state i in a Markov chain

- *absorbing*, if the transition probability (or rate) to a state $j \neq i$ from state i is zero,
- *reachable* from state j, if there is a path with nonzero transition probability (rate) from state i,
- *recurrent*, if the probability to return to i is 1,
- *transient*, if i is not recurrent.

1.4 Basic concepts of computational complexity theory

In this work, we also discuss computational complexity aspects of some problems which arise in the discussion of bounded-parameter Markov decision processes. In order to do this, we need to introduce several terms which help us to define the complexity of a problem precisely. A more thorough introduction can be found e.g., in the book of Arora and Barak [AB09], here, we concentrate on the important results.

Complexity theory deals, in general, with the resource requirements that are imposed when a certain problem has to be solved. To establish upper and lower bounds for needed resources, we need to formally define the notion of a problem. Intuitively, we associate a problem with deciding if an element x of some larger set X is also an element of a subset $Y \subseteq X$. For convenience, our definition imposes more structure upon X.

Definition 1.4.1 (Languages and problems)**.** Let Σ be a finite set. We then call Σ an *alphabet* and $L \subseteq \Sigma^*$ a *formal language* over Σ which may contain finite *words* of the form $w = \sigma_1\sigma_2 \ldots \sigma_n$ such that $\sigma_i \in \Sigma, i \in [n], |w| := n$. The *word problem* for a given language L is the task to decide, for a word $w \in \Sigma^*$, if $w \in L$. Furthermore, we define a *computational problem* to be a language L together with the word problem for L.

In the future, we use less formal language to discuss problems, however, if we talk about a problem, we implicitly assume that it can be stated as a word problem; thus, problems are by default *decision* problems. It is easy to see that this view is limited and does not (obviously) capture usual computational tasks such as computing a matrix product. However, this perspective is still sufficiently expressive: it is possible to restate almost all known problems as decision problems by asking a sequence of binary decision questions about each bit of the output.

The central notion in computational complexity is the idea of a *reduction*. Reductions introduce a partial order on problems and allow one to meaningfully compare problems with respect to their hardness and to say things like "Problem X is not harder than problem Y".

Definition 1.4.2 (Reduction)**.** Let L_1, L_2 be languages over alphabets Σ_1, Σ_2. A *reduction* from L_1 to L_2 is a function $f\colon \Sigma_1^* \to \Sigma_2^*$ that satisfies $w \in L_1 \Leftrightarrow f(w) \in L_2$.

To define complexity, i. e., the amount of needed resources, we now face the problem of picking a machine model on which the time and space requirements are measured. It is not obvious that machine models have similar expressive power and it is even less obvious that same problems require similar space and time on different machine models. However, all general-purpose computation models (except for possibly quantum computers [Sho94]) do not falsify the (extended) *Church-Turing thesis* [vEB90].

Conjecture 1.4.1 (Extended Church-Turing thesis)**.** The intuitive notion of computability is equivalent to the formal notion of computability on a Turing machine; furthermore, every deterministic machine model can be simulated on a Turing machine with at most polynomial space and time costs, measured as function of input word length $|w|$.

Hence, we assume our machine model to be a random access machine (RAM) with constant costs for any arithmetic and memory access operation. The polynomial slowdown mentioned in the extended Church-Turing thesis also motivates a notion of "acceptable" or "efficient" complexity: we consider a problem to be efficiently solvable if the computational problem can be decided in polynomial time, as polynomial speedups and slowdowns are due to machine model.

Definition 1.4.3 (Polynomial reductions and the class P)**.** Let L_1, L_2 be languages. If there exists a reduction f from L_1 to L_2 which can be computed in polynomial time with respect to input size, then we write $L_1 \leqslant_p L_2$ and say that f is a *polynomial reduction*.

The class P ("polynomial") is the class of all problems that can be solved in polynomial time.

Often, however, problems are not known to be in P. Sometimes this can be proven by showing exponential lower bounds; in some cases, there are no obvious hints. A large number of interesting and relevant computational problems are known to belong to NP, a superset of P which is similar to P insofar as it extends P by non-deterministic additional information, or advice. Intuitively, NP allows one to "guess" an advice string of at most polynomial length with which, then, the problem can be solved in polynomial time.

Definition 1.4.4 (Non-determinism and the class NP)**.** The class NP ("non-deterministically polynomial") is the class of all problems that can be solved with a non-deterministic RAM in polynomial time, i.e., a language L over Σ is in NP if and only if for each $x \in L$ there is a $y \in \Sigma^*$, $|y| = |x|^{\mathcal{O}(1)}$ such that the language $L' = (x, y)$ is in P.

The classes P and NP are important in the sense that P captures the problems which can be solved efficiently while the class NP captures the problems for which a solution can be efficiently *verified*. NP obviously contains P but there are no results that imply P = NP or P $\neq$ NP. As previously noted, there are several important and interesting computational problems in NP for which no polynomial algorithm is known; furthermore, some of these are problems which are as complex as any other problem in NP. This notion is formalized with the help of reductions.

Definition 1.4.5 (Hardness and completeness)**.** Let L be a language, F a set of reductions, and $\mathcal{C}$ any complexity class. If, for any $L' \in \mathcal{C}$ there is a reduction $f \in F$ from L' to L, then L is said to be *$\mathcal{C}$-hard under F*; if L is $\mathcal{C}$-hard and $L \in \mathcal{C}$, then L is said to be *$\mathcal{C}$-complete under F*.

We restrict ourselves to the class of polynomial reductions, so, whenever we argue about, say, NP-hardness, then we mean NP-hardness under polynomial reductions.

Many decision versions of mathematically interesting combinatorial problems are NP-complete; among others, deciding if a graph contains a Hamiltonian path, a cycle of at most given weight, or a complete subgraph of a given size, deciding if a set of integers has a subset whose sum has a given value, or deciding if a Boolean formula is satisfiable [Kar72]. For several problems for bounded-parameter Markov decision processes, we show that they are NP-hard or NP-complete, too, by reducing from known NP-hard problems. Many years of research did not deliver a polynomial-time algorithm for any of these and many other NP-complete problems. This gives a reason to assume that P $\neq$ NP implying that there is no algorithm that can efficiently solve all instances of any NP-hard problem.

It is worth noting that being NP-hard does not make a problem intractable in practice; many NP-hard problems such as Boolean satisfiability or traveling salesperson problems allow for heuristics that can solve many large instances efficiently [MZ09, Hel00]; however, a large worst-case lower bound implies at least potential problems and prohibitively slow computation of exact solutions on some large instances. In this work, we see that some of the considered problems are NP-hard or NP-complete which is in many cases a sufficient justification to switch to generic methods such as mathematical programming and use mathematical programming solvers as subroutines. Such subroutines are called *oracles* in the complexity theory world and with the help of oracles, more important problem classes can be captured.

Definition 1.4.6 (Oracles and Σ_2^p)**.** Let L, L' be languages and $\mathcal{C}$ a complexity class. If L can be decided with an algorithm that calls to a subroutine that decides L', and the complexity of the algorithm except for the call to this subroutine is in $\mathcal{C}$, then L can be decided with a $\mathcal{C}^{L'}$ algorithm, a *$\mathcal{C}$ algorithm with an oracle for L'*.

The class $\mathsf{NP}^{\mathsf{NP}}$, the class of all non-deterministically polynomial algorithms with an oracle for an NP-complete problem, is also known as the class Σ_2^p.

1.5 Markov decision processes and extensions

We now introduce the formalism around which this thesis is centered. Informally, a Markov decision process is a Markov chain with rewards and the possibility to select, after each transition, the transition probabilities from some pre-defined set. We briefly cover the terms and main results on Markov decision processes. A more in-depth discussion on the general formalism can be found in [Put94, Kal16].

1.5.1 The model

Definition 1.5.1 (Markov decision/reward process)**.** Given a set of *states* $S = [n]$, a stochastic *transition matrix* $\boldsymbol{P} \in \mathbb{R}^{n\times n}$ with row sum 1, a *reward vector* $\vec{r} \in \mathbb{R}^n$, and an initial distribution vector $\vec{q} \in \mathbb{R}_{\geqslant 0}^{1\times n}$ with $\vec{q}^\top \vec{1} = 1$, a *Markov reward process* is a tuple $(S, \boldsymbol{P}, \vec{r}, \vec{q})$ that defines a sequence of random variables $(X_i)_{i\in\mathbb{N}}$ where $\Pr[X_1 = s] = \vec{q}(s)$, and X_{i+1} for $i \in \mathbb{N}$ is subject to the probability distribution $\Pr\left[X_{i+1} = s \mid X_i = s'\right] = \boldsymbol{P}(s, s')$.

For a set of states $S = [n]$, *actions* $A = [m]$, a reward vector $\vec{r} \in \mathbb{R}^n$, m stochastic transition matrices $T = \left\{\boldsymbol{P}^1, \dots, \boldsymbol{P}^m\right\} \subset \mathbb{R}^{n\times n}$, and a stochastic vector $\vec{q}$, a *Markov decision process* is a tuple $(S, A, T, \vec{r}, \vec{q})$ which, for a sequence of actions $(a_t)_{t\in\mathbb{N}} \in A^{\mathbb{N}}$, defines sequences of random variables (X_t) and (R_t) where $X_1 \in S$ is set according to $\Pr[X_1 = s] = \vec{q}(s)$, and X_{i+1} for $i \in \mathbb{N}$ is subject to the probability distribution $\Pr\left[X_{i+1} = s \mid X_i = s', a_i\right] = \boldsymbol{P}^{a_i}(s, s')$; furthermore, R_t is defined as $R_t = \vec{r}(X_i)$.

For convenience, in the Markov decision process context, we write $\Pr\left[s' \mid s, a\right]$ for $\boldsymbol{P}^a(s, s')$ to designate the transition probabilities as probabilities and not just as real numbers.

1.5.2 Alternative definitions and formalisms

In literature such as [Put94], one often defines Markov decision processes in a different fashion. Especially, it is often assumed that a one-time reward not only depends on the state one is starting from, but also on the state after the transition and the action the controller has performed. Here, we want to argue that mathematically, our model also covers this case. Suppose there is a function $R : S \times A \times S \to \mathbb{R}$ such that $R(s, a, s')$ is the reward we get after transitioning from s to s' after choosing action a.

Then, we introduce a virtual state $\tilde{s}(s, a, s')$ for all triples $(s, a, s') \in S \times A \times S$ such that after selecting an action $a \in A$, the system transitions first into the state $\tilde{s}(s, a, s')$ with the respective probability $\Pr\left[s' \mid s, a\right]$ and then transitions into s' with probability 1 (independent of the action selected in $\tilde{s}(s,a,s')$), generating the reward $R(s,a,s')$. Together, we derive an MDP

$$\left(S \uplus \left\{\tilde{s}(s,a,s') \mid s,s' \in S, a \in A\right\}, A, T, \vec{r}, \vec{q}\right)$$

with

$$\begin{aligned} \boldsymbol{P}^a(s,\tilde{s}(s,a,s')) &= \Pr\left[s' \mid s,a\right] \\ \boldsymbol{P}^a(\tilde{s}(s,a,s'),s') &= 1 \\ \vec{r}(\tilde{s}(s,a,s')) &= R(s,a,s') \\ \vec{r}(s) &= 0 \text{ for } s \in S \end{aligned}$$

Thus, we can efficiently eliminate a possible dependence of the rewards from the after-transition states. We note that this equivalence works by also transforming the goal function; in a way, the concept of "model equivalence" is similar to the concept of reduction we have defined above in the context of computational complexity.

We observe furthermore that our definition does not allow for states to have differing numbers of possible actions. However, this limitation can be ignored by using one action more than one time in states which have less than the maximal number of actions. In some of our proofs we will construct Markov decision processes which have different numbers of actions in different states; however, as it has been said, these MDPs can be represented with the formalism above.

1.5.3 Policies and objectives

For a Markov decision process (and its variants), several performance criteria have been proposed. We discuss some of these criteria in the context of Markov decision processes with uncertain parameters. The most important term in this context is the term of a *policy* which can be judged upon with the help of one of the introduced performance criteria. Informally speaking, a policy defines actions that shall be performed given a certain condition; formally, a policy is a function $f\colon S^+ \times A \to \mathbb{R}$ that maps (finite) *histories* of states to probability distributions on the action space. A policy f generates, for a Markov decision process $M = (S, A, T, \vec{r}, \vec{q})$ an *immediate reward at step i*

$$r_i^{(f)} = \sum_{a \in A} f((H, s_i), a) \cdot r_{s_i}$$

if s_i is the current state and $H \in S^*$ is the sequence of states preceding s_i.

We call a policy f

- *stationary* if it depends only on the current state, i. e., if it is $f((H,s),a) = f((H',s),a)$ for all $H, H' \in S^*$.
- *deterministic* if it always maps a history to a Dirac distribution, i. e., $f(\cdot,a) \in \{0,1\}$,
- *pure* if it is stationary and deterministic,
- *mixed* if it is stationary, but not pure.

To clarify the nomenclature, we denote general policies with Latin identifiers, such as f; for pure policies, we use Greek identifiers, such as π or σ. Stationary policies are denoted with capital Greek identifiers such as Π. Furthermore, we introduce short notation for stationary and pure policies: we write $\pi(s)$ for the action a where $\pi(\cdot,s,a) = 1$ and $\Pi(s,a)$ for $\Pi((\cdot,s),a)$.

It is easy to see that a stationary policy π induces a Markov reward process with transition matrix $\boldsymbol{P}^{(\pi)}$ and reward vector $\vec{r}$, as the transition probabilities under π depend only on the current state; this makes it mathematically easier to handle stationary policies; thus, the interest in stationary policies is justified from an "internal", mathematical as well as from an "external", user-centric point of view.

Having defined the notion of a policy, we now introduce tools which can measure the performance of a given policy f in a given Markov decision process $(S, A, T, \vec{r}, \vec{q})$.

Many performance measures we define depend on one free parameter we have only briefly mentioned, namely the *initial distribution*. Different initial distributions will lead to different distributions over the random states in the state sequence, and thus, lead to different rewards. For performance measures that depend on the initial distribution, it is possible to aggregate all initial distributions by computing the performance measure v for every starting state, i. e., by computing a function $v^{(f)}(s) = \left[v^{(f)} \mid X_1 = s\right]$ that maps states to the reward measure with this starting state. This function is called a *value vector* $\vec{v}^{(f)} \in \mathbb{R}^n$ and, in many cases, finding a policy that maximizes the reward measure for one specific initial distribution also yields a policy that optimizes the value vector as a whole. In the following discussion, we understand under "optimization", unless explicitly stated otherwise, maximization of the value vector. This means that an optimal policy f adheres to $f = \arg\max_f \vec{q}^\top \vec{v}^{(f)}$. This notion of optimality implies that there may exist more than one policy which is optimal. Sometimes we might be interested in stationary or pure policies only; then, the existence of an optimal pure or stationary policy means the existence of a policy which is optimal in the sense defined above and, additionally, is pure or stationary.

Expected total reward A straightforward performance indicator is the *expected total reward* defined by the term

$$v_\Sigma^{(f)} = \mathrm{Ex}\left[\sum_{i=1}^{\infty} r_i^{(f)}\right]. \tag{1.2}$$

We note that this sum does not need to converge. It does not converge if there exists at least one recurrent state with positive reward, and in general, convergence can only be guaranteed if there exist absorbing states with zero reward which are reached with probability 1. This puts a limitation on the nature of models we can consider.

Expected finite-horizon total reward To keep the mathematical properties of the expected total reward measure but to ensure also finiteness we can truncate the computation of the expected total reward after a fixed amount of steps. This amount $N \in \mathbb{N}$ is called a *horizon*, and the reward term turns out to be

$$v_N^{(f)} = \mathrm{Ex}\left[\sum_{i=1}^{N} r_i^{(f)}\right]. \tag{1.3}$$

Expected average reward One useful property of the expected total reward is that this measure represents the behaviour of a policy in the infinite. For this task, we introduce two performance criteria that work independent of the nature of the underlying MDP model and converge at all times. Both of them are motivated by the expected total reward measure. As the latter does not need to be finite, it can be made finite by considering the average gain for a time step in the long run. This measure, the *expected average reward* or *expected gain*, can be formalized by

$$v_\infty^{(f)} = \mathrm{Ex}\left[\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} r_i^{(f)}\right]. \tag{1.4}$$

It can be shown [LL69] that this limit always exists for stationary policies and that there exists an optimal stationary policy, that is, a stationary policy $f = \arg\max_{f\colon v_\infty^{(f)} \text{ exists}} v_\infty^{(f)}$.

Computing the expected average reward is non-trivial for general non-stationary policies as the limit in (1.4) does not need to exist; here, we show how it can be computed for a stationary policy f. Following [Put94], it is possible to show that the expected average reward $\vec{v}_\infty^{(f)}$ has the following properties.

$$\begin{aligned} \boldsymbol{P}^{(f)}\vec{v}_\infty^{(f)} &= \vec{v}_\infty^{(f)} \\ \vec{r} - \vec{v}_\infty^{(f)} + \boldsymbol{P}^{(f)}\vec{h} &= \vec{h} \end{aligned} \tag{1.5}$$

The vector $\vec{h}$ is also known as the *bias vector* and describes the state-dependent constant term in the formula $\vec{h}(s) + t\vec{v}_\infty^{(f)}(s)$ for the expected gain after t steps, starting in state s.

Expected discounted reward There are two downsides of the expected gain measure. First, it is not guaranteed that the limit in (1.4) converges for general, not necessarily stationary policies. For example, consider a two-state MDP that is depicted in Fig. 1.1. The states offer rewards of -1 and 1, respectively, and it is possible to move to either of these states arbitrarily. In this MDP, one can devise a policy $f_\pm$ with the following behaviour: For $i \in \mathbb{N}_0$, the policy stays for 2^i time steps in s_1, gathering a total reward of -2^i, and then, $f_\pm$ stays for 2^i time steps in s_2, gathering a total reward of 2^i. Then $f_\pm$ returns to s_1 and stays there for 2^{i+1} steps, gathering a total reward of -2^{i+1} and so on. For $k \in \mathbb{N}_0$, the total reward after $2(2^{k+1}-1)$ steps will be then zero, and the total reward after $2(2^{k+1}-1)+2^{k+1}$ steps will be -2^{k+1}. Hence, the average reward after N steps is then, depending on N, somewhere between 0 and $-1/3$ and the limit in (1.4) does not exist.

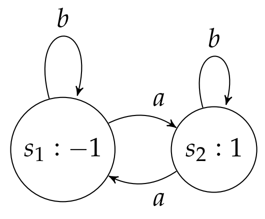


Figure 1.1: A Markov decision process with a non-convergent policy

Second, the expected average measure does not differentiate between *early* and *future* gains. To cope with both issues, we can introduce a *discount factor* $\gamma \in [0, 1)$ which describes the "importance damping" of gains, i. e., how much less important tomorrow's profits in comparison to those of today are.

To address these issues we introduce a different performance indicator, the *expected discounted total reward*

$$v_\gamma^{(f)} = \mathrm{Ex}\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_i^{(f)}\right]. \tag{1.6}$$

We note that this sum always converges as there exists an upper bound on the reward (since we consider finite-state MDPs). Furthermore, one can show that for stationary policies, the optimal policies for $\gamma \to 1$ converge to optimal policies for the expected average reward measure if the rewards are bounded [LL69] (which is always the case for finite-state and finite-action models).

Remark. In the fully general case, a policy can depend on the complete history of a Markov decision process. However, in many applications, it may require infinite memory and thus, it seems natural to consider policies that depend only on the current state. The optimality of such policies depends largely on the kind of optimality measure; for some optimality measures such as the expected gain and expected discounted total reward, there exists an optimal deterministic policy that only depends on the current state [Put94]; for other measures, only policies with access to the full history are optimal.

1.5.4 Optimization of Markov decision processes

For the reward measures described above, there exist several general algorithms that find optimal policies efficiently. We briefly describe them and their properties; a more detailed discussion can be found in [Put94]. Without limitation of generality we consider maximization to be the main optimization direction; for minimization, symmetric arguments apply.

Dynamic programming and value iteration For total reward measures, the dynamic programming approach is straightforward as well as efficient. Intuitively, the approach consists of computing the optimal decision "in the end" where no further decisions can be made and then, by backwards induction, find optimal decisions for the preceding step under the assumption that the next step has been computed optimally. Together, this yields Algorithm 1.

Algorithm 1 Dynamic programming algorithm for optimization of finite-horizon total expected reward

```
function FiniteHorizonDynamicProgramming($S, A, T, \vec{r}, N$)
    for $s \in S$ do $\vec{v}_N \leftarrow \vec{r}$
    for $i = N-1, \ldots, 1$ do
        for $s \in S$ do
            $\vec{v}_i(s) \leftarrow \vec{r}(s) + \max_{a \in A} \boldsymbol{P}^a(s\bullet)\vec{v}_{i+1}$        ▷ Compute the expected value
            $f(i, s) \leftarrow \arg\max_{a \in A} \boldsymbol{P}^a(s\bullet)\vec{v}_{i+1}$
    return $\vec{v}_1, f$
```

This approach is also known as *value iteration*. One can observe from the structure of the algorithm that the resulting policies depend on the state and the time step, and are deterministic. The algorithm needs N iterations of the outer loop and in each of these iterations, $|S|$ inner loop iterations. Together, this makes $\mathcal{O}\left(N \cdot |S||A|\right)$ time steps.

A similar approach can be used for optimizing the expected discounted reward. There, we do not have a finite amount of steps, but a convergence guarantee that stems from Banach's fixed point theorem [Cie07]. The modified algorithm is presented in Alg. 2. The algorithm stops when the difference between the successively computed vectors v and v' is smaller than $\frac{\varepsilon(1-\gamma)}{2\gamma}$, which implies, after Theorem 6.3.1 in [Put94], that the difference between the resulting vector v and the true value vector defined in (1.6) is at most $\frac{\varepsilon}{2}$ for a given precision parameter $\varepsilon > 0$. The difference between the value vector of the resulting policy π and the value vector of an optimal policy will be at most ε.

For convenience, we provide the main argument for this statement. Banach's fixed point theorem states, that for a norm $\|\cdot\|$ on $\mathbb{R}^n$ and a *contraction mapping* $L\colon \mathbb{R}^n \to \mathbb{R}^n$ which satisfies $\|L(\vec{v}) - L(\vec{u})\| \leqslant \gamma\|\vec{v} - \vec{u}\|$ for some $\gamma \in [0, 1)$, a unique *fixed point* $\vec{v}^*$ exists with $L(\vec{v}^*) = \vec{v}^*$ which can be computed by iteratively applying $L(L(\ldots L(\vec{v})))$ for any vector $\vec{v}$. Let $\vec{v}_0 = \vec{v}$ and $\vec{v}_n = L(\vec{v}_{n-1})$ for $n \in \mathbb{N}$. Using the properties of contraction mappings and the triangle inequality, it is possible to derive

$$\begin{aligned}
\|\vec{v}^* - \vec{v}_{n+1}\| &= \|L(\vec{v}^*) - \vec{v}_{n+1}\| \\
&\leqslant \|L(\vec{v}^*) - L(\vec{v}_{n+1})\| + \|L(\vec{v}_{n+1}) - \vec{v}_{n+1}\| \\
&= \|L(\vec{v}^*) - L(\vec{v}_{n+1})\| + \|L(\vec{v}_{n+1}) - L(\vec{v}_n)\| \\
&\leqslant \gamma\|\vec{v}^* - \vec{v}_{n+1}\| + \gamma\|\vec{v}_{n+1} - \vec{v}_n\| \Leftrightarrow \\
\|\vec{v}^* - \vec{v}_{n+1}\| &\leqslant \frac{\gamma}{1-\gamma}\|\vec{v}_n - \vec{v}_{n+1}\| .
\end{aligned}$$

This means that if $\|\vec{v}_n - \vec{v}_{n+1}\| \leqslant \frac{\varepsilon(1-\gamma)}{2\gamma}$, then $|\vec{v}^* - \vec{v}_{n+1}| \leqslant \frac{\varepsilon}{2}$.

Algorithm 2 Dynamic programming algorithm for optimization of expected discounted reward

```
function ValueIteration($S, A, T, \vec{r}, \gamma$)
    $\vec{v}, \vec{v}' \leftarrow \vec{0} \in \mathbb{R}^n$
    while $\delta \geqslant \frac{\varepsilon(1-\gamma)}{2\gamma}$ do
        for $s \in S$ do
            $\vec{v}'(s) \leftarrow \vec{r}(s) + \gamma \max_{a \in A} \boldsymbol{P}^a(s\bullet)\vec{v}$
            $\pi(s) \leftarrow \arg\max_{a \in A} \boldsymbol{P}^a(s\bullet)\vec{v}$
        $\delta \leftarrow \max_{s \in S} |\vec{v}(s) - \vec{v}'(s)|$
        $\vec{v} \leftarrow \vec{v}'$
    return $\vec{v}, \pi$
```

We note that the policy computed by Alg. 2 is pure. In fact, it can be shown that for this performance measure, there always exists an optimal pure policy that corresponds to the fixed point of the outer loop in Alg. 2.

Policy iteration A similar result can be shown for the expected average reward measure: in this case, too, there always exists an optimal pure policy. Furthermore, a *locality* property can be shown: a pure policy which cannot be optimized by changing its behaviour in one state, i.e. a locally optimal policy, is also globally optimal. This gives rise to the *policy iteration* approach which is a local improvement algorithm for policies which yields optimal policies for the expected average reward and expected discounted reward measures. In the pseudo-code description, the concrete reward measure is designated by a function v.

Algorithm 3 Generalized policy iteration scheme

```
function PolicyIteration($S, A, T, \vec{r}, \vec{q}, v$)
    $\pi \leftarrow \vec{0}$                                   ▷ Initialize an arbitrary policy
    while $\pi$ changes do
        for $(s, a) \in S \times A$ do
            $\pi' \leftarrow \pi$
            $\pi'(s) = a$
            if $v^{(\pi')}(S, A, T, \vec{r}, \vec{q}) > v^{(\pi')}(S, A, T, \vec{r}, \vec{q})$ then
                $\pi \leftarrow \pi'$
    return $\pi$
```

The general scheme is depicted in Algorithm 3 where v is an arbitrary optimality criterion for pure policies. The concrete formulation of the algorithm may vary with the optimality criterion. So, for the expected average reward, the policy iteration algorithm has the following form [Vei66, Put94].

Algorithm 4 Subsection 9.2.1, [Put94]: Policy iteration for the expected average criterion in MDPs

```
1: function AVERAGEPOLICYITERATION($M = (S, A, T, \vec{r}, \vec{q})$)
2:     $n \leftarrow 0$, select an arbitrary decision rule $\pi_0$
3:     repeat
4:         Compute $\vec{g} \in \mathbb{R}^n, \vec{h} \in \mathbb{R}^n$ such that $\boldsymbol{P}_{\pi_n}\vec{g} = \vec{g}, \vec{r}_{\pi_n} - \vec{g} + \left(\boldsymbol{P}_{\pi_n} - \boldsymbol{I}\right)\vec{h} = \vec{0}$
5:         Choose a $\pi_{n+1}$ that satisfies
               $\pi_{n+1} \in \arg\max_{\pi} \boldsymbol{P}_{\pi}\vec{g},$    (1.7)
           keeping $\pi_{n+1} = \pi_n$, if possible.
6:         if $\pi_{n+1} = \pi_n$ then
7:             Choose a $\pi_{n+1}$ that satisfies
                   $\pi_{n+1} \in \arg\max_{\pi} \left(\vec{r}_{\pi} + \boldsymbol{P}_{\pi}\vec{h}\right),$    (1.8)
           keeping $\pi_{n+1} = \pi_n$, if possible.
8:         $n \leftarrow n + 1$
9:     until $\pi_n = \pi_{n-1}$
10:    return $\pi_n$
```

Linear programming formulations As with most optimization problems, there also exists a linear programming formulation for Markov decision problems [Man60, d'E63, Put94, DD05]. The problem of interest here is optimizing the expected discounted reward measure. Its main property is that the value vector $\vec{v}^{(\Pi)}$ for a (stationary) policy Π can be written as a solution of a linear equation system $\vec{r} + \gamma \boldsymbol{P}^{(\Pi)}\vec{v}^{(\Pi)} = \vec{v}^{(\Pi)}$ where $\boldsymbol{P}^{(\Pi)}$ is defined by $\boldsymbol{P}^{(\Pi)}(s\bullet) = \sum_{a \in A} \Pi(s,a)\boldsymbol{P}^a(s\bullet)$. This yields a linear program which computes the optimal value vector; this formulation has been derived by Manne [Man60].

$$\begin{aligned} &\min \vec{1}^\top \vec{v} \\ \text{s.t.} \\ &\vec{r} + \gamma \boldsymbol{P}^a \vec{v} \leqslant \vec{v} \quad \forall a \in A \end{aligned} \tag{1.9}$$

The corresponding dual linear program has been introduced by d'Epenoux [d'E63, Kal83].

$$\begin{aligned} &\max \sum_{s \in S} \sum_{a \in A} x_{s,a} \vec{r}(s) \\ \text{s.t.} \\ &\sum_{a \in A} x_{s,a} - \gamma \sum_{a \in A, s' \in S} \boldsymbol{P}^a(s', s) x_{s',a} = \vec{q}(s) && \forall s \in S \\ &x_{s,a} \geqslant 0 && \forall (s,a) \in S \times A \end{aligned} \tag{1.10}$$

This formulation has several interesting properties. First, the goal function coefficients in the primal LP can be arbitrary non-negative values and lead to the same value of $\vec{v}$. Second, the optimal policy can be read from the solution by looking at the rows of the primal LP that are tight; if the constraint

$$\vec{r}(s) + \gamma \boldsymbol{P}^a(s\bullet)\vec{v} \leqslant \vec{v}(s) \tag{1.11}$$

is tight, that is, if it is $\vec{r}(s) + \gamma \boldsymbol{P}^a(s\bullet)\vec{v} = \vec{v}(s)$, then the optimal policy selects the action a in state s. Third, by complementary slackness, the variables $x_{s,a}$ in the dual formulation that correspond to the rows in (1.9) are nonzero if and only if the constraint (1.11) is tight. This means that the variable $x_{s,a}$ in the dual formulation can be used as "decision variable" that is nonzero if and only if the optimal policy selects the action a in state s. However, these are not decision variables in the sense of combinatorial optimization, as they do not have to be integral. The most common interpretation for these variables is that they describe "discounted visitation frequencies" of states that contribute to the cumulative reward.

Nevertheless, there are ways to derive integral variables from the variables of the dual formulation [DD05]. A simple way to do so is by introducing additional integer variables $d_{s,a} \in \{0,1\}$ with the constraints $d_{s,a} = 1 \Leftrightarrow x_{s,a} > 0$; alternatively, an equivalent constraint is $d_{s,a} = \frac{x_{s,a}}{\sum_{a' \in A} x_{s,a'}}$. In general, this constraint cannot be written as a linear inequality; however, here we know that $x_{s,a}$ has an upper bound of $\frac{1}{1-\gamma}$ as the rewards from a state s are bounded by $\frac{\vec{r}(s)}{1-\gamma}$. This allows us to impose the constraint $d_{s,a} \cdot \frac{1}{1-\gamma} \geqslant x_{s,a}$ for all $(s,a) \in S \times A$ and $\sum_{a \in A} d_{s,a} = 1$ for all $s \in S$. Together, we can derive the following mixed-integer linear program.

$$
\begin{aligned}
&\max \sum_{s \in S} \sum_{a \in A} x_{s,a} \vec{r}(s) \\
&\text{s.t.} \\
&\sum_{a \in A} x_{s,a} - \gamma \sum_{a \in A, s' \in S} \boldsymbol{P}^a(s', s) x_{s',a} = \vec{q}(s) && \forall s \in S \\
&\sum_{a \in A} d_{s,a} = 1 && \forall s \in S \\
&d_{s,a} \geqslant (1-\gamma) x_{s,a} && \forall (s,a) \in S \times A \\
&x_{s,a} \geqslant 0 && \forall (s,a) \in S \times A \\
&d_{s,a} \in \{0,1\} && \forall (s,a) \in S \times A
\end{aligned}
\tag{1.12}
$$

From the complexity-theoretic point of view, it is easy to see that almost all MDP problems can be solved in polynomial time. As we extend the MDP formalism, an important question is if this property can be kept.

1.5.5 Continuous-time processes and uniformization

A large body of research deals with the question what happens if the transition times in a Markovian decision process are not equal but also distributed according to a memoryless distribution which depends on the state (and sometimes the selected action). Following this research question, one arrives at the continuous-time MDP model. Its main characteristic is that the evolution of the underlying system is defined by a system of (linear) differential equations. In the next paragraphs, we give a brief overview of the formalism and describe a method to analyse some aspects of continuous-time MDPs with algorithms for discrete-time Markov decision processes.

Continuous-time Markov chains Similarly to a discrete-time Markov chain, one can define continuous-time Markov chains where the transition times are governed by exponential distributions. Concretely a continuous-time Markov chain can be represented by a stochastic vector $\vec{q}_0 \in \mathbb{R}^{1 \times n}$ and a *rate matrix* $\boldsymbol{Q} \in \mathbb{R}^{n \times n}$ for which the properties $\boldsymbol{Q}\vec{1} = \vec{0}$ and $\boldsymbol{Q}(i,j) \geqslant 0$ for all $i \neq j$ with $i,j \in [n]$ hold.

The evolution of a continuous-time Markov chain can be described as follows. The system stays in a state $s \in S$ for a time interval which is negative exponentially distributed with rate $\boldsymbol{Q}(s,s)$ and then performs a transition to another state s' with probability $\frac{\boldsymbol{Q}(s,s')}{-\boldsymbol{Q}(s,s)}$.

A global representation of this dynamics is the differential equation [Kol31]

$$\begin{aligned} \frac{\mathrm{d}\vec{p}(t)}{\mathrm{d}t} &= \vec{p}(t)\boldsymbol{Q} \\ \vec{p}(0) &= \vec{q} \end{aligned} \tag{1.13}$$

where $\vec{p}(t)$ is the probability distribution of the Markov chain being in a given state at time t. The solution to this equation is $\vec{p}(t) = \vec{q}\exp(\boldsymbol{Q}t)$ where the matrix exponential $\exp(M)$ is defined by the infinite sum

$$\exp(M) = \sum_{i=0}^{\infty} \frac{M^i}{i!}. \tag{1.14}$$

Continuous-time Markov decision processes With a continuous-time Markov chain formalism, we arrive at a formalism for continuous-time MDPs.

Definition 1.5.2. Given a Markov decision process $(S, A, T, \vec{r}, \vec{q})$ and a vector $\vec{\beta} \in \mathbb{R}^n$ with $\vec{\beta} > 0$, a *continuous-time Markov decision process* (CTMDP) is a tuple $\left(S, A, T, \vec{r}, \vec{q}, \vec{\beta}\right)$. A CTMDP defines not only a sequence of states as described in Def. 1.5.1, but also a sequence of *sojourn times* $(Y_t)_{t\in\mathbb{N}}$ where Y_t is exponentially distributed with parameter $\vec{\beta}(X_t)$.

Together, they define a family of random variables $(Y_x)_{x\in\mathbb{R}_{\geqslant 0}}$ with $Y_x = X_t$ if $\sum_{t'<t} Y_{t'} < x$ and $\sum_{t'\leqslant t} Y_{t'} \geqslant x$.

This definition expands the MDP formalism by the notion of transition times. It is easy to see that the system retains its Markovian property: The sojourn time does not depend on the starting point of the observation. Mathematically speaking, for an exponentially distributed transition time Y it is $\Pr[Y \geqslant x] = \Pr\left[Y \geqslant T + x \mid Y \geqslant T\right]$ for all $x \geqslant 0, T \geqslant 0$.

The finite-horizon total reward for a time horizon T in this model is formalized by the integral expression

$$\int_0^T \vec{r}(Y_x)\,\mathrm{d}x \tag{1.15}$$

which allows us to derive the expected average total reward

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T \vec{r}(Y_x)\,\mathrm{d}x \tag{1.16}$$

and the expected discounted total reward with a *discount rate* $\alpha > 0$

$$\int_0^\infty \exp(-\alpha x)\vec{r}(Y_x)\,\mathrm{d}x. \tag{1.17}$$

Uniformization Analysing continuous-time MDPs means, in the most general case, analyzing a continuous-time system that is governed by a set of differential equations [Put94]; especially the finite-horizon case is less simple to analyse as the number of transitions in a finite time interval can be unbounded. We note here that even for the finite-horizon case optimal policies can be computed [BDS17b], but as our main results consider "stationary" optimality criteria such as the expected discounted total reward and the expected average reward, we provide a tool that enables to analyse continuous-time MDPs with discrete-time methods with respect to these criteria [Put94].

The uniformization technique amounts to two steps: First, the stochastic process is transformed into a process with uniform sojourn time distribution in all states. Second, as the sojourn times are distributed equally, the expected rewards in each state are computed and only the transition probabilities with these new rewards are considered, which allows one to use discrete-time methods.

Informally, the first step of the uniformization procedure transforms the continuous-time process into another continuous-time process where the transition events occur with equal frequency in each state. For this, the transition probabilities are modified in order to keep the total sojourn time.

Concretely, to transform a continuous-time MDP $(S, A, T, \vec{r}, \vec{q}, \vec{\beta})$ into a discrete-time MDP $(S, A, T_u, \vec{r}_u, \vec{q})$, the following steps have to be made. First, the *uniformization rate* $\beta^* \geqslant \max_{i \in S} \vec{\beta}(i)$ is chosen. Then, a CTMDP $(S, A, T_u, \vec{r}, \vec{q}, \beta^* \vec{1})$ is computed by defining $T_u = \left\{ \boldsymbol{P}_u^1, \ldots, \boldsymbol{P}_u^m \right\}$ with

$$\boldsymbol{P}_u^a(s, s') = \begin{cases} 1 - \frac{(1-\boldsymbol{P}^a(s,s))\vec{\beta}(s)}{\beta^*} & s = s' \\ \frac{\boldsymbol{P}^a(s,s')\vec{\beta}(s)}{\beta^*} & s \neq s' \end{cases} \tag{1.18}$$

Then, the rewards have to be adjusted. For the expected average total reward, we set

$$\vec{r}_u(s) = \frac{\vec{r}(s)}{\beta^*}. \tag{1.19}$$

For the expected discounted total reward, we set

$$\vec{r}_u(s) = \frac{\vec{r}(s)}{\beta^* + \alpha} \tag{1.20}$$

and introduce the discount factor $\gamma = \frac{\beta^*}{\beta^* + \alpha}$. It can be shown that a stationary policy will yield for these *uniformized* processes the same optimality values as for the original CTMDPs, which allows us to use discrete-time analysis methods in order to find optimal policies for the expected average and expected discounted reward criteria [Ser79, Put94].

1.6 Stochastic games

It is easy to see that we can consider a Markov decision process as a game where the controller can choose actions and the randomness chooses following states. This view can be generalized in a natural way to more parties. For us, of special interest is the two-player case, where two parties can control the system, the *controller* (CON) and *Nature* (NAT).

In the literature [Sha53, FV97], these processes are known as *stochastic games*. Informally, in a stochastic game, CON and NAT choose from two pools of available *actions*; the action pair combined with the current state defines a probability distribution on the next state and a reward value.

Definition 1.6.1 (Stochastic game)**.** For a set of states $S = [n]$ and two action sets A_C, A_N, a *stochastic game* is a tuple $(S, A_C, A_N, P, \vec{r}_C, \vec{r}_N, \vec{q})$ where $P \colon S \times A_C \times A_N \to S \to \mathbb{R}$ is, for every triple (s, a_C, a_N), a probability distribution on states, $\vec{q} \in \mathbb{R}^n$ is a stochastic vector, and $\vec{r}_N, \vec{r}_C \in \mathbb{R}^n$ are *reward vectors*.

The semantics of a stochastic game is straightforward: At each discrete time step $t, t \in \mathbb{N}$, the formal system is in a state $s \in S$. CON chooses an action a_C from the *controller action set* A_C and NAT chooses an action a_N from the *nature action set* A_N. Then, an *immediate reward* $(r_C, r_N) = (\vec{r}_C(s), \vec{r}_N(s))$ is being paid off; CON gains $\vec{r}_C(s)$ reward units and NAT gains $\vec{r}_N(s)$ reward units. Then, the system performs a transition to a state $s' \in S$ with probability $P(s, a_C, a_N)(s')$. Formally, a stochastic game together with a sequence of action pairs $(a_{C,i}, a_{N,i})_{i \in \mathbb{N}}$ define a sequence $(X_i)_{i \in \mathbb{N}}$ of random variables with the distributions

$\Pr[X_1 = s] = \vec{q}(s)$ and $\Pr[X_{i+1} = s' \mid X_i = s, a_{C,i} = a_C, a_{N,i} = a_N] = P(s, a_C, a_N)(s')$. The sequence $(X_i)_{i\in\mathbb{N}}$ also defines a sequence of random reward pairs $(r_{C,i}, r_{N,i})_{i\in\mathbb{N}}$ with $r_{C,i} = \vec{r}_C(X_i)$ and $r_{N,i} = \vec{r}_N(X_i)$.

It is important to distinguish an important subclass of stochastic games with special semantics: In a *perfect information* stochastic game, NAT and CON perform actions by alternating their moves, that is, both players can observe the results of each other's action before making the next move. This is modeled by separating the state set S into two disjoint subsets, the *controller set* S_{CON} and the *nature set* S_{NAT} with $S = S_{\text{CON}} \uplus S_{\text{NAT}}$; the semantics is such that $P(s, a_C, a_N) = P(s, a_C, a'_N)$ for $s \in S_{\text{CON}}$ and all $a_N, a'_N \in A_N$, and, symmetrically, $P(s, a_C, a_N) = P(s, a'_C, a_N)$ for $s \in S_{\text{NAT}}$ and all $a_C, a'_C \in A_C$.

For a stochastic game, the most interesting question is if there is a policy for CON which maximizes her overall performance measure while NAT tries to maximize her own overall performance measure. The answer to this question mainly depends on the structure of $\vec{r}_N$ and $\vec{r}_C$ and the chosen optimality criteria. For the latter, we assume that the optimality criteria are the same for both players (and have the same optimization direction), that is, if $(r_{C,i})_{i\in\mathbb{N}}$ and $(r_{N,i})_{i\in\mathbb{N}}$ are the payoff sequences for each of the players, then the optimality criterion can be described by a single function $v\colon \mathbb{R}^{\mathbb{N}} \to \mathbb{R}$ such that the goal function v_C for CON is $v((r_{C,i})_{i\in\mathbb{N}})$ and the goal function v_N for NAT is $v((r_{N,i})_{i\in\mathbb{N}})$. For the former, we consider two cases that are most important to us, namely, the *cooperative* and the *competitive* cases.

The cooperative case $\vec{r}_C \geqslant \vec{0}, \vec{r}_N = \vec{r}_C$

The competitive case $\vec{r}_C \geqslant \vec{0}, \vec{r}_N = -\vec{r}_C$

It is easy to see that in the cooperative case, there is no conflict in the goals of CON and NAT. This means that finding the optimal policy for the cooperative case can be performed with the methods known from Markov decision process optimization. In fact, replacing both players with one party leaves us with a Markov decision process (under the assumption that the optimality criteria for both players are the same).

Optimal policies for stochastic games Previously, we have established that cooperative stochastic games are equivalent to Markov decision processes. Here, we consider competitive stochastic games and briefly outline the most important algorithms that find optimal policies. A more complete discussion is available in the original work of Shapley [Sha53] and in the textbook of Filar and Vrieze [FV97]; the facts that are mentioned here are reduced to the ones we need in the rest of our work.

A *policy pair* in a stochastic game is a pair of functions $f_C\colon S^+ \times A_C \to [0,1]$, $f_N\colon S^+ \times A_N \to [0,1]$ that define probability distributions over actions in dependence of previous visited states and random reward sequences $(r^{(f_C,f_N)}_{C,i})_{i\in\mathbb{N}}, (r^{(f_C,f_N)}_{N,i})_{i\in\mathbb{N}}$. In analogy to the MDP case, we assume that there exists an optimality criterion that yields real values $v^{(f_C,f_N)}_C, v^{(f_C,f_N)}_N$ which describe the goal functions of the two players for each policy. We call a policy pair (f^*_C, f^*_N) *optimal*, if the following conditions are met.

- $v^{(f^*_C,f^*_N)}_C \geqslant v^{(f_C,f^*_N)}_C$ for each $f_C : S^+ \times A_C \to [0,1]$
- $v^{(f^*_C,f^*_N)}_N \geqslant v^{(f^*_C,f_N)}_N$ for each $f_N : S^+ \times A_N \to [0,1]$

Now we consider different optimality criteria and the corresponding algorithms. The optimality criteria are similar to those defined for Markov decision processes.

The finite horizon total expected reward criterion For a finite horizon $N \in \mathbb{N}$, the finite horizon total expected reward criterion for stochastic games is defined by

$$\begin{aligned} v_{C,N}^{(f_C,f_N)} &= \mathrm{Ex}\left[\sum_{i=1}^{N} r_{C,i}^{(f_C,f_N)}\right] \\ v_{N,N}^{(f_C,f_N)} &= \mathrm{Ex}\left[\sum_{i=1}^{N} r_{N,i}^{(f_C,f_N)}\right] \end{aligned} \tag{1.21}$$

In the sequel, we only give the optimality criterion for CON, as the criterion for NAT is defined symmetrically.

To compute an optimal policy pair for the finite horizon total expected reward criterion, we perform a similar procedure to the one in Algorithm 1, adjusted for the two-player case. In fact, the most challenging task here is to compute a decision rule that is optimal, as we move from a simple maximization problem to a max-min problem.

In order to derive a solution, we consider simple one-step two-player games first.

Definition 1.6.2 (One-step game)**.** For finite action spaces $A_C = [m_C], A_N = [m_N]$, a *one-step game* is defined by a matrix $\boldsymbol{C} \in \mathbb{R}^{m_C \times m_N}$ with the following semantics: If CON chooses action $a_C \in A_C$ and NAT chooses action $a_N \in A_N$, then the payoff for CON is $\boldsymbol{C}(a_C, a_N)$ and the payoff for NAT is $-\boldsymbol{C}(a_C, a_N)$.

An optimal policy for a one-step game can then be computed by solving the following linear program.

$$\begin{aligned} & \max v \\ \text{s.t.} & \\ & v \leqslant \sum_{a_C \in A_C} \vec{x}(a_C)\boldsymbol{C}(a_C, a_N) \quad \forall a_N \in A_N \\ & \sum_{a_C \in A_C} \vec{x}(a_C) = 1 \\ & \vec{x} \geqslant \vec{0} \end{aligned} \tag{1.22}$$

The result of this optimization problem is the *value* of the *matrix game* defined by $\boldsymbol{C}$; the optimal policy is given implicitly in the decision vector $\vec{x}$. We observe that $\vec{x}$ does not necessarily have to define a Dirac distribution; hence, for general stochastic games policies may not necessarily be deterministic.

This result can be used in order to derive optimal policies for stochastic games with arbitrary finite horizons. In detail, knowing the optimal policy for future steps in step n and the corresponding value $v(s', n+1)$ of all states $s' \in S$ in the $n+1$st step, the choice for players CON and NAT in state s corresponds exactly to a one-step game with matrix $\boldsymbol{C}_{n,s}$ which is defined by $\boldsymbol{C}_{n,s}(a_C, a_N) = \sum_{s' \in S} P(s, a_C, a_N, s')v(s', i+1)$.

For perfect information stochastic games, the structure of the optimization problem given in (1.22) becomes radically simpler. In CON-controlled states, the matrix $\boldsymbol{C}$ has equal columns, and the optimal action is the one that chooses the row with the greatest value. In NAT-controlled states, $\boldsymbol{C}$ has equal rows and, hence, v will be the minimal value that can result from an action of NAT. In both cases, the resulting policies for CON and NAT are deterministic.

This gives rise to the following general algorithm inspired by Alg. 1.

Algorithm 5 Finite-horizon expected total reward optimization algorithm for stochastic games

```
function GameFiniteHorizon($S, A_C, A_N, P, \vec{r}_C, \vec{r}_N, N$)
    $\vec{v}_N \leftarrow \vec{r}_C$
    for $i \in \{N-1, \dots, 1\}$ do
        for $s \in S$ do
            for $(a_C, a_N) \in A_C \times A_N$ do
                $C(a_C, a_N) \leftarrow \sum_{s' \in S} P(s, a_C, a_N, s') v(s', i+1)$
            $v', \vec{x}_{s,i} \leftarrow$ solution of LP in (1.22)
            $\vec{v}_i(s) \leftarrow \vec{r}_C(s) + v'$
        $\Pi_i \leftarrow (\vec{x}_{s,i})_{s \in S}$
    return $\vec{v}_1, (\Pi_1, \dots, \Pi_{N-1})$
```

The runtime of this algorithm is polynomial in $|S|$, $|A_C|$, $|A_N|$, and N.

The expected discounted reward criterion For the expected discounted reward criterion, we can apply the same reasoning we used in the preceding discussion. The optimality criterion is defined by

$$v_{C,\gamma}^{(f_C, f_N)} = \mathrm{Ex}\left[\sum_{i=1}^{\infty} \gamma^{i-1} r_{C,i}^{(f_C, f_N)}\right]. \tag{1.23}$$

Analogously, we obtain an algorithm that computes optimal value vectors.

Algorithm 6 Finite-horizon expected total reward optimization algorithm for stochastic games

```
function GameDiscounting($S, A_C, A_N, P, \vec{r}_C, \vec{r}_N, \gamma$)
    $\vec{v}, \vec{u} \leftarrow \vec{0}$
    while $\delta > \frac{\varepsilon(1-\gamma)}{2\gamma}$ do
        for $s \in S$ do
            for $(a_C, a_N) \in A_C \times A_N$ do
                $C(a_C, a_N) \leftarrow \sum_{s' \in S} P(s, a_C, a_N, s') \vec{v}(s')$
            $\delta \leftarrow \max_{s \in S} |\vec{v}(s) - \vec{u}(s)|$
            $v', \vec{x}_s \leftarrow$ solution of LP in (1.22)
            $\vec{u}(s) \leftarrow \vec{r}_C(s) + \gamma v'$
        $\Pi \leftarrow (\vec{x}_s)_{s \in S}$
        $\vec{v} \leftarrow \vec{u}$
    return $\vec{v}, \Pi$
```

Again, the convergence of this algorithm can be shown with Banach's fixed point theorem: The convergence rate of the value vectors is γ, and we can again apply the argument in Theorem 6.3.1 from [Put94] to show the error bound of $\frac{\varepsilon}{2}$. However, there do not exist superior algorithms. Optimization of infinite-horizon stochastic games, even perfect-information stochastic games, is a major open problem [BV07] related to other "game-like" problems such as parity games [Obd06] or mean-payoff games [ZP96], and it is still unclear whether it can be done in polynomial time independently of the discount factor γ. It is known that optimization of discounted perfect-information zero-sum two-player stochastic games is in NP $\cap$ coNP [Con92], which is supposed to be generally "easier" than NP-hard problems, but no polynomial-time results are known.

The average total reward criterion This criterion is defined by

$$v_{C,\infty}^{(f_C,f_N)} = \lim_{N\to\infty} \frac{1}{N} \operatorname{Ex}\left[\sum_{i=1}^{N} r_{C,i}^{(f_C,f_N)}\right]. \tag{1.24}$$

In analogy to Markov decision processes, our approach is to consider the expected discounted reward criterion for a sequence of discount factors $\gamma_1, \gamma_2, \ldots$ with $\gamma_1 < \gamma_2 < \ldots$ and $\lim_{i\to\infty} \gamma_i = 1$. It can be shown [MN81] that the sequence of resulting policies converges; furthermore, the convergence holds also for the value vectors multiplied by $(1-\gamma)$. This results in the following algorithm.

Algorithm 7 Optimal policies for the expected average reward criterion, general scheme

```
function GAMEDISCOUNTINGLIMIT($S, A_C, A_N, P, \vec{r}_C, \vec{r}_N$)
    $i \leftarrow 0, \vec{v} \leftarrow \infty$
    while $\delta > \varepsilon$ do
        $\gamma \leftarrow 1 - 2^{-i}$
        $\vec{u}, \Pi \leftarrow$ GAMEDISCOUNTING($S, A_C, A_N, P, \vec{r}_C, \vec{r}_N, \gamma$)
        $\vec{v}' \leftarrow (1-\gamma)\,\vec{u}$
        $\delta \leftarrow \left\|\vec{v} - \vec{v}'\right\|$
        $\vec{v} \leftarrow \vec{v}'$
        $i \leftarrow i + 1$
    return $\vec{v}, \Pi$
```

However, this algorithm amounts to iterative computation of Algorithm 6. Furthermore, the current upper bound for the computation of the value (and thus, the optimal policy) of a general zero-sum stochastic game is in EXPTIME [CMH08], which is a very pessimistic estimate. Hence, we describe a specialized algorithm for perfect information zero-sum stochastic games which also yields pure policies [LL69]. It is ideologically similar to the policy iteration scheme given in Alg. 3. The idea behind it is technically simple: If one player, say, CON, has already committed to the policy, then from the perspective of NAT the problem reduces to optimizing a Markov decision process. The same happens if the players are switched, as the scenario is symmetrical, and the decisions are not simultaneous. This allows us to optimize the policies for each of the players independently. Intuitively, the algorithm alternates between computation of the optimal policy for CON and NAT with $\gamma \to 1$ until the policies do not change.

Algorithm 8 Optimal policies for the expected average reward criterion, perfect information version

```
function PI-GAMEDISCOUNTINGLIMIT($S = S_C \uplus S_N, A_C, A_N, P, \vec{r}_C, \vec{r}_N$)
    $i \leftarrow 0, v \leftarrow \infty$
    $\pi_C, \pi_N \leftarrow \vec{0}$                                  ▷ Initialize arbitrary policies
    while $\delta > \varepsilon$ do
        $\gamma \leftarrow 1 - 2^i$
        $\pi_C \leftarrow \arg\max_\pi v_{C,\gamma}^{(\pi,\pi_N)}$, $\pi_N \leftarrow \arg\max_\pi v_{N,\gamma}^{(\pi_C,\pi)}$
        $v' \leftarrow (1-\gamma) v_{C,\gamma}^{(\pi_C,\pi_N)}$
        $\delta \leftarrow \left|v - v'\right|$
        $v \leftarrow v', i \leftarrow i + 1$
    return $(\pi_C, \pi_N)$
```

1.7 Multi-objective optimization

A significant part of this work deals with properties of multi-objective problems. Prior to this, we give an overview of this topic. In many optimization problems in the engineering context, one seeks to optimize several measures, such as costs and time, simultaneously. Mathematically speaking, one may say that the optimization problem has a formulation

$$\begin{aligned} &\max c(x) \\ \text{s.t.}& \\ &x \in X \end{aligned} \tag{1.25}$$

where X is a subset of $\mathbb{R}^n$ and $c\colon \mathbb{R}^n \to \mathbb{R}^m$ is a vector-valued function. Often, X is given by an indirect definition, such as

$$X = \left\{x \in \mathbb{R}^n \mid g(x) \geqslant 0\right\}$$

where $g\colon \mathbb{R}^n \to \mathbb{R}$ is a problem-specific *constraint function*.

This immediately implies further questions. As the set of values of c is vector-valued and not totally ordered, it is unclear what to consider a maximum. Hence, the problem (1.25) is not well-defined unless there are further assumptions.

The Pareto frontier Given a multi-objective optimization problem such as (1.25), one can proceed mathematically and observe that since the element-wise order relation $\leqslant$ on $\mathbb{R}^m$ is not total, then there is a set $P \subseteq X$ with the property that for each $x \in P$, there is no other $x' \in X$ with a better goal function value $c(x')$, i. e., $P = \left\{x \mid \nexists x' \in X : c(x') \geqslant c(x) \wedge c(x') \neq c(x)\right\}$. The set P is then called the *non-dominated set* or *Pareto frontier*. Then, the problem (1.25) can be interpreted as the task of finding the set P.

The complexity of computing the set P is, however, dependent on its size. It is easy to see that if there is an exponential number of non-dominated values, then the complexity of finding P is also at least exponential. However, this is a very coarse approach. For problems with large output sizes, there has been proposed a finer classification [PY00].

- If it is possible to enumerate the desired set $P = \{x_1, \dots, x_N\}$ in such a way that the computation of the $i+1$st element x_{i+1} takes only polynomial time in terms of problem size, given the elements $x_1, \dots, x_i$, then the complexity of enumerating P is *polynomial delay* (PD).
- If it is possible to enumerate the desired set $P = \{x_1, \dots, x_N\}$ in such a way that the computation of the $i+1$st element x_{i+1} takes only polynomial time in terms of problem size and i, given the elements $x_1, \dots, x_i$, then the complexity of enumerating P is *incremental polynomial time* (IPT).

The focus on enumeration problems in this classification is motivated by practice: In most cases, even if the Pareto frontier is exponentially large, to compute at least a subset in polynomial time in order to give the decision maker useful options is considered useful; if the enumeration problem is at least in IPT, then the corresponding algorithm can be stopped after a sufficient number of solutions has been generated, without having to wait until the exponentially large Pareto frontier has been fully computed.

Canonical problem Another approach to a multi-objective problem such as (1.25) may be to define a value y and ask to find a value $x \in X$ such that $c(x) \geqslant y$. This is known as the *canonical optimization problem*. In many cases, the canonical optimization problem is NP-hard even if optimizing every single component of c is easy, as the function c can be designed in such a way that it captures multiple components of a problem in a sufficiently structured way [Ehr05].

Scalarization A structurally simple way of dealing with a multi-objective optimization problem lies in converting it into a single-objective one by assigning weights $\vec{w} = (w_1, \ldots, w_m) \in \mathbb{R}^m$ to the individual components of the objective function and optimizing $\vec{w}c(x)$. This approach is especially fruitful for linear problems. Furthermore, as for each vector $\vec{w}$ the solution of the scalarized problem lies in the Pareto frontier, then one may also consider finding all possible solutions of the scalarized problem with the weighting vector as a free parameter. It is easy to see that the resulting set W is a subset of the Pareto frontier; furthermore, one can see that in the objective function space, the values of W span the convex hull of P. A visualization of the relation between P and W can be seen in Fig. 1.2: the dots are the images of P, and the line spans the images of W. For multi-objective linear problems, there exists a generic "off-the-shelf" algorithm that computes the set W of the extreme points on the convex hull [Ben98]. We note that, unfortunately, in our setting, it is not applicable, as most of the problems considered in this thesis are non-linear.

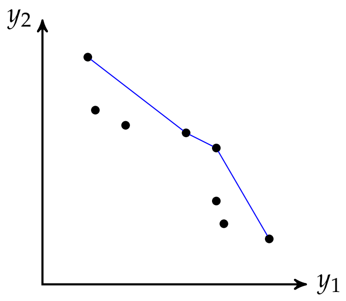


Figure 1.2: The solution space of a multi-objective optimization problem

2 Theory of parametric models

Nothing is more practical than a good theory.

— Kurt Lewin

IN this chapter, we discuss the properties of models that arise when the standard MDP model as described in [Put94] is extended with parameter uncertainty in transition probabilities and, possibly, rewards. First, we discuss known previous work on parametric models. It turns out that the model introduced in [GLD00] which is called *bounded-parameter Markov decision processes* is practical to use as a base for further extensions. Then, we explain known properties of this model, including the *interval value iteration* algorithm for the expected discounted reward criterion. Proceeding to own work, we first show partial equivalence to stochastic games, which also solves the problem of finding an optimal policy for the expected average reward criterion and then turn to the finite-horizon reward criterion. Dealing further with the model, we consider multi-objective problems that arise in its context and their complexity-theoretic properties.

Concerning the structure of this chapter, we introduce the formalism, survey known results and discuss possible alternatives to it in Section 2.1. Then, in Sections 2.2 and 2.3, we consider alternative optimality models to the expected discounted reward criterion. In Section 2.4, we discuss a related multi-objective optimization problem and respective solution approaches, and, finally, Section 2.5 introduces and analyses more general models of parameter uncertainty.

This chapter is based on the publications [Sch15, SBHH17, BS17a].

2.1 Background

The main motivation in the uncertainty models we discuss in this chapter is the insight that the parameters (and, most notably, the transition probabilities) in a Markov decision process are derived from empirical data and are therefore subject to loss of precision. Reasoning about this, a valid idea is to consider stochastic processes that may have varying parameters, or, speaking in terms of mathematical optimization, of an *uncertainty set* of parameters. This leads us into the domain of *robust optimization*, optimization of a function where we control only a part of the input while the rest of it is controlled by an adversary that tries to counter our optimization effort. If solved, such a problem will yield a *pessimistic solution* that is always feasible and optimal for each decision of a potential adversary.

However, adversaries do not need to play optimally. When modeling the environment as an adversary, we have to keep in mind that it does not inherently want to minimize our objective function, and therefore, a pessimistic solution may be too conservative. This may lead us to the question of finding policies that are "nearly always good"; however, this is a very vague term. In the following discussion, we consider two alternative definitions that try to capture this idea. The first is the concept of *percentile optimization*, that assumes a probability distribution on the uncertainty set and declares the goal to optimize the probability of reaching a given objective. While being intuitive, this definition implies optimization of volumes, which is an inherently hard problem. Another option is to consider performance in different scenarios as individual goal functions and optimize them all. The latter approach leads to multi-objective optimization, where the optimal solution is not necessary unique and optimality may not be a total order, leading to a (possible exponentially large) solution set. To evade this, one can consider a weighted sum of the goal functions, which together yields a *stochastic multi-scenario optimization* problem. This multi-scenario problem can be considered for several MDPs with shared state and action spaces which are grouped together in a *concurrent MDP*, and extended to the BMDP setting with the pessimistic and optimistic solution as (partial) objective functions. In both cases, the goal is to optimize a weighted sum of resulting value vectors.

2.1.1 Basic formalisms

In our further discussion, we use the uncertainty model introduced in [GLD00], the bounded-parameter Markov decision process. The formalism describes a set of Markov decision processes that is given by upper and lower bounds on transition probabilities. The original publication [GLD00] also included bounds on rewards on the formalism; in this work, we consider the rewards to be certain.

Definition 2.1.1 (Bounded-Parameter Markov decision/reward process [GLD00]). Given a set of states $S = [n]$, a pair of matrices $\boldsymbol{P}_\updownarrow = (\boldsymbol{P}_\downarrow, \boldsymbol{P}_\uparrow)$, an initial distribution $\vec{q} \in \mathbb{R}^n$, and a *reward vector* $\vec{r} \in \mathbb{R}^n$, a *bounded-parameter Markov reward process* is a tuple $\left(S, \boldsymbol{P}_\updownarrow, \vec{r}, \vec{q}\right)$ that defines a set of Markov reward processes $\left\{(S, \boldsymbol{P}, \vec{r}, \vec{q}) \mid \boldsymbol{P}_\downarrow \leqslant \boldsymbol{P} \leqslant \boldsymbol{P}_\uparrow\right\}$.

Analogously, for a set of states $S = [n]$, actions $A = [m]$, a reward vector $\vec{r} \in \mathbb{R}^n$, an initial distribution $\vec{q} \in \mathbb{R}^n$, and m pairs of transition matrices $T_\updownarrow = \left\{\boldsymbol{P}^1_\updownarrow, \dots, \boldsymbol{P}^m_\updownarrow\right\} \subset \mathbb{R}^{n\times n}$, a *bounded-parameter Markov decision process* $M_\updownarrow = \left(S, A, T_\updownarrow, \vec{r}, \vec{q}\right)$ is a set of Markov decision processes

$$\left\{(S, A, T, \vec{r}, \vec{q}) \mid T = \left\{\boldsymbol{P}^1, \dots, \boldsymbol{P}^m\right\}, \boldsymbol{P}^a_\downarrow \leqslant \boldsymbol{P}^a \leqslant \boldsymbol{P}^a_\uparrow \wedge \boldsymbol{P}^a\vec{1} = \vec{1}, a \in A\right\}.$$

As previously noted, this definition assumes that the rewards are certain, in contrast to the definition in [GLD00]. We argue that this does not reduce modeling power with the following construction. We introduce, for each state s three states, a low-reward state $s_\downarrow$, a high-reward state $s_\uparrow$, and a non-determinism state $\tilde{s}$ where for every action a, the probability bounds for transitioning to a state $\tilde{t}$ in $s_\downarrow$ and $s_\uparrow$ are identical to the probability bounds for transitioning from s to t under a and the probability bounds intervals for transitioning from $\tilde{s}$ to $s_\downarrow$ and $s_\uparrow$ are $[0, 1]$ for every action. This way, the states $s_\downarrow$ and $s_\uparrow$ can model uncertain rewards.

In this work, we talk about the "average performance" of bounded-parameter Markov decision processes; to do this, we need to extend the original formalism with a probability measure on the set of possible transition matrices. This probability measure will be mostly used to compute $\mathrm{Ex}_{M\in M_\updownarrow}[\boldsymbol{P}^a]$, the expected value of the transition matrices over the whole uncertainty set which produces an "average" MDP $M_\times = \mathrm{Ex}_{M\in M_\updownarrow}[M]$. Extending the formalism, we arrive at the following definition.

Definition 2.1.2 (Stochastic BMRP, BMDP [SBHH17]). For a bounded-parameter Markov reward process $\left(S, \boldsymbol{P}_\updownarrow, \vec{r}, \vec{q}\right)$ and a probability density function p on the set

$$\tilde{P} = \left\{\boldsymbol{P} \mid \boldsymbol{P}_\downarrow \leqslant \boldsymbol{P} \leqslant \boldsymbol{P}_\uparrow \wedge \boldsymbol{P}\vec{1} = \vec{1}\right\}$$

a *stochastic bounded-parameter Markov reward process* (SBMDP) is the tuple $\left(S, \boldsymbol{P}_{\updownarrow}, \vec{r}, \vec{q}, p\right)$. In a similar fashion, adding a probability measure on the transition matrices for each action we can define a *stochastic bounded-parameter Markov decision process* $\left(S, A, T_{\updownarrow}, \vec{r}, \vec{q}, p\right)$ where p is a probability density measure on

$$\hat{P} = \bigtimes_{a \in A} \left\{ \boldsymbol{P}^a \mid \boldsymbol{P}^a_{\downarrow} \leqslant \boldsymbol{P}^a \leqslant \boldsymbol{P}^a_{\uparrow} \wedge \boldsymbol{P}^a \vec{1} = \vec{1} \right\}.$$

An SBMDP induces an "average" MDP $M_{\times} = (S, A, T_{\times}, \vec{r}, \vec{q})$ with $T_{\times} = \left\{ \boldsymbol{P}^1, \ldots, \boldsymbol{P}^m \right\}$ and

$$(\boldsymbol{P}^1, \ldots, \boldsymbol{P}^m) = \mathrm{Ex}_p[P \in \hat{P}] = \int_{\hat{P}} P p(P) \, \mathrm{d}P$$

Given an uncertainty set of possible models, the main question is now to devise a notion of optimality for a policy. In the most trivial case, one could optimize policies for the average MDP $M_{\times}$; however, there is not much mathematical value in the problem, as computing an expected value and optimizing an MDP is possible with well-known mathematical methods.

Hence, we consider the complete uncertainty set $M_{\updownarrow}$ of Markov decision processes and seek for policies that are, with respect to the chosen optimality criterion, either

- optimal for all realizations of an MDP $M \in M_{\updownarrow}$ or
- optimal if $M \in M_{\updownarrow}$ optimizes the reward criterion, too.

Mathematically speaking, we search for policies $f_{\downarrow}, f_{\uparrow}$ which fulfill

$$\begin{aligned} f_{\downarrow} &= \arg\max_{f} \min_{M \in M_{\updownarrow}} \vec{v}^{(f)}(M), \\ f_{\uparrow} &= \arg\max_{f} \max_{M \in M_{\updownarrow}} \vec{v}^{(f)}(M). \end{aligned} \tag{2.1}$$

for a value function $\vec{v}^{(f)}(M)$ that maps the policy f and the MDP M to the value of f in M. For convenience, we designate by $\vec{v}^{(f)}_{\downarrow}, \vec{v}^{(f)}_{\times}, \vec{v}^{(f)}_{\uparrow}$ the value vectors

$$\min_{M \in M_{\updownarrow}} \vec{v}^{(f)}(M), \vec{v}^{(f)}(M_{\times}), \max_{M \in M_{\updownarrow}} \vec{v}^{(f)}(M).$$

We call these values the *worst case*, *average case*, and *best case*, respectively; in some cases alternative terms such as *pessimistic* and *optimistic value* are used for the lower bound and the upper bound terms.

Furthermore, we consider the multi-scenario optimization setting. There, the uncertainty set is discrete and consists of finitely many *scenarios* which may occur when the policy is executed. Mathematically, it translates into a set of several MDPs which have common action and state spaces but different transition probabilities and rewards.

Definition 2.1.3. For $K \in \mathbb{N}$, let $\mathcal{M} = \{M_1, \ldots, M_K\}$ be a set of K MDPs with a common state space S and action space A. We call $\mathcal{M}$ a *concurrent* Markov decision process. We designate its transition probability matrices by $\boldsymbol{P}^a_k$ and its reward vectors by $\vec{r}_k$ where $k \in [K]$ is the index of the MDP $M_k \in \mathcal{M}$ and $a \in A$ the corresponding action. For a policy f we write

$$\vec{v}^{(f)}_1, \ldots, \vec{v}^{(f)}_K = \vec{v}^{(f)}(M_1), \ldots, \vec{v}^{(f)}(M_K)$$

for the value vectors resulting from f in the MDPs $M_1, \ldots, M_K$.

2.1.2 Similar models

One may encounter different notions of uncertainty in MDPs in literature. One of the earliest extensions the Markov decision process formalism with uncertainty can be found in the work of Silver [Sil63]. The author considers a perspective where the transition probabilities are uncertain with a given probability distribution and tries to infer the true transition probabilities and optimize the outcomes from runtime behaviour.

The robust perspective, building on Silver's foundations, has been introduced by Satia and Lave [SL73]. They considered Silver's model with convex uncertainties (*Markov decision processes with imprecise probabilities*, MDP-IP) over the transition probabilities and formulated the robust optimization problem in the framework of stochastic games introduced by Shapley [Sha53]. Also, a policy iteration procedure for the expected discounted reward criterion has been proposed in this work. Later, White and El-Deib [WED94] developed a value iteration procedure for the same problem and showed its convergence. In these works, uncertainty is being assumed to be convex, that is, for a state s and action a, the uncertainty set U_s^a for the transition probability vector $\boldsymbol{P}^a(s\bullet)$ is assumed to be convex. This work is also important from the perspective that it (at least implicitly) states the main assumption in the uncertainty model: The uncertainty sets are independent across different states and actions. This assumption, better known as the *rectangularity property*, has been discussed in the work of Iyengar [Iye05].

More work on the rectangularity property has been done by Wiesemann et al. [WKR13]. There, the authors describe a more general rectangularity property where the uncertainty sets for the transition probabilities from a given state can depend on the chosen action in this state but not on other states. This property is motivated by the perspective of modeling uncertain Markov decision processes from long-time observation of a system behaviour under varying policies. The methods of Wiesemann et al. allow one to derive an uncertainty region with given confidence from historical observations and compute a robust policy for it; the efficiency of the provided algorithms varies depending on the shape of the uncertainty set.

A different approach to deriving MDPs from historic data has been undertaken by Nilim and El Ghaouli [NG05] where a likelihood function on the transition frequency matrix has been defined. Given a *likelihood threshold* which serves as a lower bound for the likelihood function, it is possible to define an uncertainty set in this setting and to compute corresponding robust policies.

From the application perspective, robust MDP models have been applied in a discussion of multi-armed bandit problems [CD15] and product line design problems with model uncertainty [BM17].

Several works are related to concurrent MDPs. A more general notion is a Markov decision process with coupling constraints, which is often motivated by combining several MDPs into one. An ILP formulation of specially constrained MDPs can be found in the paper of Dolgov and Durfee [DD05]; an approach to merge MDPs to control several parallel tasks with constraints on the action space can be observed in [SC97]. For MDPs with coupling constraints, mathematical relaxations of the exact optimization problems have been considered by Hawkins [Haw03] and Adleman and Merserau [AM08] where the global optimization problem is decomposed into smaller sub-problems, based on the coupling constraints. A tighter upper bound for the decomposition approach has been given by Bertsimas and Mišić [BM16]. A further coupled MDP formalism are multi-agent models where agents perform a joint task on a shared state space and the reward function depends on the combined decision of all agents [GKP01].

There exist also several applications of multi-scenario stochastic models which are not directly related to Markov decision models [RW91]. Examples include product line design problems [BM17] and healthcare decision making [BST16] where different scenarios of system evolution are defined and optimization of a compromise strategy is performed.

The framework of Givan et al. [GLD00] we are using here is derived from a state aggregation approach [DGL97, BDFS17] and assumes interval-like uncertainties. In theory, the algorithms for the robust optimization problem for BMDPs are computationally not much less complex than those for MDP-IPs, and in our further discussion we use BMDPs while keeping in mind that the same approach is in most cases transferable to the MDP-IP model.

Recently, there has been some effort in the direction of percentile optimization [DM10] for uncertain MDP models. In a way, we may perceive this as the continuation of work began by Silver, however, percentile optimization is inherently a hard problem that can be solved only for very specific probability distributions.

The multi-objective perspective for MDPs has been explored largely in the context of Markov decision processes with multi-objective rewards. White shows in [Whi82] a value iteration-based method to compute the Pareto frontier for MDPs with vector-valued rewards, which, however, may yield non-stationary policies; this approach has been extended to stochastic games by [CMH08]. A policy iteration-based IPT algorithm to enumerate the Pareto frontier is presented in [WdJ07], which, however, has been shown to work only with deterministic Markov decision processes. Different scalarization approaches that formalise the notion of a "compromise policy" have been explored by [PW10]. Following up on this, a large body of work exists on computing all compromise policies for a given scalarization metric [BN08, RSS$^+$14, RWO13]. Further research concerns itself with complex Boolean queries for policies that yield minimal or maximal rewards [HHH$^+$17].

Our work, as stated, uses mostly the model of Givan et al. and the concurrent MDP model. However, other ideas (such as convexity of the uncertainty set or the existence of a probability distribution over the uncertainty set) are considered in this chapter. Furthermore, we see in Sec. 2.5 what happens if we depart from the rectangularity assumption.

2.1.3 Interval value iteration

The work [GLD00] makes an important contribution to BMDP theory by developing an algorithm that computes $f_\downarrow$ and $f_\uparrow$ for the expected discounted reward criterion. Here, we present the algorithm as well as the arguments for correctness. In later discussion, we infer the same properties by showing a fully general (partial) equivalence result.

The core of the algorithm (Alg. 9) is a modified value iteration with an additional optimization step inside the value iteration. For better readability, the extra part is highlighted as an additional function INTERVALVALUE in lines 13–27 which is dependent on the parameter $b \in \{\downarrow, \uparrow\}$. This parameter steers the bound that is computed: if $b = \downarrow$, then we compute the optimal lower bound (and, implicitly, the optimal policy for it), i. e., $f_\downarrow$ and $\vec{v}^{(f)}_\downarrow$. If $b = \uparrow$, we compute, respectively, $f_\uparrow$ and $\vec{v}^{(f)}_\uparrow$.

Looking more carefully at the interval value iteration algorithm, we see that the function INTERVALVALUE computes, depending on b,

$$\mathrm{opt}_{\vec{p}:\vec{p}\cdot\vec{1}=1, \boldsymbol{P}_\downarrow(s\bullet)\leq\vec{p}\leq\boldsymbol{P}_\uparrow(s\bullet)}(\vec{p}\cdot\vec{v}),$$

where opt $=$ min, if $b = \downarrow$, and opt $=$ max otherwise. We note here that this procedure is very similar to the value iteration algorithm for perfect-information stochastic games; later, we elaborate on this more. What we also see here is that the policy computed by this algorithm is stationary and deterministic, and the algorithm itself relies on the convergence of $\vec{v}$ until a pre-defined precision is reached. The convergence holds by application of Banach's fixed point theorem; the convergence rate can be shown to be γ and the optimal policy is independent of $\vec{q}$ [GLD00]; this allows one again to argue, similarly to the argument for the MDP value iteration algorithm, that the difference between the computed vector $\vec{v}$ and the value vector of the optimal policy will be at most $\frac{\varepsilon}{2}$.

Algorithm 9 Interval value iteration

```
function IntervalValueIteration($S, A, T_{\updownarrow}, \vec{r}, \gamma, b \in \{\downarrow, \uparrow\}$)
    $\delta \leftarrow \infty$
    $\pi \leftarrow \vec{0} \in A^S$
    $\vec{v} \leftarrow \vec{0}$
    while $\delta > \frac{\varepsilon(1-\gamma)}{2\gamma}$ do
        for $s \in S$ do
            $a \leftarrow \max_{a \in A}$ IntervalValue($s, \boldsymbol{P}^a_{\downarrow}, \boldsymbol{P}^a_{\uparrow}, \vec{r}, \gamma, \vec{v}, b$)
            $\vec{v'}(s) \leftarrow$ IntervalValue($s, \boldsymbol{P}^a_{\downarrow}, \boldsymbol{P}^a_{\uparrow}, \vec{r}, \gamma, \vec{v}, b$)
            $\pi(s) \leftarrow a$
        $\delta \leftarrow \max_{s \in S} \left| \vec{v'}(s) - \vec{v}(s) \right|$
        $\vec{v} \leftarrow \vec{v'}$
    return $\vec{v}, \pi$
function IntervalValue($s, \boldsymbol{P}^a_{\downarrow}, \boldsymbol{P}^a_{\uparrow}, \vec{r}, \gamma, \vec{v}, b$)
    if $b = \downarrow$ then
        $(s_1, s_2, \dots, s_n) \leftarrow$ *ascending* order of states with respect to $\vec{v}$
    else
        $(s_1, s_2, \dots, s_n) \leftarrow$ *descending* order of states with respect to $\vec{v}$
    $\vec{p} \leftarrow \boldsymbol{P}_{\downarrow}(s\bullet) \in \mathbb{R}^n$          ▷ Initialize transition probability vector
    $r \leftarrow 1 - \vec{p} \cdot \vec{1}$
    for $s' \in (s_1, \dots, s_n)$ do
        if $r > 0$ then
            $m \leftarrow \boldsymbol{P}_{\uparrow}(s, s') - \vec{p}(s')$
            $d \leftarrow \min\{m, r\}$
            $\vec{p}(s') \leftarrow \vec{p}(s') + d$
            $r \leftarrow r - d$
    $v \leftarrow \vec{p} \cdot \vec{v}$
    return $\vec{r}(s) + \gamma v$
```

It is worth noting that, since the algorithm relies on convergence with the rate γ, its runtime is polynomial if γ is constant. The time complexity of Algorithm 9 is, to be precise, $\mathcal{O}\left(|S|^2 \log |S| |A| \frac{\log \varepsilon - \log |\vec{r}|}{\log \gamma}\right)$. In general, this running time complexity is not polynomial and one could ask oneself if there are faster algorithms that find optimal policies for the expected discounted reward in BMDPs. Later on, we derive results that answer this question, at least partially.

2.2 Finite-horizon properties

Previous results on uncertain MDPs are mostly concerned with the expected discounted reward measure. However, there are at least two more different optimality criteria we are interested in. For now, we consider the finite-horizon expected total reward. In particular, we show that it is, in contrast to optimization problems for "stationary" criteria such as expected discounted reward, computationally hard to find optimal policies. The reason for it lies in the observation that optimal policies for the finite-horizon total reward criterion are local, while the choice of one specific MDP from the uncertainty set is global.

We consider the complexity of the following problem: Given a bounded-parameter Markov decision process $M_{\updownarrow}$ and a number of steps $N \in \mathbb{N}$ in binary encoding, what is the optimistically optimal policy for the total reward over the finite horizon N? For a hardness result, we reformulate this question into a decision problem.

Theorem 2.2.1. *For a constant numerical precision, a given minimum reward $\vec{u} \in \mathbb{R}^n$, and a given horizon $N \in \mathbb{N}$ (in binary encoding), deciding if the optimistically optimal policy in a given bounded-parameter Markov decision process $M_\updownarrow$ with n states has expected total reward of at least $\vec{u}$ in N time steps is* NP*-complete.*

Before proving the statement above, we provide an intuition for the hardness of the problem. The problem basically asks for an optimistic Markov decision process from a set of MDPs given by a bounded-parameter Markov decision process. The problematic aspect is time: As we specifically consider a finite time horizon, at different time steps different actions in one state may lead to better performance. However, as we have to find one globally optimal MDP, the usual approach of considering each time step individually, beginning from the last one, may lead to inconsistent assignments of transition probabilities across different states. Hence, the usual dynamic programming approach does not work. On the other hand, exploiting the dependencies in the same state between different time steps leads to the desired hardness result.

Proof. We show NP-hardness first. Intuitively, our result relies on the need, for any algorithm that solves the given problem, to enforce equal parameter values in the optimistic Markov decision process at all time steps. This adds sufficient structure to the problem to encode global constraints that make NP-hard problems so hard.

We perform a reduction from the directed Hamiltonian path problem (DHP) which is known to be NP-complete [GJ79]. Let a graph $G = (V, E)$ be an instance of DHP. We construct an intermediate graph $G' = (V', E')$ as follows: V' is obtained from V by adding two special vertices s and t, and E' is obtained from E by adding for each vertex $v \in V$ two edges, (s, v) and (v, t). Having defined G', we derive a bounded-parameter Markov decision process $M_\updownarrow^{G+}$ from G by using V' as the state set, defining a single action a for each state, making t an absorbing state and setting the intervals for the transition probabilities to $[0, 1]$ wherever an edge in E' exists.

The reward function is defined as follows. We introduce transition-dependent rewards, as an MDP with transition-dependent rewards can, by construction in subsection 1.5.2, be transformed into an MDP with state-dependent rewards. For each edge in E, the reward is 1. For all outgoing edges $(s, v) \in E'$ from s, the reward is 0, and for all edges $(v, t) \in E'$ from V to t, the reward is 1.5; for the edge (t, t), the reward is 0. It is easy to see that this is a polynomial-time construction.

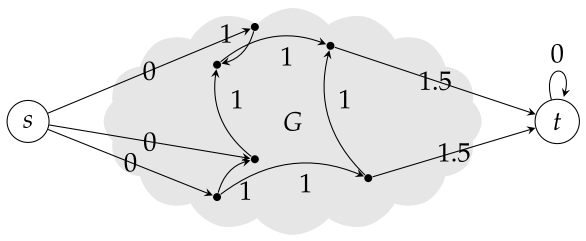


Figure 2.1: A visualization of the reduction from DHP

We observe that there exists a directed Hamiltonian path in the graph G' if and only if there is a directed Hamiltonian path in the graph G: the only possibility to construct a directed Hamiltonian path in G' is to start at s, construct a directed Hamiltonian path in G, and end in t. We also see that if there exists a directed Hamiltonian path in G', then it is possible to derive a Markov decision process $M \in M_\updownarrow^{G+}$ with expected total reward $|V| + \frac{1}{2}$ over time horizon $|V| + 1$ by setting the transition probabilities to form a Hamiltonian path in G'.

If there is no directed Hamiltonian path in G', then the expected reward is strictly less than $|V| + \frac{1}{2}$. To show this, we first note that no path of length $|V| + 1$ that starts in s can yield a reward greater than $|V| + \frac{1}{2}$, since, after reaching t, no further reward can be gained and before reaching t, one has only $|V| - 1$ transitions to gain nonzero reward which can be only 1 per transition. It follows now that in order to achieve expected reward $|V| + \frac{1}{2}$, all paths of nonzero probability have to return a reward of $|V| + \frac{1}{2}$. But, if this is the case, and no directed Hamiltonian path exists, every path of nonzero probability must contain a loop. If there exists a path $p = (s, \ldots, t)$ with a loop $(v_1, v_2, \ldots, v_1, v_k)$, then there must be also a shorter path p' that can be obtained from p by contracting the loop to a simple subpath (v_1, v_k) and extending it at the sink state t. So, another path is created which, by construction, has nonzero probability and yields a reward strictly less than $|V| + \frac{1}{2}$. Thus, the expected total reward is less than $|V| + \frac{1}{2}$ which is a contradiction to the initial conjecture.

We complete the proof by showing that the problem belongs to NP. Since the precision is constant, the number of bits that describe the distribution for the transition probabilities is at most polynomial in the input length, and thus, it is possible to non-deterministically guess the transition probability matrices $\boldsymbol{P}_1, \ldots, \boldsymbol{P}_m$ in polynomial time. To verify that the expected total reward will be at least R, one can use dynamic programming to compute the rewards iteratively for the time steps $N, N-1, \ldots, 1$. □

The construction from Theorem 2.2.1 makes stronger statements possible. It is easy to observe that the construction from Theorem 2.2.1 did not rely on multiple possible actions, thus transforming the bounded-parameter Markov decision process into a bounded-parameter Markov reward process or a bounded-parameter Markov decision process with a fixed policy.

Corollary 2.2.2. *For a given BMDP $M_\updownarrow$ with n states, $N \in \mathbb{N}$, $\vec{u} \in \mathbb{R}^n$, and a given policy f, deciding if the optimistic expected total reward over a finite horizon N under f is at least $\vec{u}$ is* NP*-complete.*

The idea from Theorem 2.2.1 allows us to show a lower bound for the complexity of determining a pessimistically optimal policy.

Theorem 2.2.3. *Given a reward $\vec{u} \in \mathbb{R}^n$, and a finite horizon $N \in \mathbb{N}$, deciding if a given BMDP $M_\updownarrow$ with n states has a lower reward bound of at most $\vec{u}$ in N time steps is* NP*-complete.*

Proof. We present a construction similar to the one shown in Theorem 2.2.1. We construct a BMDP $M_\updownarrow^{G-}$ analogously to $M_\updownarrow^{G+}$, but with an extra state t' and differences in rewards: The rewards for the transitions from s to $v \in G$ are $1/2$. The rewards inside G and from G to t are set to $1/2$, the reward for the transition from t to t' is 0 and the reward for transitioning from t' to t' is 1. The only possible transition from t leads to t'. It is easy to see that if there is a directed Hamiltonian path in G, then there is an MDP $M \in M_\updownarrow^{G-}$ that follows this path with probability 1 and gets a reward $1/2 \cdot |V|$ for the time horizon $|V| + 2$, starting from s. Conversely, if there is an MDP $M \in M_\updownarrow^{G-}$ that has expected reward at most $1/2 \cdot |V|$ over time horizon $|V| + 2$, then it must contain a deterministic path from s to t with $|V| + 2$ vertices; otherwise the reward will be either at least $\frac{|V|+1}{2}$ if the MDP induces a loop inside the graph G and at least $\frac{|V|}{2} + 1$ if the path ends in the vertex t' prematurely. The rest follows analogously from the arguments used in Theorem 2.2.1. □

Since the decision problem of finding a lower bound for all policies is at least as hard as finding a lower bound for one fixed policy, we infer the NP-hardness of finding a "worst-case" MDP for a finite time horizon. This implies optimization hardness of a given BMDP over a finite time horizon.

Corollary 2.2.4. *For a given BMDP $M_{\updownarrow}$ with n states, $\vec{u} \in \mathbb{R}^n$, and a given finite time horizon $h \in \mathbb{N}$, it is* NP*-hard to decide either if*

- *there is a policy f such that there is an MDP $M \in M_{\updownarrow}$ such that the expected total reward over the horizon h under f is at least $\vec{u}$ or*
- *there is a policy f such that for all MDPs $M \in M_{\updownarrow}$, the expected total reward over the horizon h under f is at least $\vec{u}$.*

We sketch the proof. Since it is NP-hard to decide if a BMDP $M_{\updownarrow}$ with a (possibly nonstationary) policy f has a lower reward bound of at least R or at most $R - 3\varepsilon$ after h steps, starting from a state s, we construct a BMDP $M'_{\updownarrow}$ that has the following structure: In a special state q, one can perform two actions, a and b. If a is taken, we certainly transition to the state s of $M_{\updownarrow}$ with reward 0. Starting from s, the policy f is applied. If b is taken, we transition to a sink state t with reward $R - 2\varepsilon$. Thus, a policy f' in $M'_{\updownarrow}$ that yields a worst-case reward of at least $R - \varepsilon$ after $h + 1$ steps would imply a lower bound on the reward in $M_{\updownarrow}$ under f of at least R.

Upper bound for the pessimistic optimization problem It is easy to see that a Σ_2^p algorithm can decide if there is a policy that gives a reward of at least R for all MDPs $M \in M_{\updownarrow}$: First, we non-deterministically guess a policy f. Then, we non-deterministically decide if, for all MDPs $M \in M_{\updownarrow}$ the expected total reward in M under f is at least R. This, however, leaves a gap between NP and Σ_2^p and leads to the question of the exact complexity of the problem. Here, we close this gap and show that the discussed problem is "just" NP-complete.

Theorem 2.2.5. *Given a BMDP $M_{\updownarrow}$ with n states, a finite time horizon $N \in \mathbb{N}$, and a value vector $\vec{u} \in \mathbb{R}^n$, deciding if, over the time horizon N, the robust policy for $M_{\updownarrow}$ yields a reward of at most $\vec{u}$ is* NP*-complete.*

Proof. Since NP-hardness follows from Theorem 2.2.3, we only have to show that there exists a non-deterministic polynomial-time algorithm which decides the desired property.

First, we observe that deciding if there is an MDP $M \in M_{\updownarrow}$ such that an optimal policy for M yields a reward of at most $\vec{u}$ can be done in non-deterministic polynomial time: After guessing a Markov decision process M, we can find an optimal finite-horizon policy f for M and decide if f yields a reward of at most $\vec{u}$ in polynomial time with a standard dynamical programming algorithm [Put94].

To prove correctness of this construction, we observe that if there exists a Markov decision process $M \in M_{\updownarrow}$ with value of at most $\vec{u}$ for an optimal policy f, then the pessimistically optimal policy f^* will yield a reward of at most $\vec{u}$ on M. Minimizing over $M \in M_{\updownarrow}$, we conclude that the pessimistically optimal policy f^* will also yield at most $\vec{u}$ for the pessimistic MDP. □

2.3 Stochastic games and limit-average reward properties

We observe that the BMDP formalism has several intuitive similarities with the general notion of stochastic games. In this section, we discuss these similarities and derive an equivalence result. This result allows us to reason about a wide class of optimality criteria and transfer established algorithms for stochastic games to the BMDP model. Furthermore, we establish complexity lower bounds for some problems for bounded-parameter MDPs by showing a relation to problems for stochastic games.

First of all, we can, reasoning on an intuitive level, interpret a bounded-parameter Markov decision process as a stochastic game where the reward vectors $\vec{r}_C, \vec{r}_N$ conform to either

- $\vec{r}_N = \vec{r}_C = \vec{r}$, if we seek to optimize the *optimistic* performance measure or
- $\vec{r}_N = -\vec{r}_C = -\vec{r}$, if we seek to optimize the *pessimistic* performance measure.

Then, the set of possible Markov decision processes that is described by a bounded-parameter Markov decision process $M_{\updownarrow} = (S, A, T_{\updownarrow}, \vec{r}, \vec{q})$ can be interpreted as an action space for NAT while the original action space A of a bounded-parameter Markov decision process can be interpreted as an action space for CON in a perfect-information stochastic game. Moreover, the action space for NAT can be reduced even further, as in most cases it is easier to deal with discrete action spaces and the action space A for CON is, in most applications, also discrete. For each action $a \in A$ and each state $s \in S$, the possible transition probability matrix rows $\boldsymbol{P}^a(s\bullet)$ are constrained by the stochasticity condition $\boldsymbol{P}^a(s\bullet) \cdot \vec{1} = 1$ and by the inequality conditions $\boldsymbol{P}^a_{\downarrow}(s\bullet) \leqslant \boldsymbol{P}^a(s\bullet) \leqslant \boldsymbol{P}^a_{\uparrow}(s\bullet)$. From a geometric point of view, these conditions form together a polytope of possible values for $\boldsymbol{P}^a(s\bullet)$. This implies that there are points $\vec{p}^a_1, \ldots, \vec{p}^a_k$ such that $\boldsymbol{P}^a(s\bullet)$ can be written as a convex combination $\sum_{i=1}^k \lambda_i \vec{p}^a_i$, $\sum_{i=1}^k \lambda_i = 1$. We can interpret this property in a way that, if CON chooses the action a, there are k actions $b^a_1, \ldots, b^a_k$ available for NAT such that the transition probability distribution vector for b^a_i is $\vec{p}^a_i$, for $i \in [k]$. By allowing NAT to choose a randomized policy, we can model the complete polytope of feasible values for $\vec{p}$. We note here that this construction is not complete; below we explain it in a more detailed way.

Having established the similarities between bounded-parameter Markov decision processes and stochastic games, we point out the main difference: In a general stochastic game, NAT does not need to follow a stationary policy and, hence, the transition probabilities from the same NAT-controlled state can be different after every transition; in contrast, in a bounded-parameter Markov decision process, the policy of NAT is stationary. This difference indirectly affects several results in this work, making some problems harder to solve from a computational complexity point of view. We already have seen that on the one hand, upper and lower bounds for the expected total reward after N steps in a BMDP are computationally hard to find. However, we see in this section that this difference does not play a significant role for "stationary" reward measures such as the expected discounted reward measure or the expected average total reward measure.

Although some of the results and methods for general stochastic games will not be applicable in the BMDP scenario, we can easily observe that bounded-parameter Markov decision processes and other similar formalisms are similar to stochastic games: more precisely, a bounded-parameter Markov decision process is a perfect-information stochastic game where NAT has to choose a stationary policy before the game is played. Thus, we can (where applicable) use the stochastic game formalism in order to reuse results and algorithms that find optimal policies.

This observation can be formalized as follows.

Theorem 2.3.1. *Let $C\colon X \times Y^* \to X^*$ be a mapping with $C(X, (y_1, \ldots, y_l)) = (y_k)_{k \in [l], y_k \in X}$ which restricts a sequence to a subsequence of values from a given set X.*

For a given bounded-parameter Markov decision process $M_{\updownarrow} = \left(S, A, T_{\updownarrow}, \vec{r}, \vec{q}\right)$ there exists a perfect-information stochastic game G with state space $S_G = S \uplus S_N$, action spaces $A_C = A$ for CON resp. A_N for NAT, such that for each Markov decision process $M \in M_{\updownarrow}$ there exists a stationary policy $\tau(M)$ for NAT with the following properties.

1. *The probability distribution of sequences of states under any BMDP policy $f\colon S^+ \times A \to [0, 1]$ in M is equivalent to the probability distribution of sequences of CON-controlled states under the policy pair $(f_G, \tau(M))$ with $f_G((s_1, \ldots, s_l), a) = f(C(S, (s_1, \ldots, s_l)), a)$ where the sequence $C(S, (s_1, \ldots, s_l)) = (s_i)_{i \in [l], s_i \in S}$ is the sequence $(s_1, \ldots, s_l)$ limited to CON-controlled states only and $s_1 \in S$.*

2. *For any policy $f_G\colon S_G^+ \times A_C \to [0,1]$ and any state $s_1 \in S$, the probability distribution of sequences of* CON*-controlled states under the policy pair $(f_G, \tau(M))$ is equivalent to the probability distribution of sequences of states under the policy f in M where f is defined by $f(\zeta = (s_1, s_2, \dots, s_l), a) = f_G(\xi = (s_1, \dots, s_2, \dots, s_l), a)$ where $s_i \in S$ for all i and $\zeta = C(S, \xi)$.*

Proof. We start with defining G. As already stated, G has the state space $S_G = S \uplus S_N$ with $S_N = S \times A$ and the action space A for CON. The transition from $s \in S$ with actions $a \in A$ will lead directly to $(s,a) \in S_N$. For NAT, we construct the action space as follows. For each state $s \in S$ and action $a \in A$, we define the *transition polytope*

$$P^{s,a} = \left\{ \vec{p} \mid \boldsymbol{P}^a_{\downarrow}(s\bullet) \leqslant \vec{p} \leqslant \boldsymbol{P}^a_{\uparrow}(s\bullet), \vec{p}\vec{1} = \vec{1} \right\}. \tag{2.2}$$

By definition, $P^{s,a}$ is a convex, bounded polytope. Let $\vec{p}_1^{s,a}, \dots, \vec{p}_{m_{s,a}}^{s,a}$ be the extreme points of $P^{s,a}$. Since $P^{s,a}$ is bounded, it is $P^a = \operatorname{conv}\left(\vec{p}_1^{s,a}, \dots, \vec{p}_{m_{s,a}}^{s,a}\right)$ where $\operatorname{conv}(\vec{p}_1, \dots, \vec{p}_k)$ is the convex hull of the points $\vec{p}_1, \dots, \vec{p}_k$. Finally, we set A_N, the action space for NAT, to be $A_N = \left[\max\left\{m_{s,a} \mid s \in S, a \in A\right\}\right]$, and set the transition probability P_G to

$$P_G(s, a_C, a_N, s') = \begin{cases} 1 & s \in S, a_C \in A, s' = (s, a_C) \\ \vec{p}_{a'}^{s^*,b}(s') & s = (s^*, b) \in S \times A, a' = \min\left(m_{s^*,b}, a_N\right) \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

We now show the existence of the mapping τ from an arbitrary $M \in M_{\updownarrow}$ to stationary policies of NAT. Let $M \in M_{\updownarrow}$. Then, in each state $s \in S$, for each action $a \in A$ of the controller, the row of the transition probability matrix $\boldsymbol{P}^a(s\bullet)$ is a convex combination of $\vec{p}_1^{s,a}, \dots, \vec{p}_{m_{s,a}}^{s,a}$, i. e., $\boldsymbol{P}^a(s\bullet) = \sum_{i=1}^{m_{s,a}} \vec{\lambda}_{s,a}(i)\vec{p}_i^{s,a}$ with $\vec{\lambda}_{s,a} \cdot \vec{1} = 1$. This corresponds to a stationary policy of NAT in G; indeed, it is easy to see that the stationary policy $\tau(M)$ of NAT in G that is defined by the distribution vectors $\vec{\lambda}_{s,a}$ in state s and CON action a yields, by construction, exactly the same transition probabilities to the next CON-controlled state, yielding, by the Markov property, identical distributions on sequences of CON-controlled states. Extending the result to randomized policies yields the first part of the statement.

As for the second part of the statement, we show that the policy f yields the same transition probabilities. By construction of P_G, any policy pair (f_G, τ) in G yields sequences of states $s_1, s_2, \ldots \in S_G^+$ where each CON-controlled state s_i is followed by a NAT-controlled state s_{i+1} and vice versa. We compute now the probability $\Pr_G[s' \mid s, a]$ of landing in the state $s' \in S$ after CON selects action $a \in A$ in state $s \in S$. It is $\Pr_G[s' \mid s, a] = \Pr_G[s' \mid (s,a), \tau(M)] \cdot \Pr_G[(s,a) \mid s, a] = \Pr_M[s' \mid s, a] \cdot 1 = \Pr_M[s' \mid s, a]$ where $\Pr_M$ is the probability to arrive in s' after selecting a in s in the MDP M and $\Pr_G$ is the transition probability in G. Together with the Markov property, this yields the second statement. □

It follows furthermore that, if we set

$$\vec{r}_C(s) = \begin{cases} \vec{r}(s) & s \in S_C \\ 0 & \text{otherwise,} \end{cases} \tag{2.4}$$

then, the probability distribution of the reward sequences in CON-controlled states will be identical to the reward sequence in the MDP. Furthermore, since each play will alternate between CON-controlled states and NAT-controlled states, the reward sequences for CON will predictably depend on the reward sequence in the MDP.

This proof serves as a vehicle to use the stochastic game approach wherever it yields stationary policies for NAT. Note, however, that the rewards for NAT are not defined yet;

to finalize the construction and introduce rewards for NAT, we transfer the two robust optimization scenarios we have introduced in the BMDP model to the stochastic games setting.

max-min scenario The min-max scenario corresponds to optimizing a policy subject to the pessimistic realization of a BMDP. It is easy to see that this scenario corresponds to setting the rewards in the stochastic game to r for CON, and $-r$ for NAT whenever the reward is r in the BMDP, and, hence, to the competitive stochastic game scenario.

max-max scenario The max-max scenario corresponds to optimizing both the MDP and a policy such that both yield maximal rewards. This scenario corresponds to rewards fulfilling the property $\vec{r}_C = \vec{r}_N = \vec{r}$ wherever the BMDP reward vector is $\vec{r}$. However, the max-max scenario can also be seen as an MDP since the goal functions for both players are indistinguishable.

In both scenarios, we set $\vec{r}_C(s) = \vec{r}_N(s) = 0$ for NAT-controlled states $s \in S_N$.

Since stochastic games in these two scenarios are symmetrical (for sufficiently well-behaved optimality criteria, that is, for reward criteria that are symmetrical for CON and NAT), the existence of optimal stationary policies for NAT also implies the existence of optimal stationary policies for CON; this means that any optimality criterion for which optimal stationary policies exist in stochastic games will also yield optimal stationary policies for BMDPs. Hence, we can use the following results from [LL69, FV97].

Theorem 2.3.2 (Theorem 1, [LL69])**.** *For any zero-sum two-player perfect-information stochastic game, there exists an optimal policy pair (π, τ) for the expected average reward criterion such that π and τ are pure policies of* CON *and* NAT*, respectively.*

Theorem 2.3.3 ([Sha53], Theorem 3.1.1, [FV97])**.** *For any zero-sum two-player stochastic game, there exists an optimal policy pair (Π, Θ) for the expected discounted reward criterion such that Π and Θ are stationary policies of* CON *and* NAT*, respectively. If the game has the perfect information property, then there also exist pure optimal policies π, τ for this criterion.*

We note that for these optimality criteria, we have to take care of the reward distribution. As the rewards are being generated only in CON-controlled states, the corresponding distribution changes, and with it, the value of the aggregated reward measure. We have to formally show that these changes do not affect the order of policies with respect to the optimality criteria.

Lemma 2.3.4. *The mapping in Theorem 2.3.1 and (2.4) is monotone with respect to the expected average reward objective function. With respect to the expected discounted reward goal function, monotonicity can be maintained by adjusting the discount factor γ to $\gamma' = \sqrt{\gamma}$.*

Proof. We note that the values of the rewards are kept, but in each play of the game, the rewards are generated only in every second state. Then, we can derive for the two given optimality criteria the following arguments.

Expected average reward It is easy to see that the expected average reward in the game for CON will be exactly the half of the expected average reward in the BMDP.

Expected discounted reward Let $(r_i)_{i\in\mathbb{N}}$ and $(s_i)_{i\in\mathbb{N}}$ be sequences of rewards in the BMDP. For the expected discounted reward measure, we observe that

$$\sum_{i=1}^{\infty} \gamma^{i-1} r_i \geqslant \sum_{i=1}^{\infty} \gamma^{i-1} s_i \Leftrightarrow$$
$$\sum_{i=1}^{\infty} \left(\sqrt{\gamma^2}\right)^{i-1} r_i \geqslant \sum_{i=1}^{\infty} \left(\sqrt{\gamma^2}\right)^{i-1} s_i \Leftrightarrow$$
$$\sum_{i=1}^{\infty} \gamma'^{2i-2} r_i \geqslant \sum_{i=1}^{\infty} \gamma'^{2i-2} s_i,$$

which means that the order of the expected discounted reward measures in the derived stochastic game is kept.

□

These results yield, with the help of Theorem 2.3.1, the following results. The first result concerns itself with the expected discounted reward criterion and can also be found in other publications, such as [GLD00]. However, the proof there required more technical work, that, in a sense, has already been done. Here, we can abstract away from the nature of the solution procedure, and rely on existing results for stochastic games.

Corollary 2.3.5. *For any BMDP, there exists a pure optimal policy π for the expected discounted reward criterion in the min-max and the max-max scenarios.*

The second result directly translates Theorem 2.3.2 to the BMDP scenario.

Corollary 2.3.6. *For any BMDP, there exists a pure optimal policy π for the expected average reward criterion in the min-max and the max-max scenarios.*

It is possible to automatically derive an optimization procedure from this result. In the most naïve case, we can use Theorem 2.3.1 to transform a given BMDP to a stochastic game and compute an optimal policy for it. The complexity of this approach depends on the complexity of the transformation and the complexity of computing an optimal policy for a stochastic game. The complexity of computing an optimal policy for the expected average and expected discounted reward optimality criteria in a stochastic game is still not fully determined; the best known upper bound is sub-exponential and the corresponding decision problem is known to be in NP ∩ coNP [BV07]. The complexity of the transform itself depends on the size of the resulting stochastic game, most importantly on $|A_C| = \max_{s\in S, a\in A} m_{s,a}$. This value depends on the number of vertices of the polytope given by Eq. (2.2). This polytope is known in the literature as the intersection between a hyperplane (in our case, the stochasticity constraint) and a hypercube (in our case, given by the upper and lower bounds) and there exist upper bounds on the number of its vertices. In [O'N71] the upper bound is established to be $K(n) = \left(n - \lfloor n/2 \rfloor\right) \binom{n}{\lfloor n/2 \rfloor}$; furthermore, this bound is tight. Unfortunately, this number is exponential in n and we are interested in finding a better suited method. We show now how to keep the action space for NAT in an implicit representation, without having to compute $|A_N|$ stochastic vectors $\vec{p}_1^{s,a}, \ldots, \vec{p}_{m_{s,a}}^{s,a}$ explicitly.

For the expected discounted reward criterion, we observe that the value iteration procedure in a perfect-information stochastic game relies only on computing the value

$$\min_{a_N \in A_N} \vec{r}_N(s) + \sum_{s'\in S} P(s, \cdot, a_N, s')\vec{v}(s').$$

Rephrasing this expression in terms of linear operators, the optimization problem looks like $\min_{a_N \in A_N} \vec{r}_N(s) + \vec{p}^{s,a_N} \cdot \vec{v}$ for transition probability vectors $\vec{p}^{s,a_N}$ that describe the transition probability from a NAT-controlled state s if NAT picks the action a_N. Here, we see that we optimize a linear function over all vectors $\vec{p}^{s,a_N}$. If the set of vectors is given implicitly by some linear inequality $\boldsymbol{A}\vec{p} \geqslant \vec{b}$, the complexity of this optimization step is still polynomial. Hence, the value iteration procedure for the expected discounted reward criterion can be performed efficiently.

For the expected average reward criterion, the approach we show is a straightforward application of policy iteration for zero-sum stochastic games with perfect information [BR14][1]. The approach in [BR14] relies on iterative policy improvements for MDPs; here, we show that it can be performed efficiently, even if A_N is given implicitly, as in our case, by a set of linear constraints. In the loop of the policy iteration procedure, we first choose a policy π for CON that is optimal under the current policy τ of NAT. Then, we choose a policy τ' that optimizes the actions of NAT under π. This reduces the problem to finding an optimal policy for an MDP under the expected average reward criterion. In [Vei66, Put94], Algorithm 4 is proposed as a policy iteration procedure for MDPs.

Note that in the optimization steps (1.7) and (1.8), there is no need to iterate over all possible decision rules. In step (1.7), $\vec{g}$ is fixed, and $\boldsymbol{P}_\pi$ is variable; in step (1.8), analogously, $\vec{h}$ is fixed, and $\boldsymbol{P}_\pi$ together with $\vec{r}_\pi$ are variable. Both optimizations are, in fact, linear programs, which means that the optimization steps can be performed in polynomial time depending on the number of overall constraints. Since the number of constraints that define the action space for NAT is polynomial, it is possible to efficiently implement policy iteration to optimize the expected average reward for BMDPs. The resulting procedure is depicted in Algorithm 10. It is easy to see that the optimization steps in (2.5) and (2.6) can be performed efficiently, as the optimization problem amounts to a linear program with a polynomial number of constraints and variables.

Algorithm 10 Policy iteration for expected average reward in BMDPs

```
1:  function BMDPAVERAGEPOLICYITERATION($M_\updownarrow$, opt $\in \{\max, \min\}$)
2:      $n \leftarrow 0$, select an arbitrary MDP $M_0 \in M_\updownarrow$.
3:      repeat
4:          $\pi_n \leftarrow$ MDPAVERAGEPOLICYITERATION($M$)
5:          Compute $\vec{g} \in \mathbb{R}^n, \vec{h} \in \mathbb{R}^n$ such that $\boldsymbol{P}^M_{\pi_n}\vec{g} = \vec{g}, \vec{r}_{\pi_n} - \vec{g} + \left(\boldsymbol{P}^M_{\pi_n} - \boldsymbol{I}\right)\vec{h} = \vec{0}$
6:          Choose a $M_{n+1}$ that satisfies
                $$M_{n+1} \in \arg\operatorname{opt}_M \boldsymbol{P}^M_{\pi_n}\vec{g}, \qquad (2.5)$$
            keeping $M_{n+1} = M_n$, if possible.
7:          if $M_{n+1} = M_n$ then
8:              Choose a $M_{n+1}$ that satisfies
                $$M_{n+1} \in \arg\operatorname{opt}_M \left(\vec{r}_{\pi_n} + \boldsymbol{P}^M_{\pi_n}\vec{h}\right), \qquad (2.6)$$
            keeping $M_{n+1} = M_n$, if possible.
9:          $n \leftarrow n + 1$
10:     until $M_n = M_{n-1}$
11:     return $\pi_n$
```

[1]The cited approach works well for a slightly larger class of stochastic games, perfect information games are a special case of it.

Lower bounds for BMDP problems In the preceding discussion, we have successfully transferred results on stochastic games to bounded-parameter MDPs. However, we already have noted that the complexity of the policy optimization problems for stochastic games we have relied on is not known to be polynomial [BV07]. A question arises if it is possible to do better, to derive an algorithm that performs asymptotically better, exploiting some structure of BMDPs. We show now that this is not the case, that is, every stochastic game can be transformed into a bounded-parameter Markov decision process with a sufficiently similar probability distribution on state and reward sequences.

Theorem 2.3.7. *Let $G = (S_G, A_C, A_N, P, \vec{r}_C, \vec{r}_N, \vec{q})$ be a perfect-information zero-sum stochastic game. Then there exists a bounded-parameter MDP $M_\updownarrow = (S_M \supseteq S_G, A, T_\updownarrow, \vec{q})$ and an isomorphic mapping E from policies for* Con *in G to policies in $M_\updownarrow$ such that for every stationary policy τ of* Nat*, an MDP $M = M(\tau) \in M_\updownarrow$ with the following property exists.*

For every policy pair (f, τ) with $f \colon S_G^+ \times A_C \to [0,1]$ a policy of Con *in G, the policy $E(f) \colon S_M^+ \times A \to [0,1]$ yields the same distribution of sequences of states from S_G in the MDP $M(\tau)$.*

To prove the statement, we present a construction that transforms actions of Nat into parameter uncertainty in the bounded-parameter MDP.

Proof. We present a BMDP $M_\updownarrow$ with $\mathcal{O}\left(|A_N| \cdot |S_G|\right)$ states and an action space $A = A_C$. The idea of the construction is to allow for the uncertainty in the BMDP to represent the action space of Nat. To do this, we introduce additional states $\bar{S} = S_N \times A_N$ where $S_N \subseteq S_G$ are the Nat-controlled states. Furthermore, for each Con-controlled state $s_C \in S$, we introduce an additional state $\hat{s}_C$. The transition probability intervals in the BMDP are defined by

$$\begin{aligned}
\boldsymbol{P}^a(s, (s, a_N)) &\in [0,1] && \text{if } s \in S_N, a \in A_C, a_N \in A_N\\
\boldsymbol{P}^a((s, a_N), s') &= P(s, \cdot, a_N, s') && \text{if } s \in S_N, a \in A_C, a_N \in A_N, s' \in S_G\\
\boldsymbol{P}^a(s, \hat{s}) &= 1 && \text{if } s \in S_C, a \in A_C\\
\boldsymbol{P}^a(\hat{s}, s') &= P(s, a, \cdot, s') && \text{if } s \in S_C, s' \in S_G, a \in A_C
\end{aligned}$$

An illustration of the construction can be seen in Fig. 2.2. Furthermore, we define the reward vector $\vec{r}(s)$ by

$$\vec{r}(s) = \begin{cases} \vec{r}_C(s) & s \in S_G\\ 0 & \text{otherwise.}\end{cases}$$

We observe that each stationary policy for Nat can be transformed into a realization of the interval uncertainty by choosing the corresponding transition probabilities from states $s_N \in S_N$. Furthermore, we observe, by analogy to Theorem 2.3.1, that the probability distribution of state sequences is preserved. For two states $s, s' \in S_G$ we have the following two cases.

Case 1 If $s \in S_C$, then the transition probability to s' is $P(s, a, \cdot, s')$. In the BMDP $M_\updownarrow$, the transition probability from s to s' is $\boldsymbol{P}^a(s, \hat{s}) \cdot \boldsymbol{P}^a(\hat{s}, s') = 1 \cdot P(s, a, \cdot, s')$.

Case 2 If $s \in S_N$, then the transition probability to s' is $\sum_{a \in A_N} f(s,a) P(s, \cdot, a, s')$ for a stationary policy f with $\sum_{a \in A_N} f(s,a) = 1$ for all $s \in S_N$. Then there exists an MDP $M \in M_\updownarrow$ with transition probability $\boldsymbol{P}^a(s, (s,a)) = f(s,a)$ and $\boldsymbol{P}^a((s,a), s') = P(s, \cdot, a, s')$. By law of total probability, it is $\Pr[s' \mid s, f] = \sum_{a \in A_N} f(s,a) P(s, \cdot, a, s')$ in M.

Together, this yields the statement. □

Theorem 2.3.7 allows us now to derive the following results.

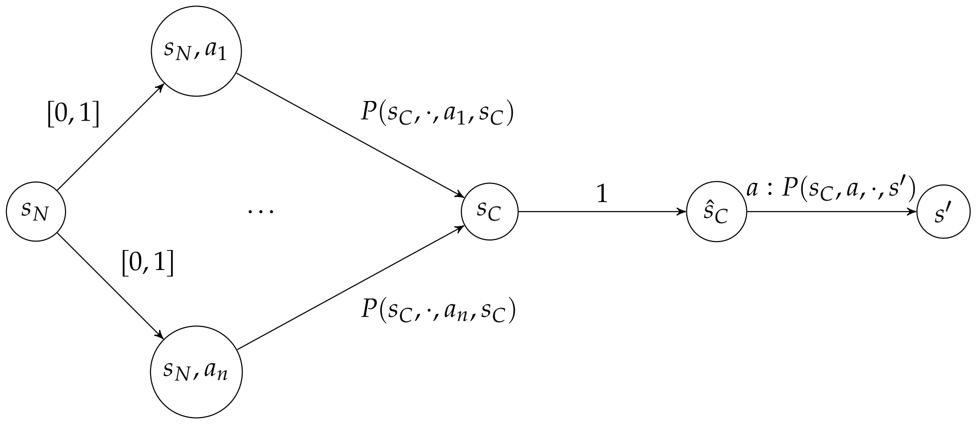

s_N, a_1
$[0,1]$
$P(s_C, \cdot, a_1, s_C)$
s_N
$\dots$
s_C
1
$\hat{s}_C$
$a : P(s_C, a, \cdot, s')$
s'
$[0,1]$
$P(s_C, \cdot, a_n, s_C)$
s_N, a_n

Figure 2.2: Construction from Theorem 2.3.7

Corollary 2.3.8. *The problem of computing an optimal policy for the expected discounted reward criterion for bounded-parameter Markov decision processes is polynomially reducible to the problem of computing an optimal policy for the expected discounted reward criterion in zero-sum perfect-information stochastic games and vice versa; hence, the problems are equivalent under polynomial-time reductions.*

Corollary 2.3.9. *The problem of computing an optimal policy for the expected average reward criterion for bounded-parameter Markov decision processes is polynomially reducible to the problem of computing an optimal policy for the expected average reward criterion in zero-sum perfect-information stochastic games and vice versa; hence, the problems are equivalent under polynomial-time reductions.*

The construction above helps to prove both results, but in each case we need a slightly different argument. For both optimality criteria, we observe that each sequence of l states in a play of the game yields a sequence of $2l$ states in the BMDP, where every second state comes from S_G with the corresponding rewards and all other states yield reward zero. Now we differentiate between the optimality criteria.

Proof of Corollary 2.3.8. For the expected discounted reward measure, the construction requires changes to the discount factor, in analogy to the discussion above. Setting the discount factor to $\gamma' = \sqrt{\gamma}$ in the BMDP yields the equivalence with respect to the reward measure and thus, equivalence with respect to policy optimality. □

Proof of Corollary 2.3.9. For the expected average reward criterion, we see that this yields a factor of exactly $\frac{1}{2}$ in the reward measure, which means equivalence with respect to optimality. □

2.4 Multi-objective approaches

In this section, we consider a different perspective on bounded-parameter MDPs. We have seen that BMDPs themselves define robust optimization problems. Here, we take the perspective that has also been explored in [KKST13] which connects research on robust optimization with research on multi-objective problems. The work in [KKST13] has discussed general robust linear programs; here, we apply similar ideas to (non-linear) BMDP problems. Concerning optimality criteria, we consider here only the expected discounted reward measure with a given discount factor $\gamma \in [0, 1)$. Knowing that the reward measure is non-ambiguous, here we use $\vec{v}$ for the reward vector instead of $\vec{v}_\gamma$ to reduce notation load.

Our discussion here starts with the observation that each policy π for a given BMDP $M_\updownarrow$ yields at least two value vectors, $\vec{v}_\downarrow^{(\pi)}$ and $\vec{v}_\uparrow^{(\pi)}$ for the (policy-dependent) minimal and maximal MDPs $M_\downarrow{}^{(\pi)}, M_\uparrow{}^{(\pi)} \in M_\updownarrow$, respectively. Later in this section, we consider stochastic BMDPs that introduce, following the same considerations, a third value vector $\vec{v}_\times^{(\pi)}$ that corresponds to the performance of π in the "expected" MDP $M_\times \in M_\updownarrow$.

This observation yields a further perspective with respect to applications. In previous sections, we consider only lower and upper bounds, resulting in computation of policies that are optimal in the "best case" and the "worst case". Yet, in an uncertainty setting, rarely the worst or best case scenarios actually happen; hence, if the controller chooses the robust policy, she will likely miss the opportunity to attain higher rewards, and if she chooses the optimistic policy, she will likely lose in the long run. Hence, knowing that all scenarios are possible, the controller might be then interested in a "compromise" policy π that yields acceptable value vectors $\vec{v}_\downarrow^{(\pi)}, \vec{v}_\uparrow^{(\pi)}$ for both scenarios.

From this discussion, several problems arise: First, how can the resulting value vectors be aggregated? Second, is it possible to compute optimal policies with respect to this aggregation function and what is the structure of the resulting optimization problem?

In this section, we consider the *theoretical* properties of the raised problems. That is, we derive here general complexity results and/or point to exact algorithms which might be not too efficient. In Chapter 3, we consider practical, mostly heuristic, implementations and present empirical results from experiments. This section is based on the publications [SBHH17, BS17a].

2.4.1 Initial considerations

Before we discuss further details, we explain specific properties of the raised problems. In the introductory chapter, we have discussed general properties of multi-objective optimization and we have formulated solution approaches for Markov decision processes; in the beginning of this chapter, we have discussed bounded-parameter Markov decision processes. For this section, we present a model that is slightly more general model than bounded-parameter MDPs.

Stochastic BMDPs Our motivation to extending the BMDP model lies in the requirement to capture the "average" performance of a system. While a BMDP can deliver upper and lower performance bounds, often also the expected case is of interest. Here, we use the SBMDP model we have introduced in the beginning; knowing the expected-case MDP in the SBMDP, we can derive, for a policy π, not only the bounds $\vec{v}_\downarrow^{(\pi)}$ and $\vec{v}_\uparrow^{(\pi)}$, but also the average value $\vec{v}_\times^{(\pi)}$. Intuitively, this model expansion does not add to the hardness of the general optimization problem yet extends the model to capture more applications.

Problem formulations From the discussion in Sec. 1.5, we can see that the optimal policy is non-linear in the value vector. Furthermore, this shows that additional constraints on the policy, such as equality of actions in different states, cannot be incorporated into the LP easily and without considerable costs in terms of additional complexity. This observation will be important when we consider the stochastic optimization problem. But prior to discussion of problem properties, we define the problems we discuss here precisely.

Definition 2.4.1 (Multi-objective SBMDP problems)**.** Given a stochastic bounded-parameter MDP $(S = [n], A, T_\updownarrow, \vec{r}, \vec{q}, p)$, we define the following problems.

Canonical multi-objective decision problem Given vectors $u_\downarrow, u_\times, u_\uparrow \in \mathbb{R}^n$, decide if a policy f exists such that it is

$$\vec{v}_\downarrow^{(f)} \geqslant u_\downarrow, \vec{v}_\times^{(f)} \geqslant u_\times, \vec{v}_\uparrow^{(f)} \geqslant u_\uparrow. \tag{2.7}$$

Stochastic optimization decision problem Given a value $u \in \mathbb{R}$ and a *weight vector* $\vec{w} = \left(w_\downarrow, w_\times, w_\uparrow\right) \in \mathbb{R}^3$, decide if there is a policy f such that it is

$$\sum_{s \in S} \vec{q}(s) \left(w_\downarrow \vec{v}_\downarrow^{(f)}(s) + w_\times \vec{v}_\times^{(f)}(s) + w_\uparrow \vec{v}_\uparrow^{(f)}(s) \right) \geqslant u \tag{2.8}$$

Stochastic optimization problem For a given weight vector $\vec{w} = \left(w_\downarrow, w_\times, w_\uparrow\right)$, compute a policy f such that

$$f = \arg\max_f \sum_{s \in S} \vec{q}(s) \left(w_\downarrow \vec{v}_\downarrow^{(f)}(s) + w_\times \vec{v}_\times^{(f)}(s) + w_\uparrow \vec{v}_\uparrow^{(f)}(s) \right) \tag{2.9}$$

Pareto frontier enumeration problem Compute the set of policies F such that

$$\begin{aligned} &\forall f \in F : \nexists f' : S^* \times A \to [0,1] : \vec{v}_\downarrow^{(f')} \geqslant \vec{v}_\downarrow^{(f)} \wedge \vec{v}_\times^{(f')} \geqslant \vec{v}_\times^{(f)} \wedge \vec{v}_\uparrow^{(f')} \geqslant \vec{v}_\times^{(f)} \wedge \\ &(\vec{v}_\downarrow^{(f')} \neq \vec{v}_\downarrow^{(f)} \vee \vec{v}_\times^{(f')} \neq \vec{v}_\times^{(f)} \vee \vec{v}_\uparrow^{(f')} \neq \vec{v}_\uparrow^{(f)}). \end{aligned} \tag{2.10}$$

The resulting set F is also called the *Pareto frontier*.

Convex hull enumeration problem Compute all policies H such that every policy in H is the solution of the stochastic optimization problem.

For the practical part of this work, we are interested in *pure* optimal policies, reducing the search space to a finite (yet still exponentially large) set. For this special case, we introduce additional notation. The Pareto frontier of pure policies is designated by $\mathcal{P}_{\text{Pareto}}$ and the corresponding set of value vectors is designated by $\mathcal{V}_{\text{Pareto}}$.

As for the complexity discussion, we consider mixed stationary policies, as additional constraints on the search space may artificially harden the problem, as it is the case, for example, in linear programming.

2.4.2 Canonical decision problem

First, we establish complexity bounds for the canonical optimization problem. We prove that it is NP-complete when we restrict ourselves to pure policies and NP-hard in the general case.

It has been shown by [CMH06] that the corresponding decision problem is NP-hard for multi-objective MDPs which have a multi-objective reward. The following theorem shows that the hardness result also holds for SBMDPs even if rewards do not depend on the chosen action and even if only two objectives are constrained by the decision problem.

Theorem 2.4.1. *The canonical multi-objective decision problem for pure policies is* NP*-complete.*

Proof. The stated problem is obviously in NP because a policy for an SBMDP can be evaluated in polynomial time. As for NP-hardness, we show a reduction from the subset sum problem. Given a subset sum instance $M = \{m_1, \ldots, m_n\}$, we construct an SBMDP and two vectors $\vec{v}_1, \vec{v}_2$ such that there is a policy π with $\vec{v}_{\downarrow}^{(\pi)} \geqslant \vec{v}_1, \vec{v}_{\times}^{(\pi)} \geqslant \vec{v}_2$ if and only if there is a subset $I \subseteq [n]$ such that $\sum_{i \in I} m_i = \frac{1}{2} \sum_{i=1}^{n} m_i$.

The SBMDP which we construct will have an arbitrary nonzero discount factor $\gamma \in (0,1)$, $4n+1$ states which have the identifiers t and $q_i, s_{\times i}, s_{\downarrow i}, s_{\uparrow i}$ for $i \in [n]$. The general idea of the reduction is the following: The SBMDP proceeds through all states q_i and ends up in the absorbing state t. In the state q_i, it is possible to choose between the (adjusted for the discount factor) reward pairs $(0, 2m_i)$ and (m_i, m_i) with the first component being the pessimistic and the second component being the average (in the SBMDP sense) reward. This way, a pure policy π will induce a subset $I \subseteq [n]$ such that the expected total rewards, if one starts in state q_1, will be $\vec{v}_{\downarrow}^{(\pi)}(q_1) = \sum_{i \notin I} m_i$ and $\vec{v}_{\times}^{(\pi)}(q_1) = \sum_{i=1}^{n} m_i + \sum_{i \in I} m_i$. Technically, we model this by enabling two actions in states q_i, a and b. These actions do not generate rewards, but lead to different outcomes: The action a leads to either $s_{\downarrow i}$ or $s_{\uparrow i}$, with probability in the interval $[0,1]$ and the expected value for this probability being $1/2$; all actions from these two states lead, in turn, to q_{i+1} (or t, if $i = n$), and the difference between the states $s_{\downarrow i}$ and $s_{\uparrow i}$ lies in the reward: the reward in $s_{\downarrow i}$ is 0, the reward in $s_{\uparrow i}$ is $\frac{4m_i}{\gamma^{2i-1}}$. The action b leads to the state $s_{\times i}$ unconditionally with reward 0, and all actions from $s_{\times i}$ lead to the state q_{i+1}, if $i < n$, or t, if $i = n$, with reward $\frac{m_i}{\gamma^{2i-1}}$.

Finally, we define the expected discounted rewards we would like to get with a pure policy π. It should be $\frac{1}{2} \sum_{i \in [n]} m_i$ in the worst case and $\frac{3}{2} \sum_{i \in [n]} m_i$ in the average case.

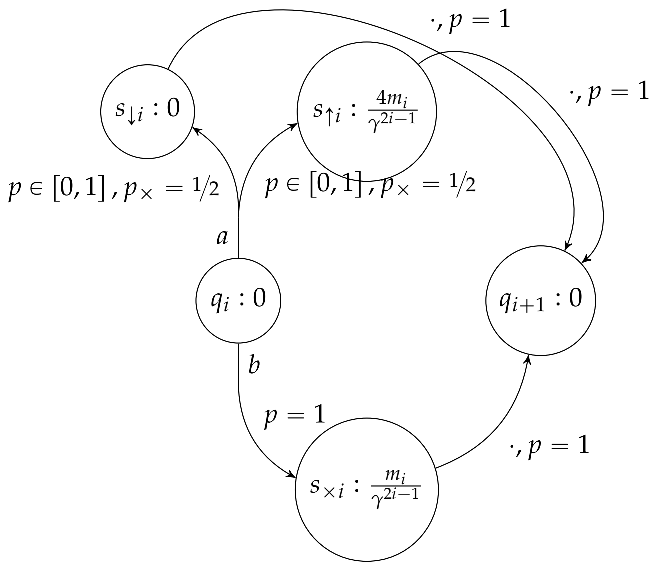

$s_{\downarrow i} : 0$
$s_{\uparrow i} : \frac{4m_i}{\gamma^{2i-1}}$
$q_i : 0$
$q_{i+1} : 0$
$s_{\times i} : \frac{m_i}{\gamma^{2i-1}}$
$\cdot, p = 1$
$\cdot, p = 1$
$p \in [0,1], p_\times = 1/2$
$p \in [0,1], p_\times = 1/2$
a
b
$p = 1$
$\cdot, p = 1$

Figure 2.3: Construction from Theorem 2.4.1

It is easy to see that the construction can be done in polynomial time. We show now its correctness. For every subset $I \subseteq [n]$ we define a policy π_I with $\pi_I(q_i) = a$ if $i \in I$ and $\pi_I(q_i) = b$ if $i \notin I$. The expected discounted total reward for policy π_I in state q_1 will then be

$$\sum_{i \in I} 2\gamma^{2i-1} \cdot \frac{m_i}{\gamma^{2i-1}} + \sum_{i \in [n] \setminus I} \gamma^{2i-1} \cdot \frac{m_i}{\gamma^{2i-1}} = \sum_{i \in [n]} m_i + \sum_{i \in I} m_i$$

in the average case and $\sum_{i\in[n]\setminus I}\gamma^{2i-1}\cdot\frac{m_i}{\gamma^{2i-1}}=\sum_{i\in[n]\setminus I}m_i$ in the pessimistic case. If there is a subset $I\subseteq[n]$ such that $\sum_{i\in I}m_i=\sum_{i\in[n]\setminus I}m_i$, then there is a policy π_I that yields the requested expected discounted rewards. Furthermore, as any pure policy π induces a subset $I\subseteq[n]$ by defining $i\in I\Leftrightarrow\pi(q_i)=a$, the existence of a pure policy π with the requested expected discounted rewards implies the existence of a subset with sum $\frac{1}{2}\sum_{i\in[n]}m_i$.

It is easy to see that the proof technique can also be applied to the combination of optimistic and expected-case measures, and optimistic and pessimistic measures. □

For the general stationary case, we consider a slightly different construction.

Theorem 2.4.2. *The canonical multi-objective decision problem is* NP*-complete for general stationary policies.*

Proof. We perform a similar reduction from the subset sum problem. Now, our SBMDP will have $4n+1$ states

$$S=\{q_i\mid i\in[n]\}\uplus\left\{s_{\downarrow i}\mid i\in[n]\right\}\uplus\left\{s_{\uparrow i}\mid i\in[n]\right\}\uplus\{s_{\times i}\mid i\in[n]\}\uplus\{s\}$$

with the following properties. For $m_i'=m_i(1-\gamma^2)$, the rewards are zero in states $s,s_{\downarrow i},q_i$, and $2m_i'$ in states $s_{\uparrow i}$, and m_i' in states $s_{\times i}$ for $i\in[n]$. Again, we have two actions, a and b, which differ only in states q_i: It is

$$\begin{aligned}
\boldsymbol{P}_{\downarrow}^{\cdot}(s,q_i)=\boldsymbol{P}_{\times}^{\cdot}(s,q_i)=\boldsymbol{P}_{\uparrow}^{\cdot}(s,q_i)&=\frac{1}{n},\\
\boldsymbol{P}_{\downarrow}^{\cdot}(s_{\uparrow i},q_i)=\boldsymbol{P}_{\times}^{\cdot}(s_{\uparrow i},q_i)=\boldsymbol{P}_{\uparrow}^{\cdot}(s_{\uparrow i},q_i)&=1,\\
\boldsymbol{P}_{\downarrow}^{\cdot}(s_{\downarrow i},s_{\downarrow i})=\boldsymbol{P}_{\times}^{\cdot}(s_{\downarrow i},s_{\downarrow i})=\boldsymbol{P}_{\uparrow}^{\cdot}(s_{\downarrow i},s_{\downarrow i})&=1,\\
\boldsymbol{P}_{\downarrow}^{b}(q_i,s_{\times i})=\boldsymbol{P}_{\times}^{b}(q_i,s_{\times i})=\boldsymbol{P}_{\uparrow}^{b}(q_i,s_{\times i})&=1,
\end{aligned}$$

and

$$\begin{aligned}
\boldsymbol{P}_{\downarrow}^{a}(q_i,s_{\downarrow i})=\boldsymbol{P}_{\downarrow}^{a}(q_i,s_{\uparrow i})&=0,\\
\boldsymbol{P}_{\times}^{b}(q_i,s_{\downarrow i})=\boldsymbol{P}_{\times}^{b}(q_i,s_{\uparrow i})&=1/2,\\
\boldsymbol{P}_{\uparrow}^{b}(q_i,s_{\downarrow i})=\boldsymbol{P}_{\uparrow}^{b}(q_i,s_{\uparrow i})&=1,
\end{aligned}$$

as depicted in Fig. 2.4. We now show the properties of the reduction. If the given subset sum instance allows for a subset $I\subseteq[n]$ such that $\sum_{i\in I}m_i=\frac{1}{2}\sum_{i\in[n]}m_i$, then the induced pure policy π_I which selects action a in s_i iff $i\in I$ will yield an expected discounted reward of $\frac{\gamma^2}{n}\left(\sum_{i\in I}2m_i+\sum_{i\in[n]\setminus I}m_i\right)=\frac{3\gamma^2}{2n}\sum_{i\in[n]}m_i$ in the best case and $\frac{\gamma^2}{n}\sum_{i\in[n]\setminus I}m_i=\frac{\gamma^2}{2n}\sum_{i\in[n]}m_i$ in the worst case in state s.

Conversely, for any stationary policy f we have in state q_i a probability p_i to choose action b. Then, for the worst-case expected discounted reward $\vec{v}_{\downarrow}(q_i)$ in this state will hold

$$\begin{aligned}
\vec{v}_{\downarrow}(q_i)&=p_i\gamma(m_i'+\gamma\vec{v}_{\downarrow}(q_i))\Leftrightarrow\\
\vec{v}_{\downarrow}(q_i)&=\frac{p_i\gamma m_i'}{1-p_i\gamma^2}
\end{aligned}$$

which, derived with respect to p_i, yields

$$\frac{\partial\vec{v}_{\downarrow}(q_i)}{\partial p_i}=\frac{\gamma m_i'}{(1-p_i\gamma^2)^2}$$

For the best-case expected discounted reward, we have, analogously,

$$\vec{v}_\uparrow(q_i) = \frac{(2-p_i)\gamma m'_i}{1-\gamma^2},$$
$$\frac{\partial \vec{v}_\uparrow(q_i)}{\partial p_i} = -\frac{\gamma m'_i}{1-\gamma^2}.$$

Considering the sum of the values, we observe that

$$\frac{\partial\left(\vec{v}_\uparrow(q_i)+\vec{v}_\downarrow(q_i)\right)}{\partial p_i} = \gamma m'_i \left(\frac{-1}{1-\gamma^2} + \frac{1}{\left(1-p_i\gamma^2\right)^2}\right).$$

Setting the derivative to zero, we have

$$1-\gamma^2 = \left(1-p_i\gamma^2\right)^2 \Rightarrow$$
$$\sqrt{1-\gamma^2} = 1-p_i\gamma^2 \Leftrightarrow$$
$$\frac{1-\sqrt{1-\gamma^2}}{\gamma^2} = p_i$$

where the first implication follows from $\gamma \in [0,1)\,, p_i \in [0,1]$. Furthermore, substituting $t := 1-\gamma^2$, we have

$$p_i = \frac{1-\sqrt{t}}{1-t} = \frac{1}{1+\sqrt{t}}$$

for $t \in (0,1]$. It is easy to see that this implies $0 < p_i \leqslant 1$, so a possible maximum may lie between zero and one. However, considering the second derivative, we observe that

$$\frac{\partial^2\left(\vec{v}_\uparrow(q_i)+\vec{v}_\downarrow(q_i)\right)}{\partial p_i^2} = \frac{2\gamma^3 m'_i}{\left(1-\gamma^2\right)^3}$$

which is positive and implies a local minimum. This means that the maximal values of $\sum_{i\in[n]} \vec{v}_\uparrow(q_i) + \vec{v}_\downarrow(q_i)$ lie at $p_i \in \{0,1\}$. This means that a policy can achieve

$$\vec{v}_\uparrow(s) \geqslant \frac{\gamma^2}{2n}\sum_{i\in[n]} m_i, \vec{v}_\downarrow(s) \geqslant \frac{3\gamma^2}{2n}\sum_{i\in[n]} m_i$$

only if there exists a subset $I \subseteq [n]$ with $\sum_{i\in I} m_i = {}^1\!/\!_2 \sum_{i\in[n]} m_i$.

$\square$

2.4.3 Stochastic optimization and the multi-scenario problem

In this section, we consider the stochastic optimization problem applied to the multi-objective perspective on CMDPs and SBMDPs. More formally, the problem statement for stochastic bounded-parameter MDPs sounds like this: Given a stochastic BMDP $\left(S=[n], A=[m], P_\updownarrow, \vec{r}, \vec{q}, p\right)$ and a weighting vector $\vec{w} = \left(w_\downarrow, w_\times, w_\uparrow\right)$ for the individual performance metrics, we seek to optimize $\vec{q}\boldsymbol{V}^{(\pi)}\vec{w}$ where $\boldsymbol{V}^{(\pi)} = \left(\vec{v}_\downarrow^{(\pi)}, \vec{v}_\times^{(\pi)}, \vec{v}_\uparrow^{(\pi)}\right) \in \mathbb{R}^{n\times 3}$ is the matrix that is built from the value vectors that result from applying the policy π and computing the optimistic, average, and pessimistic rewards.

We have seen in the proof of Theorem 2.4.2 that the stochastic optimization problem is NP-hard for stationary policies. We formally capture this result.

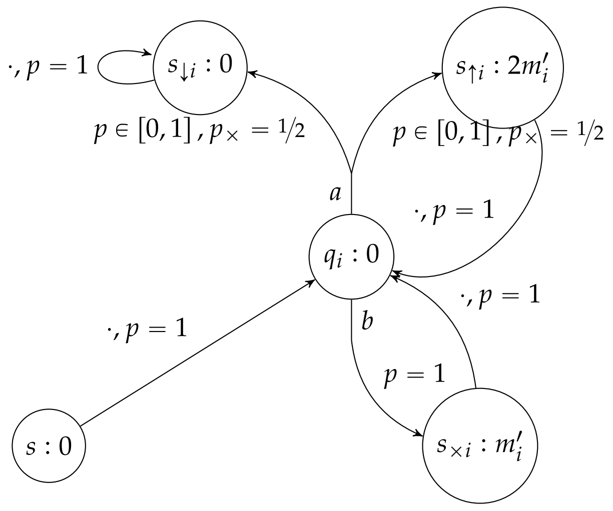


Figure 2.4: Construction from Theorem 2.4.2

Corollary 2.4.3. *The stochastic optimization decision problem for stationary and pure policies in BMDPs is* NP*-complete.*

Proof. We consider the same construction that has been used in the proof of Theorem 2.4.2. From the properties of the problem shown above, the condition $\frac{\vec{v}_\downarrow(s)+\vec{v}_\uparrow(s)}{2} \geqslant \frac{\gamma^2}{n} \sum_{i\in[n]} m_i$ is equivalent to the existence of a subset $I \subseteq [n]$ with $\sum_{i\in I} m_i = \frac{1}{2} \sum_{i\in[n]} m_i$. □

By carefully studying the stochastic optimization problem, it is possible to see that solving it also means solving a related but not exactly equal problem in *multi-scenario optimization*. That is, given a set $\mathcal{M} = \{M_1, \ldots, M_K\}$ of MDPs, one looks for a policy that performs well on all of them, using as performance measure a scalarization metric. However, the stochastic optimization problem for BMDPs seems to be slightly harder than the stochastic multi-scenario optimization for two MDPs. The reason for this assumption is the following: In the stochastic multi-scenario setting, the MDPs for which a shared policy has to be found and optimized are already known. In contrast, in the stochastic BMDP setting, there not only are coupling constraints on actions in different states, but there also is a task to simultaneously compute the MDPs which optimize the value vector for the computed policy in different directions. This means optimization of a two-stage min max (or max min, depending on the formulation) problem, with potential integer or non-linear constraints.

However, even in the seemingly easier multi-scenario case for concurrent Markov decision processes, we can show a hardness result. For it, we formalize the problem.

Definition 2.4.2. Given a concurrent Markov decision process $\mathcal{M} = \{M_1, \ldots, M_K\}$, and a *weight vector* $\vec{w} \in \mathbb{R}^K$, we call the optimization problem

$$\max_f \sum_{k\in[K]} \vec{w}(k)\vec{v}_k^{(f)}$$

the *multi-scenario stochastic optimization problem*. The set of policies may be restricted to general stochastic or pure policies.

The problem turns out to lose the structure of the usual MDP problems, especially, the unimodality property is lost. This means that a locally optimal policy may not be globally optimal, and, furthermore, this also means that the optimal policy may not be pure. This can be seen in the following example.

Let us consider a concurrent MDP $\mathcal{M} = \{M_1, M_2\}$ with $S = [2], A = \{a, b\}$. We define the reward vectors to be $\vec{r}_1 = (0,3)^\top, \vec{r}_2 = (3,9)^\top$. For the transition probability matrices, we set

$$\boldsymbol{P}_1^a = \boldsymbol{P}_2^b = \begin{pmatrix} 1 & 0 \\ 1 & 0 \end{pmatrix} \text{ and } \boldsymbol{P}_1^b = \boldsymbol{P}_2^a = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

where the matrix $\boldsymbol{P}_1^a$ corresponds to the action a in the first MDP and so on. The discount factor is $\gamma = 0.9$ and the weights are $\vec{w} = \left(7/10, 3/10\right), \vec{q} = \left(2/3, 1/3\right)^\top$. We observe that the weighted sum for the policy $\pi_{aa} = (a, a)$ is 26.5, for the policy $\pi_{ab} = (a, b)$ it is 24.8, and for the policy $\pi_{ba} = (b, a)$ it is 25.3. This means that π_{aa} is locally optimal. However, the policy $\pi_{bb} = (b, b)$ yields a weighted sum of rewards of 29.2, which is the global maximum for pure policies.

Having seen that the stochastic optimization problem for concurrent MDPs does not have the unimodality property, we show its theoretical hardness.

Definition 2.4.3 (Multi-scenario stochastic optimization decision problem)**.** Given a concurrent MDP $\mathcal{M}$ and real (represented with $\mathcal{O}(NMK)$ bits) vectors and numbers $\vec{w} \in \mathbb{R}^K, g \in \mathbb{R}$, decide if there is a stationary policy $f \in \mathcal{P}$ such that $\sum_{k=1}^K \vec{w}(k) \left(\sum_{s \in S} \vec{q}(s) \vec{v}_k^{(f)}(s) \right) \geqslant g$.

The first part of our NP-completeness proof is to show that the given problem is, in fact, in NP. One can see that the representation of a stationary policy is polynomial as long as the representation of a real number is assumed to consume $\mathcal{O}(NMK)$ bits. Then one can define a non-deterministic algorithm that guesses a policy by guessing $\mathcal{O}(NMK \times NM)$ bits that represent NM real numbers which define a stationary policy f and then verifies that f fulfills the relation above.

Now we can prove the more interesting part of the completeness statement.

Theorem 2.4.4. *The decision problem defined in Def. 2.4.3 is* NP*-complete.*

Proof. We perform a reduction from 3-SAT. Given a 3-SAT instance with n variables $x_1, \ldots, x_n$ and m clauses $C_1, \ldots, C_m$ where each clause contains three literals, we construct a concurrent MDP $\mathcal{M} = \{M_1, M_2\}$ consisting of two MDPs, a vector $\vec{w} \in \mathbb{R}^2$ and a real number $g \in \mathbb{R}$ such that the instance will be satisfiable if and only if there is a stationary policy f for concurrent MDP that yields the value g.

The first part of our construction are the states. They are arranged in three groups.

- First, we create specially designated sink states s_0, s_1 that yield 0 resp. 1 reward units in both MDPs.
- Then, we transfer the variables of the Boolean satisfiability problem into states of the MDPs: for each variable x we create two states s_x, s_x' in both MDPs. The reward is 0 in states s_x and 1 in states s_x'.
- Last, we create states for clauses: for each clause C, a state s_C is created. Again, the reward in s_C is zero for all clauses.

The second part of the construction are the actions. We create three actions $\mathcal{A} = \{1, 2, 3\}$ with the following semantics.

- In the sink state s_0, it is $\boldsymbol{P}_k^a(s_0, s_0) = 1$ for all $a \in \mathcal{A}$ and $M_k \in \mathcal{M}$.

- In the variable states, we define

$$P_1^1(s_x, s'_x) = P_1^2(s_x, s_0) = P_1^3(s_x, s_0) = 1$$

and

$$P_2^1(s_x, s_0) = P_2^2(s_x, s'_x) = P_2^3(s_x, s'_x) = 1,$$

that is, we define actions in s_x to lead to different outcomes in the MDPs. The motivation is to force a mutually exclusive choice of values for the Boolean variables in the concurrent MDP. In the auxiliary variable states, it is $P_k^a(s'_x, s_x) = 1$ for all actions $a \in \mathcal{A}$ and MDPs $k \in \mathcal{M}$; the idea behind these states is to exploit non-linearity of the problem. The construction is visualized in Fig. 2.5 where the upper part corresponds to the first MDP M_1 and the lower part corresponds to the second MDP M_2 in $\mathcal{M}$.

- In the clause states, we define actions as follows. In a clause $C = L_1 \vee L_2 \vee L_3$, the chosen action represents the literal that evaluates to true. Hence, we define $P_k^a(C, s)$ by setting $P_k^a(C, s) = 1$ in the cases
  - $L_a = x, k = 1, s = s_x$
  - $L_a = \neg x, k = 1, s = s_0$
  - $L_a = \neg x, k = 2, s = s_x$
  - $L_a = x, k = 2, s = s_0$

A graphical sketch of this setup can be seen in Fig. 2.6. Again, the upper part of the drawing corresponds to the first MDP in $\mathcal{M}$ while the lower part corresponds to the second MDP.

The idea behind this construction is to infer functions $\beta\colon \{1, \ldots, n\} \to \{0, 1\}$ that map variables to values and $\nu\colon \{1, \ldots, m\} \to \{1, 2, 3\}$ that map the clauses to the satisfying variables. This is done to create a mapping from policies to variable assignments in the SAT problem. Furthermore, we define the initial distribution $\vec{q}$ with $\vec{q}(s_C) = 1/m$ for all clauses C for both MDPs and weights $\vec{w} = (1/2, 1/2)$. Concerning the value, we set an auxiliary constant $\alpha := \frac{1}{1-\gamma^2}$ and the required value $g := \frac{\gamma^2\alpha}{2}$ where $\gamma \in (0, 1)$ is a non-zero discount factor in the concurrent MDP.

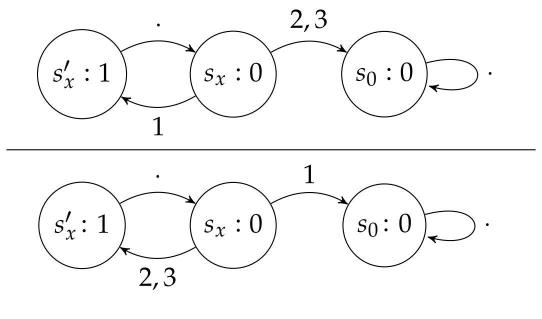


Figure 2.5: The variable gadget

We prove the validity of the reduction. First, we show that if there is an assignment $\beta\colon \{1, \ldots, n\} \to \{0, 1\}$ that satisfies the SAT instance, then there also exists a pure policy $\pi\colon S \to A$ such that $\sum_{k=1}^K \vec{w}(k) \sum_{s \in S} \vec{q}(s) \vec{v}_k^{(f)}(s) \geqslant g$. We construct the policy in two steps. In the first step, we set $\pi(s_x) = 1 \Leftrightarrow \beta(x) = 1$ for all variables x. In the second step, it follows from the existence of a satisfying assignment that in each clause, a literal is satisfied, defining a function $\nu\colon \{1, \ldots, m\} \to \{1, 2, 3\}$ that defines the number of a satisfied literal in every clause. Thus, we set $\pi(s_C) = a \Leftrightarrow \nu(C) = a$.

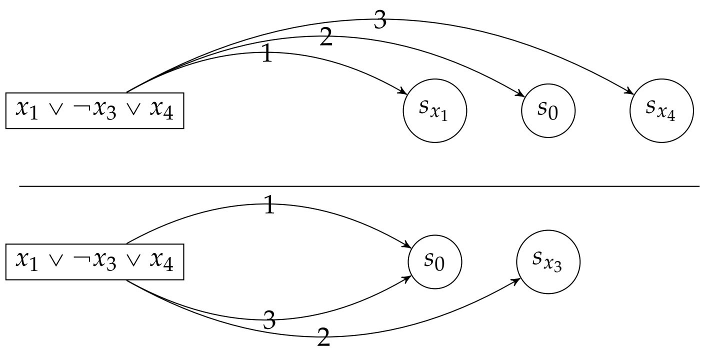


Figure 2.6: The clause gadget

We verify that the constructed policy yields the given value. As in each clause the satisfying literal is chosen, the value of this state will be 0 in one MDP and $\gamma^2\alpha$ in the other one.

Now we show that if there is no satisfying assignment, then the value of the concurrent MDP will be lower than g. Given any assignment β and any assignment ν, the induced policy will lead from at least one clause state to the sink state s_0 with nonzero probability in both MDPs, yielding a lower value. However, we must take care of stationary but not pure policies that still might induce the desired value. One can observe that if the stationary policy is not pure in a state s_x for a variable x, then the cumulative discounted reward in this state is $\frac{p\gamma}{1-p^2\gamma^2}$ for some real $0 < p < 1$. Deriving the value of a clause state from which one can arrive to this variable state, we get a summand

$$\frac{1}{2}\left(\frac{p\gamma^2}{1-p^2\gamma^2} + \frac{(1-p)\gamma^2}{1-(1-p)^2\gamma^2}\right)$$

Let $f(p) = \frac{p}{1-p^2\gamma^2} + \frac{1-p}{1-(1-p)^2\gamma^2}$. Computing the derivative, we obtain

$$f'(p) = \frac{2\gamma^2 p}{\left(1-\gamma^2p^2\right)^2} - \frac{2\gamma^2(1-p)^2}{\left(1-\gamma^2(1-p)^2\right)^2} + \frac{1}{1-\gamma^2p^2} - \frac{1}{1-\gamma^2(1-p)^2}$$

which has its roots at

$$\frac{1}{2}, \frac{1 \pm \sqrt{1-4\gamma^{-2}+4\gamma^{-2}\sqrt{4-\gamma^2}}}{2}, \frac{1 \pm i\sqrt{1-4\gamma^{-2}+4\gamma^{-2}\sqrt{4-\gamma^2}}}{2}.$$

The only roots of interest are the real ones, and thus, we investigate the pair

$$\frac{1 \pm \sqrt{1-4\gamma^{-2}+4\gamma^{-2}\sqrt{4-\gamma^2}}}{2}. \tag{2.11}$$

It can be seen that for $0 < \gamma < 1$, the value $\sqrt{4-\gamma^2}$ is at least $\sqrt{3} > 1$, and the root term in (2.11) is thus greater than one. This means that the whole term (2.11) is either greater than one or negative. Hence, the possible extreme points of f in $[0,1]$ may lie at 0, 1, or $1/2$. We can see that $f(0) = f(1) = q$ while $f(1/2) = \frac{1}{1-1/4\gamma^2} < q$. Hence, a non-pure policy in a variable state will have a lower cumulative discounted reward. Concerning the clause states, we observe that a non-pure policy cannot yield higher rewards than a pure one, as the expected discounted reward in a clause state is linear in the expected discounted rewards in the following variable states; the clause states are not visited again. □

2.4.4 Finding pure Pareto-optimal policies

We conclude the discussion on multi-objective perspectives on BMDPs by deriving an algorithm that computes the set $\mathcal{P}_{\text{Pareto}}$ exactly. The algorithm is based on policy iteration and works as follows: For each currently computed policy, neighbour policies with a Hamming distance of 1 are computed, and those policies that are not worse than the policy they are derived from are kept. This continues for $|S|$ steps, and the resulting non-dominated policies are the Pareto frontier.

The correctness of this approach is shown by proving the following lemma on the structure of $\mathcal{P}_{\text{Pareto}}$. Intuitively, we show that for each policy, there exists a "path" of policies to any policy on the Pareto frontier on which any policy is not worse than its predecessor in all components, i. e., there always is at least a partial improvement. The proof is a consequence of general MDP and stochastic game theory.

Lemma 2.4.5. *Let $\mathcal{P} = (S, A, T_{\updownarrow}, \vec{r}, \vec{q}, p)$ be a SBMDP. Let furthermore π, π' be two pure policies where π' lies on the Pareto frontier. Then there exists a finite sequence of pure policies $\pi = \pi_0, \pi_1, \ldots, \pi_N = \pi'$ where $d(\pi_i, \pi_{i+1}) = 1, \vec{v}^{(\pi_i)} \not> \vec{v}^{(\pi_{i+1})}$ and, additionally, $N \leqslant |S|$.*

Proof. We provide a proof by induction on $d(\pi, \pi')$. For $d(\pi, \pi') \in \{0, 1\}$, the statement holds obviously.

For $d(\pi, \pi') = c > 1$, the induction hypothesis is that the statement holds for $c - 1$. This means that for each policy π_1 with distance $d(\pi_1, \pi') = c - 1$ there exists a sequence of policies $\pi_1, \pi_2, \ldots, \pi_c = \pi'$ such that for any two adjacent policies π_i, π_{i+1} it is $\vec{v}^{(\pi_i)} \not> \vec{v}^{(\pi_{i+1})}$.

To show the induction step, we must infer the statement for $d(\pi, \pi') = c$. Suppose now for the sake of contradiction that it is not the case. We observe that under this assumption, for each state $s \in S$, the policy $\pi^{(s,\pi'(s))}$ that results from changing π in state s to choose action $\pi'(s)$ results in a value vector that is dominated by $\vec{v}^{(\pi)}$, i. e., $\vec{v}^{(\pi)} > \vec{v}^{(\pi^{(s,\pi'(s))})}$. Let us now consider a restricted SBMDP $\mathcal{P}^{[\pi,\pi']} = (S, A^{[\pi,\pi']}, T_{\updownarrow}^{[\pi,\pi']}, \vec{r}, \vec{q}, p^{[\pi,\pi']})$ where the available actions are only those used in either π or π', that is, $A^{[\pi,\pi']} = \{a, b\}$ and the matrices $\boldsymbol{P}$ in $T_{\updownarrow}^{[\pi,\pi']}$ are constructed with

$$\boldsymbol{P}_{\downarrow}^{[\pi,\pi']a}(s\bullet) = \boldsymbol{P}_{\downarrow}^{\pi(s)}(s\bullet), \boldsymbol{P}_{\uparrow}^{[\pi,\pi']a}(s\bullet) = \boldsymbol{P}_{\downarrow}^{\pi(s)}(s\bullet),$$
$$\boldsymbol{P}_{\downarrow}^{[\pi,\pi']b}(s\bullet) = \boldsymbol{P}_{\downarrow}^{\pi'(s)}(s\bullet), \boldsymbol{P}_{\uparrow}^{[\pi,\pi']b}(s\bullet) = \boldsymbol{P}_{\downarrow}^{\pi'(s)}(s\bullet)$$

The reward vector is kept.

It is easy to see that the policies π and π' can be executed in the new SBMDP $\mathcal{P}^{[\pi,\pi']}$. As all action changes from π lead to smaller value vectors in each component, we can see that π is locally optimal for each component, and thus, π is optimal for all components. Hence, π is an optimal policy in $\mathcal{P}^{[\pi,\pi']}$. Furthermore, π' is then dominated by π in all states and all components in $\mathcal{P}^{[\pi,\pi']}$ as well as in $\mathcal{P}$. Consequently, π' cannot lie on the Pareto frontier, which contradicts the initial assumption.

As we have arrived at a contradiction, we it follows that there must exist a state s where it is $\vec{v}^{(\pi)} \not> \vec{v}^{(\pi^{(s,\pi'(s))})}$, and, since $d(\pi^{(s,\pi'(s))}, \pi') = c - 1$ and $d(\cdot, \cdot)$ can never exceed $|S|$, there exists, by induction hypothesis, a sequence of policies $\pi^{(s,\pi'(s))} = \pi_1, \pi_2, \ldots, \pi_c = \pi'$ for which it is $\vec{v}^{(\pi_i)} \not> \vec{v}^{(\pi_{i+1})}$. As $d(\pi, \pi^{(s,\pi'(s))}) = 1$, this concludes the proof. □

This result directly leads to Algorithm 11 which is, as already said, based on policy iteration and (partially) breadth-first search.

It remains now to prove correctness of Algorithm 11.

Algorithm 11 Exact computation of $\mathcal{P}_{\text{Pareto}}$ and $\mathcal{V}_{\text{Pareto}}$

```
1: function PURE-OPT-EXACT($\mathcal{P} = (S, A, T_{\updownarrow}, \vec{r}, p), \gamma$)
2:     $P_0 \leftarrow \{\text{arbitrary policy}\}$                          ▷ initialize current policy set
3:     $F \leftarrow P_0$                                                    ▷ initialize the Pareto frontier
4:     for $i \in [|S|]$ do
5:         $P_i \leftarrow \cup_{\pi \in P_{i-1}} \left\{\pi' \mid d(\pi, \pi') = 1, \vec{v}^{\pi} \not> \vec{v}^{\pi'}\right\}$
6:         $F \leftarrow PO(F \cup P_i)$
7:     return $F$
```

Theorem 2.4.6. *Algorithm 11 correctly computes $\mathcal{P}_{\text{Pareto}}$.*

Proof. The correctness of this algorithm is guaranteed by Lemma 2.4.5. In detail, Algorithm 11 stores a set P of policies. In the i-th step, the set P is updated with policies that have distance 1 from already computed policies in P and distance i from π_0; a further constraint restricts the policies to be non-dominated by their "parent" in P. This way, after i steps P contains all policies with distance i from π_0 that follow a non-dominated path. By computing the non-dominated subset of currently found policies in line 6, we maintain a set of mutually non-dominated policies that are reachable on a non-dominated path from π_0. By Lemma 2.4.5, this captures all policies from $\mathcal{P}_{\text{Pareto}}$. □

2.5 Extending the model

In this section, we consider possible extensions of the bounded-parameter model, including different reward measures. Motivated by the assumption that the dimension of the uncertainty set is less than the number of states, we propose a formalism that introduces a *parameter space* with dimension $k < |S|$ along with a mapping from the parameter space to an uncertainty set $\mathcal{M}$ of MDPs and consider several reward measures on it.

Specifically, we assume that an uncertainty set of Markov decision processes $\mathcal{M}$ is extended by a probability distribution in form of a probability density function

$$p\colon \mathcal{M} \to \mathbb{R}, \int_{\mathcal{M}} p(M)\, \mathrm{d}M = 1$$

and, similar to our approach in Sec. 2.4, we search for a trade-off policy f. However, now f should optimize the probability of a given value vector $\vec{v}$ with respect to p. More generally, the problem can be stated as follows [DM10].

Definition 2.5.1 (Percentile optimization)**.** Let $\mathcal{M}$ be a set of MDPs with n states and a shared action space, p a probability density function on $\mathcal{M}$, $\vec{v} \in \mathbb{R}^n$ and f a policy. We call $(\mathcal{M}, p)$ an *stochastic MDP*.

The *worst-case set* $\dot{M}_{\vec{v},f} \subseteq \mathcal{M}$ is the set of all Markov decisison processes $M \in \mathcal{M}$ for which the value vector $\vec{v}^{(f)}(M)$ in M under f fulfills $\vec{v}^{(f)}(M) \geqslant \vec{v}$.

For a given probability $q \in [0,1]$ and a given value vector $\vec{v} \in \mathbb{R}^n$, the *percentile optimization problem* is the problem of deciding if a worst-case set $\dot{M}_{\vec{v},f}$ with $\int_{\dot{M}_{\vec{v},f}} p(M)\, \mathrm{d}M \geqslant q$ exists.

It is easy to see that for arbitrary probability density functions p this problem can get arbitrarily hard, and a hardness result for such an unrestricted problem may not be surprising. Thus, we restrict the problem to a more handy subclass we have hinted at in the beginning of this section.

Definition 2.5.2 (Parameterized MDP)**.** For a state set $S = [n]$, an action set $A = [m]$, and a reward vector $\vec{r}$, a *parameterized Markov decision process* is a set $\tilde{\mathcal{M}}$ of Markov decision processes with common state space S, action space A, reward vector $\vec{r}$, a probability distribution $p\colon [0,1]^k \to \mathbb{R}$, and a surjective mapping $F\colon [0,1]^k \to \tilde{\mathcal{M}}$ with the following properties.

- p is a uniform distribution on $[0,1]^k$.
- The mapping F maps values $\lambda_1, \dots, \lambda_k \in [0,1]^k$ to transition probability matrices affinely, that is, F has the form $F\colon [0,1]^k \times A \to \mathbb{R}^{n\times n}_{\geqslant 0}$ and can be represented by $(k+1)|A|$ real matrices
$$\left\{ \left(\boldsymbol{P}_0^a, \Delta_1^a, \dots, \Delta_k^a\right) \mid a \in A \right\}$$
where the *base matrices* $\boldsymbol{P}_0^a$ are stochastic matrices for all $a \in A$ and for the *influence matrices* Δ_i^a the constraints
$$\Delta_i^a \vec{1} = \vec{0} \qquad \forall a \in A, i \in [k] \tag{2.12}$$
$$\min_{\lambda_1,\dots,\lambda_k \in [0,1]^k} \left(\boldsymbol{P}_0^a(s,s') + \sum_{i=0}^{k} \lambda_i \Delta_i^a(s,s') \right) \geqslant 0 \qquad \forall a \in A, s \in S, s' \in S \tag{2.13}$$
hold. Then F can be represented by
$$F(\lambda_1, \lambda_2, \dots, \lambda_k, a) = \boldsymbol{P}_0^a + \sum_{i=1}^{k} \lambda_i \Delta_i^a$$

The arguments of F in this setting are called *parameters*.

Analogously, one can define parameterized Markov reward processes.

The constraint (2.13) requires that any affine combination of the base matrix and the influence matrices is a non-negative matrix, the constraint (2.12) implies that each such affine combination will have row sum 1. This constaint implies that the influence and base matrices are dependent on each other; by doing this, this constaint ensures that for any choice of parameters $\lambda_1, \dots, \lambda_k \in [0,1]^k$, the resulting transition probability matrix $F(\lambda_1, \dots, \lambda_k, a)$ will be stochastic. The constraint (2.13) is equivalent to
$$\boldsymbol{P}_0^a(s,s') + \sum_{i=0}^{k} \min(0, \Delta_i^a(s,s')) \geqslant 0$$
for all $s, s' \in S$, which makes it easy to verify.

To provide an intuition for parameterized MDPs, we provide an example first. Consider a two-state two-action model with $S = [3], A = [2]$, reward vector $\vec{r} = (1,3,2)^\top$ and the following transition probability matrices, written as functions of parameters $\lambda_1, \lambda_2 \in [0,1]$.

$$F(\lambda_1,\lambda_2,1) = \boldsymbol{P}^1(\lambda_1,\lambda_2) = \begin{pmatrix} \lambda_1 & 1-\lambda_1 & 0 \\ 1/2 + \lambda_1/2 & 0 & 1/2 - \lambda_1/2 \\ 1/2 + \lambda_1/6 - \lambda_2/4 & 1/4 - \lambda_1/6 & 1/4 + \lambda_2/4 \end{pmatrix},$$
$$F(\lambda_1,\lambda_2,2) = \boldsymbol{P}^2(\lambda_1,\lambda_2) = \begin{pmatrix} 1/2 + \lambda_1/2 & 1/2 - \lambda_1/2 & 0 \\ \lambda_2 & 1-\lambda_2 & 0 \\ 1-\lambda_1 & \lambda_1/2 & \lambda_1/2 \end{pmatrix}$$

In the representation defined above, it is

$$\boldsymbol{P}_0^1 = \begin{pmatrix} 0 & 1 & 0 \\ 1/2 & 0 & 1/2 \\ 1/2 & 1/4 & 1/4 \end{pmatrix}, \Delta_1^1 = \begin{pmatrix} 1 & -1 & 0 \\ 1/2 & 0 & -1/2 \\ 1/6 & -1/6 & 0 \end{pmatrix}, \Delta_2^1 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ -1/4 & 0 & 1/4 \end{pmatrix},$$
$$\boldsymbol{P}_0^2 = \begin{pmatrix} 1/2 & 1/2 & 0 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \Delta_1^2 = \begin{pmatrix} 1/2 & -1/2 & 0 \\ 0 & 0 & 0 \\ -1 & 1/2 & 1/2 \end{pmatrix}, \Delta_2^2 = \begin{pmatrix} 0 & 0 & 0 \\ 1 & -1 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

What we can see in this example is that one parameter can steer the behaviour of the model in arbitrarily many states and actions. Generalizing this observation, we can say that the PMDP formalism reduces the number of dimensions in the uncertainty model, however, it adds complexity in the form of additional constraints. The intuition is that a parameter λ_i can influence several transitions in different states; that is, we cannot rely on dynamic programming, as the assumption of independence of different states and actions, the *rectangularity property* [Iye05], does not hold. This means that a local maximization or minimization of a parameter does not necessarily imply a global maximization or minimization of all components in the value vector. This perspective is somewhat similar to the reasoning behind the hardness result in Theorem 2.2.1. Here, we show that even for a mathematically simpler performance measure, the expected discounted total reward, computing bounds and performing percentile optimization is computationally hard. The presented hardness results are worst case results, which means that the PMDP model allows for hard instances to be constructed. In the cases where the rectangularity property holds, it is still possible to apply suitable efficient algorithms; however, the generality of the model also allows one to construct instances which are as computationally complex as instances for NP-hard problems.

2.5.1 Hardness of robust optimization

First, we show that in the most general case, computation of lower and upper bounds is NP-complete.

Theorem 2.5.1. *For a given parameterized Markov reward process and a vector $\vec{v}$, deciding if the maximal expected total discounted reward is greater than $\vec{v}$ is* NP*-complete.*

Proof. We describe a reduction from 3−SAT. Given a 3−SAT instance $\phi = C_1 \wedge \cdots \wedge C_m$ with m clauses $C_i, i \in [m]$ and variables $x_1, \ldots, x_n$, we construct a PMRP with $m + 5n + 2$ states $\{s_1, \ldots, s_m, s_+, s_-\} \cup \left\{q_j^i \mid j \in [5], i \in [n]\right\}$, an arbitrary discount factor $\gamma \in (0, 1)$, and n parameters $\lambda_1, \ldots, \lambda_n$. The states s_+ and s_- are absorbing; the only nonzero reward is generated in state s_+ and is defined to be 1. For the transition probabilities, we set $\Pr\left[s_- \mid s_i, C_i = L_1 \vee L_2 \vee L_3\right] = p_1 + p_2 + p_3$ where

$$p_j = \begin{cases} \frac{1}{3} - \frac{1}{3}\lambda_t & L_j = \neg x_t \\ \frac{1}{3}\lambda_t & L_j = x_t \end{cases}$$

and $\Pr\left[s_+ \mid s_i, C_i = L_1 \vee L_2 \vee L_3\right] = q_1 + q_2 + q_3$ where

$$q_j = \begin{cases} \frac{1}{3}\lambda_t & L_j = \neg x_t \\ \frac{1}{3} - \frac{1}{3}\lambda_t & L_j = x_t, \end{cases}$$

for $j \in \{1, 2, 3\}$; an illustration can be observed in Fig. 2.7. It is easy to see that if $\lambda_i \in [0, 1]$ for all $i \in [n]$, the constraints $0 \leqslant p_1 + p_2 + p_3 \leqslant 1$ and $0 \leqslant q_1 + q_2 + q_3 \leqslant 1$ hold, satisfying (2.13).

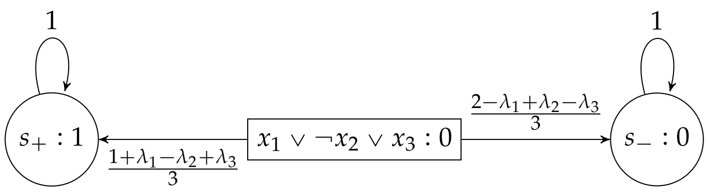

1
1
$s_+ : 1$
$x_1 \vee \neg x_2 \vee x_3 : 0$
$s_- : 0$
$\frac{1+\lambda_1-\lambda_2+\lambda_3}{3}$
$\frac{2-\lambda_1+\lambda_2-\lambda_3}{3}$

Figure 2.7: First part of the reduction in Theorem 2.5.1

One can immediately observe that if there exists a satisfying assignment for ϕ, then there also exists an assignment for $\lambda_1, \dots, \lambda_k$ such that $\lambda_i \in \{0, 1\}$ and the expected total discounted reward $\vec{v}^*$ is at least $\frac{1}{3} \cdot \frac{\gamma}{1-\gamma}$ in states $s_i, i \in [n]$. Conversely, if ϕ is unsatisfiable, then there exists no assignment for $\lambda_1, \dots, \lambda_k$ such that $\lambda_i \in \{0, 1\}$ and $\vec{v}^* \geqslant \vec{v}$. However, this disregards the possibility for the parameters to be fractional, and one can immediately see that setting $\lambda_1 = \lambda_2 = \dots = \lambda_n = \frac{1}{2}$ delivers a total expected discounted reward of $\frac{1}{2} \cdot \frac{1}{1-\gamma} \cdot (\gamma, \dots, \gamma, 1, 0)^\top$. To disallow this, we add, for each $i \in [n]$, a gadget as depicted in Figure 2.8. The only nonzero reward in this gadget is in state q_5^i, which is reached with probability $1 - \lambda_i + \lambda_i^2 = 1 - \lambda_i (1 - \lambda_i)$ which is maximal if λ_i is either 1 or 0; by requiring the expected discounted total reward in state q_1^i to be exactly $\gamma^2 \cdot \frac{1}{1-\gamma}$ we can effectively restrict λ_i to $\{0, 1\}$, which completes the argument.

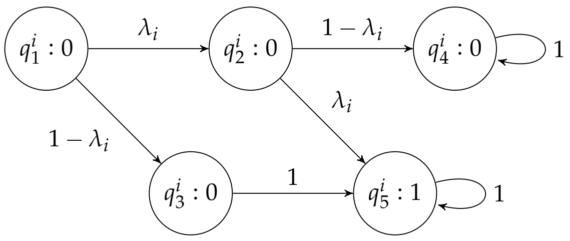


Figure 2.8: A gadget forcing $\lambda_i \in \{0, 1\}$

Completeness follows analogously from the arguments in Theorem 2.2.1. □

2.5.2 Hardness of percentile optimization

We observe that optimizing the expected discounted rewards for parameterized Markov decision processes can be considered a subclass of the polynomial optimization problem which is also NP-hard [MK87]. This yields an intuitive argument that the percentile optimization problem is at least as hard. Formally, we show via a reduction that deciding if the worst-case set is not empty is NP-hard.

Theorem 2.5.2. *The optimization of the probability of reaching a certain minimal expected discounted total reward in a parameterized Markov decision process is* NP*-hard, even if the transition probabilities for each action depend on at most one parameter.*

Proof. We show a reduction from 3−SAT. Given a 3−SAT instance ϕ consisting of k variables and n clauses $\mathcal{C} = \{C_1, \dots, C_n\}$ such that the clause C_i contains literals L_1^i, L_2^i, L_3^i, we construct a PMDP with $n + 2$ states $S = \{u, d\} \uplus \mathcal{C}$ and k parameters $\lambda_1, \dots, \lambda_k$. In the following, we use the same symbols to describe both states and clauses; the meaning will be in each case given by the context. The states u and d are both absorbing with exactly one action; the reward in u is 1 and the reward in d is 0. In any other state $C_i \in \mathcal{C}$, there are three available actions that correspond to the three literals $L_1^i, L_2^i, L_3^i \in C_i$; the reward in C_i is 1. As before, we overload the meaning of symbols and, dependent on context, describe actions or literals with the same symbols. Additionally, we assume that the instance contains clauses of the form $x_i \vee \neg x_i$ for each variable x_i.

The transition probabilities are as follows: For state C_i and action L^i_j, the probability of the next state being u is either

Case 1 λ_r, if L^i_j is a positive literal that corresponds to the variable x_r or

Case 2 $1 - \lambda_r$, if L^i_j is a negative literal that corresponds to the variable x_r.

The probability of the next state being d is set to the respective inverse, either

Case 1 $1 - \lambda_r$, if L^i_j is a positive literal that corresponds to the variable x_r or

Case 2 λ_r, if L^i_j is a negative literal that corresponds to the variable x_r.

Before we complete the reduction, we observe: The expected discounted total reward will be maximized in all states if the policy f leads to u with a high probability; however, to do this, f must imply a consistent variable assignment for the underlying 3−SAT instance.

We set the discount factor to $\gamma = 1/2$ and a minimal discounted total reward vector $\vec{u}$ to be 2 in state u, 0 in state d and $5/3$ in all other states. Now we show that the probability of gaining a reward of at least $\vec{u}$ is positive if and only if the instance for 3−SAT is satisfiable.

"$\Leftarrow$" Suppose the given instance for 3−SAT is satisfiable. Then, there is a satisfying assignment $\alpha\colon \{1,\dots,k\} \to \{0,1\}$. In every clause $C_i \in \mathcal{C}$ we can then choose a literal L^i_j such that $\alpha \models L^i_j$ and thus, we can choose the corresponding action L^i_j in the corresponding state C_j; this way, the probability over the parameters $\lambda_1,\dots,\lambda_k$ for the transition probability from C_i to u to be at least $2/3$ is $1/3$, and thus, there exists a parameter set with nonzero measure where the expected discounted total reward of $1 + 2/3 \cdot \gamma \cdot 2 = 5/3$ is reached.

"$\Rightarrow$" Suppose the given instance for 3−SAT is unsatisfiable. Then, no deterministic policy f can induce a consistent assignment and there will be two states $C_i, C_{i'}$ with actions corresponding to literals x_j in C_i and $\neg x_j$ in $C_{i'}$. It is easy to see that for every value of λ_j, either the probability to transition from C_i to u is less than $\frac{2}{3}$ or the probability to transition from $C_{i'}$ to u is less than $\frac{2}{3}$. Then, the worst-case set where a reward of at least $\vec{u}$ is gained is empty.

For randomized policies, we compute the measure of the induced worst-case sets and show that it will be smaller than the worst-case set in the case if a satisfying assignment exists. Suppose that for variable x_i, the probability that the action corresponding to $\neg x_i$ is chosen is $0 < p < 1$. Then to maximise the worst-case set measure, the corresponding action in the clause $x_i \vee \neg x_i$ has to be chosen also with probability p; then, the projection of the worst-case set on λ_i is given by $\Lambda_i = \{\lambda \mid (1-p)\lambda + p(1-\lambda) \geqslant 2/3\} \cap [0,1]$. The inequalities can be transformed into $\lambda(1-2p) \geqslant 2/3 - p$. For $p < 1/2$, we derive $1 \geqslant \lambda \geqslant \frac{2/3-p}{1-2p} = \frac{1}{2} + \frac{1}{6} \cdot \frac{1}{1-2p}$ and the measure $\mu^-(p) = \min(0, \frac{1}{2} - \frac{1}{6} \cdot \frac{1}{1-2p})$. For $p > 1/2$, it is analogously $\mu^+(p) = \min(0, \frac{1}{2} + \frac{1}{6} \cdot \frac{1}{1-2p})$. The derivatives (in the differentiable part) are zero or

$$\frac{d\mu^-}{dp} = -\frac{1}{6} \cdot \left(-\frac{-2}{(1-2p)^2}\right) = -\frac{1}{3} \cdot \frac{1}{(1-2p)^2} < 0, \frac{d\mu^+}{dp} = \frac{1}{3}\frac{1}{(1-2p)^2} > 0,$$

yielding maximal values for $\mu^\pm$ at $p \in \{0,1\}$. This implies that the maximal possible measure of the worst-case set of $\frac{1}{3^k}$ is not reachable if the given instance is unsatisfiable.

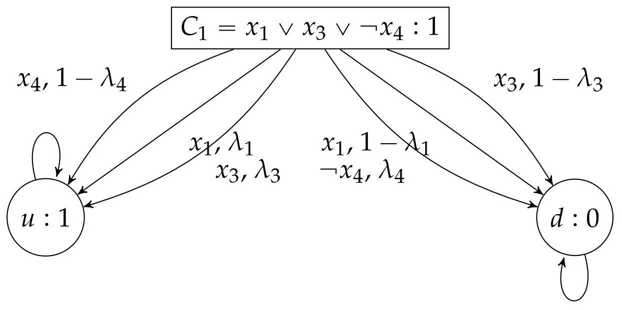


Figure 2.9: (Incomplete) construction for Theorem 2.5.2

□

It is easy to see that this result relies on the possibility to cut the k-dimensional parameter space with hyperplanes in an exponential number of regions. For a fixed number of parameters this idea cannot work, at least not in its unmodified form, because, if the number of dimensions is fixed, the space can be cut only in a polynomial number of segments (more precisely, n planes can dissect $\mathbb{R}^k$ in $\mathcal{O}\left(n^k\right)$ regions). Now, we show that even for a constant number of dimensions (at least 2) yet their influence is unconstrained, the percentile optimization problem is NP-hard.

Theorem 2.5.3. *Deciding if reaching a certain minimal expected discounted total reward is possible with a given probability p in a parameterized Markov decision process is* NP*-hard in the general case, if there are at least two parameters.*

Proof. We again construct a reduction to 3−SAT. As before, we assume a 3−SAT instance ϕ consisting of k variables $\mathcal{X} = \{x_1, \ldots, x_r\}$ and n clauses $\mathcal{C} = \{C_1, \ldots, C_s\}$ such that the clause C_i contains literals L^i_1, L^i_2, L^i_3. Without limitation of generality we assume that ϕ contains, for each variable x_i, a clause $x_i \vee \neg x_i \vee x_i$; this is a technical requirement that will be useful in the proof of the reduction. We generate a PMDP with two parameters λ_1, λ_2 and $2k + n + 2$ states, n of which correspond to the clauses in ϕ.

The general idea is, again, to use the actions to encode both the satisfying assignment and its verification. Similarly to the proof of Theorem 2.5.2, we construct a PMDP with a value vector; the constraints for the parameters that follow from the construction will describe a polygon of known size such that by choosing a value of a variable x_i to be either 0 or 1, one would simultaneously choose one of two adjacent boundaries of the polygon, as shown in Figure 2.10. In a simple case of a $4k$-gon, choosing 0 or 1 for x_i will yield one of the two hyperplanes $\lambda_1 \cos\alpha + \lambda_2 \sin\alpha \leqslant \frac{1}{4} + \frac{1}{2}(\cos\alpha + \sin\alpha)$ or $\lambda_1 \cos\beta + \lambda_2 \sin\beta \leqslant \frac{1}{4} + \frac{1}{2}(\cos\beta + \sin\beta)$ with $\alpha = \frac{2\cdot(4i+1)\pi}{4k}$ and $\beta = \frac{2\cdot(4i+2)\pi}{4k}$, respectively. Other boundaries will be forced by additional states with exactly one action. This way, an assignment will translate to k boundaries of a regular $4k$-gon, while the other $2k$ boundaries will be enforced. A policy will describe, as in Theorem 2.5.2, a (possibly contradictory) assignment that satisfies all clauses.

It is easy to see that, if the assignment is non-contradictory, the area of the polygon will equal to the area of the regular $4k$-gon with inner radius $1/4$ plus k times the area of the gray triangle in Figure 2.10. Applying basic trigonometry, we derive this area to be

$$4r\left(\frac{1}{4}\right)^2 \tan\frac{\pi}{4r} + r\cdot\left(\frac{1}{4}\right)^2 \tan^2\frac{\pi}{4r}\tan\frac{\pi}{2r} = \frac{r}{4}\tan\frac{\pi}{4r} + r\cdot\left(\frac{1}{4}\right)^2\tan^2\frac{\pi}{4r}\tan\frac{\pi}{2r}.$$

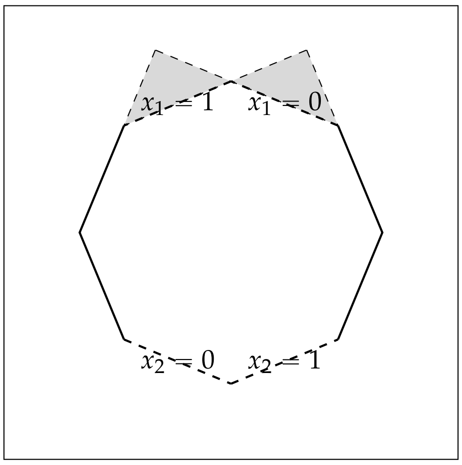


Figure 2.10: Main proof idea of Theorem 2.5.3

Now we construct the PMDP that enforces a similar construction. Since it is technically nontrivial to describe arbitrary hyperplanes of the form $\left\{\vec{x} \mid \vec{c}^{\top}\vec{x} = d\right\}$ with arbitrary coefficients in $\vec{c}$ when only non-negative coefficients $0 \leqslant \lambda_1, \lambda_2 \leqslant 1$ with an upper bound of 1 are allowed, we use a slightly different polygon. Furthermore, the structure of the underlying satisfiability instance has to be encoded in the MDP problem.

- Two states, u and d, are absorbing such that the cumulative reward in u is 1 and in d, respectively, 0.
- For $i \in [k]$, we define values

$$\begin{aligned}
\alpha_i^- &:= \frac{2 \cdot (4i+1)\pi}{16k}, \\
\alpha_i^+ &:= \frac{2 \cdot (4i+2)\pi}{16k}, \\
\psi_i^- &:= \frac{2 \cdot (4i)\pi}{16k}, \\
\psi_i^+ &:= \frac{2 \cdot (4i+3)\pi}{16k}.
\end{aligned}$$

  We observe that these values lie in the interval $\left[0, \pi/2\right]$, where the sine and cosine are both positive; we also see that this construction describes the first quadrant of a regular $16k$-gon; it is easy to see that the main properties of this figure are the same.
- For $i \in [k]$, we construct states a_i, b_i. The states a_i and b_i have only one action; the transition probabilities are

$$\begin{aligned}
\Pr\left[u \mid a_i, \cdot\right] &= \frac{1}{\sqrt{2}}\left(\lambda_1 \cos \psi_i^- + \lambda_2 \sin \psi_i^-\right) \\
\Pr\left[u \mid b_i, \cdot\right] &= \frac{1}{\sqrt{2}}\left(\lambda_1 \cos \psi_i^+ + \lambda_2 \sin \psi_i^+\right) \\
\Pr\left[d \mid a_i, \cdot\right] &= 1 - \frac{1}{\sqrt{2}}\left(\lambda_1 \cos \psi_i^- + \lambda_2 \sin \psi_i^-\right) \\
\Pr\left[d \mid b_i, \cdot\right] &= 1 - \frac{1}{\sqrt{2}}\left(\lambda_1 \cos \psi_i^+ + \lambda_2 \sin \psi_i^+\right)
\end{aligned}$$

The minimal discounted reward from a_i and b_i is set to $\frac{\gamma}{2\sqrt{2}}$. The idea behind these states is to create the "static" boundaries of the figure which cannot be changed with a decision. In the visualization in Figure 2.10, these boundaries are on the sides of the polygon. Here, these constraints create boundaries in the parameter space of the form

$$\frac{\gamma}{\sqrt{2}}\left(\lambda_1 \cos\psi_i^+ + \lambda_2 \sin\psi_i^+\right) \geqslant \frac{\gamma}{2\sqrt{2}} \Leftrightarrow$$
$$\lambda_1 \cos\psi_i^+ + \lambda_2 \sin\psi_i^+ \geqslant \frac{1}{2}$$

which corresponds to the space outside of the first quadrant of a regular $16k$-gon with radius $1/2$.

- For $j \in [n]$, we construct states s_j that correspond to clauses in the 3−SAT instance. In each state s_j, three actions are available that correspond to the literals L_1^j, L_2^j, L_3^j in the clause C_j. The transition probabilities are

$$\Pr\left[u \mid s_j, L\right] = \begin{cases} \frac{1}{\sqrt{2}}\left(\lambda_1 \cos\alpha_i^+ + \lambda_2 \sin\alpha_i^+\right) & L = x_i \\ \frac{1}{\sqrt{2}}\left(\lambda_1 \cos\alpha_i^- + \lambda_2 \sin\alpha_i^-\right) & L = \neg x_i \end{cases}$$

In analogy to the preceding discussion, we install a minimal expected discounted reward of $\frac{\gamma}{2\sqrt{2}}$. This ensures the constraint

$$\frac{1}{2} \leqslant \begin{cases} \lambda_1 \cos\alpha_i^+ + \lambda_2 \sin\alpha_i^+ & L = x_i \\ \lambda_1 \cos\alpha_i^- + \lambda_2 \sin\alpha_i^- & L = \neg x_i \end{cases}$$

We show now the correctness of this construction. If a policy yields a satisfying assignment, then the volume of the worst-case set will be at least $1 - (4kA + kS)$, where A is the area of an isosceles triangle T with top angle $\frac{2\pi}{16k}$ and side length $1/2$, and S is the area of an isosceles triangle with base length $\sin\frac{\pi}{16k}$ (which corresponds to the base length of T) and top angle $\pi - 2\alpha$. One can see that it is $A = \frac{1}{4}\sin\frac{\pi}{16k}\cos\frac{\pi}{16k}$ and $S = \frac{1}{4}\sin^2\frac{\pi}{16k}\cdot\tan\frac{\pi}{16k}$.

If there is no satisfying assignment, then there are two cases that have to be considered.

Case 1 The policy is pure and implies a contradicting assignment. Then the volume of the worst-case set will be less than $1 - (4kA + kS)$, as the additional constraints will exclude one of the polygons of area S.

Case 2 The policy is not pure in one of the states that corresponds to a clause of the form $x_i \vee \neg x_i \vee x_i$. We compute the resulting area of the corresponding part of the polygon. As the worst-case set is defined by the difference of a part of a $16k$-gon and the unit rectangle, we compute the area of the parameter space which is excluded by the chosen policy. The constraint has then the form $\lambda_1(pd_1 + (1-p)e_1) + \lambda_2(pd_2 + (1-p)e_2) \geqslant z$ where d_1, d_2 and e_1, e_2 are the coordinates of D resp. E in Figure 2.11, $p \in [0,1]$, and the resulting linear combination is the point X in Figure 2.11. The constraint is represented in the figure by the line $\overline{YZ}$. The point X' is the intersection of the constraint with the normal. It is easy to see that the distance $|OX'|$ is equal to the distance $|OD|$ scaled by the inverse of the distance $|OX|$.

Our area of interest equals to the area of the polygon $OAYZC$ in Figure 2.11, which is the difference between the area of the triangles $OX'Y'$ and $OX'Z'$ and the area of the orange triangles AYY' and CZZ'. In order to compute the complete area, we compute first several important values.

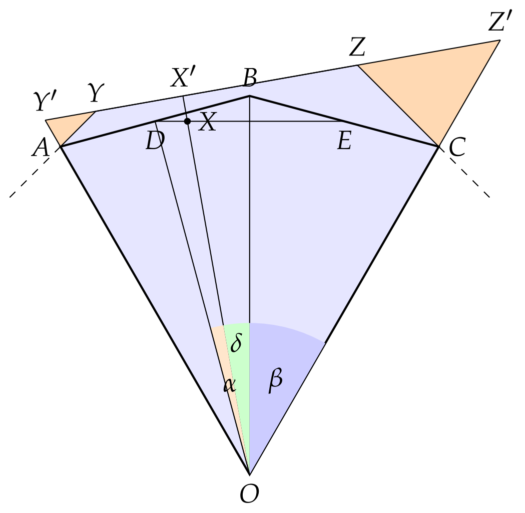


Figure 2.11: Illustration of the geometrical figure in proof of Theorem 2.5.3

First, we observe that the figure depends on the radius $r_0 = |OD|$ and the angles α and β. β and r_0 are given, and the only variable value is α. Knowing this, we can derive the different lengths and angles in the figure. The angle δ equals $\frac{\beta}{2} - \alpha$, and knowing this angle, we can derive the lengths $|OX| = \frac{|OD|\cos\frac{\beta}{2}}{\cos\delta}$ and $|OX'| = |OD|\frac{|OD|}{|OX|} = \frac{r_0\cos\delta}{\cos\frac{\beta}{2}}$. This yields the sum of the areas of the triangles $OX'Y'$ and $OX'Z'$ to be

$$\begin{aligned} A_0(\delta) &= \frac{|OX'|^2\tan\left(\beta+\delta\right) + |OX'|^2\tan\left(\beta-\delta\right)}{2} = \frac{|OX'|^2}{2}\Big(\tan\left(\beta+\delta\right) + \tan\left(\beta-\delta\right)\Big) \\ &= \frac{r_0^2\cos^2\delta}{2\cos^2\frac{\beta}{2}}\Big(\tan\left(\beta+\delta\right) + \tan\left(\beta-\delta\right)\Big). \end{aligned}$$

Now we compute the areas of the triangles CZZ' and AYY'. First, we compute the angles.

$$\begin{aligned} \measuredangle Z'CZ = \measuredangle YAY' &= \frac{\pi-\beta}{2}, \\ \measuredangle AY'Y &= \pi - \measuredangle OX'Y' - \measuredangle AOX' = \pi - \frac{\pi}{2} - \left(\beta-\delta\right) = \frac{\pi}{2} - \left(\beta-\delta\right) \\ \measuredangle CZ'Z &= \frac{\pi}{2} - \left(\beta+\delta\right) \\ \measuredangle CZZ' &= \pi - \measuredangle Z'CZ - \measuredangle CZ'Z = \pi - \frac{\pi-\beta}{2} - \left(\frac{\pi}{2} - \left(\beta+\delta\right)\right) = \frac{3\beta}{2} + \delta \\ \measuredangle AYY' &= \frac{3\beta}{2} - \delta. \end{aligned}$$

Then, we compute the lengths of the sides $|AY'|$ and $|CZ'|$. It is

$$|AY'| = |OY'| - |OA| = \frac{|OX'|}{\cos\left(\beta-\delta\right)} - \frac{r_0}{\cos\frac{\beta}{2}} = \frac{r_0\cos\delta}{\cos\left(\beta-\delta\right)\cos\frac{\beta}{2}} - \frac{r_0}{\cos\frac{\beta}{2}}.$$

and, analogously,

$$|CZ'| = \frac{r_0 \cos\delta}{\cos\left(\beta+\delta\right)\cos\frac{\beta}{2}} - \frac{r_0}{\cos\frac{\beta}{2}}.$$

The area of these two triangles is then

$$\begin{aligned}
A_1(\delta) &= \frac{|AY'|^2 \sin\frac{\pi-\beta}{2}\sin\left(\frac{\pi}{2}-(\beta-\delta)\right)}{2\sin\left(\frac{3\beta}{2}-\delta\right)} + \frac{|CZ'|^2 \sin\frac{\pi-\beta}{2}\sin\left(\frac{\pi}{2}-(\beta+\delta)\right)}{2\sin\left(\frac{3\beta}{2}+\delta\right)} \\
&= \frac{r_0^2}{2}\cdot\left(\frac{\cos\delta}{\cos\left(\beta-\delta\right)\cos\frac{\beta}{2}} - \frac{1}{\cos\frac{\beta}{2}}\right)^2 \frac{\cos\frac{\beta}{2}\cos\left(\beta-\delta\right)}{\sin\left(\frac{3\beta}{2}-\delta\right)} + \\
&+ \frac{r_0^2}{2}\cdot\left(\frac{\cos\delta}{\cos\left(\beta+\delta\right)\cos\frac{\beta}{2}} - \frac{1}{\cos\frac{\beta}{2}}\right)^2 \frac{\cos\frac{\beta}{2}\cos\left(\beta+\delta\right)}{\sin\left(\frac{3\beta}{2}+\delta\right)} \\
&= \frac{r_0^2}{2\cos\frac{\beta}{2}}\cdot\left(\frac{\left(\cos\delta-\cos\left(\beta-\delta\right)\right)^2}{\cos\left(\beta-\delta\right)\sin\left(\frac{3\beta}{2}-\delta\right)} + \frac{\left(\cos\delta-\cos\left(\beta+\delta\right)\right)^2}{\cos\left(\beta+\delta\right)\sin\left(\frac{3\beta}{2}+\delta\right)}\right)
\end{aligned}$$

Now we can derive the formula for the area of interest, which is $A(\delta) = A_0(\delta) - A_1(\delta)$. The derivative of this function is

$$\begin{aligned}
\frac{\mathrm{d}A}{\mathrm{d}\delta}\cdot\frac{2\cos\frac{\beta}{2}}{r_0^2} = &\frac{-2\cos(\delta)(\tan(\beta+\delta)+\tan(\beta-\delta))\sin(\delta)}{\cos\left(\frac{\beta}{2}\right)} \\
&+ \frac{\cos^2(\delta)(\tan^2(\beta+\delta)-\tan^2(\beta-\delta))}{\cos\left(\frac{\beta}{2}\right)} \\
&- \frac{2(\cos(\delta)-\cos(\beta-\delta))(-\sin(\delta)-\sin(\beta-\delta))}{\cos(\beta-\delta)\sin\left(\frac{3\beta}{2}-\delta\right)} \\
&+ \frac{(\cos(\delta)-\cos(\beta-\delta))^2\sin(\beta-\delta)}{\cos^2(\beta-\delta)\sin\left(\frac{3\beta}{2}-\delta\right)} \\
&- \frac{(\cos(\delta)-\cos(\beta-\delta))^2\cos\left(\frac{3\beta}{2}-\delta\right)}{\cos(\beta-\delta)\sin^2\left(\frac{3\beta}{2}-\delta\right)} \\
&- \frac{2(\cos(\delta)-\cos(\beta+\delta))(-\sin(\delta)+\sin(\beta+\delta))}{\cos(\beta+\delta)\sin\left(\frac{3\beta}{2}+\delta\right)} \\
&- \frac{(\cos(\delta)-\cos(\beta+\delta))^2\sin(\beta+\delta)}{\cos^2(\beta+\delta)\sin\left(\frac{3\beta}{2}+\delta\right)} \\
&+ \frac{(\cos(\delta)-\cos(\beta+\delta))^2\cos\left(\frac{3\beta}{2}+\delta\right)}{\cos(\beta+\delta)\sin^2\left(\frac{3\beta}{2}+\delta\right)}
\end{aligned}$$

Computing the zeros of this function with Maple [Map14], we obtain

$$\frac{\mathrm{d}A}{\mathrm{d}\delta}(\delta) = 0 \Leftrightarrow \delta \in \left\{0, \pm\frac{\pi}{2}, \pi\right\}$$

All zeros besides 0 are outside of the range $\left[-\frac{\beta}{2}, \frac{\beta}{2}\right]$ for $\beta < \pi$, and the second derivative at $\frac{\beta}{2}$ is negative. This means that for $\delta = 0$, the complement of the worst-case set is maximized, and the worst-case set itself is minimized. This concludes the proof.

□

The proofs allow us to derive results for the finite-horizon total reward measure. The idea is straightforward: As all paths in the parameterized Markov decision processes that occur in our constructions lead in a finite number of steps to an absorbing state, we can also consider the total reward over a finite horizon that is equal to the maximal path length to an absorbing state. This leads to the following corollary.

Corollary 2.5.4. *Optimizing the probability of reaching a certain minimal expected total reward over a finite horizon in a parameterized Markov decision process is* NP*-hard in the worst case if either of the following statements is true.*

- *The number of parameters is at least two and the transition probabilities for each action depend on an unconstrained number of parameters.*
- *The transition probabilities for each action depend on at most one parameter, but the number of parameters is unconstrained and the number of nonzero rows in the corresponding influence matrices is unconstrained.*

If the transition probabilities for each action and each state depend on at most one parameter and each parameter influences at most one state, then the rectangularity property holds and efficient algorithms for policy optimization for the expected discounted reward and the expected average reward measures exist.

Upper bounds Since percentile optimization involves solving function problems, upper bounds seem nontrivial. With an oracle that can compute the continuous distribution function for the value vector, percentile optimization can be performed with a NP algorithm by guessing a sufficiently good policy. For a constant discount factor and polynomial precision, even non-stationary policies can be represented in polynomial space; thanks to the oracle, verification of the policy can be done in polynomial time.

Corollary 2.5.5. *Deciding if the probability of reaching a certain minimal expected total reward over a finite horizon in a PMDP can be surpassed with a policy is* NP*-complete under the conditions in Corollary 2.5.4 if there exists an oracle for the continuous distribution function on the PMDP.*

2.5.3 Relevance to computational geometry

It is easy to see that Theorem 2.5.2 and Theorem 2.5.3 share a common idea and a common construction that is more general and can be applied outside of the usual Markov decision process context. It is possible to extract this idea in form of a corollary.

Definition 2.5.3. Let $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_n$ be finite sets of halfspaces which are bounded by hyperplanes in $\mathbb{R}^k$ and $V \in \mathbb{R}$ a real number. We call the problem of deciding if there is a set of halfspaces $P_1, \ldots, P_n$ such that $P_i \in \mathcal{P}_i$ and the volume of the intersection $\left\{x \mid x \in \cap_{i=1}^{n} P_i\right\}$ is at least V the *polytope volume optimization problem.*

Corollary 2.5.6. *The polytope volume optimization problem is* NP*-hard, even if at most one of two simplifying conditions is met:*

- *Each hyperplane is orthogonal to a basis vector $\vec{e}_i$.*

- *The number of dimensions k is fixed and it is $k \geqslant 2$.*

We furthermore can show that the conditions for NP-hardness are tight.

Theorem 2.5.7. *For a constant number of dimensions k, if each hyperplane is orthogonal to a basis vector $\vec{e}_i$, then the polytope volume optimization problem can be solved in polynomial time.*

Proof. The first observation on our way to the proof is that if each hyperplane that bounds a halfspace is orthogonal to a basis vector, then the resulting polytope will be a box, with the volume $\prod_{i=1}^{k}(u_i - l_i)$, where u_i is the tightest upper bound for the i-th coordinate inside the box and l_i is the respective tightest lower bound. This reduces the problem to optimizing each difference $u_i - l_i$ individually. We now show that there are only polynomially many options for the upper and lower bounds for each coordinate.

We compute, for all $i \in [k]$, the number N_i of possible feasible intervals

$$\mathcal{I}_i := \left\{ \left[l_i^1, u_i^1\right], \ldots, \left[l_i^{N_i}, u_i^{N_i}\right] \right\}$$

for the coordinate x_i: For any assignment of hyperplanes $\pi \in \mathcal{P}_1 \times \cdots \times \mathcal{P}_n$, there will be one set $\mathcal{P}_+$ that defines the tightest upper bound for x_i and at most one set $\mathcal{P}_-$ that defines the tightest lower bound for x_i; these bounds are defined solely by the selected hyperplane in the respective sets. Since in each set, there can be only polynomially many halfspaces, there can be at most $m := \sum_{i=1}^{k} |\mathcal{P}_i|$ lower and m upper bounds for each coordinate. Combining all sets and all coordinates together, we get an upper bound of $\left(\sum_{i=1}^{k} |\mathcal{P}_i|\right)^k$ for the total number of possible bounding polytopes, which is polynomial in the input size if k is constant.

However, not every polytope $P \in \left\{I_1, \ldots, I_k \mid I_i \in \mathcal{I}_i, i \in [k]\right\}$ that results from a possible combination of intervals in $\mathcal{I}_1, \ldots, \mathcal{I}_k$ can be the result of a valid selection of halfspaces since the halfspaces can be taken from one set; thus, we will have to additionally check for feasibility, which can be done by keeping a data structure that maps upper and lower bounds to the respective pair of set $\mathcal{P}$ and halfspace.

This consideration delivers an algorithm to find the largest bounding polytope. First, we compute the sets $\mathcal{I}_1, \ldots, \mathcal{I}_k$ which can be done in time $\mathcal{O}\left(m \log m\right)$ and then, we can sort the set $\left\{I_1, \ldots, I_k \mid I_i \in \mathcal{I}_i, i \in [k]\right\}$ with respect to the volume of the polytope. This can be done in time $\mathcal{O}\left(k \cdot m^k \log m\right)$; then, we can check for every polytope beginning with the smallest if it corresponds to a valid assignment. The total time complexity is thus $\mathcal{O}\left(k \cdot m^k \log m\right)$ which is polynomial for fixed k; furthermore, the chosen perspective makes the polytope volume optimization problem fixed-parameter tractable. □

2.5.4 The simple case: One-state one-parameter models

We finish this section with a discussion of a simple case of percentile optimization in stochastic MDPs. Here, we assume that there exists only one state where the transition probabilities are uncertain and $k = 1$, i.e., that the set T^a of all transition matrices can be expressed as $\left\{\boldsymbol{P}_0^a + \lambda \vec{e}_i \vec{z}^a \mid \lambda \in [0,1]\right\}$ for some $i \in S$ where $\vec{z}^a$ is a row vector subject to $\vec{z}^a \vec{1} = 0$ and $\boldsymbol{P}_0^a(i\bullet) + \vec{z} \geqslant \vec{0}$. For the influence matrix $\Delta^a = \vec{e}_i \vec{z}^a$, it is $\operatorname{rk} \Delta^a = 1$ as Δ^a has only one nonzero row. Then, given a sum of expected discounted total rewards d, we compute a parameter value λ such that the relation $d = \vec{1}^\top \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a - \gamma \lambda \Delta^a\right)^{-1} \vec{r}$ holds. Using the result in [Mil81] and the fact that $\operatorname{rk} \Delta^a = 1$, we obtain

$$d = \vec{1}^\top \left(\left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} + \frac{\left(I - \gamma \boldsymbol{P}_0^a\right)^{-1} \lambda \gamma \Delta \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1}}{1 + \operatorname{tr}\left(\lambda \gamma \Delta^a \left(\boldsymbol{I} - \gamma \boldsymbol{P}^{a\prime}\right)^{-1}\right)} \right) \vec{r}$$

By linearity of $\operatorname{tr}(\cdot)$ and general properties of linear operators, we can further simplify the term and solve it for λ.

$$\begin{aligned} d &= \vec{1}^\top \left(\left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} + \frac{\lambda \cdot \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} \gamma \Delta^a \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1}}{1 + \lambda \gamma \operatorname{tr}\left(\Delta^a \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1}\right)} \right) \vec{r} \\ &= \vec{1}^\top \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} \vec{r} + \frac{\lambda \gamma \vec{1}^\top \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} \Delta^a \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} \vec{r}}{1 + \lambda \gamma \operatorname{tr}\left(\Delta^a \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1}\right)} \end{aligned}$$

Introducing the variables

$$\begin{aligned} \alpha &= \gamma \vec{1}^\top \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} \Delta^a \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} \vec{r}, \\ \beta &= \vec{1}^\top \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1} \vec{r}, \\ g &= \operatorname{tr}\left(\Delta \left(\boldsymbol{I} - \gamma \boldsymbol{P}_0^a\right)^{-1}\right), \end{aligned}$$

we get

$$\begin{aligned} d - \beta = \alpha \cdot \frac{\lambda}{1 + \lambda g} &\Leftrightarrow \\ \lambda \alpha = (d - \beta) + (d - \beta) \cdot \lambda g &\Leftrightarrow \\ \lambda \left(\alpha - (d - \beta) \cdot g\right) = (d - \beta) &\Leftrightarrow \\ \lambda = \frac{d - \beta}{\alpha - (d - \beta) \cdot g}&, \end{aligned} \tag{2.14}$$

thus inferring a term for the value of λ from a given expected discounted total reward d. Using the fact that the mapping (2.14) is continuous and the values of α, β, and g are independent of d, we can establish the following for the cumulative probability $\Pr_{\mathcal{M},p}\left[d \in \left[d^-, d^+\right]\right]$:

$$\begin{aligned} \Pr_{\mathcal{M},p}\left[d \in \left[d^-, d^+\right]\right] &= \Pr\left[x \in \left[\lambda\left(d^-\right), \lambda\left(d^+\right)\right]\right] \\ &= \int_{\lambda\left(d^-\right)}^{\lambda\left(d^+\right)} p_\lambda(x)\, \mathrm{d}x \\ &= \int_{\frac{d^- - \beta}{\alpha - (d^- - \beta) \cdot g}}^{\frac{d^+ - \beta}{\alpha - (d^+ - \beta) \cdot g}} p_\lambda(x)\, \mathrm{d}x, \end{aligned} \tag{2.15}$$

where p_λ is the continuous distribution function of the sole parameter λ, the values α, β, and g are as defined above and d^- and d^+ are such that $\frac{x-\beta}{\alpha-(x-\beta)\cdot g} \in [0, 1]$.

Example 2.5.1. We can compute the exact value of the integral (2.15) for several probability distributions. For the uniform distribution $\lambda \sim \mathcal{U}(\lambda^-, \lambda^+)$ we have

$$\begin{aligned} \Pr_{\mathcal{M},p}\left[d \in \left[d^-, d^+\right]\right] &= \int_{\frac{d^- - \beta}{\alpha - (d^- - \beta) \cdot g}}^{\frac{d^+ - \beta}{\alpha - (d^+ - \beta) \cdot g}} p_\lambda(x)\, \mathrm{d}x = x \Bigg|_{\frac{d^- - \beta}{\alpha - (d^- - \beta) \cdot g}}^{\frac{d^+ - \beta}{\alpha - (d^+ - \beta) \cdot g}} \cdot \frac{1}{\lambda^+ - \lambda^-} \\ &= \frac{1}{\lambda^+ - \lambda^-} \cdot \left(\frac{d^+ - \beta}{\alpha - (d^+ - \beta) g} - \frac{d^- - \beta}{\alpha - (d^- - \beta) g} \right). \end{aligned}$$

From an applied perspective, we might be interested in modeling uncertainty and thus, the distribution of λ with a "practical" probability distribution such as the Beta distribution. Then, for parameters $a > 0, b > 0$ and $\lambda \sim \mathrm{Beta}(a, b)$ we get

$$\Pr_{\mathcal{M},p}\left[d \in \left[d^-, d^+\right]\right] = \int_{\frac{d^- - \beta}{\alpha - (d^- - \beta)\cdot g}}^{\frac{d^+ - \beta}{\alpha - (d^+ - \beta)\cdot g}} p_\lambda(x)\,\mathrm{d}x$$
$$= \frac{B\left(\frac{d^+ - \beta}{\alpha - (d^+ - \beta)g}; a, b\right) - B\left(\frac{d^- - \beta}{\alpha - (d^- - \beta)g}; a, b\right)}{B(a, b)}$$

where $B(a, b) = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$ is the beta function and

$$B(x; a, b) = \int_0^x t^{a-1}(1-t)^{b-1}\,\mathrm{d}t$$

is the incomplete beta function.

Using the method discussed above, we can now compute "almost robust" policies which are characterized by a minimal expected total discounted reward d^- if, for all policies, uncertainty occurs in at most one state and has a linear form. Furthermore, we consider a special case where the policy does not change with λ, if the action in state s is constant. This can be done by extension of general MDP and BMDP methods as follows. From BMDP theory [GLD00], we know that the maximal and minimal values are observed at extreme values of λ, as the choice of λ that maximizes or minimizes the value depends on the order of the values in the states. Hence, if s is the state where the uncertainty can occur, then, for each action a and the two extreme cases of the uncertainty, that is, $\lambda \in \{0, 1\}$, we can compute the value $d_{a,\lambda}$ if λ is fixed and the action in s is set to a. Thus, we can compute, given d^-, the values

$$q_a = \Pr_{\mathcal{M},p}\left[d \in \left[d^-, \max(d_{a,0}, d_{a,1})\right]\right],$$

obtaining the optimal action $a^* = \arg\max_a q_a$. The whole procedure, takes $|A|$ MDP optimizations and $|A|$ evaluations of Eq. 2.15, which, given an integration oracle for the density function p, can be performed in polynomial time. Together, we obtain the following algorithm.

Algorithm 12 Percentile optimization in the simple case

```
function PercentileOptimization($S, A, s, (\boldsymbol{P}_0^a)_{a\in A}, (\Delta^a)_{a\in A}, \vec{r}, \gamma$)
    for $a \in A, \lambda \in \{0, 1\}$ do
        $\vec{v}_{a,\lambda} \leftarrow \vec{v}_\gamma$with parameter $\lambda$ and $\pi(s) = a$
        $\pi_a \leftarrow$ optimal policy with $\pi(s) = a$
    for $a \in A$ do
        $q_a \leftarrow \Pr_{\mathcal{M},p}\left[d \in \left[d^-, \max(d_{a,0}, d_{a,1})\right]\right]$
    $a^* \leftarrow \arg\max_a q_a$
    return $\pi_{a^*}$
```

3

Algorithms for multi-objective problems

The difference between theory and practice is smaller in theory than in practice.

— Roger Pate

NOW we turn our attention to the practical part of our results. In particular, we consider the multi-objective problems mentioned in Chapter 2. We have seen there that they are theoretically hard to solve, but now we present algorithms that aim to be practically efficient. In the first part of this chapter, we consider different approaches to the *multi-scenario stochastic optimization problem* for concurrent Markov decision processes which we have introduced in Sec. 2.4. Then, we consider the *enumeration problem* for the Pareto frontier for stochastic bounded-parameter MDPs. There, we consider algorithms that compute the Pareto frontier, i. e., the set of all non-dominated policies in value vector space. In both cases, we consider the expected discounted reward optimality measure with a given discount factor $\gamma \in [0, 1)$.

The results are based on the findings in [SBHH17, BS17a]. Here, they have, in contrast to the results in the previous chapter, mostly empirical nature, as we consider algorithms for computationally hard problems. This means that we design heuristic algorithms and evaluate them empirically, by defining test cases, running in silico experiments, and describing the performance of different algorithmic approaches on the considered test cases.

3.1 Stochastic multi-scenario optimization

First, we discuss the *stochastic multi-scenario optimization problem* whose theoretical properties are discussed in the previous chapter. Here, we consider the practical possibilities to optimize the weighted sum of rewards in concurrent MDPs. We design and empirically compare several algorithms that optimize the weighted sum of rewards in several independent MDPs with shared state and action spaces under a shared policy. We concentrate ourselves on pure and stationary policy classes, as non-stationary policies are hard to store in practice.

In the previous chapter (Def. 2.1.3), we define a concurrent MDP to be a collection $\mathcal{M}$ of finitely many, $K \in \mathbb{N}$ MDPs with a common state space $S = [n]$, a common action space $A = [m]$, a common initial distribution $\vec{q} \in \mathbb{R}^n$, non-negative reward vectors $(r_k)_{k\in[K]}$ and transition probability matrices $(\boldsymbol{P}_k^a)_{a\in A, k\in[K]}$. In the following paragraphs, we solve the stochastic multi-scenario optimization problem as defined above. As Theorem 2.4.4 shows, the corresponding decision problem is NP-hard and there is little hope for a polynomial-time optimization algorithm. Hence, we turn to generic methods. In the following discussion, we apply generic-purpose methods for the stochastic optimization problem as described in Def. 2.4.2 and compare their performance empirically.

A QCLP formulation Before we consider more problem-specific formulations, we first derive a constrained mathematical program. Starting with a linear program for a single MDP as given in Eq. (1.9), we consider policy matrices $\boldsymbol{\Pi} \in \mathbb{R}^{n \times m}$ where $\boldsymbol{\Pi}(s, a)$ is the probability to choose action a in state s under a given stationary policy Π. We introduce the matrices $\boldsymbol{P}_k^{\boldsymbol{\Pi}} \in \mathbb{R}^{n \times n}$ and $\boldsymbol{C}_k^{\boldsymbol{\Pi}} \in \mathbb{R}^{n \times n}$ with

$$\boldsymbol{P}_k^{\boldsymbol{\Pi}}(s, s') = \sum_{a \in A} \boldsymbol{\Pi}(s, a) \boldsymbol{P}_k^a(s, s')$$

and

$$\boldsymbol{C}_k^{\boldsymbol{\Pi}} = \boldsymbol{I} - \gamma \boldsymbol{P}_k^{\boldsymbol{\Pi}}$$

for $k \in [K]$, with which we define the following mathematical optimization program

$$\begin{aligned}
&\max_{\boldsymbol{\Pi}} \left(\min_{\vec{v}_k} \sum_{k \in [K]} \vec{w}(k) \vec{q}^{\top} \vec{v}_k \right) \\
&\text{s.t.} \\
&\quad \boldsymbol{C}_k^{\boldsymbol{\Pi}} \vec{v}_k \geqslant \vec{r}_k && \forall k \in [K] \\
&\quad \sum_{a \in A} \boldsymbol{\Pi}(s, a) = 1 && \forall s \in S \\
&\quad \boldsymbol{\Pi}(s, a) \geqslant 0 && \forall s \in S, a \in A
\end{aligned} \tag{3.1}$$

This formulation is a two-stage non-linear program with $2Kn + nm$ inequality constraints and n equality constraints. The non-linearity can be explicitly seen in the first set of constraints, $\boldsymbol{C}_k^{\boldsymbol{\Pi}} \vec{v}_k \geqslant \vec{r}_k$, where the left hand side contains products of variables from $\boldsymbol{\Pi}$ and $\vec{v}_k$.

To constrain non-linearity and generality of the program, we first get rid of the min-max term. To do so, we use the dual LP formulation (1.10) for the MDP optimization problem and obtain the following optimization problem.

$$\begin{aligned}
&\max_{\boldsymbol{\Pi}} \left(\max_{\vec{x}_k} \sum_{k \in [K]} \vec{w}(k) \vec{x}_k^{\top} \vec{r}_k \right) \\
&\text{s.t.} \\
&\quad (\boldsymbol{C}_k^{\boldsymbol{\Pi}})^{\top} \vec{x}_k = \vec{q} && \forall k \in [K] \\
&\quad \sum_{a \in A} \boldsymbol{\Pi}(s, a) = 1 && \forall s \in S \\
&\quad \boldsymbol{\Pi}(s, a) \geqslant 0 && \forall s \in S, a \in A
\end{aligned} \tag{3.2}$$

Having got rid of the min-max goal, we can re-formulate the problem as a *quadratically constrained linear program* (QCLP) [BV04, ABZ07]. For this, we observe that the non-linear inequality constraints have a bilinear term on the left hand side. Defining the matrix

$$\boldsymbol{A}_{s,k} = \begin{pmatrix} \vec{e}_s - \gamma \boldsymbol{P}_k^1(s\bullet) \\ \vdots \\ \vec{e}_s - \gamma \boldsymbol{P}_k^m(s\bullet) \end{pmatrix}$$

and observing that

$$\boldsymbol{C}_k^{\boldsymbol{\Pi}} = \begin{pmatrix} \boldsymbol{\Pi}(1\bullet)\boldsymbol{A}_{1,k} \\ \vdots \\ \boldsymbol{\Pi}(n\bullet)\boldsymbol{A}_{n,k} \end{pmatrix}$$

we can represent each inequality constraint as

$$\sum_{s' \in S} \vec{x}_k(s') \sum_{a \in A} \boldsymbol{\Pi}(s', a)\boldsymbol{A}_{s',k}(a, s) = \sum_{s' \in S} \vec{x}_k(s')\boldsymbol{\Pi}(s'\bullet)\boldsymbol{A}_{s',k}(\bullet s) \leqslant \vec{w}(k)\vec{q}(s). \tag{3.3}$$

All these inequality constraints are at most bilinear in the variables $\vec{x}_k$ and $\boldsymbol{\Pi}$, and can thus be represented as a quadratic constraint in the following way. For a combined variable vector $\vec{y}_k = \left(\boldsymbol{\Pi}(1\bullet), \ldots, \boldsymbol{\Pi}(n\bullet), \vec{x}_k^\top\right) \in \mathbb{R}^{(mn+n)\times 1}$ and a combined objective vector $\vec{c}_k = \left(0, \ldots, 0, \vec{r}_k^\top\right)$ we define matrices $\boldsymbol{F}_{s,k}$ for which the constraint (3.3) can be represented with $\frac{1}{2}\vec{y}_k\boldsymbol{F}_{s,k}\vec{y}_k^\top \leqslant \vec{w}(k)\vec{q}(s)$. By defining

$$\boldsymbol{G}_{s,k} = \begin{pmatrix} \boldsymbol{A}_{1,k}(\bullet s) & \boldsymbol{0} & \ldots & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & & \vdots \\ \vdots & & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \ldots & \boldsymbol{0} & \boldsymbol{A}_{n,k}(\bullet s) \end{pmatrix} \in \mathbb{R}^{nm \times n}$$

and

$$\boldsymbol{F}_{s,k} = \begin{pmatrix} \boldsymbol{0} & \boldsymbol{G}_{s,k} \\ \boldsymbol{G}_{s,k}^\top & \boldsymbol{0} \end{pmatrix} \in \mathbb{R}^{(nm+n)\times(nm+n)}$$

we obtain the following QCLP.

$$\begin{aligned} & \max\, (\vec{c}_1, \ldots, \vec{c}_K)\left(\vec{y}_1, \ldots, \vec{y}_K\right)^\top \\ \text{s.t.} \quad & \\ & \frac{1}{2}\vec{y}_k\boldsymbol{F}_{s,k}\vec{y}_k^\top \leqslant \vec{w}(k)\vec{q}(s) && \forall s \in S, k \in [K] \\ & \boldsymbol{\Pi}(s\bullet)\vec{1} = 1 && \forall s \in S \\ & \boldsymbol{\Pi}(s, a) \geqslant 0 && \forall s \in S, a \in A \\ & \vec{x}_k(s) \geqslant 0 && \forall s \in S, k \in [K] \end{aligned} \tag{3.4}$$

A general argument for the usage of constrained classes of mathematical programming formulations is the possibility to solve them more efficiently. Most prominently, this is the case for linear and convex programs, that is, optimization problems where the constraints describe a convex set and the objective function is convex [NN94]. Furthermore, it is known that quadratically constrained linear programs are also convex if the quadratic constraint matrices are positive definite. Unfortunately (and not much surprisingly, considering the hardness result above), the matrices $\boldsymbol{F}_{s,k}$ are indefinite in the general case. In this situation, we resort to non-problem-specific approaches for general quadratically constrained LPs, such as those described in [QBM12].

A general non-linear formulation If we are not constrained by specific classes of mathematical programming problems, we can use most general formulations and feed them into general global optimizers, providing them additionally with information on partial derivatives, that is, $\frac{\partial f(\vec{x})}{\partial \vec{x}(i)}$, if f is the function to optimize and $\vec{x}$ is the vector of input variables. In the following, we provide such a formulation alongside with partial derivatives.

In our formulation we represent the policy Π directly as a set of decision variables $\mathbf{\Pi}$, as discussed above. Knowing that the value vector $\vec{v}^{(\Pi)}$ adheres to $\vec{v}^{(\Pi)} = (\boldsymbol{I} - \gamma \boldsymbol{P}^{\mathbf{\Pi}})^{-1}\vec{r}$, we can derive the following mathematical program for a given weight vector $\vec{w} \in \mathbb{R}^K$.

$$
\begin{aligned}
&\max f(\mathbf{\Pi}) = \sum_{k\in[K]} \vec{w}(k)\vec{q}^\top \vec{v}_k^\Pi \\
&\text{s.t.} \\
&\boldsymbol{P}_k^{\mathbf{\Pi}}(s,s') = \sum_{a\in A} \mathbf{\Pi}(s,a)\boldsymbol{P}_k^a(s,s') && \forall s,s' \in S, k \in [K] \\
&(\boldsymbol{I} - \gamma \boldsymbol{P}_k^{\mathbf{\Pi}})\vec{v}_k^\Pi = \vec{r}_k && \forall k \in [K] \\
&\mathbf{\Pi}(s,a) \geqslant 0 && \forall s \in S, a \in A \\
&\sum_{a\in A} \mathbf{\Pi}(s,a) = 1 && \forall s \in S
\end{aligned}
\tag{3.5}
$$

Additionally, we have to provide partial derivatives $\frac{\partial \vec{v}_k^\Pi(s)}{\partial \mathbf{\Pi}(s,a)}$. For this, we observe that computing the partial derivatives implies computing the derivative of a matrix inverse. Let $\boldsymbol{C}_k^{\mathbf{\Pi}} = (\boldsymbol{I} - \boldsymbol{P}_k^{\mathbf{\Pi}})^{-1}$. If we change $\mathbf{\Pi}(s,a)$ by adding λ, we hereby add $\lambda \vec{e}_s \boldsymbol{P}_k^a(s\bullet)$ to $\boldsymbol{P}_k^{\mathbf{\Pi}}$, and, equivalently, $\lambda \vec{e}_s(\vec{e}_s^\top - \gamma \boldsymbol{P}_k^a(s\bullet))$ to the matrix $(\boldsymbol{I} - \gamma \boldsymbol{P}_k^{\mathbf{\Pi}})$ we have to invert. This is a rank-one update, for which we can apply the formula from [Hag89] that describes the result of a matrix inversion after a rank-ℓ update. For a rank-one update of a matrix $\boldsymbol{M}$ in the i-th row by a (transposed column) vector $\vec{x}^\top$, it is

$$
\begin{aligned}
(\boldsymbol{M} + \vec{e}_i\vec{x}^\top)^{-1} &= \boldsymbol{M}^{-1} - \frac{\boldsymbol{M}^{-1}\vec{e}_i\vec{x}^\top\boldsymbol{M}^{-1}}{1 + \vec{x}^\top\boldsymbol{M}^{-1}\vec{e}_i} \\
&= \boldsymbol{M}^{-1} - \frac{\boldsymbol{M}^{-1}(\bullet i)\vec{x}^\top\boldsymbol{M}^{-1}}{1 + \vec{x}^\top\boldsymbol{M}^{-1}(\bullet i)}
\end{aligned}
$$

Applying this to the function $f(\mathbf{\Pi})$ that is given in (3.5), we derive the following. For

$$
\mathbf{\Pi}'(s',a') = \begin{cases} \mathbf{\Pi}(s,a) + \lambda & s' = s \wedge a = a \\ \mathbf{\Pi}(s',a') & \text{otherwise} \end{cases}
$$

we have

$$
f(\mathbf{\Pi}') = f(\mathbf{\Pi}) - \lambda \sum_{k=1}^{K} \frac{\vec{w}(k)\vec{q}^\top \boldsymbol{C}_k^{\mathbf{\Pi}}(\bullet s)(\vec{e}_s^\top - \gamma \boldsymbol{P}_k^a(s\bullet))\boldsymbol{C}_k^{\mathbf{\Pi}}\vec{r}_k}{1 + \lambda(\vec{e}_s^\top - \gamma \boldsymbol{P}_k^a(s\bullet))\boldsymbol{C}_k^{\mathbf{\Pi}}(\bullet s)} \tag{3.6}
$$

if the resulting inverse matrix exists for the given value of λ. Hence, the partial derivatives of $f(\mathbf{\Pi})$ are, by computing $\lim_{\lambda\to 0} \frac{f(\mathbf{\Pi}')-f(\mathbf{\Pi})}{\lambda}$,

$$
\frac{\partial f(\mathbf{\Pi})}{\partial \mathbf{\Pi}(s,a)} = -\sum_{k=1}^{K} \vec{w}(k)\vec{q}^\top \boldsymbol{C}_k^{\mathbf{\Pi}}(\bullet s)\left(\vec{e}_s^\top - \gamma \boldsymbol{P}_k^a(s\bullet)\right)\boldsymbol{C}_k^{\mathbf{\Pi}}\vec{r}_k \tag{3.7}
$$

and the gradient is

$$
\begin{aligned}
\nabla f(\mathbf{\Pi}) = \Bigg(&-\sum_{k=1}^{K} \vec{w}(k)\vec{q}^\top \boldsymbol{C}_k^{\mathbf{\Pi}}(\bullet 1)(\vec{e}_1^\top - \gamma \boldsymbol{P}_k^1(1\bullet))\boldsymbol{C}_k^{\mathbf{\Pi}}\vec{r}_k, , \\
&\ldots, \\
&-\sum_{k=1}^{K} \vec{w}(k)\vec{q}^\top \boldsymbol{C}_k^{\mathbf{\Pi}}(\bullet n)(\vec{e}_n^\top - \gamma \boldsymbol{P}_k^m(n\bullet))\boldsymbol{C}_k^{\mathbf{\Pi}}\vec{r}_k \Bigg)
\end{aligned}
\tag{3.8}
$$

Together, the functions f, ∇f, and the constraints can be passed to a non-linear global optimization routine. The constraints are now linear, however, at the cost of the goal function complexity.

A mixed-integer linear program formulation If we constrain ourselves to pure policies only, we may use integer programming approaches. We remember that for general MDPs, it is possible, starting with the dual LP for MDPs (1.10), to derive integer decision variables, and, thus, an integer programming formulation (1.12). For concurrent MDPs, the integer program (1.12) can be extended. We get the following mixed-integer program.

$$
\begin{aligned}
&\max \sum_{k=1}^{K} \vec{w}(k) \sum_{s \in S} x_{k,s,a} \vec{r}_k(s) \\
\text{s.t.}\quad & \\
&\sum_{a \in A} x_{k,s,a} - \gamma \sum_{a \in A, s' \in S} \boldsymbol{P}^a(s', s) x_{k,s',a} = \vec{q}(s) && \forall s \in S, k \in [K] \\
&\sum_{a \in A} d_{s,a} = 1 && \forall s \in S \\
&d_{s,a} \geqslant (1-\gamma) x_{k,s,a} && \forall s \in S, a \in A, k \in [K] \\
&d_{s,a} \in \{0,1\} && \forall s \in S, a \in A
\end{aligned}
\tag{3.9}
$$

By using common decision variables $d_{s,a}$, we ensure that $x_{k,s,a} > 0$ holds if and only if $d_{s,a} = 1$, as $x_{k,s,a}$ has an upper bound of $\frac{1}{1-\gamma}$. This ILP has Kmn real variables, mn Boolean variables, $(K+1)n$ equality constraints, and Kmn inequality constraints. This means that instances with large state and action spaces are computation-heavy and may be intractable for analysis even with modern (M)ILP solvers on large machines [SAB+15].

Local optimization heuristics Knowing that none of the exact formulations of the stochastic multi-scenario optimization problem can be solved quickly, we turn to heuristics that promise adequate performance in exchange for a sacrifice in solution quality. In particular, we consider local optimization methods that iteratively improve a policy until no further simple, local improvement is possible. As the problem we face is non-unimodal in the general case, this does not guarantee an optimal policy. Later we empirically evaluate how big our sacrifice in precision here is.

The local optimization methods are based on the mathematical program (3.5). First, we consider a policy matrix $\boldsymbol{\Pi}$ and what happens if we apply local modifications to it. Let $s \in S$ be a state, $a, a' \in A$ be actions, and $\lambda \in [0, \boldsymbol{\Pi}(s,a)]$. Let furthermore $\boldsymbol{\Pi}'$ result from $\boldsymbol{\Pi}$ by setting $\boldsymbol{\Pi}'(s,a)$ to zero and adding $\boldsymbol{\Pi}(s,a)$ to $\boldsymbol{\Pi}(s,a')$ with

$$
\boldsymbol{\Pi}'(t,b) = \begin{cases} 0 & t = s \wedge b = a \\ \boldsymbol{\Pi}(s,a) + \boldsymbol{\Pi}(s,a') & s = t \wedge b = a' \\ \boldsymbol{\Pi}(t,b) & \text{otherwise} \end{cases}
$$

In analogy to our observations in the derivation of the gradient of f in (3.6), the change from $\boldsymbol{\Pi}$ to $\boldsymbol{\Pi}'$ performs a rank-one update to the matrices we invert in the mathematical program, and we can again use the formula from [Hag89]. The difference now is that we not only change $\boldsymbol{\Pi}(s,a)$, but also simultaneously change $\boldsymbol{\Pi}(s,a')$; so, in the computation of the resulting value of f we have to adjust the formulas we have derived in (3.6) accordingly.

Let $\vec{u}_k = \gamma(\boldsymbol{P}_k^{a'}(s\bullet) - \boldsymbol{P}_k^{a}(s\bullet))$, then we have, in analogy to (3.6),

$$\begin{aligned} f(\lambda\boldsymbol{\Pi}' + (1-\lambda)\boldsymbol{\Pi}) &= \sum_{k=1}^{K} \vec{w}(k)\vec{q}^{\top}(\boldsymbol{I} - \gamma\boldsymbol{P}_k)^{-1}\vec{r}_k \\ &= \sum_{k=1}^{K} \vec{w}(k)\vec{q}^{\top}\boldsymbol{C}_k^{\lambda\boldsymbol{\Pi}'+(1-\lambda)\boldsymbol{\Pi}}\vec{r}_k \\ &= f(\boldsymbol{\Pi}) - \sum_{k=1}^{K} \lambda \frac{\vec{w}(k)\vec{q}^{\top}\boldsymbol{C}_k^{\boldsymbol{\Pi}}(\bullet s)\vec{u}_k\boldsymbol{C}_k^{\boldsymbol{\Pi}}\vec{r}_k}{1 + \lambda\vec{u}_k\boldsymbol{C}_k^{\boldsymbol{\Pi}}(\bullet s)} \end{aligned} \tag{3.10}$$

We observe that λ is a free parameter in this term that can be locally optimized. Now let

$$\begin{aligned} \zeta_k &= \vec{w}(k)\vec{q}^{\top}\boldsymbol{C}_k^{\boldsymbol{\Pi}}(\bullet s)\vec{u}_k\boldsymbol{C}_k^{\boldsymbol{\Pi}}\vec{r}_k, \\ \eta_k &= \vec{u}_k\boldsymbol{C}_k^{\boldsymbol{\Pi}}(\bullet s) \end{aligned}$$

Furthermore let g_k be the difference in the k-th MDP. It is now

$$g_k(\lambda) = -\frac{\lambda\zeta_k}{1 + \lambda\eta_k}$$

and

$$f(\lambda\boldsymbol{\Pi}' + (1-\lambda)\boldsymbol{\Pi}) = f(\boldsymbol{\Pi}) + \sum_{k=1}^{K} g_k(\lambda)$$

To optimize the reward locally, it is now needed to optimize the function

$$g(\lambda) = \sum_{k=1}^{K} g_k(\lambda) \tag{3.11}$$

Its derivatives are

$$g_k'(\lambda) = \frac{-\zeta_k}{(1+\lambda\eta_k)^2} \text{ and } g_k''(\lambda) = \frac{-2\zeta_k\eta_k}{(1+\lambda\eta_k)^3}$$

Extreme values of (3.11) are found either at the endpoints, i. e., at $\lambda \in \{0, \boldsymbol{\Pi}(s,a)\}$ or at the points where g_k fulfill

$$\sum_{k=0}^{K} g_k'(\lambda) = 0 \text{ and } \sum_{k=0}^{K} g_k''(\lambda) < 0$$

As finding the roots of the first derivative implies finding roots of the term

$$\sum_{k=1}^{K} \zeta_k \prod_{k' \neq k} (1 + \lambda\eta_k)^2, \tag{3.12}$$

which is a polynomial of degree $2K-2$, the roots in question can be computed efficiently, for example, with the Jenkins-Traub algorithm [JT70, PTVF07] for $K \leqslant 20$ [Goe94] or other numerical methods for greater values of K [KVY11].

Suppose now that λ^* is the extremal point of (3.11). Then, the new policy matrix $\boldsymbol{\Pi}'$ can be computed from $\boldsymbol{\Pi}$ by changing $\boldsymbol{\Pi}(s,a)$ by $-\lambda^*$ and $\boldsymbol{\Pi}(s,a')$ by λ^*. The resulting value of the objective function is then given by $f(\boldsymbol{\Pi}) + g(\lambda^*)$.

Considering local optimality, let $\boldsymbol{\Pi}$ and $\boldsymbol{\Pi}'$ be given such that $\boldsymbol{\Pi}(s'\bullet) = \boldsymbol{\Pi}'(s'\bullet)$ for all $s' \neq s$. Then the differences $\gamma(\boldsymbol{P}_k^{\boldsymbol{\Pi}'}(s\bullet) - \boldsymbol{P}_k^{\boldsymbol{\Pi}}(s\bullet))$ between the matrices to be inverted can be expressed by $\sum_{i\in I} \lambda_i\vec{u}_{k,i}$ where each $\vec{u}_{k,i}$ has the form $\vec{u}_{k,i} = \gamma(\boldsymbol{P}_k^{a_i}(s\bullet) - \boldsymbol{P}_k^{a_i'}(s\bullet))$ for some

$\lambda_i \in [0,1]$, $a_i, a_i' \in A$ and some index set $I, |I| \leqslant |A|$. As the matrix update is still rank-one, we have

$$f(\mathbf{\Pi}') = f(\mathbf{\Pi}) - \sum_{i \in I} \lambda_i \sum_{k=1}^{K} \frac{\vec{w}(k)\vec{q}^{\top} \boldsymbol{C}_k^{\mathbf{\Pi}}(\bullet s)\vec{u}_{k,i}\boldsymbol{C}_k^{\mathbf{\Pi}}\vec{r}_k}{1 + \lambda_i \vec{u}_{k,i}\boldsymbol{C}_k^{\mathbf{\Pi}}(\bullet s)}$$

which implies that at least one of the inner sums has to be positive, if $f(\mathbf{\Pi}') > f(\mathbf{\Pi})$. Conversely, if for some state $s \in S$ no pair of actions a, a' exists where $\mathbf{\Pi}(s,a) > 0$ and (3.10) yields a better objective function value, then the decision in state s is optimal.

Together, this results in Algorithm 13 that searches for locally optimal stationary policies. The algorithm looks for states where an action pair can be found such that $g(\lambda^*) > 0$ is observed and loops until no improvement can be made.

Algorithm 13 Local optimization heuristic for concurrent MDPs

```
1: function CPolicyOpt($\vec{w}, \vec{q}, ((\boldsymbol{P}_k^a)_{a \in A}, \vec{r}_k)_{k \in \{1,\dots,K\}}$)
2:     $\mathbf{\Pi} \leftarrow \mathbf{0} \in \mathbb{R}^{n \times m}$
3:     $\mathbf{\Pi}(\bullet 1) = \vec{1}$
4:     Compute $\boldsymbol{C}_1^{\mathbf{\Pi}}, \dots, \boldsymbol{C}_K^{\mathbf{\Pi}}$
5:     repeat
6:         for $s \in S$ do
7:             for $a \in \{a \in A \mid \mathbf{\Pi}(s,a) > 0\}$ do
8:                 for $a' \in A, a' \neq a$ do
9:                     $\lambda^* = \arg\max_{\lambda} g(\lambda)$
10:                    if $\lambda^* > 0$ then
11:                        $\mathbf{\Pi}(s,a) \leftarrow \mathbf{\Pi}(s,a) - \lambda^*$
12:                        $\mathbf{\Pi}(s,a') \leftarrow \mathbf{\Pi}(s,a') + \lambda^*$
13:                        Update $\boldsymbol{C}_1^{\mathbf{\Pi}}, \dots, \boldsymbol{C}_K^{\mathbf{\Pi}}$ with (3.10)
14:                        Break
15:    until $\mathbf{\Pi}$ does not change
16:    return $\mathbf{\Pi}$
```

If only pure policies have to be computed, a variant of Alg. 13 can be used that is provided as Algorithm 14. It considers only pure policies and, hence, only improvements with $\lambda = 1$.

Algorithm 14 Local pure policy optimization heuristic for concurrent MDPs

```
1: function CPurePolicyOpt($\vec{w}, \vec{q}, ((\boldsymbol{P}_k^a)_{a \in A}, \vec{r}_k)_{k \in \{1,\dots,K\}}$)
2:     $\mathbf{\Pi} \leftarrow \frac{1}{m}\mathbf{1} \in \mathbb{R}^{n \times m}$
3:     Compute $\boldsymbol{C}_1^{\mathbf{\Pi}}, \dots, \boldsymbol{C}_K^{\mathbf{\Pi}}$
4:     repeat
5:         for $s \in S$ do
6:             Let $a \in A$ be such that $\mathbf{\Pi}(s,a) = 1$
7:             for $a' \in A, a' \neq a$ do
8:                 Evaluate $g(1)$ with (3.10)
9:                 if $g(1) > 0$ then
10:                    $\mathbf{\Pi}(s,a) \leftarrow 0$
11:                    $\mathbf{\Pi}(s,a') \leftarrow \mathbf{\Pi}(s,a') + \lambda$
12:                    Update $\boldsymbol{C}_1^{\mathbf{\Pi}}, \dots, \boldsymbol{C}_K^{\mathbf{\Pi}}$ with (3.10)
13:                    Break
14:    until $\mathbf{\Pi}$ does not change
15:    return $\mathbf{\Pi}$
```

Evaluation The algorithms are implemented in C using several optimization libraries. For mixed-integer programming, CPLEX [IBM15] is used. The quadratically constrained linear program is, as it has been mentioned, not convex, which implies that generic non-linear routines are needed. Using the COBYLA [Pow94] method from the NLOpt [Joh14] library delivers here the most promising results. For the derivative-based non-linear optimization formulation, the code uses the Ipopt [Wä02] with the HSL [hsl17] library for low-level optimization routines. The local search heuristics are written from scratch and use linear algebra and polynomial solving primitives from the GNU scientific library (GSL) [Con16] and the Intel MKL [int17] implementation of basic linear algebra subroutines (BLAS). The roots of the polynomial (3.12) are computed with the standard formulas if the degree of the polynomial is smaller than 3. Further code optimization include caching of the $\vec{u}_k$ vectors and the values of η and ζ from (3.10) to avoid unnecessary re-computation of already known values. The code base for the algorithms can be found at [SB17], the test case generator and data analysis routines can be found at [Sch17a]. The results are generated with the collider [Sch17b] tool. All results are produced on a machine with two 10-core Intel Xeon E5-2690 v2 CPUs running at 3.00GHz clock rate and 126GB of RAM; the total time needed for producing the results is around two weeks.

The evaluation describes the behaviour of algorithms on two distributions of instances. One set of instances is generated by taking random dense stochastic matrices as probability distributions and uniformly distributed reward vectors. The second set of test cases is generated by creating random deterministic MDPs, that is, transition matrices with random unit row vectors. Furthermore, different discount factors $\gamma \in \{0.9, 0.999\}$ are considered.

In total, 30 runs for each parameter set have been performed. Every algorithm had a time budget of 5000s and, in the case of iterative solvers (that is, for the heuristics and NLP and QCLP formulations), 10 000 function evaluations to complete; if the algorithm did not converge and failed to produce any result, the run was flagged as erroneous. The results can be observed in tables A.1–A.4 in Appendix A. In the tables, t and σ_t denote, respectively, the average and the standard deviation of non-erroneous runs' runtimes, *err* denotes the absolute number of instances with failures (which were only observed during the runs of the QCLP solver), and *diff* denotes the average relative difference to the optimal solution across the runs. A visualization of the results is available in Figure 3.1–Figure 3.4. In these plots, the performance of the heuristic approaches is compared to the one of the respective exact algorithms (MIP and NLP). The dots correspond to the averages, the dashed lines correspond to the interpolated performance, the error bars depict measured standard deviations.

There are several observations to be made with these results. First, the gradient-based non-linear program and CPLEX deliver the best policies. Additionally, the non-linear solver often converges quickly, but the convergence time varies greatly across launches. The gradient-free QCLP formulation, in contrast, seems to be ill-suited for the task as the solver does not converge in many cases[1]. The local optimization heuristic in Alg. 13 is generally faster than the non-linear solver, but slows down on larger action spaces; the mean deviation of the heuristic solution from the optimal solution is very small and lies almost always under 2%, often even under 1%.

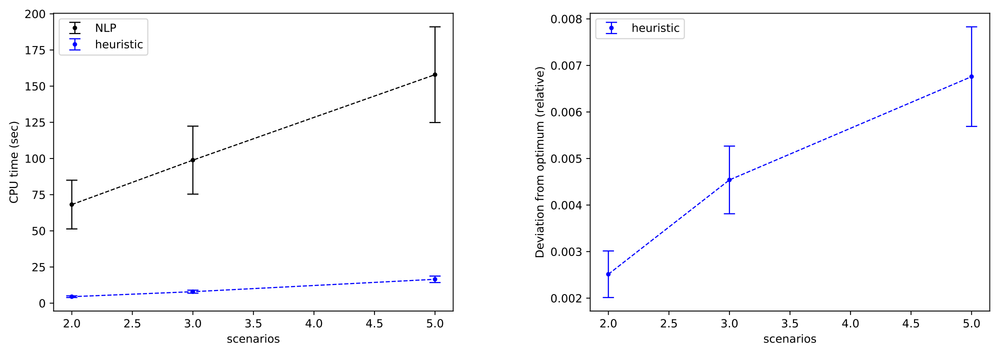
A similar picture arises with pure policies. Here, all algorithms converge, but CPLEX takes significantly more time, using the allowed time budget on models with more than 20 states with only an insignificant improvement over the local heuristic. The optimization logs show that on these instances, CPLEX arrives at a reasonably good solution very fast and takes a long time closing a very small (around 1%) gap between the upper and the lower bound. Again, the local search heuristic takes more time with larger action spaces.

Comparing the performance on different discount factors, we do not observe a major difference. Another picture occurs when comparing the performance on deterministic and

[1]This behaviour was responsible for the rather long time for evaluation.

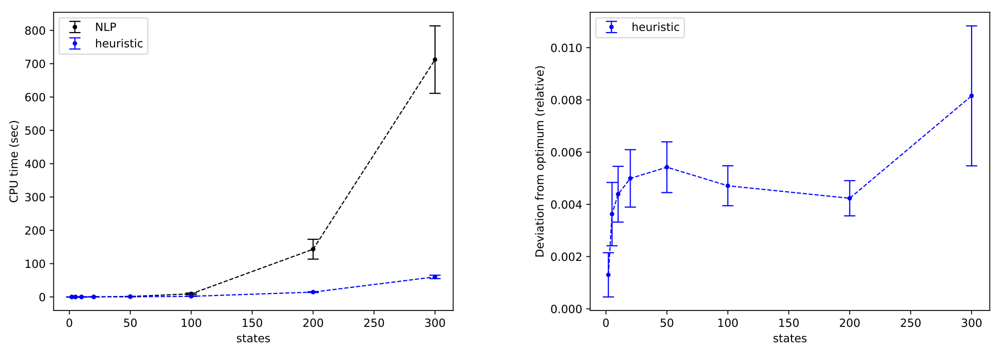
stochastic CMDP models: One can see that deterministic models take more time to be optimized, and the heuristics sacrifice more in terms of solution quality.

In all cases, the heuristics seem well-suited to be used, especially on larger state and action spaces. When comparing the two heuristics, a further observation can be made. As one can see, the pure policy optimization heuristic has a slightly poorer solution quality, but is around four times faster. The reason lies in a simplified test for possible improvement and less iterations until an improvement is actually found. This means that for even larger state and action spaces, using the heuristic from Algorithm 14 may be preferential if a slight sacrifice (of about 1%) in the quality of the resulting policy can be made.



(a) CPU time in dependence of scenarios for stationary policies (b) Relative deviation from the optimum in dependence of scenarios for stationary policies

Figure 3.1: Performance of the heuristic for stationary policies as a function of scenarios

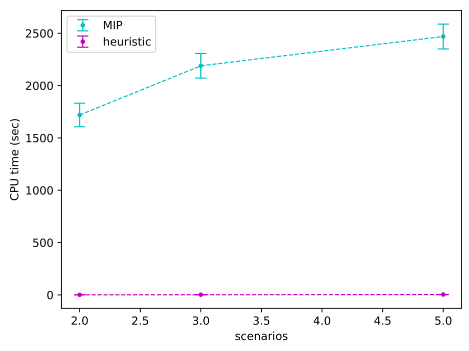


(a) CPU time in dependence of states for stationary policies (b) Relative deviation from the optimum in dependence of states for stationary policies

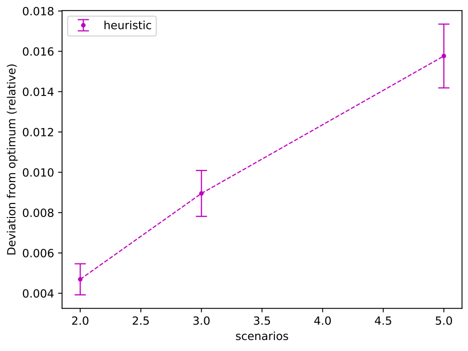
Figure 3.2: Performance of the heuristic for stationary policies as a function of states

3.2 Pareto frontier enumeration

Now we consider the enumeration problem for the set of non-dominated policies for the multi-scenario stochastic MDPs. This section is based on the publication [SBHH17] and extends it with some minor remarks.

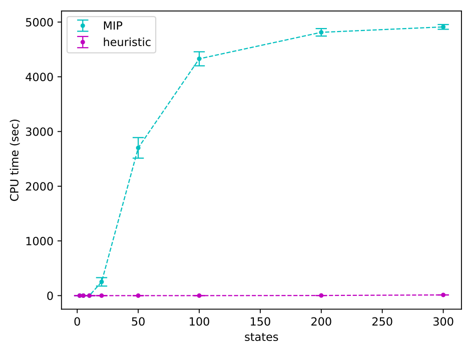


(a) CPU time in dependence of scenarios for pure policies

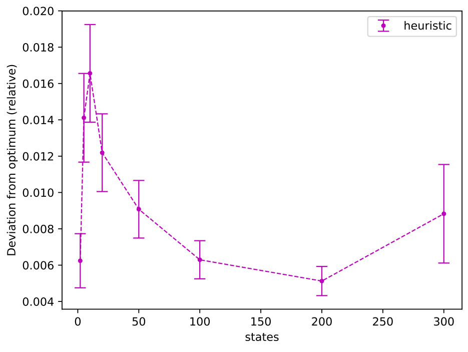


(b) Relative deviation from the optimum in dependence of scenarios for pure policies

Figure 3.3: Performance of the heuristic for pure policies as a function of scenarios



(a) CPU time in dependence of states for pure policies



(b) Relative deviation from the optimum in dependence of states for pure policies

Figure 3.4: Performance of the heuristic for pure policies as a function of states

We begin our discussion with noting that the computation of all Pareto optimal policies in multi-objective MDPs, as given in Eq. 2.10, is a computational and algorithmic challenge. Since the number of optimal policies can be large or even infinite (if non-stationary policies are considered), one can usually not expect to compute the whole set of Pareto optimal policies. Here, we devise algorithms which heuristically compute sets of non-dominated policies which are, in turn, likely to belong to the Pareto frontier, efficiently.

Currently, there are several approaches in literature to solve this problem for multi-objective MDPs, which are MDPs with several reward vectors but one fixed transition probability distribution function. Hence, the approaches in [PW10, BN08, CMH06, Whi82, WdJ07, RWO14] consider an MDP setting with multi-objective rewards. In the SBMDP setting and for the enumeration problem, we have to face optimization of, ultimately, several MDPs with related but not identical transition probabilities; furthermore, the MDPs for which the optimization has to take place also have to be computed separately, as they depend on the chosen policy, which makes the general problem harder to solve.

Here, we limit ourselves to only pure policies. However, a problem we are faced with is that even considering only pure policies might yield an exponential runtime. Even the most efficient algorithm that enumerates all pure Pareto optimal policies has to compute them all, and this numer is bounded only by $\mathcal{O}\left(|A|^{|S|}\right)$. We suspect that for most non-trivial SBMDP models, the number of Pareto optimal policies is still much too large to compute them all; however, we expect that many policies show similar behaviour and considering all of them is not necessary for practical purposes.

3.2.1 The algorithm

From a practical point of view it should be sufficient to compute a subset of the Pareto optimal policies if the corresponding value vectors are equally distributed over the Pareto frontier. Equally distributed means here that the nearest-neighbour distances between the corresponding value vectors are similar.

A valid initial approach is here to consider value and policy iteration and base the algorithms on these ideas. In literature, this approach has been undertaken by [WdJ07] where all value vectors from $\mathcal{V}_{\text{Pareto}}$ have been computed. However, even disregarding theoretical correctness, value iteration has the major disadvantage that the number of intermediate value vectors can become prohibitively large even before the policies will be completely evaluated. It is possible to stop the value iteration at some point when one believes that the Pareto frontier is approximated sufficiently well, but this raises several questions. First, the resulting policies themselves have to be fully evaluated and their value vectors may significantly differ from the intermediate values. Second, equal policies with differing value vectors have to be treated adequately. Third, it is hard to get a guarantee that the resulting policies in value vector space will be approximated sufficiently well by the current policy set and, furthermore, the approximation bounds for the true Pareto frontier have to be proven separately. Finally, the policies generated in such a way may not be pure. Hence, we consider an approach based on policy iteration similar to the one described in Algorithm 11.

The main disadvantage of Algorithm 11 is its runtime complexity. In the (algorithmic) worst case, the algorithm will produce large numbers of temporarily optimal policies that will be dominated by a few Pareto-optimal policies in the end, making the worst-case complexity $\mathcal{O}\left(|A|^{|S|}\right)$, which is then theoretically independent of the final size of $\mathcal{P}_{\text{Pareto}}$. A further problem is that even if the number of Pareto optimal policies is this large, $\mathcal{P}_{\text{Pareto}}$ may have Hamming distance $\frac{|S|}{2}$ from the initial policy, which implies exponential runtime even before the first Pareto optimal policy is generated. To circumvent this, we propose a slightly different algorithm that computes a set of policies that seem to be a reasonably good approximation of $\mathcal{P}_{\text{Pareto}}$. It is important to note that the following approach is an heuristic; later, we discuss its empirical quality.

Solution approach Our heuristic is based on a simplification of Algorithm 11. The simplification lies in including only those policies into the non-dominated set that are not dominated by previously computed policies. This step radically decreases the number of candidate policies and decreases the runtime at the cost of possible imprecision. The main steps of the simplified method are outlined in Algorithm 15.

We briefly analyse the complexity of the proposed algorithm. In lines 4–6, $|P| = \mathcal{O}\left(|F||S||A|\right)$ policies are generated. Policy evaluation is possible in $\mathcal{O}(\frac{|S|^2}{-\log\gamma})$ time steps. The complexity of computing the resulting non-dominated set that will replace F is, with efficient data structures to compute the Pareto frontier [ZTCJ15], $\mathcal{O}(\frac{|F||S|^3|A|}{-\log\gamma} + (|F||S||A|)^2)$. Thus, the total runtime can be bounded by $\mathcal{O}(R((I|S||A|)^2 + \frac{I|S|^3|A|}{-\log\gamma}))$, if R is the result set size and I is the size of the largest intermediate set of non-dominated solutions. This implies cubic complexity in the size of the largest intermediate set in the worst case. Heuristically, it seems also intuitive to assume that I cannot be much larger than the resulting set, i. e.,

Algorithm 15 A heuristic for $\mathcal{P}_{\text{Pareto}}$ and $\mathcal{V}_{\text{Pareto}}$

```
1: function PURE-OPT-HEURISTIC($\mathcal{P} = (S, A, T_\updownarrow, R_\updownarrow, Pr), \gamma$)
2:     $F \leftarrow \left\{\pi_\uparrow\right\}$                              ▷ initialize the Pareto frontier
3:     repeat
4:         for $\pi \in F$ do
5:             $P \leftarrow \{\pi' \mid d(\pi, \pi') = 1\}$                 ▷ consider all neighbours
6:             $F \leftarrow PO(F \cup P)$                                  ▷ keep only non-dominated in $F$
7:     until no new policies are added to $F$
8:     return $F$
```

I and R are coupled by the relation $I \leqslant cR$ where c is a small constant. This assumption reduces the runtime to

$$\mathcal{O}\left(R^3|S|^2|A|^2 + \frac{R^2|S|^3|A|}{-\log\gamma}\right)$$

which is roughly cubic in the resulting set size and quadratic in the size of state and action spaces for constant discount factors as it is $|S|, |A| = o(R)$.

An important feature of the algorithm is that it can be terminated when the non-dominated set F reaches a predefined size; at this point, the generated policies can be guaranteed to be mutually non-dominated. In our implementation, we keep this feature, but also introduce additional sophistication for performance reasons as well as to yield a more evenly distributed non-dominated set.

Implementation The practical implementation includes two additional improvements over Algorithm 15. First, the set of the initial solutions contains not arbitrary policies, but the policies which are optimal for the pessimistic, the optimistic and the average cases. It is easy to see that these policies definitively belong to $\mathcal{P}_{\text{Pareto}}$. We denote by $(\pi, \vec{v}_\downarrow^{(\pi)}, \vec{v}_\times^{(\pi)}, \vec{v}_\uparrow^{(\pi)})$ the tuple containing the policy π and the corresponding value vectors. Let $(\pi_\downarrow, \vec{v}_\downarrow^{(\pi_\downarrow)}, \vec{v}_\times^{(\pi_\downarrow)}, \vec{v}_\uparrow^{(\pi_\downarrow)})$, $(\pi_\times, \vec{v}_\downarrow^{(\pi_\times)}, \vec{v}_\times^{(\pi_\times)}, \vec{v}_\uparrow^{(\pi_\times)})$, and $(\pi_\uparrow, \vec{v}_\downarrow^{(\pi_\uparrow)}, \vec{v}_\times^{(\pi_\uparrow)}, \vec{v}_\uparrow^{(\pi_\uparrow)})$ be the policies and value vectors resulting from the optimization of the lower bound, average case and upper bound of the discounted reward, respectively. It is known that these policies are in $\mathcal{P}_{\text{Pareto}}$. Starting with them makes the algorithm walk through the policy space from the extreme points of the Pareto frontier, which, as we hope, yields an evenly distributed (in value vector space) non-dominated set of policies.

Second, we improve on the policy evaluation step to classify candidate policies in order to evade additional policy evaluations. Here, our idea is the following observation. We define first for a policy π and a state-action pair (s, a) the gradient with

$$\begin{aligned}\operatorname{grad}(\pi, s, a) = \Bigg(&\vec{r}(s) + \gamma \min_{\vec{p} \in S(\boldsymbol{P}_\downarrow^a(s\bullet), \boldsymbol{P}_\uparrow^a(s\bullet))} \vec{p}\vec{v}_\downarrow^{(\pi)} - \vec{v}_\downarrow^{(\pi)}(s), \\ &\vec{r}(s) + \gamma \boldsymbol{P}_\times^a(s\bullet)\vec{v}_\times^{(\pi)} - \vec{v}_\times^{(\pi)}(s), \vec{r}(s) + \gamma \max_{\vec{p} \in S(\boldsymbol{P}_\downarrow^a(s\bullet), \boldsymbol{P}_\uparrow^a(s\bullet))} \vec{p}\vec{v}_\uparrow^{(\pi)} - \vec{v}_\uparrow^{(\pi)}(s)\Bigg)\end{aligned} \tag{3.13}$$

where $S(\vec{p}_1, \vec{p}_2)$ is the set of stochastic row vectors between $\vec{p}_1$ and $\vec{p}_2$ defined by

$$S(\vec{p}_1, \vec{p}_2) = \left\{\vec{p} \in \mathbb{R}^{1\times n} \mid \vec{p}\vec{1} = 1 \wedge \vec{p}_1 \leqslant \vec{p} \leqslant \vec{p}_2\right\}$$

If, for some policy π, a state s and an action $a \in A\backslash\{\pi(s)\}$ can be found such that

$$\operatorname{grad}(\pi, s, a) >_P (0, 0, 0) \tag{3.14}$$

then a policy $\pi^{(s,a)}$ can be defined with $\pi^{(s,a)}(t) = \pi(t)$ for $t \neq s$ and $\pi^{(s,a)}(s) = a$ and it is $\left(\vec{v}_{\downarrow}^{(\pi^{(s,a)})}, \vec{v}_{\times}^{(\pi^{(s,a)})}, \vec{v}_{\uparrow}^{(\pi^{(s,a)})}\right) >_P \left(\vec{v}_{\downarrow}^{(\pi)}, \vec{v}_{\times}^{(\pi)}, \vec{v}_{\uparrow}^{(\pi)}\right)$. We define an operator

$$\pi' = \mathrm{popt}\left(\pi, \vec{v}_{\downarrow}^{(\pi)}, \vec{v}_{\times}^{(\pi)}, \vec{v}_{\uparrow}^{(\pi)}\right)$$

that generates a new policy from π by selecting for each s an action a for which the relation (3.14) holds, whenever this is possible and keeping π otherwise.

The function popt considers only pairs (s, a) where the gradient is non-negative and non-zero. If the gradient contains no positive elements, then the corresponding policy $\pi^{(s,a)}$ is dominated by π. If the gradient is non-negative and non-zero, then $\pi^{(s,a)}$ dominates π. In all other cases, some components of the value vectors are improved over their predecessors and other will become worse in comparison to the value vectors associated with π.

From the gradient, the direction of the different value vectors of a new policy can be estimated without evaluating it fully. Policy evaluation is performed by a function eval which solves the following three sets of equations to compute the value vectors for some pure policy π.

$$\begin{aligned}
\vec{v}_{\downarrow}^{(\pi)} &= \vec{r} + \gamma \min_{\boldsymbol{P} \in \boldsymbol{P}_{\updownarrow}^{(\pi)}} \left(\boldsymbol{P}\vec{v}_{\downarrow}^{(\pi)}\right), \\
\vec{v}_{\times}^{(\pi)} &= \vec{r} + \gamma \boldsymbol{P}_{\times}^{(\pi)} \vec{v}_{\times}^{(\pi)}, \\
\vec{v}_{\uparrow}^{(\pi)} &= \vec{r} + \gamma \max_{\boldsymbol{P} \in \boldsymbol{P}_{\updownarrow}^{(\pi)}} \left(\boldsymbol{P}\vec{v}_{\uparrow}^{(\pi)}\right)
\end{aligned} \tag{3.15}$$

The equations for the average values are the standard MDP equations and define a set of linear equations which can be solved with standard means. For the vector of the pessimistic and optimistic values, a fixed-point iteration is performed. First, the vector $\vec{v}_{\downarrow}^{(\pi)}$ (resp. $\vec{v}_{\uparrow}^{(\pi)}$) is initialized with an arbitrary value, then, the minimum (or maximum) is computed and with this minimum (or maximum), a new vector is computed which is then used to find a new minimum (or maximum). This procedure defines a sequence of value vectors that converges to a unique fixed point, as follows from the interval value iteration procedure described in Algorithm 9 in [GLD00]. In fact, this is an application of the interval value iteration algorithm for an action space with one element.

In the following procedure, given in Algorithm 16 we use a set PV that contains tuples $(\pi, \vec{v}_{\downarrow}^{(\pi)}, \vec{v}_{\times}^{(\pi)}, \vec{v}_{\uparrow}^{(\pi)})$. Additionally, we use a set P where all evaluated policies are stored in order to avoid a re-evaluation of a policy.

The algorithm itself is an optimized version of the policy iteration approach in Algorithm 15 to heuristically compute $\mathcal{P}_{\text{Pareto}}$ and the corresponding value vectors. In the current description, new policies are generated starting from available policies by maximizing one direction of the gradient; one can also think of other, more sophisticated strategies to derive promising policies. The algorithm stops if in the current set of non-dominated policies, the neighbours of each policy are either explored or dominated. A further stopping condition is if a predefined number of policies is explored, that is, if $|PV|$ surpasses a given threshold.

As the algorithm is a heuristic, it is difficult to argue about guaranteed performance in terms of quality of the output, that is, if the resulting policies $P_r = \{\pi \mid (\pi, \cdot, \cdot, \cdot) \in PV\}$ fulfill $\mathcal{P}_{\text{Pareto}} \subseteq P_r$ and $P_r \subseteq \mathcal{P}_{\text{Pareto}}$. It is still possible to make several observations. The main difference between Algorithm 16 and Algorithm 11 lies in the bookkeeping that disallows Algorithm 16 to explore policies that are already evaluated, or, more importantly, are dominated by some other already explored policy. By doing this, Algorithm 16 may ignore policies that are dominated yet lead (by choosing an appropriate sequence of policy changes) to the Pareto frontier; if there are no other ways to the Pareto frontier, this makes the output of Algorithm 16 incomplete. On the other hand, one can provide an heuristic

Algorithm 16 Policy iteration to heuristically compute $\mathcal{P}_{\text{Pareto}}$ and $\mathcal{V}_{\text{Pareto}}$

```
1:  function PURE-OPT-PRACTICAL-HEURISTIC($\mathcal{P} = (S, A, T_{\updownarrow}, R_{\updownarrow}, Pr), \gamma$)
2:      $PV \leftarrow \left\{(\pi_{\downarrow}, \vec{v}_{\downarrow}^{(\pi_{\downarrow})}, \vec{v}_{\times}^{(\pi_{\downarrow})}, \vec{v}_{\uparrow}^{(\pi_{\downarrow})}, (\pi_{\times}, \vec{v}_{\downarrow}^{(\pi_{\times})}, \vec{v}_{\times}^{(\pi_{\times})}, \vec{v}_{\uparrow}^{(\pi_{\times})}), (\pi_{\uparrow}, \vec{v}_{\downarrow}^{(\pi_{\uparrow})}, \vec{v}_{\times}^{(\pi_{\uparrow})}, \vec{v}_{\uparrow}^{(\pi_{\uparrow})})\right\}$
3:      $P \leftarrow \left\{\pi_{\downarrow}, \pi_{\times}, \pi_{\uparrow}\right\}$
4:      while $|PV| <$ max. number of policies do
5:          $\Pi = \varnothing$
6:          for $\pi \in PV$ and all $(s, a)$ where $\pi^{(s,a)} \notin P$ do
7:              $\left(g_{\downarrow}^{(\pi,s,a)}, g_{\times}^{(\pi,s,a)}, g_{\uparrow}^{(\pi,s,a)}\right) = \text{grad}(\pi, s, a)$
8:              $\Pi = \Pi \cup \{(\pi, s, a)\}$
9:          if no non-negative gradient exists for $(\pi, s, a) \in \Pi$ then
10:             break (all Pareto optimal solutions have been found)
11:         for $g \in \left\{g_{\downarrow}, g_{\times}, g_{\uparrow}\right\}$ do
12:             if $g^{(\pi,s,a)} \nleq \vec{0}$ exists then
13:                 $(\pi, s, a) \leftarrow \arg\max_{(\pi,s,a)\in\Pi} \left\{g^{(\pi,s,a)}\right\}$
14:                 $\pi' \leftarrow \pi^{(s,a)}$
15:                 repeat
16:                     $(\vec{v}_{\downarrow}^{(\pi')}, \vec{v}_{\times}^{(\pi')}, \vec{v}_{\uparrow}^{(\pi')}) \leftarrow \text{eval}(\mathcal{P}, \pi')$
17:                     $\pi' \leftarrow \text{popt}\left(\pi', \vec{v}_{\downarrow}^{(\pi')}, \vec{v}_{\times}^{(\pi')}, \vec{v}_{\uparrow}^{(\pi')}\right)$
18:                 until $\pi'$ does not change
19:                 $PV \leftarrow PO\left(PV \cup \{(\pi', \vec{v}_{\downarrow}^{(\pi')}, \vec{v}_{\times}^{(\pi')}, \vec{v}_{\uparrow}^{(\pi')})\}\right)$ $P = P \cup \{\pi'\}$
20:         if $PV$ was not changed then
21:             break (all new policies are explored or dominated)
22:     return $PV$
```

argument: Since the initial set of policies contains known "extreme points" $\pi_{\downarrow}, \pi_{\times}, \pi_{\uparrow}$, the policies that are found by Alg. 16 in realistic settings will stem from a gradual transition from one extreme policy to another, as, following Lemma 2.4.5, there always exists a path of policies that improves one of the objectives until an optimum is reached. This way, we can expect that in real-life problems, the resulting set P_r will cover the Pareto frontier or at least the space between the value vectors

$$\left(\vec{v}_{\downarrow}^{(\pi_{\downarrow})}, \vec{v}_{\times}^{(\pi_{\downarrow})}, \vec{v}_{\uparrow}^{(\pi_{\downarrow})}\right), \left(\vec{v}_{\downarrow}^{(\pi_{\times})}, \vec{v}_{\times}^{(\pi_{\times})}, \vec{v}_{\uparrow}^{(\pi_{\times})}\right), \left(\vec{v}_{\downarrow}^{(\pi_{\uparrow})}, \vec{v}_{\times}^{(\pi_{\uparrow})}, \vec{v}_{\uparrow}^{(\pi_{\uparrow})}\right)$$

adequately, i. e., the resulting set of value vectors will be evenly distributed in the space between the extreme value vectors stemming from $\pi_{\downarrow}, \pi_{\times}, \pi_{\uparrow}$. Furthermore, we expect that for practical problems, the following assumption will hold: If for a set of policies P it is $P \subseteq \mathcal{P}_{\text{Pareto}}$ and P contains not all Pareto optimal policies, then there exists a policy $\pi \in P$ and a state-action pair $(s, a) \in S \times A$ such that $\pi^{(s,a)} \notin P$ and $\pi^{(s,a)}$ is not dominated by any other policy in P. This especially means that there always is a "globally" non-dominated path of policies from a set P of mutually non-dominating policies to a policy in $\mathcal{P}_{\text{Pareto}} \backslash P$ if $\mathcal{P}_{\text{Pareto}} \neq P$. We expect that this assumption holds for practical instances. Furthermore, we conjecture that our assumption is also true for the problem in general.

Concerning the general complexity of Algorithm 16, we note that, as this is a variation of Alg. 11, the same reasoning applies. However, the practical complexity should be lower than that of Alg. 11, as less SBMDP evaluations have to be performed.

3.2.2 Evaluation

We present a series of experiments where we consider several questions. First, we compare the performance of Algorithm 16 against a black-box multi-objective optimization method as reference. We choose *SPEA2* [ZLT01] as reference since it is a well-studied, simple black-box optimization algorithm with practical use [LMSM14]. Second, we evaluate the general performance of the algorithm with respect to problem size and the number of computed solutions. Third, we test our algorithm on a small example from the literature.

For the evaluation, we use a machine with an eight-core Intel Core i7-4790 CPU and 16 GB RAM. We set a time limit of 1000 s for *SPEA2* and a limit for Algorithm 16 of 50 000 checked policies. The archive size for *SPEA2* is set to 50 000. Concerning the implementations, we use OpenMP parallelization methods to use multiple CPU cores when possible. Furthermore, we use advanced numerical algorithms to evaluate (3.15). Specifically, for large instances, we substitute the direct LU solver [Ste94] by preconditioned GMRES [Saa93, Ste94] with an ILU0-preconditioner. The code and testing infrastructure are available at [SB17, Sch17a].

The SPEA2 algorithm In detail, SPEA2 works, as each black-box algorithm, with potential solution from some solution space I. The algorithm keeps two sets of candidate solutions: a *population* P and an *archive* A where the non-dominated solutions are stored. In each iteration of the optimization cycle, A is updated with non-dominated elements of P. Then, a selection step takes place in which first, all elements of $A \cup P$ are assigned a *fitness value* and then, the solutions with lower fitness values are chosen to generate new solutions by application of mutation and crossover operators[2]. The newly generated solutions are then the new population.

The distinctive feature of this algorithm is its approach to fitness evaluation: The fitness of an individual solution p depends on the *strength* of other solutions p' that cover, i. e. dominate or are equal to p. The strength itself is defined as the number of covered solutions; thus, the non-dominated solutions have maximal strength and minimal fitness values by definition, otherwise the ranking aims at picking more diverse solutions, i. e., solutions that are more evenly distributed in the objective value space. A formal description of the heuristic is given in Algorithm 17.

In our SPEA2 implementation for multi-objective SBMDP optimization, we use problem-specific mutation and crossover operators. As possible solutions are pure policies, and, ultimately, integer vectors, the operators can be defined in a straightforward fashion. Mutation affects a decision in one state with probability $1/n$, if n is the number of states in the MDP, and replaces the previous action in the policy with a uniformly randomly chosen one. Crossover takes two "parent" policies and chooses for the result an action from either of the original policies with probability $1/2$.

The Multi-Server Queue Model For the first case study, we choose a parameterizable model instances of which can be easily generated. Concretely, we consider a multi-server queue model where servers can be switched off to save energy and switched on if the load in the system increases. Such queues are abstract models for server farms [GHA10]. The goal of this model is to find a compromise between small response times and low energy consumption; the uncertainty lies in the model dynamics.

[2]Minimizing the fitness value (instead of intuitive maximizing) is due to traditions in optimization, where minimization problems are considered natural.

Algorithm 17 The SPEA2 evolutionary multiobjective optimization heuristic

```
1: function SPEA2($N_{\text{population}} \in \mathbb{N}, N_{\text{archive}} \in \mathbb{N}, T \in \mathbb{N}$)
2:     $P \leftarrow$ initial population of size $N_{\text{population}}$
3:     $A \leftarrow \varnothing$
4:     $t \leftarrow 0$
5:     $k \leftarrow \lfloor\sqrt{N_{\text{population}} + N_{\text{archive}}}\rfloor$
6:     $p \leftarrow$ selection pressure $> \frac{1}{2}$
7:     while true do
8:         Initialize working set $W \leftarrow P \cup A$
9:         Calculate for each $i \in W$ its strength $S(i) = \left|\{j \mid j \in W \wedge i \geqslant j\}\right|$ where $\geqslant$ is the dominance relation.
10:        Calculate for each $i \in W$ its raw fitness $R(i) = \sum_{j \in W, j \geqslant i} S(j)$
11:        Calculate for each $i \in W$ the distance $\sigma_k(i)$ to its $k$-th nearest neighbour in the objective space
12:        Calculate for each $i \in W$ its fitness $F(i) = R(i) + \frac{1}{\sigma_k(i)+2}$
13:        $A \leftarrow PO(W)$                ▷ Copy all non-dominated solutions into the archive
14:        Truncate $A$ to $N_{\text{archive}}$ items
15:        if $t \geqslant T$ or other termination criteria are met then
16:            return $A$
17:        $M \leftarrow \varnothing$                ▷ initialize mating pool
18:        while $|M| < N_{\text{population}}$ do
19:            $i, j \leftarrow$ random solutions from $A$
20:            $M \leftarrow M \cup \left\{\text{with probability } p \text{ the solution with lower fitness from } \{i, j\}\right\}$
21:        $P \leftarrow$ recombination and mutation on $M$
22:        $t \leftarrow t + 1$
```

We consider a system where customers arrive according to a Poisson process with rate λ and require an exponentially distributed service with mean μ^{-1}. As our algorithms are designed for discrete-time BMDPs, we apply uniformization as described in (1.18) and (1.20) in order to derive a discrete model where the probability of arrival of a customer in a time unit is p, the service probability is q and, thus λ and μ are multiples of p^{-1} resp. q^{-1}. The system has a capacity of m and contains c servers. Each server can be in one of three states *on*, *off* and *start*. A server can be switched off after the end of a service or if it is idle. A server that is switched off immediately changes its state from *on* to *off*. Servers in state *off* can be switched on which means that they change their state to *start*. The duration of the starting period is exponentially distributed with rate ν, then the server changes its state to *on* and is ready to serve customers. A state of the system can be described by (i, j, k, l) where $i \in [0, m]$ describes the number of customers, j, k, l include the number of servers in state *on*, *start* and *off*, respectively. Consequently, $j + k + l = c$ has to hold. The number of states equals $n = (m+1)(c+2)(c+1)/2$. The reward in the state (i, j, k, l) equals $(m-i)/(j\omega_1 + k\omega_2 + l\omega_3)$ where $\omega_1, \omega_2, \omega_3$ describe the energy consumption in the *on*, *start*, and *off* state.

The transition probabilities in this model are fixed for constant λ, μ, ν, m, c; in order to introduce uncertainty in the form of bounds, we set the upper bound $\boldsymbol{P}^a_{\uparrow}$ by adding random Gaussian noise with mean 0.01 and variance 0.005 to the "theoretical" transition probabilities in $\boldsymbol{P}^a_{\times}$ defined above. The lower bounds $\boldsymbol{P}^a_{\downarrow}$ is generated analogously by subtracting Gaussian noise, also with mean 0.01 and variance 0.005 from the transition probabilities. All bounds are guaranteed to lie between 0 and 1.

Comparison to a generic heuristic As multi-scenario optimization for (stochastic) BMDPs is a new problem with little research on the topic, we choose to compare our approach to a black-box heuristic for lack of other known approaches. More specifically, we compare our algorithm to SPEA2 as it is a well-studied evolutionary optimization algorithm that is specifically designed to compute non-dominated sets for multi-objective optimization problems.

Comparison metrics To quantify performance differences, we use the coverage metric that has been introduced in [ZT99]. This performance metric is designed to compare two output sets of (heuristic) multi-objective optimization algorithms on the same problem and computes the fraction of one output set that is covered (that is, is dominated by or equal to) by an element of the other output set. Concretely, for two sets of vectors X and Y, the coverage metric $C(X,Y)$ is defined by

$$C(X,Y) = \frac{\left|\{y \in Y \mid \exists x \in X : x \geqslant y\}\right|}{|Y|}. \tag{3.16}$$

$C(X,Y) = 1$ means that all points in Y are dominated by or equal to points in X whereas $C(X,Y) = 0$ means that no point in Y is covered by a point in X. It is worth noting that the coverage metric is asymmetric and in most cases complete information about the dominance relation between X and Y can be derived only from both $C(X,Y)$ and $C(Y,X)$.

Results The results of the comparison can be found in Figures 3.5 and 3.6. The first figure describes the coverage metric where the first argument is the policy set computed by Alg. 16, the second figure describes the converse. In the run, *SPEA2* always uses the time budget of 1000s while Alg. 16 never takes more than 330s.

The numeric results can be observed in Table 3.1. The number t denotes the test case number, T the time in seconds that Algorithm 16 has used. The times for *SPEA2* are not shown as the algorithm stops on the timeout condition. P_H and P_S are the sets of computed policies by our heuristic (Alg. 16) resp. by *SPEA2* (Alg. 17); $|P_H|$ and $|P_S|$ are then the respective numbers of policies computed by each algorithm. C denotes the coverage metric as defined in (3.16): $C(P_H, P_S)$ denotes of the coverage of P_H by P_S, and vice versa, $C(P_S, P_H)$ denotes the coverage of P_H by P_S.

Table 3.1: Performance comparison of *SPEA2* and Algorithm 16

m	c	$\lvert S\rvert$	t	T (Alg. 16)	$\lvert P_H\rvert$ (Alg. 16)	$\lvert P_S\rvert$ (SPEA2)	$C(P_H, P_S)$	$C(P_S, P_H)$
2	1	9	1	0.1	4	661	1.0	1.0
			2	0.03	4	831	1.0	1.0
			3	< 0.01	2	557	1.0	1.0
			4	< 0.01	8	1168	1.0	1.0
2	2	18	1	0.04	144	700	1.0	0.0
			2	0.02	90	273	1.0	0.0
			3	0.01	16	399	1.0	0.0
			4	0.06	192	235	1.0	0.0
2	3	30	1	3.16	2048	1385	1.0	0.0
			2	8.83	3456	1678	1.0	0.0
			3	27.9	6480	2304	1.0	0.0
			4	32.09	6912	2289	1.0	0.0
3	1	12	1	0.01	4	498	1.0	1.0
			2	< 0.01	8	562	1.0	1.0
			3	< 0.01	8	494	1.0	1.0
			4	< 0.01	8	663	1.0	1.0

m	c	$\|S\|$	t	T (Alg. 16)	$\|P_H\|$ (Alg. 16)	$\|P_S\|$ (SPEA2)	$C(P_H, P_S)$	$C(P_S, P_H)$
3	2	24	1	30.01	10368	1987	1.0	0.0
			2	1.21	1728	1586	1.0	0.0
			3	0.95	1536	1834	1.0	0.0
			4	0.57	1024	1227	1.0	0.0
3	3	40	1	134.25	38626	3904	1.0	0.0
			2	170.0	43446	1448	1.0	0.0
			3	163.02	43555	3407	1.0	0.0
			4	158.31	40326	2234	1.0	0.0
4	1	15	1	0.01	16	298	1.0	0.8125
			2	< 0.01	8	412	1.0	0.75
			3	< 0.01	4	839	1.0	0.5
			4	< 0.01	12	193	1.0	0.583
4	2	30	1	124.77	17281	1372	1.0	0.0
			2	15.94	6144	1680	1.0	0.0
			3	0.86	1152	2679	1.0	0.0
			4	0.18	256	1145	1.0	0.0
4	3	50	1	182.5	44040	3110	1.0	0.0
			2	202.11	43835	2873	1.0	0.0
			3	209.75	39181	1345	0.9993	0.0
			4	196.69	39348	2720	1.0	0.0
5	1	18	1	0.01	16	1466	1.0	0.25
			2	0.01	16	1138	1.0	0.3125
			3	0.01	32	1545	1.0	0.21875
			4	0.01	16	1375	1.0	0.0625
5	2	36	1	172.42	44852	1931	1.0	0.0
			2	37.11	8192	2310	1.0	0.0
			3	102.63	13824	1867	1.0	0.0
			4	128.32	38334	1970	0.9995	0.0
5	3	60	1	195.88	43520	3555	1.0	0.0
			2	233.39	41321	3594	1.0	0.0
			3	203.18	41957	2995	1.0	0.0
			4	186.82	36129	2493	1.0	0.0
6	1	21	1	0.02	64	605	1.0	0.09375
			2	0.01	32	977	1.0	0.03125
			3	0.01	64	1130	1.0	0.03125
			4	0.01	64	756	1.0	0.0625
6	2	42	1	193.48	43583	2307	0.9996	0.0
			2	197.19	46716	2687	1.0	0.0
			3	148.53	41420	1631	1.0	0.0
			4	254.92	18432	2657	1.0	0.0
6	3	70	1	355.77	40955	2563	1.0	0.0
			2	321.47	41570	2506	1.0	0.0
			3	330.95	41739	3310	1.0	0.0
			4	330.95	36618	3288	1.0	0.0

It is easy to see that Algorithm 16 almost always delivers a significantly better performance with respect to both time complexity as well as quality of computed policies. In detail, in almost all test instances Algorithm 16 computes a set of policies that completely covers all solutions generated by *SPEA2*; the evolutionary heuristic, however, never produces a policy that strictly dominates a policy from Alg. 16 and is only able to yield comparable solutions on small instances with state space size of at most 20.

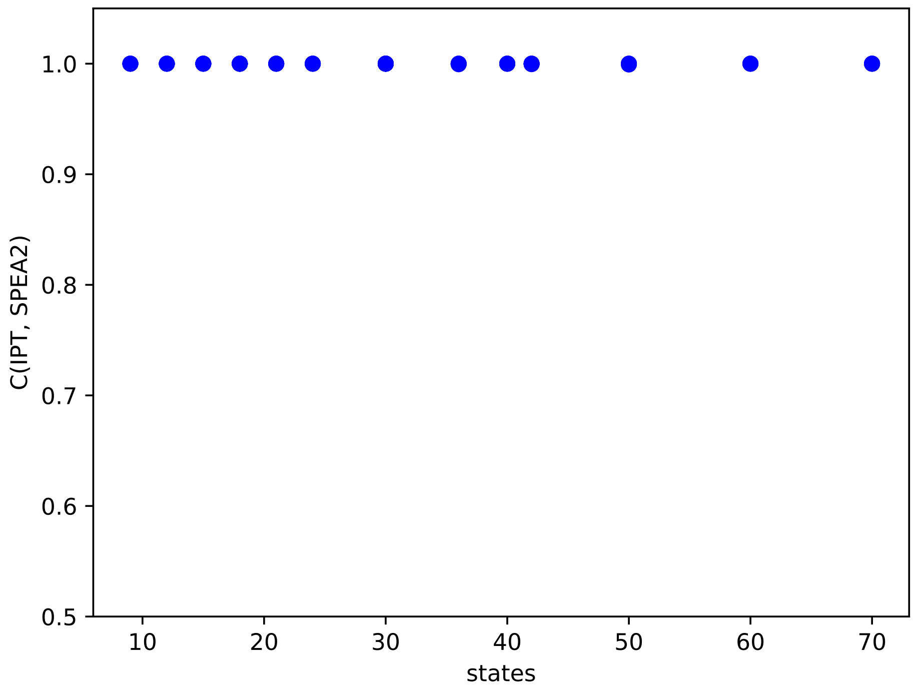


Figure 3.5: $C(\text{heuristic}, \text{evolutionary})$ in dependence of state space size

Comparison to an exact computation For some instances, we furthermore compare the performance of Algorithm 16 to the exact approach in Algorithm 11. Concretely, we consider the case $m = 2, c = 3$. It turns out that for this case, the coverage metric is always 1. This suggests that Algorithm 16 may compute the complete Pareto frontier not only heuristically but also in theory. This is, however, a conjecture subject to further investigation. Formally, the conjecture can be stated as follows.

Conjecture 3.2.1. Let $P \subset \mathcal{P}_{\text{pure}}$. If for every $\pi \in P$ and every $\pi' \in \mathcal{P}_{\text{pure}}$ with $d(\pi', \pi) = 1$ holds that

$$\vec{v}_{\downarrow}^{(\pi')} \leqslant \vec{v}_{\downarrow}^{(\pi^*)}, \vec{v}_{\times}^{(\pi')} \leqslant \vec{v}_{\times}^{(\pi^*)}, \vec{v}_{\uparrow}^{(\pi')} \leqslant \vec{v}_{\uparrow}^{(\pi^*)}$$

for some other $\pi^* \in P$, then $P = \mathcal{P}_{\text{Pareto}}$.

Time complexity As the number of policies is bounded by an upper limit of 50 000 and $|A| = o(|S|)$, the complexity can be roughly estimated by a cubic term in $|S|$. For practical applications, we are also interested in runtimes on real-life instances. To get an impression, we estimate the complexity by considering a number of test cases in a different, but more general and scalable model.

Grid model We consider here a model that resembles a grid with $n \cdot m$ states $S = \left\{ s_{i,j} \mid i \in [n], j \in [m] \right\}$ and m actions $A = [m]$. The rewards for actions in each state are normally distributed with mean 100 and variance 20. The transition probabilities are also chosen randomly according to the Dirichlet distribution. Concretely, the transition probability vector from state $s_{i,j}$ to states $s_{\min(n,i+1),j'}$ for action a is Dirichlet-distributed

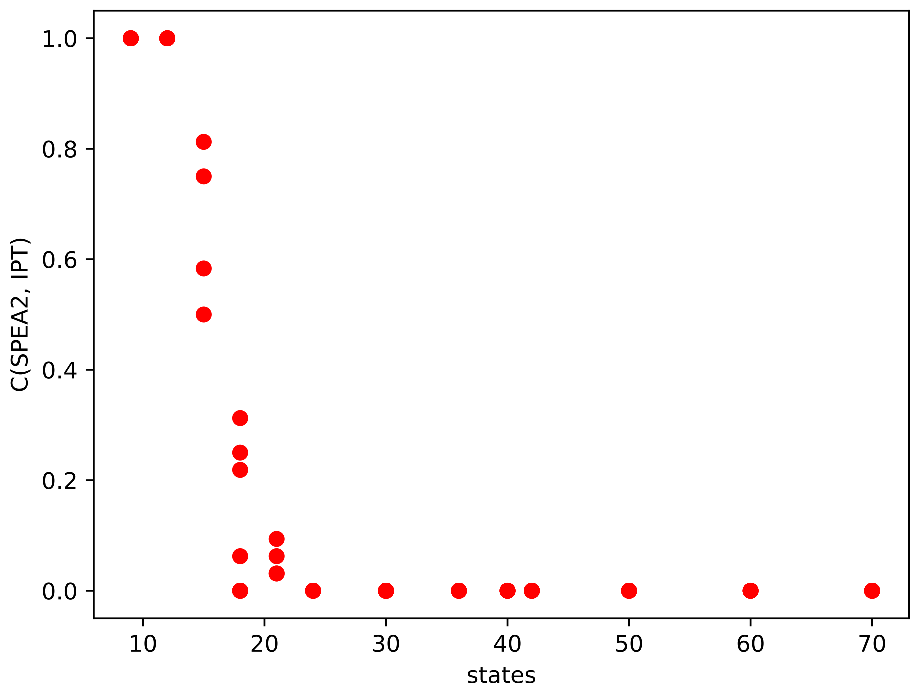


Figure 3.6: $C(\text{evolutionary}, \text{heuristic})$ in dependence of state space size

with concentration parameters $(\alpha_1^a, \ldots, \alpha_m^a)$ where $\alpha_{j'}^a = 10$ if $a = j'$ and $\alpha_{j'}^a = 1$ otherwise which yields an (expected) 10 times larger probability to land in $s_{i+1,a}$ than in other states. The upper and lower bounds are, as before, generated by adding and subtracting Gaussian noise.

Complexity We present the results in graphical form. The results themselves stem from runs of Algorithm 16 on instances of the grid model with up to 400 states, with n and m between 5 and 20. For each pair of values (n, m), we create 4 instances to achieve a more representative data set. Our algorithm stops when either no non-dominated policy can be created or when the limit of 50 000 evaluated policies is reached.

The results can be seen in Fig. 3.7. The red dots are the empirical data, the blue bars describe the mean along with a confidence interval that stems from a (scaled) t-distribution guess. The green line is the cubic regression term for convenience. The runtime complexity is polynomial, as the total number of policies that can be evaluated is bounded from above. The more interesting details in Fig. 3.7 are the values: One can see that even on large instances, the mean time until a non-dominated solution is generated lies under a second. This means that it is possible to incrementally compute the Pareto frontier, stopping the computation when the controller believes that the current set of policies is acceptable.

3.2.3 The Model of Autonomous Non-deterministic Tour Guides

Our second case study is inspired by "Autonomous Nondeterministic Tour Guides" (ANTG) in [CRI07, HHS16]. Models in [CRI07] are MDPs. In our experiment, we insert some uncertainties into the original MDP which integrate into the problem setting.

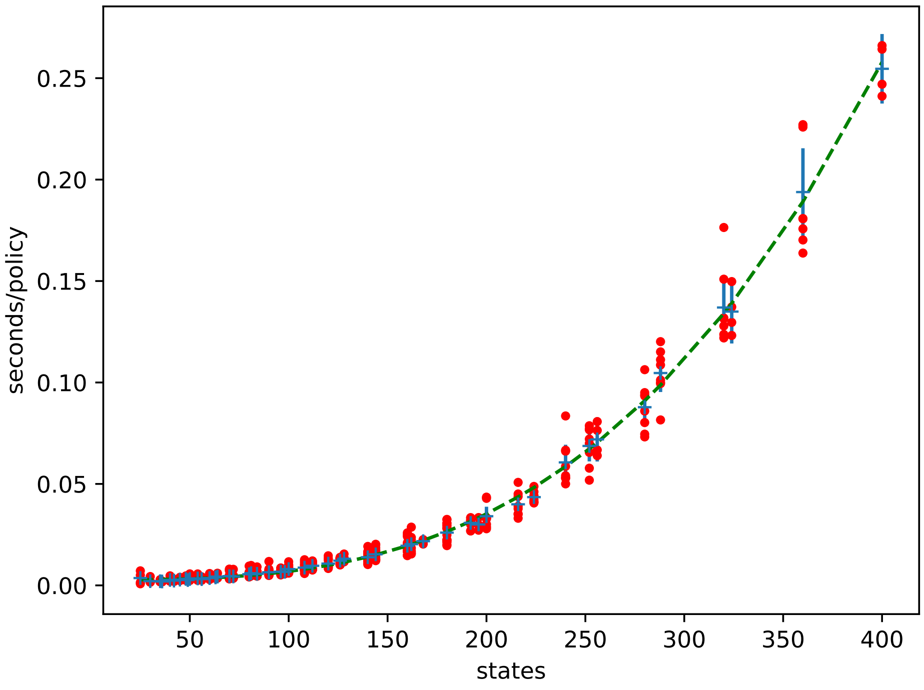


Figure 3.7: Mean time for a policy in dependence of problem size

The ANTG case study models a complex museum with a variety of collections. Due to the popularity of the museum, there are many visitors at the same time. Different visitors may have different preferences of arts. We assume the museum divides all collections into different categories which are separated into different rooms, and visitors can choose what they would like to visit and pay tickets according to their preferences. In order to obtain the best experience, a visitor can, prior to her visit, assign a predefined weight to each category denoting her preferences to the museum, and then design the best strategy for a visit. The problem with this approach is that the preference weights depend on many time-dependent factors such as price, weather, or the length of queue at that moment and are hard to compute in advance. In order to account for this, we allow uncertainties of preferences such that their values may lie in an interval and ask for the best strategies for a given museum. The solution in the form of the best policy or policies can then be used by the museum's administration for fare design decisions or load analysis; for a visitor, the policies can serve as a decision support for optimal museum experience.

For simplicity we assume all collections are organized in an $n \times n$ square with $n \geqslant 10$. Let $m = \frac{n-1}{2}$. We assume all collections at (i, j) are assigned with a weight 1 if $|i-m| > \frac{n}{5}$ or $|j-m| > \frac{n}{5}$, with a weight 2 if $|i-m| \in (\frac{n}{10}, \frac{n}{5}]$ or $|j-m| \in (\frac{n}{10}, \frac{n}{5}]$; otherwise they are assigned with a weight interval $[3, 4]$. In other words, we expect collections in the middle to be more popular and subject to more uncertainties than others. Furthermore, we assume that people at each location (i, j) have two non-deterministic choices: either move to the north and east, that is, $(\min(n, i+1), \min(n, j+1))$ or to the north and west, that is, $(\min(n, i+1), \max(0, j-1))$ if $i \geqslant j$, while if $i \leqslant j$, they can move either to the north and east, $(\min(n, i+1), \min(n, j+1))$, or to the south and east, that is, $(\max(0, i-1), \min(n, j+1))$. The transitions also depend on the location of the collection. For the collections in the middle, the main direction of transition is chosen with probability $[0.8, 1]$ while the probability to move to some other neighbour collection is $[0, 0.2]$. In the expected case, we set the probability to move to the collection in the main direction to 0.8 and distribute the remaining probability mass evenly among other neighbour collections. For collections outside the middle, the main direction (for example, north and west) is chosen with probability 1.

Therefore a model with parameter n has n^2 states in total and roughly $2n^2$ transitions, 2% of which are associated with uncertain weights and uncertain transition probabilities. Notice that a transition with uncertain weights essentially corresponds to several transitions with concrete weights.

We define a reward structure denoting the reward one can obtain by visiting each collection. For simplicity, we let the reward be the same as the weight of a collection. We can ask for the optimal policy for the expected discounted reward criterion, that is, in the scenario where it is preferable to make better rewarding moves early, which seems to be intuitively consistent with what museum visitors want to experience, if one assumes scarcity of cognitive resources [LKS+17].

Evaluation of the ANTG model We present an evaluation of the model for $10 \leqslant n \leqslant 20$, with the results depicted in Fig. 3.8. Again, the algorithms stop after no new policy can be constructed or after the number of evaluated policies exceeds 50 000. For convenience, the runtime and the number of policies are plotted in dependence of the number of states, which is n^2. We see that on large instances, the problem structure yields a large number of optimal policies, thus increasing the required runtime. On small instances, however, the number of optimal policies generated is small, which allows for a fast computation of the Pareto frontier. Furthermore, we see that the Pareto frontier is small when n is odd, which can be explained as an inherent property of the model; this also means that for odd n, the choice among the optimal policies is small and (cognitively) easier for the controller.

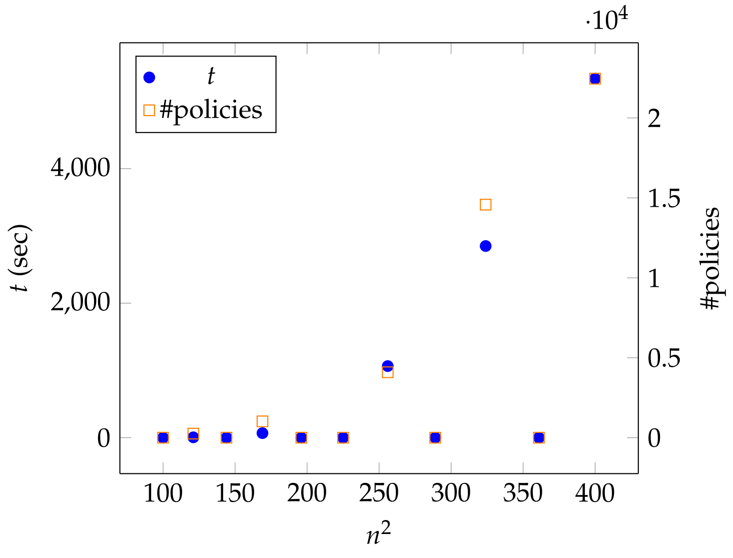


Figure 3.8: Evaluation of the ANTG model

4

A case study

Don't tell me the moon is shining; show me the glint of light on broken glass.

— Anton Chekhov

IN this chapter, we present an application of our algorithms to a more involved model in contrast to "playground" models which we have used for performance evaluation. The model which we choose for analysis is a multi-component system with several degradation stages of the individual components which occur according to a phase-type distribution. We perform an analysis of this model by transforming it into concurrent and stochastic bounded-parameter models, applying our analysis techniques and, finally, comparing the resulting policies.

4.1 Model details

We consider a continuous-time model of $M \in \mathbb{N}$ components where each component with a given index $i \in [M]$ may proceed through $N_i \in \mathbb{N}$ stages of degradation, called *operational phases* $o_{i,1}, \ldots, o_{i,N_i}$. In each operational phase, the component may either change into an operational phase that corresponds to a higher degree of degradation or fail completely. Failure is modeled as a replacement process in a dedicated *renewal phase* n_i after which the component restarts in the operational phase o_{i,N_i} that corresponds to a "new" state of the component. Furthermore, a maintenance action on the i-th component can be undertaken, which moves the i-th component into a *maintenance phase* $m_{i,j}$ from the operational phase $o_{i,j}$. In this phase, maintenance is performed which moves the i-th component into an operational stage which corresponds to a smaller degree of degradation after the maintenance process is done. Thus, the total set of phases of the i-th component can be described by $S_i = \left\{o_{i,N_i}, m_{i,N_i}, \ldots, m_{i,2}, o_{i,1}, n_i\right\}$. We note that with this definition, there is no maintenance phase that can be reached from operational phase with the highest degree of degradation, and hence, no phase $m_{i,1}$ in the model.

In detail, the operational, repair, and replacement processes are modeled as absorbing Markov chains. The (positive real-valued) probability distribution associated with the absorption time in a Markov chain is known in literature as a phase-type distribution (PHD) [Neu79]. In general, phase-type distributions are very popular in modeling and analysis literature for applicability of analytical methods [He14] as well as for being a flexible modeling tool [BKF14, BKS14]. We note that approaches to approximate transition time distributions with the help of phase-type distributions have been previously considered in [YS04]; algorithms for the resulting *semi-Markov decision processes* where the Markovian property holds for the transition probabilities, but not for transition times, have been experimentally evaluated in [BDS17a]. For further reference on semi-Markov processes and their applications, we refer to [KBT75, Bar08]; here, we consider Markov decision models.

Mathematically, a phase-type distribution can be represented by a *subgenerator matrix* $\boldsymbol{D} \in \mathbb{R}^{n\times n}$ and a vector $\vec{\phi} \in \mathbb{R}^{1\times n}_{\geqslant 0}$ with the constraints $\vec{\phi}\vec{1} = 1$, $\boldsymbol{D}\vec{1} \leqslant 0$, and $\boldsymbol{D}(i,j) \geqslant 0 \Leftrightarrow i \neq j$. This representation induces also a vector $\vec{d} = -\boldsymbol{D}\vec{1}$. The matrix $\boldsymbol{D}$ can be seen as a submatrix of a rate matrix $\boldsymbol{Q} \in \mathbb{R}^{(n+1)\times(n+1)}$ in a continuous-time Markov chain with

$$\boldsymbol{Q} = \begin{pmatrix} \boldsymbol{D} & \vec{d} \\ \boldsymbol{0} & 0 \end{pmatrix}.$$

Using the solution of the CTMC differential equation in (1.13) using matrix-exponential expressions as defined in (1.14), we can derive the corresponding probability density function $p(t)$ and the cumulative distribution function $F(t)$.

$$p(t) = \vec{\phi}\exp(\boldsymbol{D}t)\vec{d} \tag{4.1}$$

$$F(t) = 1 - \vec{\phi}\exp(\boldsymbol{D}t)\vec{1} \tag{4.2}$$

For the i-th component and the j-th operational phase, the sojourn time is modeled by a phase-type distribution with representation $(\boldsymbol{D}_o^{i,j}, \vec{\phi}_o^{i,j})$. On an operational phase change, the controller may choose to *repair* the component. This choice invokes a maintenance process which lasts for a time that is distributed according to a PHD with a representation $(\boldsymbol{D}_r^{i,j}, \vec{\phi}_r^{i,j})$, if repair occurred after the j-th operational phase. If no maintenance is performed, the component eventually fails and is replaced; similarly, the failure process in this *failure phase* is modeled by a PHD with representation $(\boldsymbol{D}_f^{i}, \vec{\phi}_f^{i})$. The failure probability for the i-th component in the j-th operational phase is $f_{i,j}$, with $f_{i,1} = 1$. For compactness reasons we define $\bar{f}_{i,j} := 1 - f_{i,j}$. Note that this implies no maintenance phase after the operational phase with index 1.

The rewards of the i-th component are given by $r_{o,i,j}$ for the j-th operational phase and by $r_{r,i,j}$ for the j-th maintenance phase $m_{i,j}$; $r_{f,i}$ denotes the (usually negative) reward in the failure state. Figure 4.1 depicts the inner workings of a single component.

We note that up to now, the consequences of degradation and maintenance are deterministic. To model stochastic transitions, we introduce a strictly lower triangular *degradation matrix* $\boldsymbol{H}_i \in \mathbb{R}^{N_i\times N_i}$ where $\boldsymbol{H}_i(s,t)$ describes the probability to change to the t-th operational phase after the s-th operational phase is completed. Mathematically, we have to enforce the constraints $\boldsymbol{H}_i(s\bullet)\vec{1} = 1$ for all $s > 1$. Semantically, one can assume that components only degrade, that is, $\boldsymbol{H}_i(s,t) > 0$ holds only for $s > t$. For example, in a component with four operational phases, the degradation matrix can look like

$$\boldsymbol{H}_i = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0.1 & 0.9 & 0 & 0 \\ 0.01 & 0.1 & 0.89 & 0 \end{pmatrix}.$$

In a similar way, consequences of maintenance actions can be modeled by an upper triangular *repair matrix* $\boldsymbol{R}_i \in \mathbb{R}^{N_i\times N_i}$ where the value $\boldsymbol{R}_i(s,t)$ denotes the probability of returning to the t-th operational phase from the s-th maintenance phase. Again, the mathematical constraint on this matrix is $\boldsymbol{R}_i(s\bullet)\vec{1} = 1$ for $s > 0$ (as there is no maintenance phase with index 1). A semantic constraint that can be imposed is that maintenance cannot degrade a component, which translates to the condition that $\boldsymbol{R}_i(s,t) > 0$ holds only for $s \leqslant t$. An example repair matrix in a component with four operational phases is given below.

$$\boldsymbol{R}_i = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.5 & 0.4 & 0.1 \\ 0 & 0 & 0.6 & 0.4 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

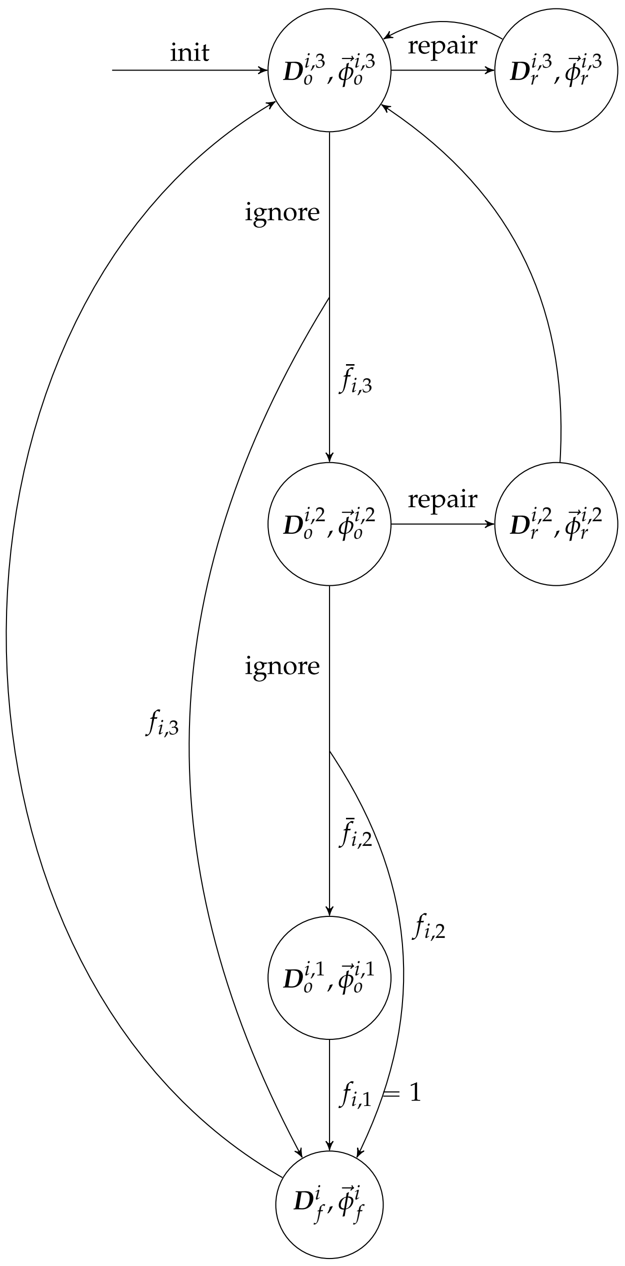

init
$\boldsymbol{D}_o^{i,3}, \vec{\phi}_o^{i,3}$
repair
$\boldsymbol{D}_r^{i,3}, \vec{\phi}_r^{i,3}$
ignore
$\bar{f}_{i,3}$
$\boldsymbol{D}_o^{i,2}, \vec{\phi}_o^{i,2}$
repair
$\boldsymbol{D}_r^{i,2}, \vec{\phi}_r^{i,2}$
ignore
$f_{i,3}$
$\bar{f}_{i,2}$
$f_{i,2}$
$\boldsymbol{D}_o^{i,1}, \vec{\phi}_o^{i,1}$
$f_{i,1} = 1$
$\boldsymbol{D}_f^{i}, \vec{\phi}_f^{i}$

Figure 4.1: Visualization of operational, maintenance and repair phases in a single component

4.2 Towards an uncertain MDP

A naïve model can describe an M-component system by considering, first, $\prod_{i=1}^{M}(2N_i)$ observable phases that can be identified with tuples $(s_1, \dots, s_M)$ which describe individual states of each component. Each observable phase $(s_1, \dots, s_M)$ can then be identified with a product state space of the underlying phase-type distributions of size $\prod_{i=1}^{M}(\dim \boldsymbol{D}^{i,s_i})$ if $\boldsymbol{D}^{i,s_i}$ is the subgenerator matrix of the phase-type distribution in phase s_i. It is easy to see that, while being complete, this model grows exponentially large with growing number of components. In order to illustrate the size of the resulting model, we sketch a complete construction first and then discuss possibilities to reduce the model size by sacrificing some of the precision and explicitly assuming that decisions depend on the current (operational) phase but not on the detailed state of the phase-type distribution processes.

4.2.1 Sketch of a complete MDP model

In detail, a state of one component can be modeled as one state in a continuous-time Markov decision process with $\sum_{j=0}^{N_i} \dim \boldsymbol{D}^{i,j}$ states.

The states are encoded as follows: The first $(\dim \boldsymbol{D}_o^{i,N_i})$ states correspond to the N_ith operational phase, the next $(\dim \boldsymbol{D}_r^{i,N_i})$ states correspond to the N_ith repair phase, the next $(\dim \boldsymbol{D}_o^{i,N_i-1})$ states correspond to the $N_i - 1$st operational phase and so on, with the last $(\dim \boldsymbol{D}_f^i)$ states corresponding to the failure phase. The process starts in the first state block, with the initial distribution vector being $\phi_o^{i,N_i} \otimes \vec{e}_{1,N_i}$.

Concerning the transition rate matrices, we first describe how the "ignore" action affects the dynamics of the system. In the "ignore" case, the transition rate matrix has the shape

$$\boldsymbol{Q}_i^{\text{ignore}} =$$
$$\begin{pmatrix} \boldsymbol{D}_o^{i,N_i} & \boldsymbol{0} & \boldsymbol{H}_i(N_i, N_i-1)\bar{f}_{i,N_i}\vec{d}_o^{i,N_i}\vec{\phi}_o^{i,N_i-1} & \dots & f_{i,N_i}\vec{d}_o^{i,N_i}\vec{\phi}_f^i \\ \boldsymbol{R}_i(N_i, N_i)\vec{d}_r^{i,N_i}\vec{\phi}_o^{i,N_i} & \boldsymbol{D}_r^{i,N_i} & \boldsymbol{R}_i(N_i, N_i-1)\vec{d}_r^{i,N_i}\vec{\phi}_o^{i,N_i} & \dots & \vdots \\ \vdots & \ddots & & & \\ \boldsymbol{0} & \dots & & \boldsymbol{D}_o^{i,1} & \vec{d}_o^{i,1}\vec{\phi}_f^i \\ \vec{d}_f^i\vec{\phi}_o^{i,N_i} & \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{D}_f^i \end{pmatrix}$$

while for the "repair" action, we have

$$\boldsymbol{Q}_i^{\text{repair}} = \begin{pmatrix} \boldsymbol{D}_o^{i,N_i} & \vec{d}_o^{i,N_i}\vec{\phi}_r^{i,N_i} & \boldsymbol{0} & \dots & \boldsymbol{0} \\ \boldsymbol{R}_i(N_i, N_i)\vec{d}_r^{i,N_i}\vec{\phi}_o^{i,N_i} & \boldsymbol{D}_r^{i,N_i} & \boldsymbol{R}_i(N_i, N_i-1)\vec{d}_r^{i,N_i}\vec{\phi}_o^{i,N_i} & \dots & \\ \vdots & \ddots & & & \\ \boldsymbol{0} & \dots & & \boldsymbol{D}_o^{i,1} & \vec{d}_o^{i,1}\vec{\phi}_f^i \\ \vec{d}_f^i\vec{\phi}_o^{i,N_i} & \boldsymbol{0} & \dots & & \boldsymbol{D}_f^i \end{pmatrix}.$$

We note here that the upper rows of the transition rate matrices correspond to the phases with higher indices. This also means that the matrices $\boldsymbol{R}_i$ and $\boldsymbol{H}_i$ which describe the stochastic consequences of maintenance and degradation, are read in the "opposite" direction, that is, the entries in lower rows of $\boldsymbol{R}_i$ and $\boldsymbol{H}_i$ influence the entries in the upper rows of $\boldsymbol{Q}_i^{\text{ignore}}$ and $\boldsymbol{Q}_i^{\text{repair}}$.

The reward vector $\vec{r}_i$ is then

$$\vec{r}_i = \left(r_{o,i,N_i}\vec{1}^\top_{\dim \boldsymbol{D}_o^{i,N_i}}, r_{r,i,N_i}\vec{1}^\top_{\dim \boldsymbol{D}_r^{i,N_i}}, \dots, r_{i,f}\vec{1}^\top_{\dim \boldsymbol{D}_f^i} \right)^\top .$$

Then, to model the complete system, one combines the matrices with Kronecker operators. The matrix for the global "ignore" action is

$$\boldsymbol{Q}^{\text{ignore}} = \bigoplus_{i=1}^{M} \boldsymbol{Q}_i^{\text{ignore}},$$

the transition rate matrix for the "repair component i" action is

$$\boldsymbol{Q}^{\text{repair},i} = \left(\bigoplus_{j=1}^{i-1} \boldsymbol{Q}_j^{\text{ignore}}\right) \oplus \boldsymbol{Q}_i^{\text{repair}} \oplus \left(\bigoplus_{j=i+1}^{M} \boldsymbol{Q}_j^{\text{ignore}}\right).$$

The transition rate matrix for the action "repair components $Z = \{i_1, \dots, i_\ell\}$" with $i_1 < i_2 < \cdots < i_\ell$, if such an action is part of the system, is

$$\boldsymbol{Q}^{\text{repair},Z} = \left(\bigoplus_{j=1}^{i_1-1} \boldsymbol{Q}_j^{\text{ignore}}\right) \oplus \boldsymbol{Q}_{i_1}^{\text{repair}} \oplus \left(\bigoplus_{k=1}^{\ell-1} \left(\left(\bigoplus_{j=i_k+1}^{i_{k+1}-1} \boldsymbol{Q}_j^{\text{ignore}}\right) \oplus \boldsymbol{Q}_{i_{k+1}}^{\text{repair}}\right)\right) \oplus$$
$$\oplus \left(\bigoplus_{j=i_\ell+1}^{M} \boldsymbol{Q}_j^{\text{ignore}}\right).$$

The combined reward vector $\vec{r}$ is defined by

$$\vec{r}((s_1, \dots, s_M)) = \sum_{i=1}^{M} \vec{r}_i(s_i).$$

Analogously, one can define the matrices for the individual "repair" actions; modeling a limited maintenance resource can be done by not allowing further "repair" actions in states of the system where no additional maintenance can be performed.

4.2.2 Model aggregation

We observe that the complete model has the downside of suffering from the notorious *state space explosion* that occurs in most Markov models where several dimensions of state space are combined. Furthermore, the resulting optimal policy may depend on the states of the phase-type processes in each operational phase, which, however, are modeling artifacts and do not correspond to physical properties of the system. To cope with the latter issue, we assume that the controller's decisions depend only on the current operational phase of each component.

Following this assumption, and in order to keep the analysis computationally tractable, we consider a state space reduction method which approximates the phase-type distributions by exponential distributions, reducing the model size of a sigle component to $2N_i$ states and the model size of M components to $\prod_{i=1}^{M}(2N_i)$. This number is still very large for $M \gg 2$, but this at least allows us to analyse two- and three-component systems.

Rate bounds for phase-type distributions In order to compute bounds for a phase-type distribution with representation $(\boldsymbol{D}, \vec{\phi})$, we have to compute rates $\lambda_\downarrow$ and $\lambda_\uparrow$ that approximate the time-dependent rate

$$\lambda(t) = \frac{\vec{\phi} \exp(t\boldsymbol{D})\vec{d}}{\vec{\phi} \exp(t\boldsymbol{D})\vec{1}} \tag{4.3}$$

by bounding it from above and below with

$$\lambda_\downarrow \leqslant \frac{\vec{\phi} \exp(t\boldsymbol{D})\vec{d}}{\vec{\phi} \exp(t\boldsymbol{D})\vec{1}} \leqslant \lambda_\uparrow \tag{4.4}$$

Little can be said about the matrix $\exp(t\boldsymbol{D})$, as a phase-type distribution can approximate any other distribution with positive real support [O'C99]. However, we observe that $\vec{\phi}\exp(t\boldsymbol{D})\vec{1}$ is the sum of the entries of the vector $\vec{\phi}\exp(t\boldsymbol{D})$. Knowing that the vector $\vec{\phi}\exp(t\boldsymbol{D})$ is nonnegative, we conclude that $\vec{\psi}(t) = \frac{\vec{\phi}\exp(t\boldsymbol{D})}{\vec{\phi}\exp(t\boldsymbol{D})\vec{1}}$ is a distribution vector, i. e., $\vec{\psi}(t) \geqslant \vec{0}, \vec{\psi}(t)\vec{1} = 1$. Thus, $\lambda(t)$ is the product of a stochastic vector and $\vec{d}$, which can be bounded by

$$\min_{i\in[n]} \vec{e}_i\vec{d} \leqslant \vec{\psi}(t)\vec{d} \leqslant \max_{i\in[n]} \vec{e}_i\vec{d} = \lambda_\uparrow. \tag{4.5}$$

However, the lower bound approximation for $\lambda(t)$ in (4.5) may be of little use to us, as the lower bound may be zero, if $\vec{d} = -\boldsymbol{D}\vec{1}$ has a zero entry. In this case, one can resort to an empirical sampling procedure such as the one described in Algorithm 18. This algorithm, given a discretization precision δ, iteratively computes the minimal value of $\lambda(t)$ as defined in (4.3) for $t = i\delta, i \in \mathbb{N}$. This is repeated for all values of i until the residual probability mass $1 - F(t)$ is lower than a predefined threshold ε. $F(t)$ is here the continuous distribution function of the phase-type distribution, as defined in (4.2). Finally, Algorithm 18 yields a lower bound $\lambda_\downarrow$ for the time-dependent rate $\lambda(t)$.

Algorithm 18 A simple sampling procedure to compute the minimal rate of a given phase-type distribution

```
1: function PhaseTypeLowerBound($\vec{\phi}, \boldsymbol{D}, \delta, \varepsilon$)
2:     $P \leftarrow 0, i \leftarrow 0$
3:     $\lambda_\downarrow \leftarrow \infty$
4:     while $P > \varepsilon$ do        ▷ Iterate only until the residual probability mass is significant
5:         $P \leftarrow \vec{\phi}\exp(\boldsymbol{D}i\delta)\vec{1}$        ▷ Compute the residual probability mass
6:         $\lambda(t) \leftarrow \frac{\vec{\phi}\exp(t\boldsymbol{D})\vec{d}}{\vec{\phi}\exp(t\boldsymbol{D})\vec{1}}$
7:         $\lambda_\downarrow \leftarrow \min\left\{\lambda(t), \lambda_\downarrow\right\}$
8:         $i \leftarrow i + 1$
    return $\lambda_\downarrow$
```

Furthermore, the average rate $\lambda_\times$ can be computed with

$$\lambda_\times = -(\vec{\phi}\boldsymbol{D}^{-1}\vec{1})^{-1}. \tag{4.6}$$

Having approximated the phase-type distributions in the operational phases by exponential distributions, we derive a stochastic bounded-parameter MDP model and a concurrent MDP model. The concurrent MDP model consists of several MDPs that correspond to different approximations of the phase-type distributions. That is, if we approximate the phase-type distribution $(\boldsymbol{D}^{i,j}, \vec{\phi}^{i,j})$ with an exponential distribution with rate $\lambda^{i,j}$, we get, for a single component, the following transition rate matrices for the "ignore" and "repair" actions. Again, we keep the state ordering we described previously: the first state corresponds to the N_ith operational phase, the second state corresponds to the N_ith maintenance phase and so on, with the $2N_i$th state corresponding to the failure and replacement phase.

$$\begin{aligned}
\tilde{\boldsymbol{Q}}_i^{\text{ignore}} &= \begin{pmatrix} -\lambda_o^{i,N_i} & 0 & \lambda_o^{i,N_i}\boldsymbol{H}_i(N_i,N_i-1)\bar{f}_{i,N_i} & \dots & \lambda_o^{i,N_i} f_{i,N_i} \\ \lambda_r^{i,N_i}\boldsymbol{R}_i(N_i,N_i) & -\lambda_r^{i,N_i} & \lambda_r^{i,N_i}\boldsymbol{R}_i(N_i,N_i-1) & \dots & \vdots \\ \vdots & \ddots & & & \\ 0 & \dots & & -\lambda_o^{i,1} & \lambda_o^{i,1} \\ \lambda_f^i & 0 & \dots & 0 & -\lambda_f^i \end{pmatrix}, \\
\tilde{\boldsymbol{Q}}_i^{\text{repair}} &= \begin{pmatrix} -\lambda_o^{i,N_i} & \lambda_o^{i,N_i} & 0 & \dots & 0 \\ \lambda_r^{i,N_i}\boldsymbol{R}_i(N_i,N_i) & -\lambda_r^{i,N_i} & \lambda_r^{i,N_i}\boldsymbol{R}_i(N_i,N_i-1) & \dots & \\ \vdots & \ddots & & & \\ 0 & \dots & & -\lambda_o^{i,1} & \lambda_o^{i,1} \\ \lambda_f^i & 0 & \dots & 0 & -\lambda_f^i \end{pmatrix}.
\end{aligned} \tag{4.7}$$

The reward vector $\vec{s}_i$ in the i-th component is defined by

$$\vec{s}_i = \left(r_{o,i,N_i}, r_{r,i,N_i}, r_{o,i,N_i-1}, \dots, r_{o,i,1}, r_{i,f}\right)^\top . \tag{4.8}$$

Composition with limited maintenance resources If the number of components which can be in a maintenance phase is unbounded, then there is little motivation in considering composed models: The optimal policy for one component can be just executed in parallel for every component. The use of a combined state space becomes justified when the amount of repair resources is limited. We model this limitation by allowing at most $L \in \mathbb{N}$ components to be in maintenance phases.

In order to compose the components with the limitation of at most $1 \leqslant L \leqslant M$ components in maintenance phases, we proceed as follows. For simplicity, we assume that the replacement process in the failure phase can be entered by any number of components simultaneously. Otherwise, it is easy to see that if the number of simultaneously repairable components is limited, the number of reachable states is reduced; the total number of states in the aggregated model is then

$$\prod_{i=1}^M N_i + \sum_{S \subseteq [M]\colon |S| \leqslant L} \left(\prod_{j \in S} N_j - 1\right)\left(\prod_{i \in [M]\setminus S} N_i\right)$$

which can be bounded by $\left(\sum_{k=0}^L \binom{M}{k}\right) \prod_{i=1}^M N_i$. For $L \ll M/2$, this is less than the number of all states in the unrestricted model, which is $2^M \prod_{i=1}^M N_i$. In order to compute the transition matrices $\bar{\boldsymbol{Q}}^{\text{ignore},L}$, $\bar{\boldsymbol{Q}}^{\text{repair},L}$ in the reduced state spaces efficiently, we propose the following approach. Intuitively, the idea is to compute the reachable state space for increasing number of components and truncating the matrices if the corresponding states are not reachable.

For $i \in [M]$, we define the sets T_i^L, C_i^L, and C_L by

$$\begin{aligned}
T_1^L &= S_1, T_{i+1}^L = \left\{(s_1,\dots,s_{i+1}) \in T_i \times S_{i+1} \mid \text{at most } L \text{ items are maintenance phases}\right\}, \\
C_i^L &= \left\{(s_1,\dots,s_i) \in T_i \mid \text{exactly } L \text{ items are maintenance phases}\right\},
\end{aligned}$$

and $C_L = C_M^L$. Here, the set T_i^L is the set of reachable phases in the first i components. The set C_i^L is the set of phases in the first i components where L components are already in maintenance phases; for simplicity we assume that no further maintenance operation can then be applied. Then, the transition rate matrices for the individual actions can be computed iteratively by setting

$$\begin{aligned}
\bar{Q}_1^{\text{ignore},L} &= \tilde{Q}_1^{\text{ignore}} \\
\bar{Q}_{i+1}^{\text{ignore},L} &= \left(\bar{Q}_i^{\text{ignore},L} \oplus \tilde{Q}_{i+1}^{\text{ignore}}\right)|_{T_{i+1}} \\
X_i^{L,i} &= \left(\bar{Q}_{i-1}^{\text{ignore},L} \oplus \tilde{Q}_i^{\text{repair}}\right)|_{T_i} \\
X_{j+1}^{L,i} &= \left(X_j^{L,i} \oplus \tilde{Q}_{i+1}^{\text{ignore}}\right)|_{T_{j+1}} \\
\bar{Q}^{\text{ignore},L} &= \bar{Q}_M^{\text{ignore},L} \\
\bar{Q}^{\text{repair},L,i}(s\bullet) &= \begin{cases} X_M^{L,i}(s\bullet) & s \notin C_L \\ \bar{Q}^{\text{ignore},L}(s\bullet) & s \in C_L \end{cases}
\end{aligned} \tag{4.9}$$

where $A|_S$ is the matrix A constrained to the rows and columns from the set S. If more than one component can be sent into maintenance simultaneously, then the transition rate matrix $\tilde{Q}^{\text{repair},L,Z}$ for the action "send components $Z = \{i_1, \ldots, i_\ell\}$ with $|Z| \leqslant L$ and $i_1 < i_2 < \cdots < i_\ell$ into maintenance" can be computed analogously as follows. As in (4.9), we compute a set of helper matrices $\left(X_j^{Z,L}\right)_{i_1-1 \leqslant j \leqslant M}$ by

$$X_j^{L,Z} = \begin{cases} \bar{Q}_{i-1}^{\text{ignore},L} & j = i_1 - 1 \\ \left(X_{j-1}^{L,Z} \oplus \tilde{Q}_i^{\text{repair}}\right)|_{T_j} & j \in Z \\ \left(X_{j-1}^{L,Z} \oplus \tilde{Q}_i^{\text{repair}}\right)|_{T_j} & j \in \{i_1+1, \ldots, M\} \setminus Z \end{cases} \tag{4.10}$$

and define a helper function $m\colon \times_{i \in [M]} S_i \to [M]$ with

$$m(s_1, s_2, \ldots, s_M) = \{i \mid s_i \text{ is a maintenance phase}\}$$

which computes the set of components currently in maintenance phases. From this we can derive the transition rate matrix

$$\tilde{Q}^{\text{repair},L,Z}(s\bullet) = \begin{cases} X_M^{L,Z}(s\bullet) & |m(s) \cup Z| \leqslant L \\ \bar{Q}^{\text{ignore},L}(s\bullet) & \text{otherwise} \end{cases} \tag{4.11}$$

The reward vector in this continuous-time process is $\vec{s}$ with $\vec{s}(s_1, \ldots, s_M) = \sum_{i=1}^{M} \vec{s}_i(s_i)$.

We observe that for every choice of exponential distribution parameters, we get a continuous-time MDP with at most $M+1$ different actions and transition probability matrices

$$P^{\text{ignore},L}, P^{\text{repair},1,L}, \ldots, P^{\text{repair},M,L}$$

that are generated from the action matrices

$$\bar{Q}^{\text{ignore},L}, \bar{Q}^{\text{repair},1,L}, \ldots, \bar{Q}^{\text{repair},M,L}$$

by deriving the exponential distribution parameter $\vec{\beta} = \operatorname{diag}\left(-\bar{Q}^{\text{ignore},L}\right)$[1] and setting

$$P^a(s, s') = \begin{cases} \frac{\bar{Q}^a(s,s')}{\vec{\beta}(s)} & s \neq s', \\ 0 & s = s'. \end{cases}$$

[1] or any other matrix, as the sojourn times in each state have the same distribution independent of the action

We note that the exact values in $\boldsymbol{P}^a$ are dependent on the choice of the distribution parameters, and, ultimately, on the rate approximations for the phase-type distributions. Choosing minimal, maximal, and average rates as aggregates for the PH distributions, we obtain in total at most $3^{\sum_{i=1}^{M} N_i}$ scenarios which we can consider as a concurrent MDP.

Usually, we are interested in a limited number of scenarios in order to simplify decision making. Here, it seems natural to choose minimal, maximal, and average rates for all distributions, which results in three scenarios.

Pessimistic scenario Choose the maximal rate for the degradation processes and the minimal rate for the repair and replacement processes

Optimistic scenario Choose the minimal rate for the degradation processes and the maximal rate for the repair and replacement processes

Average scenario Choose the average rates for all processes

For the stochastic bounded-parameter model, the upper and lower bounds can be derived as follows. By choosing the maximal and minimal rate bounds as computed by (4.5) and Algorithm 18, we obtain two Markov decision processes with corresponding transition probability matrices $\boldsymbol{P}^a_{\max}, \boldsymbol{P}^a_{\min}$. The lower and upper bound matrices $\boldsymbol{P}^a_{\downarrow}, \boldsymbol{P}^a_{\uparrow}$ can be then computed element-wise with

$$\begin{aligned}\boldsymbol{P}^a_{\downarrow}(s,s') &= \min\left\{\boldsymbol{P}^a_{\max}(s,s'), \boldsymbol{P}^a_{\min}(s,s')\right\},\\ \boldsymbol{P}^a_{\uparrow}(s,s') &= \max\left\{\boldsymbol{P}^a_{\max}(s,s'), \boldsymbol{P}^a_{\min}(s,s')\right\}.\end{aligned}$$

The average scenario described above, derived from the average rate model, serves as the average case in the stochastic BMDP.

Using the uniformization technique described in subsection 1.5.5, we can translate this continuous-time model into a discrete-time model with equal distribution over sequences of states, which allows us to analyse it with the methods discussed in the previous chapter.

4.3 Evaluation

We consider two and three-component models with two and four operational phases in each component. The goal of this experiment is to observe how well the algorithms perform in practice and what kind of solutions they generate. For the CMDP model, the topic of interest is the performance of the compromise policy against the scenario-specific policies in each scenario. For the BMDP model, the question of interest is the shape and size of the Pareto frontier.

Model parameters We consider a general model for a varying number of operational phases and components and one simultaneous maintenance dispatch which is computed deterministically. The parameters of the model are the number of operational phases $N \in \mathbb{N}$ and the number of components $M \in \mathbb{N}$. Only one maintenance process can be launched with an action. Given these numbers, we compute a component with N operational phases with the following transition rates and rewards and initial distribution vector $\vec{q} = \vec{e}_1$. To keep things simple, we consider deterministic transitions after degradation and maintenance.

M	N	states	actions
2	2	15	3
2	4	55	3
3	2	54	4
3	4	350	4

Table 4.1: State and action space sizes of the generated models

$$
\begin{aligned}
\boldsymbol{D}_o^{i,j} &= \begin{pmatrix} -j & j & 0 \\ 0 & -j & j \\ 0 & 0 & -j \end{pmatrix}, \vec{\phi}_o^{i,j} = (1,0,0) \\
\boldsymbol{D}_r^{i,j} &= 10 \cdot \boldsymbol{D}_o^{i,j}, \vec{\phi}_r^{i,j} = (1,0,0) \\
\boldsymbol{D}_f^{i} &= \begin{pmatrix} -3 & 2.9 & 0 & 0 \\ 0 & -1 & 0.9 & 0 \\ 0 & 0 & -0.5 & 0.4 \\ 0 & 0 & 0 & -1.0 \end{pmatrix}, \vec{\phi}_f^{i} = (1,0,0,0) \\
f_{i,j} &= \frac{1}{j} \\
\boldsymbol{R}_i &= \begin{pmatrix} 0 & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I} \end{pmatrix} \\
\boldsymbol{H}_i &= \begin{pmatrix} \boldsymbol{0} & 0 \\ \boldsymbol{I} & \boldsymbol{0} \end{pmatrix} \\
r_{o,i,j} &= 10 - \frac{1}{j}, r_{r,i,j} = \frac{r_{o,i,j}}{2}, r_{f,i} = 0
\end{aligned}
$$

Experiment setup We generate the composed component model for $N \in \{2, 4\}$ and $M \in \{2, 3\}$ and transform it into a BMDP and CMDP model as described above. The set of non-dominated solutions is computed with Algorithm 16. A solution to the concurrent MDP with three scenarios (fast degradation and slow maintenance, average case and slow degradation and fast maintenance) with respective weights $(0.3, 0.4, 0.3)$ is generated by Algorithm 14. It is important to note here that the CMDP and BMDP results are not necessarily comparable to each other, as the lower bound for the value in a BMDP can be achieved in our model by not only fast degradation, but also fast maintenance in operational phases $o_{i,1}$ that correspond to nearly dysfunctional components. We describe thus results for the different optimization procedures individually.

Results The models have three and four actions and between 15 and 350 states (cf. Table 4.1). On the reference machine with an Intel Xeon E5-2690v2 CPU with 3.0GHz and 126GB RAM, the CMDP algorithms are able to compute the result in under one minute for all instances, while the BMDP algorithm needs five hours to complete on the largest instance and under one minute on all other instances.

For BMDPs, a set of three-dimensional result values that correspond to the pessimistic, average, and optimistic values of the policy in the state $s = (o_{1,N_1}, o_{2,N_2}, \ldots, o_{M,N_M})$ where all components are new is computed. Here, we give its projections on two dimensions in three plots. In the first plot, we show the performance of the policies in the pessimistic case and the average case. In the second plot, we show the performance of the policies in the average case and the optimistic case. In the third plot, the projection on the optimistic and pessimistic cases is shown. As the plots show two-dimensional projections of three-

v_{pes}	v_{avg}	v_{opt}
545.822	1177.582	1899.990

Table 4.2: Non-dominated solution of the composed component model with 2 components, 2 operational phases each, and one repair worker.

	$\pi_{\mathrm{Compromise}}$	$\pi_{\mathrm{Pessimistic}}$	π_{Average}	$\pi_{\mathrm{Optimistic}}$
v_{pes}	569.044 53	569.044 53	564.984 63	499.345 18
v_{avg}	1177.582 47	1143.101 26	1177.582 47	893.532 51
v_{opt}	1899.970 09	1899.970 09	1899.970 08	1899.970 09

Table 4.3: Concurrent MDP evaluation of the composed component model with 2 components, 2 operational phases each, and one repair worker.

	$\pi_{\mathrm{Compromise}}$	$\pi_{\mathrm{Pessimistic}}$	π_{Average}	$\pi_{\mathrm{Optimistic}}$
v_{pes}	581.289 22	582.674 27	581.289 15	571.317 13
v_{avg}	1354.232 28	1202.471 53	1354.232 27	1145.747 65
v_{opt}	1949.991 21	1949.991 21	1949.991 21	1949.991 21

Table 4.4: Concurrent MDP evaluation of the composed component model with 2 components, 4 operational phases each, and one repair worker.

dimensional values, some of the policies might seem dominated in the figures while in reality, they are non-dominated either in other projections or in states different from the initial state.

For the model with two components and two operational phases each, only one non-dominated solution has been generated. The resulting values can be seen in Table 4.2. The difference in values between this table and Table 4.3 is explained by noting that the pessimistic scenario in the BMDP model depends on the policy and may differ from the pessimistic scenario in the concurrent MDP model.

The results can be seen in Figure 4.2–Figure 4.4. The blue dots depict the computed solutions in value space. Note that as the values are three-dimensional and some of them are non-dominated in components that correspond to other states than s, some of the computed policies may seem dominated in the two-dimensional projections.

One can see in these figures that the BMDP problem yields a multitude of non-dominated solutions with different values. In the case where the relative costs of the best and worst cases are not known to the user initially, she can in each individual case select a solution that suits her needs best from the non-dominated set.

For CMDPs, we compare the solution to the concurrent MDP problem to the best solution in each individual scenario, with the results visible in Table 4.3–Table 4.6. The rows of the table correspond to the scenarios (pessimistic scenario: fast degrading and slow maintenance, average scenario: average rates, optimistic scenario: slow degrading and fast maintenance) and the columns correspond to the computed policies (compromise policy and the policies optimal for the individual scenarios). The entries of the tables correspond to the achieved values in the respective scenarios by the corresponding policies. We observe that the computed compromise policy behaves well in all situations. This means that in the case where the relative costs of the worst, best and average cases are known and the scenarios which provide these cases are also known, the solution of the concurrent MDP formulation can be used in all situations without sacrificing much in comparison to the optimal solution and surpassing the policy for an "average" scenario.

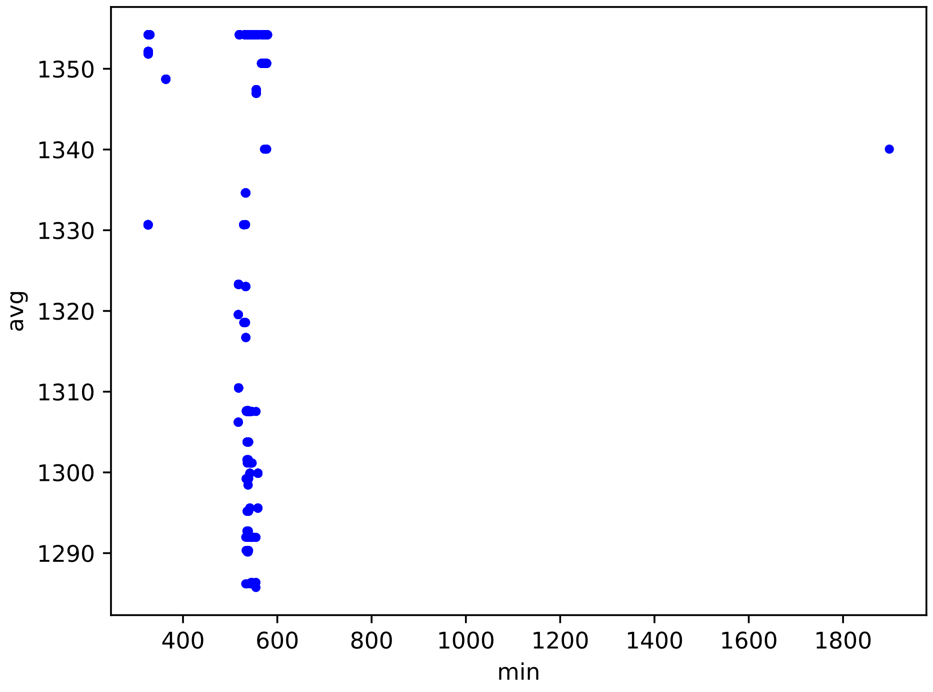
Summing up, we can say that our approaches are suitable to compute adequate policies for realistic models.

	$\pi_{\text{Compromise}}$	$\pi_{\text{Pessimistic}}$	π_{Average}	$\pi_{\text{Optimistic}}$
v_{pes}	1984.301 52	1996.445 31	1914.444 55	1649.699 97
v_{avg}	2735.989 54	2403.179 11	2735.989 54	2350.537 65
v_{opt}	3799.700 48	3799.700 48	3799.700 47	3799.700 23

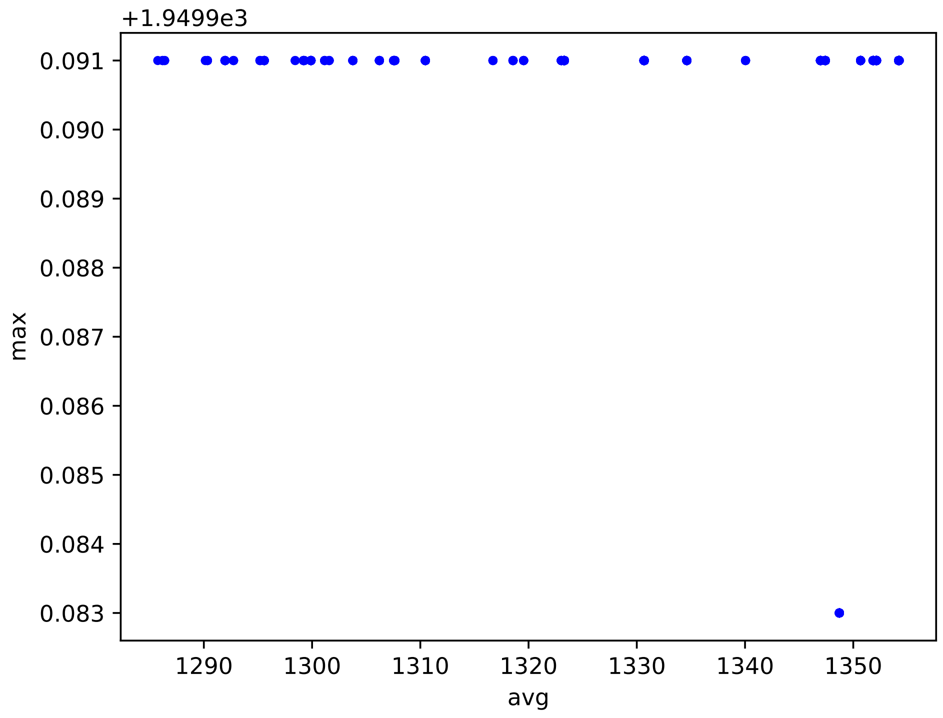
Table 4.5: Concurrent MDP evaluation of the composed component model with 3 components, 2 operational phases each, and one repair worker.

	$\pi_{\text{Compromise}}$	$\pi_{\text{Pessimistic}}$	π_{Average}	$\pi_{\text{Optimistic}}$
v_{pes}	2819.329 31	2834.989 59	2154.405 26	1825.5998
v_{avg}	2988.009 42	2873.922 78	3039.021 19	2452.738 66
v_{opt}	3898.511 07	3898.511 07	3899.613 43	3899.613 43

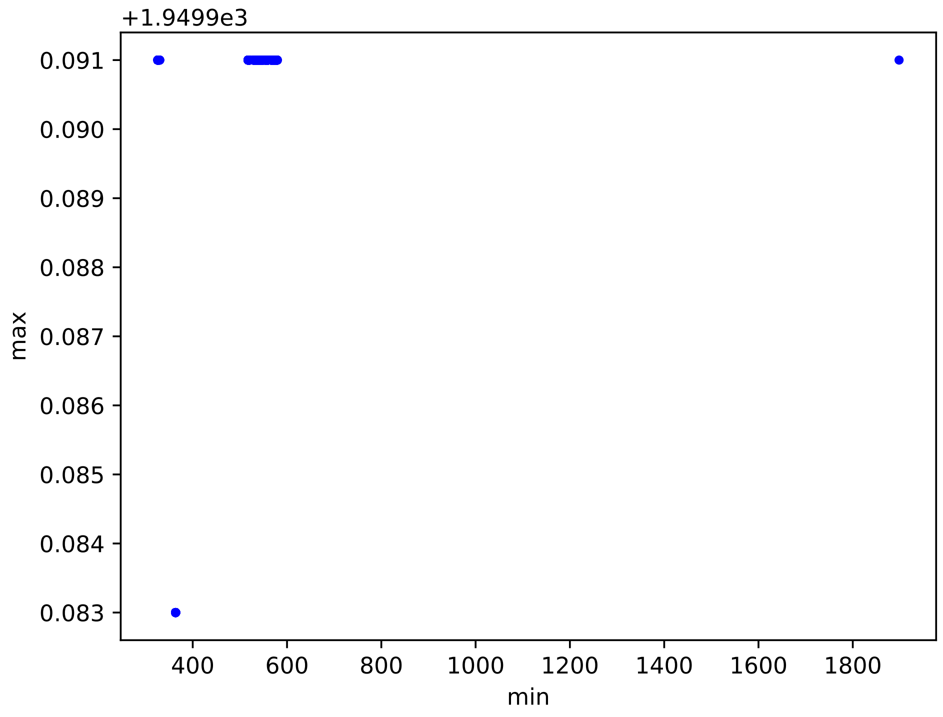
Table 4.6: Concurrent MDP evaluation of the composed component model with 3 components, 4 operational phases each, and one repair worker.



(a) Minimum and average values.

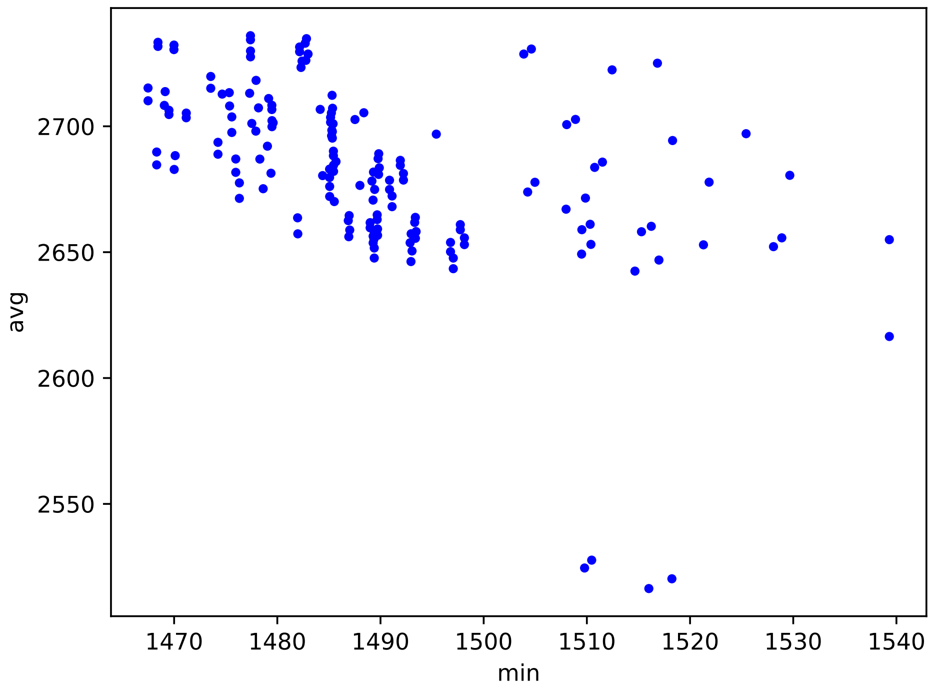


(b) Average and maximum values.

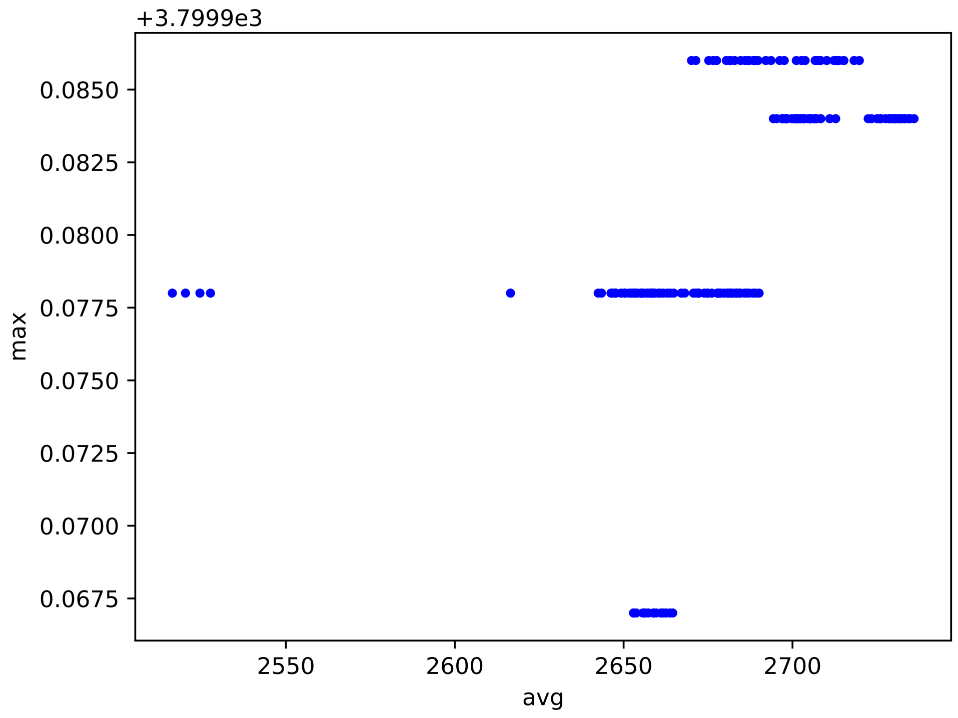


(c) Minimum and maximum values.

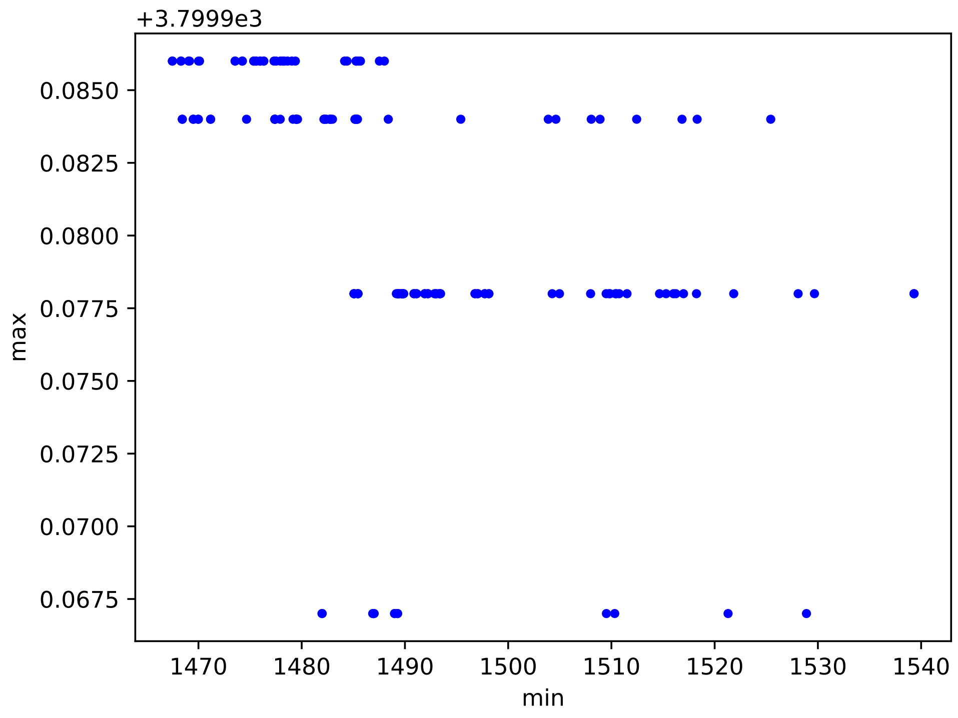
Figure 4.2: Evaluation of the composed component model with 2 components, 4 operational phases each, and one repair worker.



(a) Minimum and average values.

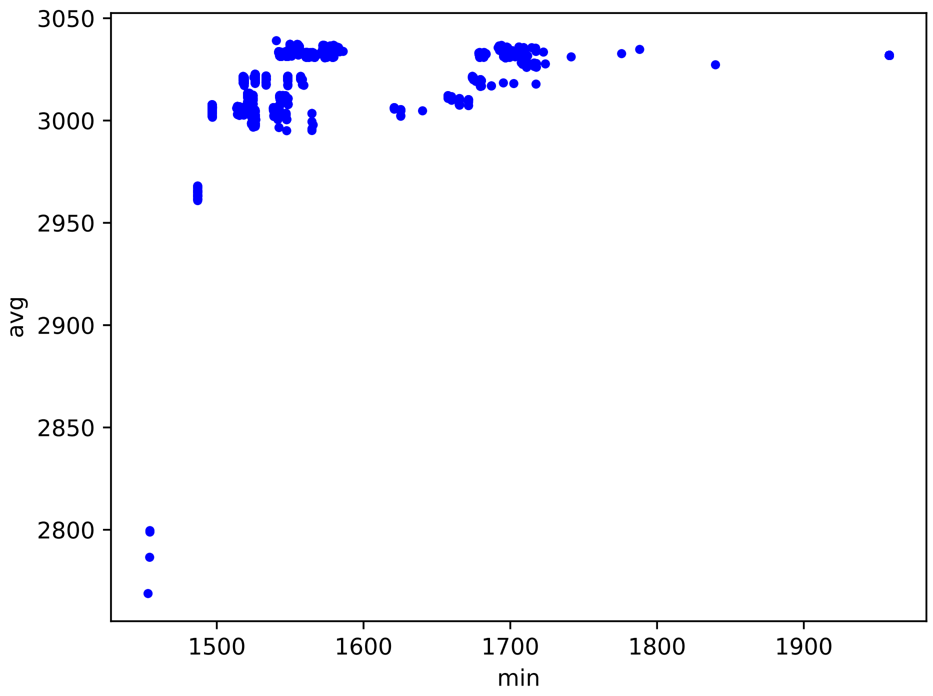


(b) Average and maximum values.

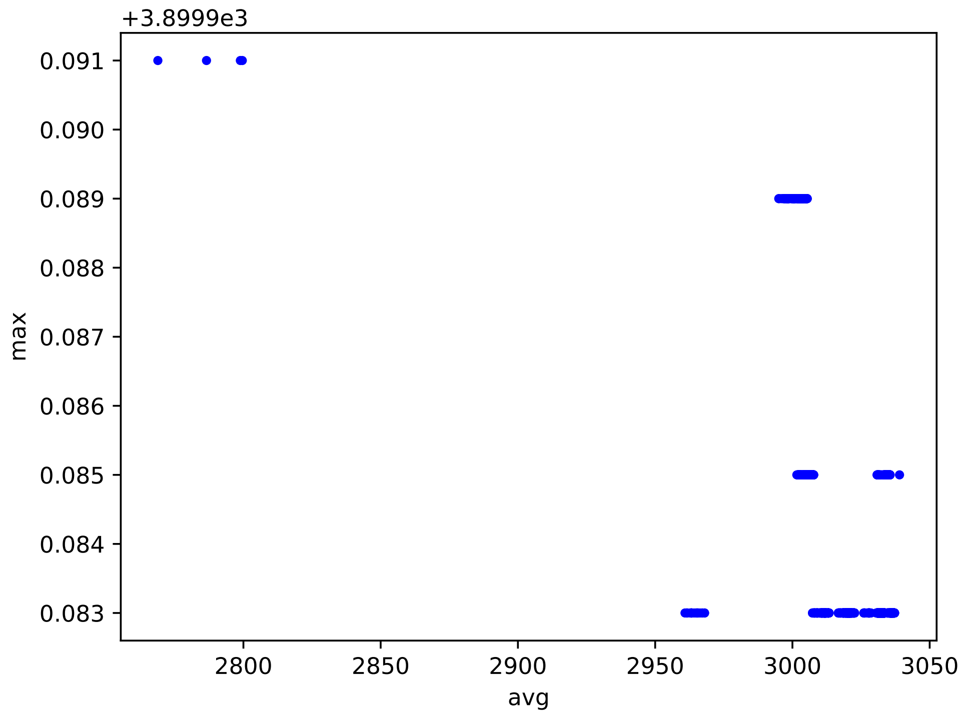


(c) Minimum and maximum values.

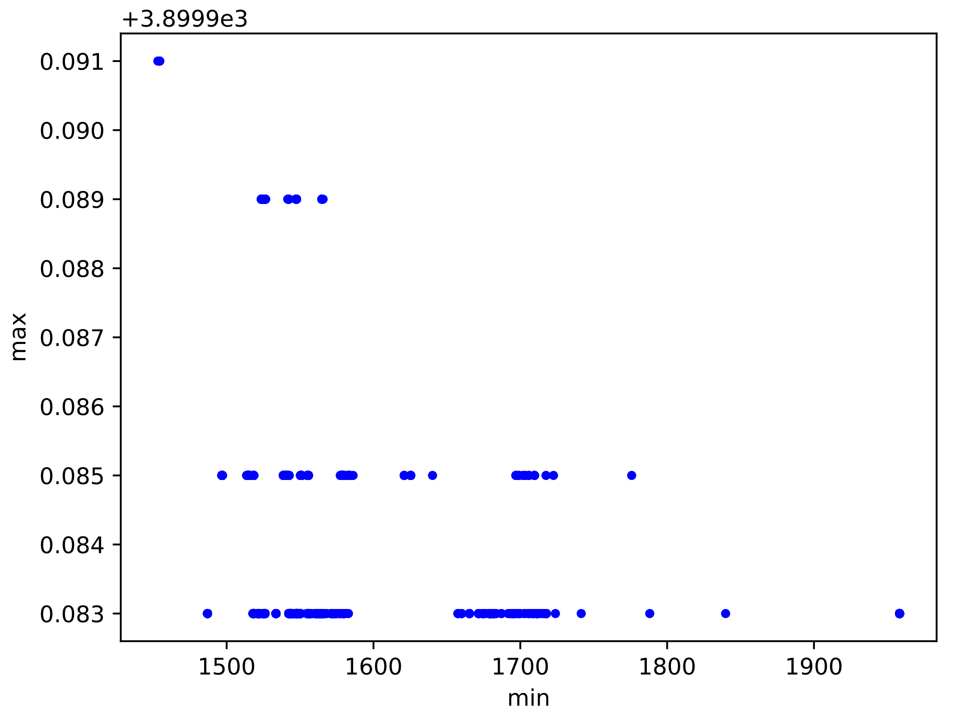
Figure 4.3: Evaluation of the composed component model with 3 components, 2 operational phases each, and one repair worker.



(a) Minimum and average values.



(b) Average and maximum values.



(c) Minimum and maximum values.

Figure 4.4: Evaluation of the composed component model with 3 components, 4 operational phases each, and one repair worker.

5 Discussion

Failure is not an option—it is mandatory. The option is whether or not to let failure be the last thing you do.

— Howard Tayler

We briefly overview our results. No scientific work, especially no PhD thesis, is a monolithic piece of research. A PhD thesis in computer science is in most cases a compilation of several publications with some further results on top, and this work is no exception; it also cannot be claimed that this work closes all questions on model uncertainty in Markov decision processes for good[1]. To put this in a less philosophical manner, this work extends previous literature, answers some of newly arisen questions, and motivates further research. Here, we discuss our results in the previous research context and consider possible future research directions.

5.1 Conclusions

This work started as an investigation of model uncertainty in MDPs. Basic formalisms which capture this notion have been considered by Silver [Sil63], Satia and Lave [SL73], White and El-Deib [WED94], Givan et al. [GLD00], and many others such as [Iye05]. The goal of this work was to establish theoretical and practical results for the different formalisms of model uncertainty in MDPs, with an emphasis on the bounded-parameter MDP and the concurrent MDP models.

Theoretical results In the beginning, we pursued the theoretical implications of uncertainty in Markov decision processes, including the aforementioned models, but also parametric formalisms and related questions. This has led to a set of theoretical results which we have discussed in chapter 2 and which can be summarized as follows. First, bounded-parameter models (and some other models with convex uncertainty sets) can be reduced to zero-sum two-player stochastic games with perfect information and vice versa, which establishes some (albeit already proven) results on stationary policies with zero further cost. Second, additional extensions of the uncertainty model which allow dependencies in the transition probabilities across states result in NP-hard decision problems. Generalizing the result, we can conclude that violation of the Markovian assumption in the model makes policy optimization a hard task. Moreover, the Markovian assumption is violated when considering the finite-horizon expected total reward criterion on the bounded-parameter (BMDP) uncertainty model, which also leads to NP-hardness of policy optimization. Third,

[1]To quote an acquaintance who put it more laconically: "Remember, a dissertation does not need to be the last word on the topic, not even *your* last word."

multi-scenario optimization problems also turned out be theoretically hard. In detail, hardness of optimization of a single policy for multiple (at least two) MDPs with shared state and action spaces (stochastic multi-scenario MDP problem) has been shown, as well as hardness of finding a policy for a BMDP subject to given value vector constraints. Finally, we extend the bounded-parameter MDP model to capture a prior distribution on the uncertainty set in order to allow for more multi-objective applications.

Algorithms The hardness results in chapter 2 have motivated us to consider empirical approaches to uncertain MDP problems, with focus on the expected discounted reward optimality criterion. In chapter 3, we have picked out two problems from multi-scenario optimization and have applied methods from algorithm engineering to them. For the stochastic multi-scenario MDP problem, we have discussed different exact and heuristic approaches; it turns out that the heuristics are significantly faster in terms of time complexity and do not sacrifice much of the solution quality.

The second problem in chapter 3 is the Pareto frontier enumeration problem for BMDPs (and their extension, stochastic BMDPs), if the optimistic, pessimistic and average cases are considered simultaneously, resulting in a multi-objective problem. For this problem, we design an algorithm that is theoretically correct as well as an heuristic. Unfortunately, the theoretically correct algorithm cannot be applied in practice for reasons of prohibitive complexity, but the evaluation of the heuristic shows, again, promising and practically usable results.

Applications In chapter 4 we have considered a complex stochastic model of several components sharing limited maintenance resources. Given the model, we have derived concurrent and bounded-parameter MDP formulations for the problem of finding an optimal maintenance schedule. Furthermore, for some instances of this problem, we have applied the algorithms derived in chapter 3 and showed the numerical results. It turns out that the BMDP formulation yields a large number of mutually non-dominated policies from which the user may select the one that suits her best. For the solution of the CMDP formulation we can say that it behaves well in all given scenarios, in contrast to the scenario-specific policies.

5.2 Future work

Obvious starting points for future work on uncertain MDPs are different optimality criteria. For instance, one may ask for optimal policies in the case of hyperbolic discounting or other aggregated measure of an infinite sequence of rewards. However obvious, these questions seem to be more of an academic interest to a researcher, as common MDP literature [Kal83, Put94, Kal16] concerns itself mainly with finite-horizon, expected average and expected discounted reward criteria.

A question that has been raised while submitting some of the results from chapter 3 by a reviewer was to consider the formalism of stochastic BMDPs in a different fashion. As a SBMDP is, ultimately, a set of MDPs $M_\updownarrow$ with a prior distribution with some density p, one can ask for a policy that maximises the reward measure subject to the distribution, that is, given a reward measure $v\colon M_\updownarrow \times F \to \mathbb{R}$, maximize $\int_{M_\updownarrow} v(M, f)p(M)\,\mathrm{d}M$, where F is a set of policies. This is, ultimately, a continuous stochastic programming problem, which boils down to volume optimization. While it is widely assumed that volume optimization is a computationally hard problem [Kha93], knowledge about the empirical performance of current approaches would be beneficial. A positive result would encourage a comparison to the BMDP algorithms presented in chapter 3, a negative result would show that this approach does not scale to practical problems.

We note here that our current results already allow one to approximate the solution of the continuous stochastic optimization problem. A valid solution approach would be to subdivide the uncertainty set into a finite number of subsets and solve a discrete stochastic optimization problem for aggregates over these subsets. It is easy to see that the latter sub-problem is exactly the stochastic multi-scenario problem for MDPs we have considered in the first part of chapter 3. The quality of this approximation depends largely on the properties of the probability distribution over the uncertainty set; for a continuous probability density function the limit of the approximation (with number of subsets of the uncertainty set going to infinity) is exactly the desired value, but the convergence speed is not clear and subject to future research.

Furthermore, for the stochastic optimization problem for bounded-parameter MDPs, where a weighted sum of the lower bound, upper bound, and average value has to be optimized, the same approach can be used without much change if the MDPs from the uncertainty set which provide lower and upper bounds are independent of the chosen policy, that is, if there exist global MDPs $M_\downarrow, M_\uparrow \in M_\updownarrow$ which minimize resp. maximize the value function for all policies. For general BMDPs, this problem is open yet we conjecture that a policy iteration-based approach would still be suitable, at least as a heuristic.

Similar questions, of course, could be asked for other NP-hard problems we have discussed in chapter 2. Again, empirical performance measurements are an interesting topic for future work. In particular, we believe that it is possible to devise an efficient algorithm for the following problem: Given a BMDP and a constraint v on a value function for the lower or upper bound, optimize the other value function. Formally speaking, the task is to compute

$$\max_{\pi \in \mathcal{P}_{\text{pure}}, \vec{q}^\top \vec{v}_\uparrow^{(\pi)} \geqslant v} \vec{q}^\top \vec{v}_\downarrow^{(\pi)}$$

and, symmetrically,

$$\max_{\pi \in \mathcal{P}_{\text{pure}}, \vec{q}^\top \vec{v}_\downarrow^{(\pi)} \geqslant v} \vec{q}^\top \vec{v}_\uparrow^{(\pi)}.$$

We believe that it is possible to solve this problem with a policy iteration-like approach efficiently [BS17b].

Furthermore, there is always a question of further applicability of established results. We believe that the empirical results from chapter 3 can be applied to multi-objective MDPs which are modeled as MDPs with more than one reward component. Existing literature on that topic concentrates on deterministic MDPs [PW10] and stochastic optimization problems in this context [RSS+14, RWO14] or, in our view, largely theoretical value iteration-like approaches [Whi82] and we believe it would be fruitful to transfer our approaches to plain multi-objective MDP models.

A

Concurrent MDP algorithms evaluation

There is no data like more data.

— Robert Mercer

On the following pages, we present the data obtained while evaluating the algorithms for concurrent Markov decision process optimization.

Table A.1: CMDP algorithms for $\gamma = 0.9$ on stochastic models

Parameters			CPolicyOpt, Alg. 13			CPurePolicyOpt, Alg. 14			QCLP (3.4)				MIP (3.9)			NLP (3.5), (3.8)		
K	n	m	t	σ_t	diff	t	σ_t	diff	t	σ_t	err	diff	t	σ_t	diff	t	σ_t	diff
2	2	2	0.01s	0.00s	0.15%	0.01s	0.00s	0.16%	0.13s	0.18s	0	0.01%	0.03s	0.04s	0.16%	0.01s	0.00s	0.13%
2	2	3	0.01s	0.00s	0.10%	0.01s	0.00s	0.13%	0.26s	0.24s	0	0.01%	0.03s	0.01s	0.13%	0.01s	0.00s	0.09%
2	2	5	0.01s	0.00s	0.00%	0.01s	0.00s	0.01%	0.53s	0.18s	0	0.00%	0.04s	0.02s	0.01%	0.02s	0.00s	0.00%
2	5	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	1.84s	1.09s	0	0.00%	0.06s	0.05s	0.00%	0.01s	0.00s	0.00%
2	5	3	0.01s	0.00s	0.02%	0.01s	0.00s	0.02%	3.82s	1.14s	0	0.01%	0.11s	0.08s	0.00%	0.02s	0.01s	0.00%
2	5	5	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	7.49s	2.03s	0	0.08%	0.17s	0.10s	0.00%	0.02s	0.01s	0.01%
2	10	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	18.02s	2.66s	0	0.04%	0.12s	0.08s	0.00%	0.03s	0.05s	0.00%
2	10	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	23.30s	5.96s	0	0.11%	0.29s	0.10s	0.00%	0.02s	0.00s	0.00%
2	10	5	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	47.38s	7.53s	0	0.27%	0.44s	0.08s	0.00%	0.03s	0.01s	0.00%
2	20	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	126.27s	8.69s	0	0.18%	0.42s	0.11s	0.00%	0.05s	0.08s	0.00%
2	20	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	177.77s	31.81s	0	0.37%	0.90s	0.53s	0.00%	0.05s	0.05s	0.00%
2	20	5	0.03s	0.00s	0.00%	0.01s	0.00s	0.00%	311.98s	31.39s	0	0.80%	7.56s	12.18s	0.00%	0.08s	0.11s	0.00%
2	50	2	0.03s	0.00s	0.00%	0.02s	0.00s	0.00%	1549.57s	78.02s	15	0.90%	13.83s	13.57s	0.00%	0.94s	1.44s	0.00%
2	50	3	0.05s	0.01s	0.00%	0.02s	0.00s	0.00%	2042.48s	147.55s	19	1.92%	1428.89s	2062.73s	0.00%	0.84s	2.03s	0.00%
2	50	5	0.20s	0.02s	0.00%	0.04s	0.01s	0.00%	4757.31s	288.29s	19	3.00%	4498.39s	1500.01s	0.00%	0.72s	1.33s	0.00%
2	100	2	0.12s	0.01s	0.00%	0.06s	0.01s	0.00%	4985.54s	N/A	29	1.19%	3722.95s	2101.85s	0.00%	5.12s	11.26s	0.00%
2	100	3	0.29s	0.03s	0.00%	0.10s	0.02s	0.00%	N/A	N/A	30	N/A	4330.59s	1695.82s	0.00%	8.45s	29.63s	0.00%
2	100	5	1.26s	0.10s	0.00%	0.19s	0.04s	0.00%	N/A	N/A	30	N/A	4835.52s	832.06s	0.01%	12.14s	37.16s	0.00%
2	200	2	1.06s	0.14s	0.00%	0.42s	0.12s	0.00%	N/A	N/A	30	N/A	4964.15s	22.57s	0.00%	65.99s	186.64s	0.19%
2	200	3	3.06s	0.30s	0.00%	0.63s	0.24s	0.00%	N/A	N/A	30	N/A	4763.11s	880.47s	0.01%	19.79s	28.01s	0.00%
2	200	5	10.78s	0.56s	0.00%	1.31s	0.34s	0.00%	5030.34s	N/A	29	1.74%	4987.08s	1.65s	0.02%	161.92s	369.07s	0.00%
2	300	2	4.46s	0.55s	0.00%	1.32s	0.55s	0.00%	4999.95s	2.88s	28	0.76%	4965.71s	20.83s	0.00%	369.16s	798.14s	0.22%
2	300	3	12.72s	0.84s	0.00%	2.61s	0.94s	0.00%	5013.44s	N/A	29	1.07%	4828.58s	842.55s	0.01%	1206.53s	1372.55s	0.32%
2	300	5	42.15s	2.23s	0.00%	4.39s	1.89s	0.00%	5105.98s	25.37s	29	0.75%	4986.83s	1.61s	0.02%	196.75s	243.60s	0.00%
3	2	2	0.01s	0.00s	0.04%	0.01s	0.00s	0.19%	0.24s	0.28s	0	0.00%	0.03s	0.05s	0.19%	0.01s	0.00s	0.02%
3	2	3	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	0.40s	0.28s	0	0.01%	0.03s	0.02s	0.01%	0.02s	0.00s	0.00%
3	2	5	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	0.80s	0.38s	0	0.02%	0.05s	0.02s	0.00%	0.02s	0.00s	0.01%
3	5	2	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	3.57s	1.86s	0	0.01%	0.08s	0.05s	0.00%	0.01s	0.00s	0.00%
3	5	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	7.06s	0.87s	0	0.01%	0.23s	0.07s	0.00%	0.02s	0.00s	0.00%
3	5	5	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	11.66s	3.04s	0	0.23%	0.36s	0.07s	0.00%	0.02s	0.01s	0.01%
3	10	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	35.44s	2.77s	1	0.04%	0.31s	0.08s	0.00%	0.02s	0.01s	0.00%
3	10	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	47.28s	7.36s	0	0.17%	0.52s	0.15s	0.00%	0.03s	0.04s	0.00%
3	10	5	0.02s	0.00s	0.00%	0.01s	0.00s	0.01%	82.87s	14.46s	1	0.15%	1.26s	0.87s	0.00%	0.04s	0.04s	0.00%
3	20	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	247.76s	27.59s	6	0.43%	0.78s	0.34s	0.00%	0.04s	0.03s	0.00%
3	20	3	0.02s	0.00s	0.00%	0.01s	0.00s	0.00%	300.79s	47.58s	3	0.76%	7.55s	6.25s	0.00%	0.07s	0.07s	0.00%
3	20	5	0.07s	0.01s	0.00%	0.01s	0.00s	0.00%	453.55s	66.34s	12	1.58%	202.32s	314.09s	0.00%	0.13s	0.15s	0.00%
3	50	2	0.04s	0.00s	0.00%	0.03s	0.00s	0.00%	3786.02s	164.11s	27	1.67%	1895.67s	2313.69s	0.00%	0.90s	1.87s	0.00%

K	n	m	t	σ_t	diff	t	σ_t	diff	t	σ_t	err	diff	t	σ_t	diff	t	σ_t	diff
3	50	3	0.09s	0.01s	0.00%	0.04s	0.01s	0.00%	4282.30s	317.32s	28	2.62%	4935.95s	270.49s	0.00%	2.64s	8.75s	0.00%
3	50	5	0.48s	0.04s	0.00%	0.06s	0.01s	0.00%	4987.57s	1.05s	25	3.59%	4988.46s	1.31s	0.00%	4.38s	11.81s	0.00%
3	100	2	0.19s	0.02s	0.00%	0.10s	0.02s	0.00%	N/A	N/A	30	N/A	4965.70s	21.06s	0.00%	7.74s	20.70s	0.02%
3	100	3	0.51s	0.05s	0.00%	0.16s	0.02s	0.00%	N/A	N/A	30	N/A	4984.43s	4.53s	0.01%	5.74s	9.13s	0.00%
3	100	5	2.37s	0.21s	0.00%	0.28s	0.05s	0.00%	N/A	N/A	30	N/A	4987.57s	1.27s	0.03%	28.15s	74.30s	0.00%
3	200	2	1.67s	0.18s	0.00%	0.76s	0.18s	0.00%	4965.24s	16.19s	28	50.31%	4966.39s	20.18s	0.00%	230.30s	422.91s	0.28%
3	200	3	4.70s	0.46s	0.00%	1.09s	0.33s	0.00%	5009.91s	3.82s	28	99.90%	4982.78s	4.44s	0.06%	126.32s	316.00s	0.00%
3	200	5	17.11s	0.91s	0.00%	1.99s	0.49s	0.00%	N/A	N/A	30	N/A	4987.48s	1.73s	0.16%	249.27s	452.84s	0.00%
3	300	2	6.98s	0.73s	0.00%	2.67s	0.75s	0.00%	N/A	N/A	30	N/A	4966.62s	20.03s	0.00%	588.26s	1218.54s	0.18%
3	300	3	19.96s	1.21s	0.00%	4.55s	1.33s	0.00%	N/A	N/A	30	N/A	4983.46s	4.18s	0.06%	1700.03s	2021.05s	0.28%
3	300	5	66.08s	3.00s	1.57%	7.91s	2.32s	1.57%	5022.89s	87.07s	29	0.00%	4984.24s	4.21s	1.73%	711.06s	1276.55s	1.57%
5	2	2	0.01s	0.00s	0.02%	0.01s	0.00s	0.06%	0.46s	0.47s	0	0.00%	0.03s	0.02s	0.06%	0.01s	0.00s	0.01%
5	2	3	0.01s	0.00s	0.02%	0.01s	0.00s	0.01%	0.89s	0.58s	0	0.01%	0.03s	0.02s	0.01%	0.01s	0.00s	0.00%
5	2	5	0.01s	0.00s	0.01%	0.01s	0.00s	0.04%	1.58s	0.77s	0	0.00%	0.07s	0.05s	0.04%	0.02s	0.00s	0.00%
5	5	2	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	9.92s	4.36s	0	0.20%	0.17s	0.05s	0.01%	0.02s	0.01s	0.01%
5	5	3	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	18.02s	2.33s	0	0.05%	0.36s	0.05s	0.01%	0.02s	0.01s	0.00%
5	5	5	0.02s	0.00s	0.01%	0.01s	0.00s	0.03%	24.60s	6.00s	0	0.37%	0.54s	0.08s	0.01%	0.02s	0.01s	0.01%
5	10	2	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	94.24s	8.17s	2	0.13%	0.40s	0.07s	0.01%	0.05s	0.06s	0.01%
5	10	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	113.01s	19.22s	6	0.51%	0.89s	0.32s	0.00%	0.05s	0.09s	0.00%
5	10	5	0.05s	0.01s	0.00%	0.01s	0.00s	0.00%	166.78s	32.43s	3	0.51%	8.02s	5.49s	0.00%	0.04s	0.03s	0.00%
5	20	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	688.23s	148.87s	21	1.07%	3.02s	1.63s	0.00%	0.07s	0.06s	0.00%
5	20	3	0.03s	0.00s	0.00%	0.01s	0.00s	0.00%	735.80s	48.32s	17	1.48%	64.46s	72.25s	0.00%	0.17s	0.19s	0.00%
5	20	5	0.16s	0.01s	0.00%	0.02s	0.00s	0.00%	989.43s	81.16s	22	2.62%	4209.90s	1499.16s	0.00%	0.37s	0.92s	0.00%
5	50	2	0.06s	0.01s	0.00%	0.04s	0.01s	0.00%	N/A	N/A	30	N/A	4703.48s	1031.89s	0.00%	5.51s	15.54s	0.00%
5	50	3	0.16s	0.02s	0.00%	0.06s	0.01s	0.00%	N/A	N/A	30	N/A	4984.52s	4.39s	0.00%	1.10s	1.08s	0.00%
5	50	5	0.89s	0.07s	0.00%	0.09s	0.01s	0.00%	N/A	N/A	30	N/A	4987.82s	1.46s	0.01%	8.12s	19.71s	0.00%
5	100	2	0.33s	0.03s	0.00%	0.17s	0.02s	0.00%	4989.03s	N/A	29	0.79%	4965.27s	22.18s	0.01%	8.77s	17.46s	0.02%
5	100	3	0.95s	0.08s	0.00%	0.27s	0.04s	0.00%	4991.08s	5.48s	28	1.15%	4984.10s	4.14s	0.03%	11.93s	12.98s	0.00%
5	100	5	4.30s	0.33s	0.00%	0.55s	0.09s	0.00%	4990.06s	2.93s	28	1.55%	4987.39s	1.62s	0.13%	41.80s	88.64s	0.00%
5	200	2	2.81s	0.32s	0.00%	1.25s	0.30s	0.00%	N/A	N/A	30	N/A	4967.77s	19.81s	0.00%	310.89s	626.46s	0.21%
5	200	3	8.31s	0.83s	0.00%	2.14s	0.48s	0.00%	N/A	N/A	30	N/A	4983.55s	4.02s	0.10%	500.51s	882.03s	0.16%
5	200	5	30.06s	1.26s	0.00%	4.00s	0.80s	0.00%	N/A	N/A	30	N/A	4989.20s	3.27s	0.25%	693.50s	1188.73s	0.00%
5	300	2	12.03s	1.34s	0.00%	4.78s	1.22s	0.00%	N/A	N/A	30	N/A	4970.98s	17.98s	0.00%	754.66s	1700.54s	0.13%
5	300	3	34.66s	1.99s	0.00%	8.71s	1.92s	0.00%	5000.49s	15.81s	30	N/A	4983.03s	4.58s	0.10%	2378.41s	2499.40s	0.25%
5	300	5	117.77s	4.50s	0.00%	16.28s	3.51s	0.00%	4990.12s	3.73s	30	N/A	4989.14s	0.99s	0.29%	1889.33s	2165.58s	0.24%

Table A.2: CMDP algorithms for $\gamma = 0.9$ on deterministic models

Parameters			CPolicyOpt, Alg. 13			CPurePolicyOpt, Alg. 14			QCLP (3.4)				MIP (3.9)			NLP (3.5), (3.8)		
K	n	m	t	σ_t	diff	t	σ_t	diff	t	σ_t	err	diff	t	σ_t	diff	t	σ_t	diff
2	2	2	0.01s	0.00s	0.14%	0.01s	0.00s	0.26%	0.10s	0.16s	0	0.19%	0.02s	0.01s	0.26%	0.01s	0.00s	0.13%
2	2	3	0.01s	0.00s	0.19%	0.01s	0.00s	1.32%	0.12s	0.16s	0	0.00%	0.02s	0.01s	1.32%	0.01s	0.01s	0.03%
2	2	5	0.01s	0.00s	0.09%	0.01s	0.00s	0.15%	0.41s	0.24s	0	0.00%	0.03s	0.02s	0.15%	0.02s	0.01s	0.07%
2	5	2	0.01s	0.00s	0.93%	0.01s	0.00s	0.96%	1.21s	0.81s	0	0.11%	0.03s	0.03s	0.77%	0.02s	0.00s	0.48%
2	5	3	0.01s	0.00s	0.27%	0.01s	0.00s	1.07%	3.11s	1.43s	0	0.09%	0.06s	0.06s	0.40%	0.02s	0.01s	0.08%
2	5	5	0.01s	0.00s	0.46%	0.01s	0.00s	1.06%	6.99s	1.53s	0	0.59%	0.13s	0.07s	0.53%	0.03s	0.01s	0.37%
2	10	2	0.01s	0.00s	0.21%	0.01s	0.00s	1.28%	18.34s	1.78s	0	0.52%	0.08s	0.05s	0.51%	0.02s	0.01s	0.16%
2	10	3	0.01s	0.00s	0.09%	0.01s	0.00s	0.27%	25.95s	3.47s	0	0.58%	0.24s	0.11s	0.07%	0.03s	0.01s	0.40%
2	10	5	0.01s	0.00s	0.19%	0.01s	0.00s	0.76%	44.51s	5.50s	0	2.17%	0.31s	0.09s	0.19%	0.06s	0.14s	0.11%
2	20	2	0.01s	0.00s	0.13%	0.01s	0.00s	1.16%	128.16s	9.53s	0	1.35%	0.26s	0.07s	0.10%	0.24s	1.11s	0.37%
2	20	3	0.01s	0.00s	0.29%	0.01s	0.00s	0.72%	177.28s	26.70s	0	1.87%	0.43s	0.08s	0.09%	0.04s	0.01s	0.12%
2	20	5	0.03s	0.00s	0.47%	0.01s	0.00s	0.63%	281.89s	19.64s	0	3.81%	1.16s	0.68s	0.03%	0.91s	2.78s	0.30%
2	50	2	0.02s	0.00s	1.02%	0.01s	0.00s	1.17%	1228.97s	276.58s	17	16.15%	1.11s	0.59s	0.03%	0.83s	3.63s	0.30%
2	50	3	0.05s	0.01s	0.39%	0.01s	0.00s	0.70%	1838.67s	217.44s	13	14.90%	20.11s	36.63s	0.01%	0.17s	0.10s	0.43%
2	50	5	0.18s	0.02s	0.44%	0.02s	0.01s	1.08%	4510.64s	383.20s	12	29.56%	2704.16s	2036.70s	0.01%	0.26s	0.14s	0.70%
2	100	2	0.12s	0.02s	0.50%	0.05s	0.02s	1.13%	4972.13s	3.98s	27	27.61%	41.82s	110.17s	0.02%	3.78s	16.81s	0.39%
2	100	3	0.35s	0.05s	0.35%	0.09s	0.03s	0.68%	4986.96s	N/A	29	35.47%	3551.61s	2109.59s	0.00%	4.81s	20.13s	0.41%
2	100	5	1.37s	0.10s	0.52%	0.18s	0.05s	0.41%	4991.09s	0.70s	27	40.25%	4982.85s	13.31s	0.00%	16.60s	43.22s	0.41%
2	200	2	1.45s	0.23s	0.37%	0.53s	0.23s	0.59%	N/A	N/A	30	N/A	4384.69s	1505.36s	0.00%	33.61s	110.80s	0.32%
2	200	3	5.06s	0.65s	0.47%	1.10s	0.42s	0.78%	N/A	N/A	30	N/A	4987.55s	2.75s	0.01%	82.29s	198.82s	0.34%
2	200	5	13.73s	0.71s	0.47%	1.81s	0.64s	0.49%	N/A	N/A	30	N/A	4986.54s	1.82s	0.00%	115.10s	257.24s	0.46%
2	300	2	7.05s	0.83s	0.44%	3.41s	2.16s	0.64%	N/A	N/A	30	N/A	4961.23s	12.35s	0.00%	401.08s	705.86s	0.23%
2	300	3	25.00s	2.12s	0.64%	5.61s	2.12s	0.79%	N/A	N/A	30	N/A	4986.36s	3.36s	0.00%	898.27s	1114.78s	0.25%
2	300	5	57.99s	2.62s	0.51%	9.07s	2.88s	0.41%	N/A	N/A	30	N/A	4884.58s	538.21s	0.00%	599.02s	1045.03s	0.34%
3	2	2	0.00s	0.00s	0.34%	0.01s	0.00s	0.80%	0.14s	0.24s	0	0.01%	0.02s	0.01s	0.80%	0.02s	0.01s	0.30%
3	2	3	0.01s	0.00s	0.57%	0.01s	0.00s	0.95%	0.26s	0.19s	0	0.04%	0.03s	0.02s	0.93%	0.02s	0.01s	0.27%
3	2	5	0.01s	0.00s	0.13%	0.01s	0.00s	0.73%	0.53s	0.22s	0	0.04%	0.03s	0.02s	0.73%	0.02s	0.01s	0.01%
3	5	2	0.01s	0.00s	0.55%	0.01s	0.00s	1.50%	1.97s	1.05s	0	0.63%	0.05s	0.03s	1.45%	0.02s	0.01s	0.28%
3	5	3	0.01s	0.01s	0.51%	0.01s	0.00s	1.44%	6.08s	1.87s	0	0.20%	0.14s	0.10s	1.18%	0.02s	0.01s	0.23%
3	5	5	0.01s	0.01s	1.28%	0.01s	0.00s	1.19%	11.30s	2.45s	0	0.84%	0.30s	0.10s	0.57%	0.03s	0.01s	0.41%
3	10	2	0.01s	0.00s	1.07%	0.01s	0.00s	2.14%	36.66s	2.27s	0	1.30%	0.18s	0.07s	0.67%	0.03s	0.01s	0.71%
3	10	3	0.01s	0.00s	0.67%	0.01s	0.00s	2.59%	46.82s	7.90s	0	2.00%	0.37s	0.07s	0.39%	0.03s	0.01s	0.69%
3	10	5	0.02s	0.01s	1.28%	0.01s	0.00s	1.79%	75.97s	9.70s	0	2.97%	0.56s	0.15s	0.04%	0.04s	0.01s	1.10%
3	20	2	0.01s	0.00s	0.76%	0.01s	0.00s	1.45%	247.40s	39.87s	1	3.25%	0.39s	0.10s	0.12%	0.03s	0.01s	0.35%
3	20	3	0.02s	0.00s	1.35%	0.01s	0.00s	1.58%	309.74s	65.68s	2	6.07%	1.05s	0.54s	0.12%	0.05s	0.02s	0.69%
3	20	5	0.06s	0.01s	1.05%	0.01s	0.00s	1.16%	484.39s	75.57s	2	9.11%	13.70s	23.53s	0.08%	0.40s	1.82s	0.67%
3	50	2	0.03s	0.01s	0.96%	0.01s	0.01s	1.71%	2648.98s	908.89s	26	29.92%	20.17s	25.13s	0.05%	0.17s	0.10s	0.46%

K	n	m	t	σ_t	diff	t	σ_t	diff	t	σ_t	err	diff	t	σ_t	diff	t	σ_t	diff
3	50	3	0.08s	0.01s	1.17%	0.02s	0.00s	1.12%	3590.40s	483.84s	25	32.77%	1757.35s	1928.38s	0.01%	1.34s	5.88s	0.52%
3	50	5	0.42s	0.02s	0.98%	0.04s	0.01s	1.06%	4986.71s	0.99s	24	38.64%	4905.61s	453.92s	0.02%	1.37s	5.46s	0.88%
3	100	2	0.19s	0.03s	0.58%	0.09s	0.03s	1.28%	N/A	N/A	30	N/A	4466.86s	1244.72s	0.01%	1.46s	1.68s	0.50%
3	100	3	0.65s	0.10s	1.20%	0.16s	0.05s	1.44%	N/A	N/A	30	N/A	4985.92s	3.58s	0.01%	7.37s	29.01s	0.77%
3	100	5	2.52s	0.22s	0.72%	0.29s	0.10s	1.01%	N/A	N/A	30	N/A	4988.36s	2.59s	0.01%	17.91s	47.81s	0.90%
3	200	2	2.33s	0.32s	0.79%	1.37s	0.39s	0.97%	N/A	N/A	30	N/A	4957.38s	14.41s	0.00%	54.00s	175.51s	0.40%
3	200	3	8.48s	0.84s	1.20%	2.32s	0.63s	1.24%	N/A	N/A	30	N/A	4986.40s	3.24s	0.01%	141.53s	334.81s	0.83%
3	200	5	22.17s	1.15s	0.76%	3.78s	0.91s	0.99%	N/A	N/A	30	N/A	4985.35s	1.94s	0.00%	371.15s	594.26s	0.65%
3	300	2	11.78s	1.17s	0.87%	6.38s	2.17s	0.87%	N/A	N/A	30	N/A	4964.27s	11.91s	0.01%	423.54s	767.27s	0.57%
3	300	3	39.38s	2.77s	0.85%	10.93s	2.13s	1.24%	N/A	N/A	30	N/A	4985.92s	3.34s	0.03%	1053.02s	1531.67s	0.53%
3	300	5	95.03s	4.86s	0.88%	20.16s	4.65s	0.96%	N/A	N/A	30	N/A	4981.78s	2.41s	0.00%	1320.11s	1787.25s	0.67%
5	2	2	0.01s	0.00s	0.05%	0.01s	0.00s	0.72%	0.42s	0.32s	0	0.06%	0.03s	0.03s	0.52%	0.01s	0.00s	0.04%
5	2	3	0.01s	0.00s	0.05%	0.01s	0.00s	1.10%	0.56s	0.33s	0	0.05%	0.03s	0.02s	1.10%	0.02s	0.00s	0.04%
5	2	5	0.01s	0.00s	0.01%	0.01s	0.00s	0.87%	1.25s	0.66s	0	0.35%	0.07s	0.06s	0.87%	0.02s	0.01s	0.30%
5	5	2	0.01s	0.00s	0.16%	0.01s	0.00s	1.64%	8.26s	3.41s	0	1.25%	0.09s	0.05s	1.01%	0.02s	0.00s	0.19%
5	5	3	0.01s	0.00s	0.78%	0.01s	0.00s	2.95%	15.96s	4.59s	0	0.85%	0.28s	0.08s	1.64%	0.03s	0.01s	0.69%
5	5	5	0.03s	0.01s	0.90%	0.01s	0.00s	2.78%	25.15s	2.09s	0	0.93%	0.47s	0.08s	0.85%	0.03s	0.01s	0.48%
5	10	2	0.01s	0.00s	0.89%	0.01s	0.00s	2.48%	94.82s	15.40s	3	2.12%	0.30s	0.09s	1.07%	0.03s	0.01s	0.54%
5	10	3	0.02s	0.01s	0.58%	0.01s	0.00s	4.22%	118.39s	18.87s	0	5.79%	0.59s	0.14s	0.84%	0.04s	0.01s	0.87%
5	10	5	0.06s	0.01s	0.97%	0.01s	0.00s	2.17%	165.02s	32.18s	2	6.29%	2.85s	1.38s	0.50%	0.05s	0.02s	1.28%
5	20	2	0.01s	0.01s	1.25%	0.01s	0.00s	2.52%	565.03s	120.32s	19	6.05%	0.80s	0.31s	0.27%	0.05s	0.02s	0.59%
5	20	3	0.04s	0.01s	1.00%	0.01s	0.00s	2.56%	769.73s	106.37s	20	11.75%	13.19s	13.75s	0.19%	0.07s	0.01s	0.74%
5	20	5	0.18s	0.03s	1.72%	0.01s	0.00s	2.33%	1110.30s	86.86s	17	15.24%	480.61s	625.97s	0.04%	0.11s	0.05s	0.82%
5	50	2	0.06s	0.02s	1.22%	0.02s	0.01s	2.64%	N/A	N/A	30	N/A	679.38s	817.30s	0.01%	0.25s	0.11s	0.87%
5	50	3	0.20s	0.03s	1.43%	0.03s	0.01s	2.11%	4985.40s	N/A	29	29.38%	4985.93s	4.40s	0.01%	0.31s	0.08s	1.39%
5	50	5	1.10s	0.10s	1.86%	0.06s	0.01s	1.89%	N/A	N/A	30	N/A	4986.73s	0.85s	0.06%	1.07s	2.41s	1.55%
5	100	2	0.35s	0.05s	1.02%	0.15s	0.05s	1.57%	N/A	N/A	30	N/A	4923.34s	94.52s	0.02%	2.93s	4.08s	0.71%
5	100	3	1.30s	0.14s	1.51%	0.29s	0.08s	1.68%	N/A	N/A	30	N/A	4984.48s	3.34s	0.02%	3.33s	1.31s	1.21%
5	100	5	5.53s	0.26s	1.28%	0.58s	0.10s	1.59%	N/A	N/A	30	N/A	4986.74s	1.34s	0.04%	7.29s	6.46s	0.98%
5	200	2	4.04s	0.42s	1.16%	2.38s	0.85s	1.39%	N/A	N/A	30	N/A	4958.96s	14.56s	0.02%	40.43s	28.92s	0.65%
5	200	3	15.12s	1.38s	1.18%	4.30s	1.04s	1.89%	N/A	N/A	30	N/A	4985.67s	2.95s	0.04%	70.17s	46.28s	0.77%
5	200	5	41.73s	1.89s	1.12%	7.46s	1.31s	1.41%	N/A	N/A	30	N/A	4984.59s	1.73s	0.05%	185.70s	222.72s	0.98%
5	300	2	20.08s	1.82s	0.94%	11.96s	2.91s	1.65%	N/A	N/A	30	N/A	4958.07s	15.02s	0.00%	493.13s	887.17s	0.74%
5	300	3	69.09s	4.86s	1.18%	22.25s	4.01s	1.59%	N/A	N/A	30	N/A	4985.30s	2.95s	0.10%	1703.45s	2032.93s	0.73%
5	300	5	170.09s	8.94s	1.08%	37.71s	6.28s	1.23%	N/A	N/A	30	N/A	4986.54s	1.31s	0.02%	1579.82s	1972.90s	0.73%

Table A.3: CMDP algorithms for $\gamma = 0.999$ on stochastic models

Parameters			CPolicyOpt, Alg. 13			CPurePolicyOpt, Alg. 14			QCLP (3.4)				MIP (3.9)			NLP (3.5), (3.8)		
K	n	m	t	σ_t	diff	t	σ_t	diff	t	σ_t	err	diff	t	σ_t	diff	t	σ_t	diff
2	2	2	0.01s	0.00s	0.01%	0.01s	0.00s	0.11%	0.19s	0.16s	0	13.56%	0.02s	0.01s	0.11%	0.01s	0.00s	0.00%
2	2	3	0.01s	0.00s	0.03%	0.01s	0.00s	0.04%	0.43s	0.18s	0	4.10%	0.03s	0.01s	0.04%	0.02s	0.00s	0.00%
2	2	5	0.01s	0.00s	0.02%	0.01s	0.00s	0.02%	0.75s	0.21s	0	7.60%	0.04s	0.03s	0.02%	0.02s	0.01s	0.01%
2	5	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	1.73s	0.93s	0	2.69%	0.04s	0.02s	0.00%	0.02s	0.00s	0.00%
2	5	3	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	3.08s	1.34s	0	11.55%	0.10s	0.06s	0.01%	0.02s	0.00s	0.00%
2	5	5	0.01s	0.00s	0.00%	0.01s	0.00s	0.01%	7.08s	1.67s	0	3.20%	0.21s	0.09s	0.00%	0.02s	0.01s	0.02%
2	10	2	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	18.80s	5.24s	0	0.13%	0.13s	0.09s	0.01%	0.02s	0.00s	0.01%
2	10	3	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	28.72s	4.64s	0	0.51%	0.28s	0.10s	0.01%	0.02s	0.01s	0.01%
2	10	5	0.01s	0.00s	0.01%	0.01s	0.00s	0.00%	50.50s	13.38s	0	0.75%	0.44s	0.12s	0.00%	0.02s	0.00s	0.00%
2	20	2	0.01s	0.00s	0.05%	0.01s	0.00s	0.05%	142.11s	40.90s	0	0.10%	0.41s	0.09s	0.05%	1.48s	7.99s	0.05%
2	20	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	211.66s	56.54s	3	0.22%	1.10s	1.33s	0.00%	0.03s	0.00s	0.00%
2	20	5	0.03s	0.00s	0.00%	0.01s	0.00s	0.00%	403.56s	72.55s	0	0.21%	9.08s	9.00s	0.00%	0.04s	0.01s	0.00%
2	50	2	0.03s	0.00s	0.03%	0.02s	0.00s	0.03%	1768.33s	321.01s	23	0.14%	11.26s	14.93s	0.03%	0.09s	0.11s	0.03%
2	50	3	0.05s	0.01s	0.03%	0.03s	0.01s	0.03%	3583.94s	520.03s	22	0.06%	1509.47s	1979.04s	0.03%	0.11s	0.10s	0.03%
2	50	5	0.21s	0.01s	0.00%	0.04s	0.01s	0.00%	4908.50s	221.20s	22	0.51%	3889.11s	1962.86s	0.00%	0.15s	0.09s	0.00%
2	100	2	0.13s	0.01s	0.02%	0.07s	0.01s	0.02%	4988.81s	1.36s	23	0.53%	4159.98s	1822.15s	0.03%	0.64s	1.51s	0.02%
2	100	3	0.31s	0.03s	0.00%	0.10s	0.02s	0.00%	4986.25s	3.93s	24	0.65%	4513.03s	1453.39s	0.00%	0.70s	0.78s	0.00%
2	100	5	1.20s	0.08s	0.00%	0.18s	0.05s	0.00%	4986.28s	3.91s	28	2.90%	4845.01s	783.31s	0.01%	1.24s	1.38s	0.00%
2	200	2	1.28s	0.17s	0.00%	0.41s	0.13s	0.00%	4995.82s	4.71s	26	1.04%	4599.40s	1210.58s	0.00%	2.99s	4.33s	0.00%
2	200	3	3.26s	0.29s	0.00%	0.70s	0.28s	0.00%	5001.21s	12.13s	22	1.64%	4662.92s	1233.65s	0.01%	5.34s	5.96s	0.00%
2	200	5	9.65s	0.43s	0.00%	1.18s	0.48s	0.00%	5038.45s	35.34s	26	2.11%	4827.51s	873.36s	0.02%	6.75s	6.87s	0.00%
2	300	2	5.47s	0.63s	0.00%	1.38s	0.57s	0.00%	5021.09s	20.36s	20	0.91%	4666.60s	1198.98s	0.00%	9.81s	11.25s	0.00%
2	300	3	13.89s	1.20s	0.00%	2.65s	1.12s	0.00%	5063.96s	37.92s	25	1.20%	4985.65s	1.30s	0.01%	14.36s	16.46s	0.00%
2	300	5	38.51s	2.07s	0.00%	4.14s	1.98s	0.00%	5040.03s	37.89s	20	9.20%	4982.83s	5.59s	0.03%	28.32s	36.84s	0.00%
3	2	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.01%	0.32s	0.22s	0	18.53%	0.03s	0.04s	0.01%	0.01s	0.00s	0.00%
3	2	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.02%	0.56s	0.30s	0	14.26%	0.03s	0.02s	0.02%	0.02s	0.00s	0.00%
3	2	5	0.01s	0.00s	0.00%	0.01s	0.00s	0.05%	1.17s	0.30s	0	4.40%	0.05s	0.03s	0.05%	0.02s	0.01s	0.25%
3	5	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	3.50s	2.00s	0	2.26%	0.08s	0.04s	0.00%	0.02s	0.00s	0.00%
3	5	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	6.75s	1.90s	0	0.61%	0.20s	0.07s	0.00%	0.02s	0.00s	0.00%
3	5	5	0.01s	0.00s	0.03%	0.01s	0.00s	0.05%	11.06s	3.07s	0	2.34%	0.34s	0.07s	0.01%	0.02s	0.01s	0.02%
3	10	2	0.01s	0.00s	0.02%	0.01s	0.00s	0.02%	39.60s	10.51s	2	0.15%	0.28s	0.07s	0.02%	0.02s	0.00s	0.02%
3	10	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	58.55s	12.85s	1	0.19%	0.43s	0.07s	0.00%	0.02s	0.00s	0.00%
3	10	5	0.02s	0.00s	0.01%	0.01s	0.00s	0.01%	95.13s	15.03s	5	0.20%	1.33s	1.20s	0.00%	0.03s	0.01s	0.01%
3	20	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	290.71s	58.66s	9	0.14%	0.71s	0.25s	0.00%	0.02s	0.01s	0.00%
3	20	3	0.02s	0.00s	0.00%	0.01s	0.00s	0.00%	382.45s	99.49s	9	0.27%	6.85s	6.44s	0.00%	0.03s	0.01s	0.00%
3	20	5	0.07s	0.01s	0.00%	0.02s	0.00s	0.00%	655.95s	180.59s	14	0.21%	286.39s	431.43s	0.00%	0.04s	0.01s	0.00%
3	50	2	0.04s	0.01s	0.04%	0.02s	0.00s	0.04%	3904.61s	303.57s	26	0.01%	1281.28s	1846.00s	0.04%	0.08s	0.02s	0.04%

K	n	m	t	σ_t	diff	t	σ_t	diff	t	σ_t	err	diff	t	σ_t	diff	t	σ_t	diff
3	50	3	0.10s	0.01s	0.00%	0.03s	0.01s	0.00%	4988.79s	1.20s	26	0.32%	4433.82s	1501.25s	0.00%	0.12s	0.03s	0.00%
3	50	5	0.49s	0.04s	0.00%	0.06s	0.01s	0.00%	4986.91s	4.19s	23	0.46%	4988.96s	0.74s	0.00%	0.23s	0.26s	0.00%
3	100	2	0.21s	0.02s	0.00%	0.10s	0.01s	0.00%	N/A	N/A	30	N/A	4853.40s	709.60s	0.00%	0.71s	0.93s	0.00%
3	100	3	0.58s	0.05s	0.00%	0.16s	0.03s	0.00%	4995.73s	N/A	29	1.21%	4988.10s	0.98s	0.00%	0.99s	1.03s	0.00%
3	100	5	2.47s	0.14s	0.00%	0.31s	0.04s	0.00%	N/A	N/A	30	N/A	4987.61s	1.06s	0.04%	2.11s	1.79s	0.00%
3	200	2	1.99s	0.25s	0.00%	0.70s	0.16s	0.00%	5012.92s	N/A	29	0.83%	4982.47s	3.56s	0.00%	9.21s	17.60s	0.00%
3	200	3	5.11s	0.43s	0.00%	1.16s	0.28s	0.00%	5010.28s	20.05s	27	1.26%	4987.68s	0.90s	0.06%	7.00s	7.71s	0.00%
3	200	5	17.11s	0.68s	0.00%	2.09s	0.60s	0.00%	5071.77s	5.09s	28	1.74%	4986.92s	3.14s	0.12%	79.41s	298.69s	0.00%
3	300	2	8.52s	0.76s	0.01%	2.72s	0.78s	0.01%	5034.59s	35.23s	23	0.60%	4981.22s	3.70s	0.01%	60.37s	99.73s	0.01%
3	300	3	22.56s	2.00s	0.40%	4.63s	1.29s	0.40%	5089.28s	80.84s	25	0.88%	4987.62s	0.94s	0.44%	23.49s	24.23s	0.40%
3	300	5	63.77s	2.82s	0.00%	9.40s	2.78s	0.00%	5042.46s	93.05s	18	26.16%	4980.54s	6.59s	0.26%	64.32s	77.07s	0.00%
5	2	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.06%	0.50s	0.28s	0	10.56%	0.03s	0.02s	0.06%	0.01s	0.00s	0.00%
5	2	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.11%	0.86s	0.43s	0	2.33%	0.05s	0.03s	0.11%	0.02s	0.00s	0.02%
5	2	5	0.01s	0.00s	0.01%	0.01s	0.00s	0.08%	1.83s	0.72s	0	3.96%	0.09s	0.05s	0.08%	0.02s	0.00s	0.08%
5	5	2	0.01s	0.00s	0.01%	0.01s	0.00s	0.02%	11.97s	5.18s	2	0.02%	0.13s	0.06s	0.02%	0.02s	0.00s	0.01%
5	5	3	0.01s	0.00s	0.00%	0.01s	0.00s	0.01%	19.32s	4.60s	3	0.07%	0.31s	0.06s	0.01%	0.02s	0.00s	0.00%
5	5	5	0.02s	0.00s	0.03%	0.01s	0.00s	0.01%	26.28s	4.63s	3	0.80%	0.47s	0.06s	0.01%	0.02s	0.00s	0.02%
5	10	2	0.01s	0.00s	0.01%	0.01s	0.00s	0.01%	113.23s	15.79s	8	0.03%	0.39s	0.05s	0.01%	0.02s	0.00s	0.01%
5	10	3	0.02s	0.00s	0.00%	0.01s	0.00s	0.00%	137.64s	22.04s	14	0.09%	0.83s	0.21s	0.00%	0.02s	0.00s	0.00%
5	10	5	0.07s	0.02s	0.01%	0.01s	0.00s	0.01%	209.59s	29.55s	14	0.15%	7.38s	6.34s	0.00%	0.03s	0.00s	0.00%
5	20	2	0.02s	0.00s	0.00%	0.01s	0.00s	0.00%	767.41s	231.43s	18	0.03%	2.17s	0.94s	0.00%	0.03s	0.01s	0.00%
5	20	3	0.04s	0.01s	0.02%	0.01s	0.00s	0.02%	1085.76s	65.13s	22	0.03%	92.28s	67.47s	0.02%	0.04s	0.01s	0.02%
5	20	5	0.20s	0.02s	0.02%	0.02s	0.00s	0.02%	1497.93s	324.21s	23	0.45%	3565.01s	1808.88s	0.02%	0.06s	0.01s	0.02%
5	50	2	0.07s	0.01s	0.00%	0.04s	0.00s	0.00%	4986.72s	N/A	29	0.09%	4871.32s	451.94s	0.00%	0.12s	0.02s	0.00%
5	50	3	0.18s	0.02s	0.00%	0.06s	0.01s	0.00%	4988.84s	1.61s	28	0.37%	4987.87s	1.09s	0.00%	0.19s	0.07s	0.00%
5	50	5	1.00s	0.05s	0.00%	0.10s	0.01s	0.00%	N/A	N/A	30	N/A	4988.21s	1.22s	0.01%	0.40s	0.30s	0.00%
5	100	2	0.39s	0.04s	0.00%	0.17s	0.02s	0.00%	N/A	N/A	30	N/A	4982.34s	3.65s	0.00%	1.12s	1.43s	0.00%
5	100	3	1.01s	0.09s	0.04%	0.28s	0.05s	0.04%	4991.07s	N/A	29	0.00%	4987.65s	0.83s	0.06%	1.62s	1.23s	0.04%
5	100	5	4.50s	0.31s	0.00%	0.52s	0.06s	0.00%	4984.09s	N/A	29	1.85%	4987.54s	1.34s	0.16%	4.64s	4.92s	0.00%
5	200	2	3.26s	0.43s	0.00%	1.30s	0.22s	0.00%	N/A	N/A	30	N/A	4982.64s	3.95s	0.04%	22.55s	37.21s	0.00%
5	200	3	8.82s	0.77s	0.00%	2.28s	0.49s	0.00%	5130.07s	N/A	29	1.00%	4987.81s	2.36s	0.11%	18.84s	23.92s	0.00%
5	200	5	29.59s	1.37s	0.00%	4.01s	0.84s	0.00%	N/A	N/A	30	N/A	4985.72s	4.25s	0.28%	32.42s	30.84s	0.00%
5	300	2	13.88s	1.12s	0.83%	4.68s	1.04s	0.83%	5152.61s	15.06s	28	0.00%	4984.19s	4.10s	0.88%	24.41s	21.61s	0.83%
5	300	3	36.22s	2.31s	10.26%	8.78s	1.88s	10.26%	5078.68s	128.00s	19	0.00%	4987.13s	1.51s	10.34%	302.16s	942.93s	10.26%
5	300	5	108.95s	4.72s	0.72%	16.04s	3.26s	0.72%	4993.12s	2.05s	13	55.31%	4979.26s	6.12s	1.15%	182.41s	305.18s	0.72%

Table A.4: CMDP algorithms for $\gamma = 0.999$ on deterministic models

Parameters			CPolicyOpt, Alg. 13			CPurePolicyOpt, Alg. 14			QCLP (3.4)				MIP (3.9)			NLP (3.5), (3.8)		
K	n	m	t	σ_t	diff	t	σ_t	diff	t	σ_t	err	diff	t	σ_t	diff	t	σ_t	diff
2	2	2	0.01s	0.00s	0.00%	0.01s	0.00s	0.00%	0.35s	0.18s	0	4.75%	0.02s	0.01s	0.00%	0.01s	0.00s	0.00%
2	2	3	0.01s	0.00s	0.95%	0.01s	0.00s	1.18%	0.40s	0.19s	0	13.99%	0.03s	0.01s	1.18%	0.02s	0.01s	0.00%
2	2	5	0.01s	0.00s	0.63%	0.01s	0.00s	1.85%	0.75s	0.13s	0	10.69%	0.03s	0.02s	1.85%	0.02s	0.01s	0.01%
2	5	2	0.01s	0.00s	1.02%	0.01s	0.00s	2.06%	1.77s	0.83s	0	16.50%	0.03s	0.02s	1.39%	0.03s	0.01s	0.11%
2	5	3	0.01s	0.00s	0.55%	0.01s	0.00s	1.83%	3.23s	1.11s	0	20.25%	0.04s	0.03s	1.66%	0.03s	0.01s	0.26%
2	5	5	0.01s	0.00s	0.73%	0.01s	0.00s	1.01%	5.97s	1.56s	0	22.65%	0.06s	0.03s	0.93%	0.04s	0.01s	0.00%
2	10	2	0.01s	0.00s	1.41%	0.01s	0.00s	4.10%	16.37s	3.65s	0	14.99%	0.06s	0.04s	2.25%	0.05s	0.05s	0.02%
2	10	3	0.01s	0.00s	0.81%	0.01s	0.00s	1.89%	24.30s	3.29s	0	17.44%	0.11s	0.06s	0.81%	0.05s	0.02s	0.48%
2	10	5	0.01s	0.00s	0.52%	0.01s	0.00s	1.31%	41.76s	2.59s	0	21.28%	0.26s	0.10s	0.14%	0.07s	0.05s	0.24%
2	20	2	0.01s	0.00s	0.80%	0.01s	0.00s	1.27%	123.18s	14.34s	3	15.48%	0.12s	0.06s	0.26%	0.08s	0.09s	0.53%
2	20	3	0.01s	0.00s	0.47%	0.01s	0.00s	0.82%	158.55s	7.99s	2	20.43%	0.35s	0.09s	0.13%	0.08s	0.04s	0.55%
2	20	5	0.03s	0.01s	0.94%	0.01s	0.00s	0.82%	278.54s	11.98s	0	31.23%	0.62s	0.30s	0.00%	0.19s	0.41s	1.10%
2	50	2	0.03s	0.01s	0.52%	0.01s	0.00s	1.41%	1542.76s	138.87s	21	27.62%	0.64s	0.27s	0.03%	0.18s	0.10s	1.21%
2	50	3	0.06s	0.01s	0.96%	0.02s	0.01s	1.61%	2138.30s	135.94s	18	27.75%	3.04s	2.02s	0.00%	0.54s	1.15s	1.03%
2	50	5	0.21s	0.02s	0.29%	0.03s	0.01s	0.38%	4399.17s	461.27s	21	33.80%	123.94s	450.40s	0.02%	1.49s	5.18s	0.40%
2	100	2	0.22s	0.05s	0.18%	0.07s	0.03s	1.16%	N/A	N/A	30	N/A	176.94s	902.74s	0.01%	1.33s	0.61s	0.44%
2	100	3	0.55s	0.11s	1.19%	0.13s	0.06s	0.36%	4988.41s	N/A	29	33.52%	778.50s	1639.40s	0.00%	9.68s	27.91s	0.70%
2	100	5	1.54s	0.18s	0.20%	0.21s	0.07s	0.28%	4990.97s	2.52s	28	43.12%	3514.67s	2082.84s	0.00%	23.19s	41.85s	0.49%
2	200	2	4.04s	0.96s	0.16%	0.94s	0.37s	0.45%	N/A	N/A	30	N/A	2439.54s	2420.76s	0.00%	85.73s	199.13s	0.87%
2	200	3	8.87s	1.80s	0.36%	1.47s	0.58s	0.31%	N/A	N/A	30	N/A	3966.67s	1966.57s	0.00%	78.40s	170.08s	0.77%
2	200	5	17.38s	2.83s	0.21%	2.82s	1.07s	0.32%	N/A	N/A	30	N/A	4495.85s	1501.10s	0.00%	197.26s	241.89s	0.47%
2	300	2	21.16s	4.48s	0.18%	5.65s	1.78s	0.44%	N/A	N/A	30	N/A	4351.08s	1515.83s	0.00%	252.28s	526.35s	0.78%
2	300	3	42.79s	9.28s	0.51%	8.08s	3.38s	0.58%	N/A	N/A	30	N/A	4509.95s	1461.03s	0.00%	620.38s	856.25s	0.30%
2	300	5	71.03s	8.92s	0.43%	13.95s	4.86s	0.41%	N/A	N/A	30	N/A	4196.90s	1801.85s	0.00%	994.34s	1072.42s	1.00%
3	2	2	0.01s	0.00s	0.16%	0.01s	0.00s	0.50%	0.30s	0.21s	0	5.59%	0.03s	0.02s	0.50%	0.02s	0.01s	0.01%
3	2	3	0.01s	0.00s	0.30%	0.01s	0.00s	1.39%	0.54s	0.22s	0	11.95%	0.03s	0.02s	1.39%	0.02s	0.01s	0.00%
3	2	5	0.01s	0.00s	0.27%	0.01s	0.00s	2.10%	0.93s	0.32s	0	11.94%	0.04s	0.02s	2.10%	0.03s	0.01s	0.17%
3	5	2	0.01s	0.00s	0.17%	0.01s	0.00s	3.71%	2.87s	1.37s	0	13.17%	0.04s	0.03s	3.39%	0.03s	0.01s	0.12%
3	5	3	0.01s	0.01s	0.97%	0.01s	0.00s	5.11%	6.25s	1.45s	0	16.23%	0.08s	0.04s	3.08%	0.04s	0.03s	1.23%
3	5	5	0.02s	0.02s	0.96%	0.01s	0.00s	3.68%	10.28s	1.68s	0	20.53%	0.20s	0.09s	2.55%	0.05s	0.03s	0.75%
3	10	2	0.01s	0.01s	0.38%	0.01s	0.00s	6.19%	34.63s	7.93s	0	14.25%	0.11s	0.05s	3.00%	0.07s	0.11s	0.66%
3	10	3	0.02s	0.03s	1.82%	0.01s	0.00s	3.54%	45.03s	7.57s	0	13.39%	0.29s	0.09s	2.03%	0.08s	0.05s	0.59%
3	10	5	0.05s	0.02s	1.04%	0.01s	0.00s	3.40%	68.87s	5.67s	2	25.75%	0.50s	0.15s	1.12%	0.14s	0.19s	0.32%
3	20	2	0.03s	0.01s	1.23%	0.01s	0.00s	4.89%	250.55s	34.10s	13	16.12%	0.34s	0.07s	1.63%	0.12s	0.10s	1.00%
3	20	3	0.04s	0.02s	1.39%	0.01s	0.00s	3.55%	306.79s	26.10s	8	23.32%	0.98s	0.36s	0.24%	0.12s	0.08s	1.40%
3	20	5	0.13s	0.07s	1.04%	0.01s	0.00s	0.94%	488.24s	12.38s	9	30.90%	2.67s	2.87s	0.09%	0.17s	0.15s	1.20%
3	50	2	0.13s	0.08s	0.83%	0.01s	0.00s	2.10%	4020.56s	25.94s	28	27.86%	3.57s	3.26s	0.21%	1.06s	1.68s	0.76%

K	n	m	t	σ_t	diff	t	σ_t	diff	t	σ_t	err	diff	t	σ_t	diff	t	σ_t	diff
3	50	3	0.25s	0.11s	1.42%	0.02s	0.01s	1.70%	4442.99s	N/A	29	28.42%	346.99s	906.95s	0.04%	0.57s	0.42s	1.14%
3	50	5	0.74s	0.21s	0.44%	0.05s	0.01s	0.74%	4985.23s	5.53s	27	34.98%	4249.93s	1653.49s	0.03%	1.41s	1.77s	1.16%
3	100	2	0.88s	0.56s	0.94%	0.11s	0.03s	2.05%	N/A	N/A	30	N/A	2635.57s	2289.28s	0.01%	3.25s	1.65s	0.77%
3	100	3	1.70s	0.63s	0.94%	0.23s	0.07s	1.12%	N/A	N/A	30	N/A	4819.88s	897.47s	0.00%	5.55s	4.93s	1.67%
3	100	5	3.58s	0.73s	1.13%	0.40s	0.08s	0.74%	N/A	N/A	30	N/A	4987.66s	1.93s	0.00%	10.15s	7.91s	1.37%
3	200	2	11.23s	3.73s	0.97%	1.72s	0.44s	1.16%	N/A	N/A	30	N/A	4630.02s	1257.92s	0.00%	43.26s	45.63s	1.90%
3	200	3	18.34s	4.23s	1.14%	3.17s	0.95s	0.67%	N/A	N/A	30	N/A	4987.91s	1.93s	0.01%	127.20s	208.02s	1.13%
3	200	5	31.14s	4.19s	0.93%	6.61s	1.55s	0.48%	N/A	N/A	30	N/A	4985.01s	1.86s	0.00%	317.84s	430.49s	1.62%
3	300	2	44.76s	12.02s	0.64%	9.94s	3.09s	0.90%	N/A	N/A	30	N/A	4973.39s	9.33s	0.00%	274.08s	383.98s	1.16%
3	300	3	87.87s	16.26s	0.78%	19.51s	5.17s	1.02%	N/A	N/A	30	N/A	4986.05s	1.13s	0.00%	365.82s	241.72s	1.45%
3	300	5	134.40s	19.44s	1.01%	27.42s	6.02s	0.61%	N/A	N/A	30	N/A	4977.91s	1.62s	0.00%	1042.52s	1038.33s	1.13%
5	2	2	0.01s	0.00s	0.04%	0.01s	0.00s	1.16%	0.66s	0.37s	0	7.38%	0.04s	0.02s	1.16%	0.02s	0.01s	0.00%
5	2	3	0.01s	0.00s	0.17%	0.01s	0.00s	2.65%	0.90s	0.40s	0	10.98%	0.04s	0.02s	2.65%	0.02s	0.01s	0.08%
5	2	5	0.07s	0.34s	0.18%	0.01s	0.00s	3.63%	1.96s	0.47s	0	11.79%	0.05s	0.03s	3.63%	0.03s	0.01s	0.11%
5	5	2	0.01s	0.01s	0.04%	0.01s	0.00s	5.32%	9.39s	3.58s	0	8.74%	0.08s	0.05s	4.57%	0.04s	0.01s	0.41%
5	5	3	0.04s	0.10s	1.40%	0.01s	0.00s	7.01%	16.28s	4.37s	0	10.11%	0.26s	0.09s	6.12%	0.05s	0.02s	0.60%
5	5	5	0.18s	0.58s	1.21%	0.01s	0.00s	6.28%	24.16s	2.84s	5	14.94%	0.38s	0.08s	4.53%	0.08s	0.05s	0.71%
5	10	2	0.05s	0.05s	1.37%	0.01s	0.00s	9.17%	91.87s	16.27s	7	12.13%	0.29s	0.07s	6.53%	0.18s	0.25s	0.43%
5	10	3	0.11s	0.12s	0.65%	0.01s	0.00s	7.73%	116.75s	10.07s	8	12.94%	0.57s	0.23s	4.63%	0.50s	1.34s	0.64%
5	10	5	0.36s	0.31s	1.76%	0.01s	0.00s	4.48%	159.57s	26.00s	12	22.28%	1.32s	0.90s	1.48%	0.28s	0.28s	0.42%
5	20	2	0.25s	0.24s	0.74%	0.01s	0.00s	6.71%	698.50s	154.70s	22	20.27%	0.64s	0.30s	2.97%	0.87s	0.91s	1.54%
5	20	3	0.58s	0.48s	1.89%	0.01s	0.00s	7.38%	810.26s	37.60s	26	23.22%	13.36s	41.83s	1.15%	0.71s	0.79s	1.36%
5	20	5	1.47s	0.85s	1.33%	0.01s	0.00s	3.29%	1095.96s	34.22s	26	34.00%	64.97s	264.12s	0.16%	0.90s	1.43s	1.14%
5	50	2	1.04s	0.79s	1.67%	0.02s	0.01s	5.30%	N/A	N/A	30	N/A	21.56s	50.88s	0.20%	3.59s	4.95s	1.65%
5	50	3	1.88s	0.92s	2.24%	0.04s	0.01s	3.47%	N/A	N/A	30	N/A	4043.33s	1622.24s	0.07%	2.87s	3.10s	2.48%
5	50	5	4.78s	2.07s	1.58%	0.08s	0.03s	2.38%	N/A	N/A	30	N/A	4977.74s	12.25s	0.02%	3.47s	4.71s	1.39%
5	100	2	4.84s	2.99s	1.03%	0.20s	0.06s	2.66%	N/A	N/A	30	N/A	4973.40s	9.24s	0.02%	33.92s	62.77s	1.86%
5	100	3	6.83s	2.22s	2.21%	0.47s	0.12s	1.93%	N/A	N/A	30	N/A	4987.36s	2.00s	0.00%	20.22s	32.27s	2.48%
5	100	5	13.09s	5.72s	1.40%	0.92s	0.18s	1.50%	4997.73s	N/A	29	43.92%	4986.68s	1.35s	0.00%	20.46s	17.65s	2.20%
5	200	2	31.55s	12.10s	1.49%	3.62s	0.78s	2.12%	N/A	N/A	30	N/A	4972.23s	9.61s	0.00%	116.99s	110.23s	1.46%
5	200	3	53.42s	18.38s	1.67%	7.55s	2.06s	1.83%	N/A	N/A	30	N/A	4987.03s	0.83s	0.00%	207.14s	232.72s	2.00%
5	200	5	74.08s	12.74s	0.80%	12.00s	2.49s	1.37%	N/A	N/A	30	N/A	4984.07s	2.47s	0.00%	347.10s	267.83s	2.53%
5	300	2	127.82s	43.56s	1.22%	22.07s	4.45s	1.95%	N/A	N/A	30	N/A	4971.33s	11.30s	0.00%	628.76s	451.35s	1.66%
5	300	3	218.41s	47.18s	1.69%	40.76s	9.63s	1.41%	N/A	N/A	30	N/A	4985.13s	1.22s	0.00%	675.07s	446.59s	2.11%
5	300	5	293.08s	56.89s	1.73%	70.15s	16.55s	1.29%	N/A	N/A	30	N/A	4980.24s	4.44s	0.00%	1813.84s	1122.05s	2.08%

Bibliography

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity - A Modern Approach*. Cambridge University Press, 2009.

[ABZ07] Christopher Amato, Daniel S. Bernstein, and Shlomo Zilberstein. Solving POMDPs using quadratically constrained linear programs. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 2418–2424, 2007.

[AM08] Daniel Adelman and Adam J. Mersereau. Relaxations of weakly coupled stochastic dynamic programs. *Operations Research*, 56(3):712–727, 2008.

[Bar08] Vlad Barbu. *Semi-Markov chains and hidden semi-Markov models toward applications their use in reliability and DNA analysis*. Springer, New York, 2008.

[BDFS17] Peter Buchholz, Iryna Dohndorf, Alexander Frank, and Dimitri Scheftelowitsch. Bounded aggregation for continuous time Markov decision processes. In *14th European Performance Engineering Workshop*. Springer LNCS, 2017.

[BDS17a] Peter Buchholz, Iryna Dohndorf, and Dimitri Scheftelowitsch. Analysis of Markov decision processes under parameter uncertainty. In *14th European Performance Engineering Workshop*. Springer LNCS, 2017.

[BDS17b] Peter Buchholz, Iryna Dohndorf, and Dimitri Scheftelowitsch. Optimal decisions for continuous time Markov decision processes over finite planning horizons. *Computers & Operations Research*, 77:267 – 278, 2017.

[Ben98] Harold P. Benson. An outer approximation algorithm for generating all efficient extreme points in the outcome set of a multiple objective linear programming problem. *Journal of Global Optimization*, 13(1):1–24, Jan 1998.

[BKF14] Peter Buchholz, Jan Kriege, and Iryna Felko. *Input Modeling with Phase-Type Distributions and Markov Models*. Springer International Publishing, 2014.

[BKS14] Peter Buchholz, Jan Kriege, and Dimitri Scheftelowitsch. Model checking stochastic automata for dependability and performance measures. In *44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, DSN 2014, Atlanta, GA, USA, June 23-26, 2014*, pages 503–514. IEEE, 2014.

[BM16] Dimitris Bertsimas and Velibor V. Mišić. Decomposable Markov decision processes: A fluid optimization approach. *Operations Research*, 64(6):1537–1555, 2016.

[BM17] Dimitris Bertsimas and Velibor V. Mišić. Robust product line design. *Operations Research*, 65(1):19–37, 2017.

[BN08] Leon Barrett and Srini Narayanan. Learning all optimal policies with multiple criteria. In William W. Cohen, Andrew McCallum, and Sam T. Roweis, editors, *ICML*, volume 307 of *ACM International Conference Proceeding Series*, pages 41–47. ACM, 2008.

[BR14] Matthew Bourque and T. E. S. Raghavan. Policy improvement for perfect information additive reward and additive transition stochastic games with discounted and average payoffs. *Journal of Dynamics and Games*, 1(3):347–361, 2014.

[BS17a] Peter Buchholz and Dimitri Scheftelowitsch. Computation of Weighted Sums of Rewards for Concurrent MDPs. *submitted for publication*, ?(?), 2017.

[BS17b] Peter Buchholz and Dimitri Scheftelowitsch. Light robustness in the optimization of Markov decision processes with uncertain parameters. *submitted for publication*, 2017.

[BST16] Dimitris Bertsimas, John Silberholz, and Thomas Trikalinos. Optimal healthcare decision making under multiple mathematical models: application in prostate cancer screening. *Health Care Management Science*, pages 1–14, 2016.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.

[BV07] Henrik Björklund and Sergei Vorobyov. A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. *Discrete Applied Mathematics*, 155(2):210 – 229, 2007. 29th Symposium on Mathematical Foundations of Computer Science MFCS 2004.

[CD15] Felipe Caro and Aparupa Das Gupta. Robust control of the multi-armed bandit problem. *Annals of Operations Research*, pages 1–20, 2015.

[Cie07] Krzysztof Ciesielski. On stefan banach and some of his results. *Banach J. Math. Anal.*, 1(1):1–10, 2007.

[CMH06] Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. Markov decision processes with multiple objectives. In Bruno Durand and Wolfgang Thomas, editors, *STACS*, volume 3884 of *Lecture Notes in Computer Science*, pages 325–336. Springer, 2006.

[CMH08] Krishnendu Chatterjee, Rupak Majumdar, and Thomas A. Henzinger. Stochastic limit-average games are in EXPTIME. *Int. J. Game Theory*, 37(2):219–234, 2008.

[Con92] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203 – 224, 1992.

[Con16] GSL Project Contributors. GSL - GNU scientific library - GNU project - free software foundation (FSF). http://www.gnu.org/software/gsl/, 2016.

[CRI07] Andrew S. Cantino, David L. Roberts, and Charles L. Isbell. Autonomous nondeterministic tour guides: improving quality of experience with TTD-MDPs. In *AAMAS*, page 22. IFAAMAS, 2007.

[DD05] Dmitri A. Dolgov and Edmund H. Durfee. Stationary deterministic policies for constrained MDPs with multiple rewards, costs, and discount factors. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI-05, Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, Edinburgh,* *Scotland, UK, July 30-August 5, 2005*, pages 1326–1331. Professional Book Center, 2005.

[d'E63] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[DGL97] Thomas L. Dean, Robert Givan, and Sonia M. Leach. Model reduction techniques for computing approximately optimal solutions for Markov decision processes. In Dan Geiger and Prakash P. Shenoy, editors, *UAI '97: Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, Brown University, Providence, Rhode Island, USA, August 1-3, 1997*, pages 124–131. Morgan Kaufmann, 1997.

[DM10] Erick Delage and Shie Mannor. Percentile optimization for Markov decision processes with parameter uncertainty. *Operations Research*, 58(1):203–213, 2010.

[Ehr05] Matthias Ehrgott. *Multicriteria Optimization*. Springer-Verlag Berlin Heidelberg, 2005.

[FV97] Jerzy Filar and Koos Vrieze. *Competitive Markov Decision Processes*. Springer New York, 1997.

[GHA10] Anshul Gandhi, Mor Harchol-Balter, and Ivo J. B. F. Adan. Server farms with setup costs. *Perform. Eval.*, 67(11):1123–1138, 2010.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractibility, A Guide to the Theory of* NP*-Completeness*. W. H. Freeman and Company, New York, 1979.

[GKP01] Carlos Guestrin, Daphne Koller, and Ronald Parr. Multiagent planning with factored MDPs. In Thomas G. Dietterich, Suzanna Becker, and Zoubin Ghahramani, editors, *Advances in Neural Information Processing Systems 14 [Neural Information Processing Systems: Natural and Synthetic, NIPS 2001, December 3-8, 2001, Vancouver, British Columbia, Canada]*, pages 1523–1530. MIT Press, 2001.

[GLD00] Robert Givan, Sonia M. Leach, and Thomas L. Dean. Bounded-parameter Markov decision processes. *Artif. Intell.*, 122(1-2):71–109, 2000.

[Goe94] S. Goedecker. Remark on algorithms to find roots of polynomials. *SIAM Journal on Scientific Computing*, 15(5):1059–1063, 1994.

[Hag89] William W. Hager. Updating the inverse of a matrix. *SIAM Review*, 31(2):221–239, 1989.

[Han95] Sven Ove Hansson. Decision theory – a brief introduction. Technical report, 1995.

[Haw03] Jeffrey Thomas Hawkins. *A Lagrangian decomposition approach to weakly coupled dynamic optimization problems and its applications*. PhD thesis, 2003.

[He14] Qi-Ming He. *Fundamentals of Matrix-Analytic Methods*. Springer-Verlag New York, 2014.

[Hel00] Keld Helsgaun. An effective implementation of the Lin-Kernighan traveling salesman heuristic. *European Journal of Operational Research*, 126(1):106–130, 2000.

[HHH$^+$17] Ernst Moritz Hahn, Vahid Hashemi, Holger Hermanns, Morteza Lahijanian, and Andrea Turrini. Multi-objective robust strategy synthesis for interval Markov decision processes. *CoRR*, abs/1706.06875, 2017.

[HHS16] Vahid Hashemi, Holger Hermanns, and Lei Song. Reward-bounded reachability probability for uncertain weighted MDPs. In *International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 351–371. Springer, 2016.

[hsl17] HSL. A collection of Fortran codes for large scale scientific computation. `http://www.hsl.rl.ac.uk/`, 2017.

[IBM15] IBM Corp. CPLEX. `https://www.ibm.com/analytics/cplex-optimizer`, 2015.

[int17] Intel Math Kernel Library. `http://software.intel.com/en-us/articles/intel-mkl/`, 2017.

[Iye05] Garud N. Iyengar. Robust dynamic programming. *Mathematics of Operations Research*, 30(2):257–280, 2005.

[Joh14] Steven G. Johnson. The nlopt nonlinear-optimization package. `http://ab-initio.mit.edu/nlopt`, 2014.

[JT70] Michael A. Jenkins and Joseph F. Traub. A three-stage algorithm for real polynomials using quadratic iteration. *SIAM journal on Numerical Analysis*, 7(4):545–566, 1970.

[Kal83] Lodewijk C. M. Kallenberg. *Linear programming and finite Markovian control problems*. Amsterdam : Mathematisch Centrum, 1983. Slightly revised version of PhD thesis.

[Kal16] Lodewijk C. M. Kallenberg. Markov decision processes. `https://www.math.leidenuniv.nl/~kallenberg/Lecture-notes-MDP.pdf`, October 2016.

[Kar72] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations*, The IBM Research Symposia Series, pages 85–103. Plenum Press, New York, 1972.

[KBT75] Vladimir S. Korolyuk, Stepan M. Brodi, and Anatoly F. Turbin. Semi-Markov processes and their applications. *Journal of Soviet Mathematics*, 4(3):244–280, Sep 1975.

[Kha93] Leonid Khachiyan. Complexity of polytope volume computation. In János Pach, editor, *New Trends in Discrete and Computational Geometry*, volume 10 of *Algorithms and Combinatorics*, pages 91–101. Springer Berlin Heidelberg, 1993.

[KKST13] Kathrin Klamroth, Elisabeth Köbis, Anita Schöbel, and Christiane Tammer. A unified approach for different concepts of robustness and stochastic programming via non-linear scalarizing functionals. *Optimization*, 62(5):649–671, 2013.

[Kol31] Andrei Kolmogoroff. Über die analytischen methoden in der Wahrscheinlichkeitsrechnung. *Mathematische Annalen*, 104(1):415–458, Dec 1931.

[KVY11] Narayan Kamath, Irina Voiculescu, and Chee K. Yap. Empirical study of an evaluation-based subdivision algorithm for complex root isolation. In *Proceedings of the 2011 International Workshop on Symbolic-Numeric Computation*, SNC '11, pages 155–164, New York, NY, USA, 2011. ACM.

[LKS$^+$17] Joel Lanir, Tsvi Kuflik, Julia Sheidin, Nisan Yavin, Kate Leiderman, and Michael Segal. Visualizing museum visitors' behavior: Where do they go and what do they do there? *Personal Ubiquitous Comput.*, 21(2):313–326, April 2017.

[LL69] Thomas M. Liggett and Steven A. Lippman. Stochastic games with perfect information and time average payoff. *SIAM Review*, 11(4):604–607, 1969.

[LMSM14] Pascal Libuschewski, Peter Marwedel, Dominic Siedhoff, and Heinrich Müller. Multi-objective, energy-aware GPGPU design space exploration for medical or industrial applications. In *2014 Tenth International Conference on Signal-Image Technology and Internet-Based Systems*, pages 637–644, Nov 2014.

[Man60] Alan S. Manne. Linear programming and sequential decisions. *Management Science*, 6(3):259–267, 1960.

[Map14] Maplesoft. Maple user manual, September 04 2014.

[Mil81] Kenneth S. Miller. On the inverse of the sum of matrices. *Math. Mag.*, 54(2):67–72, 1981.

[MK87] Katta G. Murty and Santosh N. Kabadi. Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, 39(2):117–129, Jun 1987.

[MN81] Jean-François Mertens and Abraham Neyman. Stochastic games. *International Journal of Game Theory*, 10:53–66, 1981.

[MZ09] Sharad Malik and Lintao Zhang. Boolean satisfiability from theoretical hardness to practical success. *Commun. ACM*, 52(8):76–82, 2009.

[Neu79] Marcel F. Neuts. A versatile Markovian point process. *Journal of Applied Probability*, 16(4):pp. 764–779, 1979.

[NG05] Arnab Nilim and Laurent El Ghaoui. Robust control of Markov decision processes with uncertain transition matrices. *Operations Research*, 53(5):780–798, 2005.

[NN94] Yurii Nesterov and Arkadii Nemirovskii. *Interior-Point Polynomial Algorithms in Convex Programming*. Society for Industrial and Applied Mathematics, 1994.

[Obd06] Jan Obdržálek. *Algorithmic Analysis of Parity Games*. PhD thesis, University of Edinburgh, 2006. Submitted: January 31, 2006. Examined: May 29, 2006.

[O'C99] Colm Art O'Cinneide. Phase-type distributions: open problems and a few properties. *Communications in Statistics. Stochastic Models*, 15(4):731–757, 1999.

[O'N71] Patrick E. O'Neil. Hyperplane cuts of an n-cube. *Discrete Mathematics*, 1(2):193 – 195, 1971.

[Pow94] Michael J. D. Powell. *A Direct Search Optimization Method That Models the Objective and Constraint Functions by Linear Interpolation*, pages 51–67. Springer Netherlands, Dordrecht, 1994.

[PTVF07] William H. Press, Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery. *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[PW10] Patrice Perny and Paul Weng. On finding compromise solutions in multiobjective Markov decision processes. In Helder Coelho, Rudi Studer, and Michael Wooldridge, editors, *ECAI*, volume 215 of *Frontiers in Artificial Intelligence and Applications*, pages 969–970. IOS Press, 2010.

[PY00] Christos H. Papadimitriou and Mihalis Yannakakis. On the approximability of trade-offs and optimal access of web sources. In *41st Annual Symposium on Foundations of Computer Science, FOCS 2000, 12-14 November 2000, Redondo Beach, California, USA*, pages 86–92. IEEE Computer Society, 2000.

[QBM12] Andrea Qualizza, Pietro Belotti, and François Margot. *Linear programming relaxations of quadratically constrained quadratic programs*, chapter Mixed Integer Nonlinear Programming, pages 407–426. Springer, 2012.

[RSS$^+$14] Diederik Marijn Roijers, Joris Scharpff, Matthijs T. J. Spaan, Frans A. Oliehoek, Mathijs de Weerdt, and Shimon Whiteson. Bounded approximations for linear multi-objective planning under uncertainty. In Steve Chien, Minh Binh Do, Alan Fern, and Wheeler Ruml, editors, *ICAPS*. AAAI, 2014.

[RW91] R. Tyrrell Rockafellar and Roger J.-B. Wets. Scenarios and policy aggregation in optimization under uncertainty. *Math. Oper. Res.*, 16(1):119–147, 1991.

[RWO13] Diederik M. Roijers, Shimon Whiteson, and Frans A. Oliehoek. Computing convex coverage sets for multi-objective coordination graphs. In Patrice Perny, Marc Pirlot, and Alexis Tsoukiàs, editors, *ADT*, volume 8176 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2013.

[RWO14] Diederik M. Roijers, Shimon Whiteson, and Frans A. Oliehoek. Linear support for multi-objective coordination graphs. In *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-agent Systems*, AAMAS '14, pages 1297–1304, Richland, SC, 2014. International Foundation for Autonomous Agents and Multiagent Systems.

[Saa93] Youcef Saad. A flexible inner-outer preconditioned GMRES algorithm. *SIAM J. Scientific Computing*, 14(2):461–469, 1993.

[SAB$^+$15] Yuji Shinano, Tobias Achterberg, Timo Berthold, Stefan Heinz, Thorsten Koch, and Michael Winkler. Solving open MIP instances with paraSCIP on supercomputers using up to 80,000 cores. Technical Report 15-53, ZIB, Takustr. 7, 14195 Berlin, 2015.

[SB17] Dimitri Scheftelowitsch and Peter Buchholz. Bounded-parameter and concurrent MDP analysis tool. https://gitlab.com/dreval/bmdp-analysis, 2017. Accessed: 2016-11-22.

[SBHH17] Dimitri Scheftelowitsch, Peter Buchholz, Vahid Hashemi, and Holger Hermanns. Multi-criteria approaches to Markov decision processes with uncertain transition parameters. In *VALUETOOLS 2017: 11th EAI International Conference on Performance Evaluation Methodologies and Tools, December 5–7, 2017, Venice, Italy*. ACM, New York, USA, 2017.

[SC97] Satinder P. Singh and David Cohn. How to dynamically merge Markov decision processes. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]*, pages 1057–1063. The MIT Press, 1997.

[Sch15] Dimitri Scheftelowitsch. The complexity of uncertainty in Markov decision processes. In *SIAM Conference on Control & its Applications CT15*, Paris, France, July 2015.

[Sch17a] Dimitri Scheftelowitsch. Bounded-parameter and concurrent MDP testing infrastructure. https://gitlab.com/dreval/bmdp-python-scripts, 2017. Accessed: 2016-11-22.

[Sch17b] Dimitri Scheftelowitsch. collider: An in silico experimental infrastructure. https://gitlab.com/dreval/collider, 2017.

[Ser79] Richard F. Serfozo. An Equivalence between Continuous and Discrete Time Markov Decision Processes. *Operations Research*, 27(3):616–620, 1979.

[Sha53] Lloyd S. Shapley. Stochastic games. *Proceedings of the National Academy of Sciences*, 39:1095–1100, 1953.

[Sho94] Peter W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 124–134, Nov 1994.

[Sil63] Edward A. Silver. *Markovian Decision Processes with Uncertain Transition Probabilities of Rewards*. Technical report (Massachusetts Institute of Technology. Operations Research Center). M.I.T. Operations Research Center, 1963.

[SL73] Jay K. Satia and Roy E. Lave. Markovian decision processes with uncertain transition probabilities. *Operations Research*, 21(3):728–740, 1973.

[Ste94] William J. Stewart. *Introduction to the numerical solution of Markov Chains*. Princeton University Press, 1994.

[Ste00] Neal Stephenson. *Cryptonomicon*. Arrow, London, 2000.

[vEB90] Peter van Emde Boas. Machine models and simulation. In *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A)*, pages 1–66. 1990.

[Vei66] Arthur F. Veinott. On finding optimal policies in discrete dynamic programming with no discounting. *Ann. Math. Statist.*, 37(5):1284–1294, 10 1966.

[vL13] Moritz v. Looz. Discovery of latent features and clusters based on similarities in brain function. Diploma thesis, Karlsruhe Institute of Technology, Institut für Theoretische Informatik, Fakultät für Informatik, 2013.

[WBG06] Charles M. Weber, C. Neil Berglund, and Patricia Gabella. Mask cost and profitability in photomask manufacturing: An empirical analysis. *IEEE Transactions on Semiconductor Manufacturing*, 19(4):465–474, Nov 2006.

[WdJ07] Marco A. Wiering and Edwin D. de Jong. Computing optimal stationary policies for multi-objective Markov decision processes. In *ADPRL 2007, IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*, pages 158–165, April 2007.

[WED94] Chelsea C. White and Hany K. El-Deib. Markov decision processes with imprecise transition probabilities. *Operations Research*, 42(4):739–749, 1994.

[Whi82] D. J. White. Multi-objective infinite-horizon discounted Markov decision processes. *Journal of mathematical analysis and applications*, 89(2):639–647, 1982.

[WKR13] Wolfram Wiesemann, Daniel Kuhn, and Berç Rustem. Robust Markov decision processes. *Mathematics of Operations Research*, 38(1):153–183, 2013.

[Wä02] Andreas Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Carnegie Mellon University, 2002.

[YS04] Håkan L. S. Younes and Reid G. Simmons. Solving generalized semi-Markov decision processes using continuous phase-type distributions. In Deborah L. McGuinness and George Ferguson, editors, *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence, July 25-29, 2004, San Jose, California, USA*, pages 742–748. AAAI Press / The MIT Press, 2004.

[ZLT01] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland, 2001.

[ZP96] Uri Zwick and Mike Paterson. The complexity of mean payoff games on graphs. *Theoretical Computer Science*, 158(1):343 – 359, 1996.

[ZT99] Eckart Zitzler and Lothar Thiele. Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach. *IEEE Trans. Evolutionary Computation*, 3(4):257–271, 1999.

[ZTCJ15] Xingyi Zhang, Ye Tian, Ran Cheng, and Yaochu Jin. An efficient approach to nondominated sorting for evolutionary multiobjective optimization. *IEEE Trans. Evolutionary Computation*, 19(2):201–213, 2015.