Mathematical Methods of Operations Research (2019) 89:1–42
https://doi.org/10.1007/s00186-018-0653-1

ORIGINAL ARTICLE

Computation of weighted sums of rewards for concurrent MDPs

Peter Buchholz[1] · Dimitri Scheftelowitsch[1]

Received: 21 August 2017 / Accepted: 17 October 2018 / Published online: 31 October 2018

Abstract
We consider sets of Markov decision processes (MDPs) with shared state and action spaces and assume that the individual MDPs in such a set represent different *scenarios* for a system's operation. In this setting, we solve the problem of finding a single policy that performs well under each of these scenarios by considering the weighted sum of value vectors for each of the scenarios. Several solution approaches as well as the general complexity of the problem are discussed and algorithms that are based on these solution approaches are presented. Finally, we compare the derived algorithms on a set of benchmark problems.

Keywords Markov decision processes · Optimization · Multi-objective optimization · Non-linear programming

1 Introduction

Markov decision processes (MDPs) are commonly used in modeling probabilistic state-based systems with non-deterministic controller actions where the goal is to find a control policy that optimizes some time-based reward measure. Briefly, a Markov decision process describes a Markov chain with inputs that are called *actions* and *rewards* that make it possible for a controller to, first, control the evolution of the Markov chain and, second, assess its performance. The controller can then define a *policy* which selects the right action and, thus, defines a Markov chain with rewards. The existing theory allows one to compute optimal policies for Markov decision processes with respect to various reward measures, such as expected discounted reward, expected average reward, or expected finite-horizon reward (Puterman 2005). Here,

✉ Peter Buchholz
peter.buchholz@tu-dortmund.de

Dimitri Scheftelowitsch
dimitri.scheftelowitsch@tu-dortmund.de

1 Informatik IV, TU Dortmund, Dortmund, Germany

the expected discounted reward measure is considered, that is, given a *discount factor* $\gamma \in [0, 1)$, we weight the reward from the ith step in the future with γ^i, and compute the expected sum of discounted rewards.

The general theory of MDPs is well established and applied in various branches of operations research or artificial intelligence. Several textbooks (Feinberg and Schwartz 2002; Puterman 2005; Sigaud and Buffet 2010) and tutorial papers (White and White 1989) on the subject exist. The final goal is to find an optimal control strategy for a system using the optimal policy resulting from the MDP. However, MDPs, like many other models, are an abstract image of reality which is superposed by several levels of uncertainty. This is reflected in a limited knowledge of the exact values of transition probabilities or rewards. A wide variety of approaches exists to consider uncertainty in MDPs (see the overview in Sect. 1.1). Often uncertainty is coded in the model and then the modified model is analyzed with some assumptions in mind, e.g., being pessimistic or optimistic with respect to the realization of uncertainty. The resulting formalisms that capture additional uncertainty in MDPs share the general property that the computational complexity of finding an optimal policy becomes higher when compared to a "plain" MDP, as it is, for example, the case with stochastic games (Björklund and Vorobyov 2007).

In this paper, we present a specific approach to introduce uncertainty in MDPs by so-called *scenarios*. A scenario is one possible realization of uncertainty and may result from possible realizations of a system or some observation of a system or from a simulation model. Each scenario defines a new transition probability matrix for an action in the MDP and the complete information specifies a multi-scenario MDP. In this work, we shall concentrate on the weighted optimization problem for this class of models. Formally, this means to compute, given a set of K Markov decision processes with common state and action spaces (but different transition probabilities and rewards) and K weight coefficients, a policy that optimizes the weighted sum of expected discounted rewards assuming that the policy has to be selected before the scenario which finally occurs is known.

This problem can be interpreted in a different way as follows: given a probability distribution on Markov decision processes denoted as scenarios, compute a policy that optimizes the expected discounted reward, without knowing the scenario which is finally realized. This is often denoted as *non-anticipative* behavior in stochastic optimization (Colvin and Maravelias 2010). We will later show that, on the one hand, this problem is hard to solve but, on the other hand, simple heuristic optimization approaches usually yield optimal or almost optimal results. Before we outline the rest of the paper and introduce our approach, a short overview of related work is given.

1.1 Related work

As already mentioned, there is an enormous amount of work that has been published to describe uncertainty in MDPs. Consequently, we can only give a short summary of approaches that are related to our approach and we briefly outline the differences.

Multi-scenario problems The use of scenarios is one possibility to describe uncertainty. In this case, examples for the realization of uncertainty are given instead of characterizing the whole uncertainty set. Scenarios are commonly used in two-stage or multi-stage stochastic programming (Rockafellar and Wets 1991; Dupacová et al. 2000). In this setting it is usually assumed that information about the realization of a specific scenario is gained over time. Thus, future decisions can take into account additional information by, for example, reducing the set of possible scenarios. This is different from our approach where we assume that a policy has been selected in advance before any information about the concrete scenario becomes available.

Scenarios in MDPs are mentioned in Nilim and Ghaoui (2005, Sect. 7.1) as a simple uncertainty model. In contrast to the scenarios defined here, it is assumed that the scenarios observe the (s, a)-rectangularity property, i.e., they are defined independently for every state-action pair. Concrete application examples for the use of scenarios are product line design problems (Bertsimas and Mišić 2017) where finitely many models for customer behavior are considered, healthcare decision making (Bertsimas et al. 2016), where different screening strategies to detect cancer are available, and project scheduling (Mercier and Hentenryck 2008) where different projects are described by Markov chains. The combination of scenarios and partially observable MDPs is considered in Walraven and Spaan (2015) in the context of planning problems. If one cannot distinguish the current scenario during operation of the system or decisions cannot anticipate the scenario, one has to find a policy that behaves well in all scenarios. This is different from the computation of robust policies which behave good in the worst case but may have a bad performance in the average case.

In our case, every scenario defines its own MDP but all MDPs are coupled by a single policy. The same model has been concurrently developed by Steimle et al. (2018) under the name *multi-model MDP*. However, Steimle et al. (2018) considers optimal policies for finite horizons whereas we consider optimal stationary policies for discounted infinite horizons. Both problems are shown to be NP-hard. The optimal policy in the finite horizon case depends on the current time step and is shown to be deterministic in Steimle et al. (2018), whereas the optimal stationary policy for the infinite horizon case can be randomized and strictly better than any deterministic policy as shown here. This is similar to partially observable MDPs where the optimal policy can also be randomized (Singh et al. 1994). This interesting relation has also been recognized in Steimle et al. (2018, Sect. 4). Another difference between (Steimle et al. 2018) and this paper are the heuristic algorithms proposed for approximate policies, Steimle et al. (2018) uses an extension of value iteration based on the mean value relaxation of the original problem whereas we propose an approach based on policy iteration on the unrelaxed problem with a guaranteed local convergence.

Furthermore, a similar model has been considered in Raskin and Sankur (2014), however, in a model-checking context of reachability, safety, and parity properties. The results in Raskin and Sankur (2014) show that the qualitative model-checking problems (such as limit-sure reachability) can be solved efficiently while the quantitative problems (such as quantitative reachability and safety) are NP-hard, which corresponds to the hardness results in Steimle et al. (2018) and this paper.

Multi-scenario optimization problems can be interpreted as *multi-objective optimization* which amounts to optimizing given functions $f_1(x), \ldots, f_K(x)$ subject to

$x \in X$ simultaneously. As multi-dimensional spaces can only be partially ordered in a "natural" way, there may exist multiple incomparable optimal solutions. Hence, for multi-objective problems, several approaches are known including the computation of the Pareto frontier, the generation of the convex hull of Pareto optimal solution, the analysis whether a solution exceeds a lower bound and the computation of a weighted sum of function values (Ehrgott 2005). Among these problems, the weighted optimization problem is often the simplest one. Furthermore, the computation of optimal policies for different weight vectors is the central step to characterize the convex hull of Pareto optimal solutions (Roijers et al. 2014).

MDPs with uncertainties Scenarios are only one way of expressing parameter uncertainty. Parameter uncertainty in MDPs has been discussed in a more general context for more than 40 years by various authors. Early references are Satia and Lave (1973) and White and Eldeib (1994). Givan et al. (2000), Iyengar (2005), Klamroth et al. (2013), Nilim and Ghaoui (2005) and Wiesemann et al. (2013) are more recent and important publications. Different forms of representing uncertainty have been proposed. Usually, so-called *uncertainty sets* of the transition probabilities for the underlying Markov chain and sometimes also the set of rewards are defined. These sets then characterize a usually infinite and not even countable set of MDPs. Very often the optimization problem is then interpreted as a *robust optimization* problem. Mathematically, robust optimization means finding a policy that maximizes the rewards under the worst possible realization of uncertainty; that is, we solve a problem of the type $\max_{x \in X} \min_{y \in Y} f(x, y)$. It is intuitive that robust mathematical programs are much harder than simple maximization problems. However, in some cases, like for *Bounded Parameter MDPs*, it is still possible to find optimal policies efficiently (Givan et al. 2000). In contrast to our approach with a fixed number of scenarios, the uncertainty sets define an infinite and usually not countable set of transition kernels. However, the minimum operator usually maps this continuous set on a finite set of optimal transition kernels which then have to be considered in the maximum operation. From this perspective, one can consider the robust problem as a two-stage optimization problem where a scenario-based approach is applied after the inner optimization problem has been solved or an approximate solution for this problem is known. Usually the robust problem is solved for the discounted reward over an infinite horizon.

Concerning the theoretical assumptions behind most MDP problems with uncertainties about the model parameters, we refer to the work of Iyengar (2005). There, the (s, a)-*rectangularity property* is defined as a generally desirable property of uncertainty sets for transition probability matrices which, intuitively, amounts to separability of the uncertainty set into a product of uncertainty sets of transition probability vectors for each state and each action. Under this assumption, robust policies for MDPs are studied and it is shown that robust policies are pure and can be computed with an effort only modestly larger than the effort of computing optimal policies in MDPs (for details see Iyengar 2005). It is important to note that this assumption holds for the uncertainty models in Givan et al. (2000), Nilim and Ghaoui (2005), Satia and Lave (1973) and White and Eldeib (1994). The most prominent uncertainty model are intervals such that rows of the transition matrix are chosen from sets of probabilistic vectors which are defined by lower and upper bounds for the elements (Givan et al.

2000). Other uncertainty models are a Bayesian formulation (Satia and Lave 1973), likelihood or entropy models (Nilim and Ghaoui 2005).

The (s, a)-rectangularity property has been extended in Wiesemann et al. (2013) to an s-rectangularity property. This intuitively means that dependencies between the uncertain parameters for different decisions in a single state are allowed. It can be shown that for convex uncertainty sets observing the s-rectangularity property, a stationary randomized policy can be found that minimizes the discounted reward. Computation of the optimal policy is harder than for uncertainty sets having the (s, a)-rectangularity property but the effort remains polynomial for convex uncertainty sets and the robust results can be much better as the uncertainty is constrained. However, in Wiesemann et al. (2013) it is also shown that without rectangularity, the problem becomes hard to solve.

Concerning applications of robust MDP models, we refer to a discussion of robust multi-armed bandit problems which have been transformed into MDPs with uncertain parameters observing the rectangularity property (Caro and Das-Gupta 2015), and to the work on product line design problems under model uncertainty (Bertsimas and Mišić 2017).

Stochastic games The max min problems mentioned above can also be interpreted as stochastic games with two players. MDPs with uncertainty can be interpreted in this way by assuming that two players which decide in an alternating sequence and one player is trying to minimize the goal function whereas the other one tries to maximize it (Filar and Vrieze 1997). Therefore the approach becomes similar to the robust optimization of MDPs with uncertainties.

Partially observable MDPs The scenario based perspective can also be interpreted as a variant of *partially observable MDPs* (Kaelbling et al. 1998), where the controller may have only indirect information about the exact state, such as that the state belongs with some probability to a subset of states. Partially observable Markov decision processes have more expressive power and thus high computational solution cost in the general case (Papadimitriou and Tsitsiklis 1987). Here, on the other hand, we have a very specific form of uncertainty since we know that the uncertainty set consists of finitely many possible realizations only.

Coupled MDPs Different MDPs that are combined by *coupling constraints* are considered for example in Singh and Cohn (1997) to analyze parallel tasks. These models differ from scenario based models because the state and action space of the coupled model consists of the cross product of the MDPs which is different from the scenario based approach where a single state and action space are considered.

Optimization methods Since a major aspect of this paper is the finding of efficient methods to compute or approximate the optimal policy and reward for combined MDPs, we briefly review optimization methods which are relevant for our work. Standard methods for MDPs can be found in Puterman (2005). Convex problems have been discussed in the book of Nesterov and Nemirovskii (1994), further methods of interest are mixed-integer programming (Vielma 2015) and quadratically constrained quadratic programming (Qualizza et al. 2012). Specifically for Markov decision problems, quadratic programming methods have been proposed in Amato et al. (2007). The relationship between robustness and multi-objective problems has been explored in the work of Klamroth et al. (2013).

1.2 Contribution of the paper

The class of *concurrent MDPs* is defined in this paper. It is based on a finite set of different scenarios which define different transition kernels. The goal is to find a common policy that minimizes or maximizes the weighted sum of rewards if applied to the concurrent MDPs. The approach assumes that the policy has to be found without knowing the scenario. Weights may be interpreted as probabilities that scenarios occur and the expected discounted reward is optimized. The resulting problem is different from two-stage or multi-stage problems, where information about the scenario becomes available during optimization. Problems of the form considered in this paper are important for situations where decisions have to be made beforehand and cannot be modified later when the scenario is known or can be estimated. This is for example the case for decisions made in the planning phase when the coming demands can only be estimated or during the design of a vaccine with only statistical information about future virus variants.

In contrast to Nilim and Ghaoui (2005), where scenarios observe the (s, a)-rectangularity property, scenarios here are general. This allows us to capture also those cases, where the parameters are defined for some model which is a high level description of an MDP. In this case, a single parameter, like the service rate of a queue, appears several times in the transition matrix and has to have the same value for each appearance. This property cannot be described by uncertainty sets that observe s-rectangularity or (s, a)-rectangularity.

The price for introducing this more general form of uncertainty is an increase in the complexity of the resulting optimization problem which is shown to be NP-hard. We will present five different algorithms to compute or approximate the optimal weighted sum of rewards for the concurrent MDPs under a common policy. These algorithms belong to three different classes, namely *integer linear programming* (ILP), *non-linear programming* (NLP) and problem-specific *local search heuristics*. The algorithms are compared empirically by means of several examples. The experiments show that non-linear programming approaches, which might be the natural choice for problems like the one we consider here, often show the worst performance and have convergence issues whereas the problem-specific local search heuristics are guaranteed to find local optima and are much more efficient than general non-linear programming algorithms. A second finding is that, even if the problems are non-convex and pure policies are not sufficient in general, the best pure policy is most times as good as the best general policy computed by a non-linear programming algorithm or by a local search heuristic. Furthermore, it is shown how the best pure policy can be computed from an integer linear program.

1.3 Structure of the work

The formal problem statement is given in Sect. 2, and its mathematical structure is explained and discussed in Sect. 3. Before we dive into optimization approaches, we briefly discuss complexity aspects in Sect. 3.1 (with a proof in "Appendix B"). In Sect. 4, we propose several approaches to solving the optimization problem; the practical evaluation of these algorithms is given in Sect. 5. Finally, Sect. 6 summarizes the work done and gives ideas for future research.

1.4 Notation

Over the course of this work, we shall use a couple of custom notation conventions for brevity and clarity purposes. In particular, multi-dimensional identifiers such as vectors and matrices are written in bold script, such as $\boldsymbol{v}$ or $\boldsymbol{M}$; furthermore, matrices are capitalized. To access the individual rows of a matrix $\boldsymbol{M}$, we use the $\boldsymbol{M}(i\bullet)$ notation to identify the ith row of $\boldsymbol{M}$. Analogously, the jth column is identified by $\boldsymbol{M}(\bullet j)$.

We also shall use some special identifiers for sets and constants. $\mathbb{B}$ is the set $\{0, 1\}$, $\mathbf{0}$ is, depending on the context, a matrix or a row vector of zeros (in case of ambiguity, the exact meaning is given), and $\mathbb{1}$ is a column vector of ones; in some cases, we shall write $\mathbb{1}_n$ to clarify that the vector has n dimensions. $\boldsymbol{e}_i$ is the ith basis row vector. For sets of multi-dimensional values over some set $\mathcal{K}$ we shall use $\mathcal{K}^{n\times m}$ to designate the set of matrices with n rows and m columns where each entry is in $\mathcal{K}$. $\boldsymbol{I}_N$ is the identity matrix of dimension N.

At some points in this work, Kronecker matrix-analytic operators will be used. We designate by $\otimes$ the Kronecker product.

2 Basic problem

We consider a set $\mathcal{M}$ of Markov decision processes (MDPs) with a joint finite state space $\mathcal{S}$ and a joint finite action space $\mathcal{A}$. Let K be the number of MDPs, N be the number of states and M be the number of actions. We identify the sets by consecutive integers, i.e., $\mathcal{M} = \{1, \ldots, K\}$, $\mathcal{S} = \{1, \ldots, N\}$ and $\mathcal{A} = \{1, \ldots, M\}$. Each MDP $k \in \mathcal{M}$ is defined by

$$\left(\mathcal{S}, \boldsymbol{\alpha}, \left(\boldsymbol{P}_k^a\right)_{a\in\mathcal{A}}, \boldsymbol{r}_k\right)$$

where $\boldsymbol{\alpha} \in \mathbb{R}_{\geq 0}^{1\times N}$ with $\boldsymbol{\alpha}\mathbb{1} = 1$ is the common initial distribution,

$$\boldsymbol{P}_k^a = \begin{pmatrix} \boldsymbol{P}_k^a(1\bullet) \\ \vdots \\ \boldsymbol{P}_k^a(N\bullet) \end{pmatrix} \text{ where } \boldsymbol{P}_k^a(i\bullet) \in \mathbb{R}_{\geq 0}^{1\times N},\ \boldsymbol{P}_k^a(i\bullet)\mathbb{1} = 1$$

is the stochastic transition matrix for action $a \in \mathcal{A}$, and $\boldsymbol{r}_k \in \mathbb{R}_{\geq 0}^{N \times 1}$ are the non-negative reward vectors. Furthermore, we assume a given discount factor $\gamma \in (0, 1)$.

Here, we consider rewards that are not action-dependent; this does not impose a limitation with respect to modeling power but does simplify some of the mathematical derivations. MDPs with action-dependent rewards can be transformed into equivalent MDPs with rewards that depend only on the state by enlarging the state space. This is formally shown in “Appendix A”.

We consider stationary policies which can be represented by $N \times M$ matrices

$$\boldsymbol{\Pi} = \begin{pmatrix} \boldsymbol{\pi}_1 \\ \vdots \\ \boldsymbol{\pi}_N \end{pmatrix} \text{ where } \boldsymbol{\pi}_i = (\boldsymbol{\pi}_i(1), \ldots, \boldsymbol{\pi}_i(M)) \in \mathbb{R}^{1 \times M}$$

and $\boldsymbol{\pi}_i(a)$ is the probability of choosing $a \in \mathcal{A}$ in state $i \in \mathcal{S}$. We have $\boldsymbol{\pi}_i(a) \geq 0$ and $\sum_{a \in \mathcal{A}} \boldsymbol{\pi}_i(a) = 1$. If for all $i \in \mathcal{S}$ it is $\boldsymbol{\pi}_i(a_i) = 1$ for some $a_i \in \mathcal{A}$, the policy is *pure* or *deterministic* otherwise it is *randomized*. We shall identify policies by the corresponding $N \times M$ matrices. In theory, a pure policy could be represented by a vector of length N which contains in position i the action chosen in state i; however, for the sake of uniformity in the mathematical formalism we will use a more general notation. Let $\mathcal{P}$ be the set of stationary policies and $\mathcal{P}_{pure}$ the set of pure policies. Policy $\boldsymbol{\Pi}$ defines for MDP $k \in \mathcal{M}$ a stochastic transition matrix

$$\boldsymbol{P}_k^{\boldsymbol{\Pi}} = \begin{pmatrix} \sum_{m=1}^{M} \boldsymbol{\pi}_1(m) \boldsymbol{P}_k^m(1\bullet) \\ \vdots \\ \sum_{m=1}^{M} \boldsymbol{\pi}_N(m) \boldsymbol{P}_k^j(N\bullet) \end{pmatrix}.$$

Define $\boldsymbol{C}_k^{\boldsymbol{\Pi}} = \boldsymbol{I} - \gamma \boldsymbol{P}_k^{\boldsymbol{\Pi}}$ with discount factor $\gamma \in (0, 1)$. $\boldsymbol{C}_k^{\boldsymbol{\Pi}}$ is a non-singular M-matrix. The inverse matrix $\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} = \left(\boldsymbol{C}_k^{\boldsymbol{\Pi}}\right)^{-1}$ exists and is non-negative (Berman and Plemmons 1994). The discounted gain for policy $\boldsymbol{\Pi}$ and discount factor $\gamma \in (0, 1)$ is given by

$$\boldsymbol{g}_k^{\boldsymbol{\Pi}} = \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k \quad \text{and} \quad G_k^{\boldsymbol{\Pi}} = \boldsymbol{\alpha} \boldsymbol{g}_k^{\boldsymbol{\Pi}}. \tag{1}$$

$\boldsymbol{g}_k^{\boldsymbol{\Pi}}$ is the *value function* or *value vector* of the policy $\boldsymbol{\Pi}$ for scenario k and $G_k^{\boldsymbol{\Pi}}$ is denoted as the (scalar) gain of $\boldsymbol{\Pi}$ for scenario k under initial distribution $\boldsymbol{\alpha}$. For further details about MDPs we refer to Puterman (2005); the expected discounted reward criterion is covered in Chapter 6 of the textbook.

A set of MDPs of the above form is denoted as a set of *concurrent MDPs*. We consider the computation of the weighted discounted gain for some weight vector $\boldsymbol{w} \in \mathbb{R}_{\geq 0}^{1 \times K}$ with $\boldsymbol{w}^T \mathbb{1} = 1$ which is given by

$$G^*(\boldsymbol{w}) = \max_{\boldsymbol{\Pi}\in\mathcal{P}} \left(\sum_{k=1}^{K} \boldsymbol{w}(k) G_k^{\boldsymbol{\Pi}}\right) \text{ and } \boldsymbol{\Pi}^* = \arg\max_{\boldsymbol{\Pi}\in\mathcal{P}} \left(\sum_{k=1}^{K} \boldsymbol{w}(k) G_k^{\boldsymbol{\Pi}}\right) \tag{2}$$

which can be considered a stochastic programming problem (Ruszczyński and Shapiro 2009, Chapter 1) as shown in the following section.

If we consider a single MDP $k \in \mathcal{M}$, then

$$G_k^* = \max_{\boldsymbol{\Pi}\in\mathcal{P}} \left(G_k^{\boldsymbol{\Pi}}\right) \quad \text{and} \quad \boldsymbol{\Pi}_k^* = \arg\max_{\boldsymbol{\Pi}\in\mathcal{P}} \left(G_k^{\boldsymbol{\Pi}}\right). \tag{3}$$

$\boldsymbol{\Pi}_k^*$ is not necessarily unique but it is known that a pure policy exists with gain G_k^* (Puterman 2005).

3 Mathematical structure of the optimization problem

We begin with the computation of an optimal policy for a single MDP. The optimal policy and value vectors can then be computed from a linear program (LP). Thus, the optimal gain for (3) results from the LP (Puterman 2005, Section 6.9.)

$$\min_{\boldsymbol{g}_k\in\mathbb{R}^{N\times 1}} \left(\boldsymbol{\alpha}\boldsymbol{g}_k\right) \text{ subject to } \begin{pmatrix} \left(\boldsymbol{I} - \gamma \boldsymbol{P}_k^1\right) \\ \vdots \\ \left(\boldsymbol{I} - \gamma \boldsymbol{P}_k^M\right) \end{pmatrix} \boldsymbol{g}_k \geq \mathbb{1}_M \otimes \boldsymbol{r}_k. \tag{4}$$

The LP has N variables and NM constraints. All variables are real-valued, which makes usage of standard LP solving techniques possible. The resulting vector $\boldsymbol{g}_k$ is the value function of the optimal policy. It is important to note that none of the variables in the linear program is a decision variable in the sense that it directly reflects a controller's action; the decisions are only indirectly visible. Concretely, in an optimal solution, the constraint for state i and action a is tight, i.e., it is $(\boldsymbol{e}_i - \gamma \boldsymbol{P}_k^a(i\bullet))\boldsymbol{g}_k(i) = \boldsymbol{r}_k(i)$ if action a is selected in state i.[1] The dual LP is given, following d'Epenoux (1963), by

$$\begin{aligned} &\max_{\boldsymbol{f}_k\in\mathbb{R}^{MN\times 1}} \boldsymbol{f}_k^T \left(\mathbb{1}_M \otimes \boldsymbol{r}_k\right) \\ &\text{subject to } \left(\left(\boldsymbol{I} - \gamma \boldsymbol{P}_k^1\right)^T \cdots \left(\boldsymbol{I} - \gamma \boldsymbol{P}_k^M\right)^T\right) \boldsymbol{f}_k = \boldsymbol{\alpha}^T \text{ and } \boldsymbol{f}_k \geq 0. \end{aligned} \tag{5}$$

The optimal gain G_k^* and a corresponding optimal pure policy $\boldsymbol{\Pi}_k^*$ can be computed from the LP or its dual. However, to compute the global optimum $G^*(\boldsymbol{w})$, the LPs have to be coupled, resulting in a non-linear program. To define the non-linear program we first define for $i \in \mathcal{S}$, $k \in \mathcal{M}$ the matrices

[1] This corresponds also to the corresponding slack variable being zero.

$$\boldsymbol{A}_k^i = \begin{pmatrix} \boldsymbol{e}_i - \gamma \boldsymbol{P}_k^1(i\bullet) \\ \vdots \\ \boldsymbol{e}_i - \gamma \boldsymbol{P}_k^M(i\bullet) \end{pmatrix} \text{ and vectors } \boldsymbol{\alpha}_k = \boldsymbol{w}(k)\boldsymbol{\alpha}. \tag{6}$$

Then every matrix $\boldsymbol{C}_k^{\boldsymbol{\Pi}}$ can be represented as

$$\boldsymbol{C}_k^{\boldsymbol{\Pi}} = \begin{pmatrix} \boldsymbol{\pi}_1 \boldsymbol{A}_k^1 \\ \vdots \\ \boldsymbol{\pi}_N \boldsymbol{A}_k^N \end{pmatrix}.$$

We now derive a non-linear program for the global optimization problem. Equation (4) can be equivalently formulated as

$$\min_{\boldsymbol{\Pi} \in \mathcal{P}} \left(\boldsymbol{\alpha} \boldsymbol{g}_k \right) \quad \text{subject to } \boldsymbol{\pi}_i \boldsymbol{A}_k^i \boldsymbol{g}_k = \boldsymbol{C}_k^{\boldsymbol{\Pi}}(i\bullet)\boldsymbol{g}_k \geq \boldsymbol{r}_k(i) \text{ for all } i \in \mathcal{S}. \tag{7}$$

To see this, let $\boldsymbol{g}_k^*$ be the minimal solution of (4). Then

$$a_i^* = \arg\min_{a \in \mathcal{A}} \left(\boldsymbol{C}_k^a(i\bullet)\boldsymbol{g}_k^* \right)$$

exists for $1 \leq i \leq N$, but must not be unique. By selecting $\boldsymbol{\pi}_i(a_i^*) = 1$, the vector $\boldsymbol{g}_k^*$ becomes the solution of $\boldsymbol{C}_k^{\boldsymbol{\Pi}} \boldsymbol{g}_k^* = \boldsymbol{r}_k$ and $\boldsymbol{g}_k^*$ becomes a feasible solution of (7). This solution is also optimal because every feasible solution $\boldsymbol{g}_k$ has to observe

$$\sum_{m=1}^{M} \boldsymbol{\pi}_i(m) \boldsymbol{C}_k^m(i\bullet)\boldsymbol{g}_k \geq \boldsymbol{r}_k$$

for each $i \in \mathcal{S}$ which implies that the choice of the minimum among $a \in \mathcal{A}$ results in the smallest vector $\boldsymbol{g}_k$. Observe that the vector $\boldsymbol{g}_k$ depends on $\boldsymbol{\Pi}$ due to the constraints. We do not use $\boldsymbol{\Pi}$ as an additional index for $\boldsymbol{g}_k$ to avoid an overloading of notation.

The reformulation of the LP into an NLP (due to the dependencies between $\boldsymbol{g}_k$ and $\boldsymbol{\Pi}$) is of no use for single problems but allows one to describe the coupling of MDPs via a common policy. The following formulation describes the weighted optimization problem for concurrent MDPs.

$$\begin{aligned} &\max\nolimits_{\boldsymbol{\Pi} \in \mathcal{P}} \left(\min\nolimits_{\boldsymbol{g}_k \in \mathbb{R}^{N \times 1}} \sum\nolimits_{k \in \mathcal{M}} \boldsymbol{\alpha}_k \boldsymbol{g}_k \right) \\ &\text{subject to } \forall k \in \mathcal{M}, \forall i \in \mathcal{S}, \forall a \in \mathcal{A}: \\ &\boldsymbol{\pi}_i \boldsymbol{A}_k^i \boldsymbol{g}_k = \boldsymbol{C}_k^{\boldsymbol{\Pi}}(i\bullet)\boldsymbol{g}_k \geq \boldsymbol{r}_k(i), \ \sum_{m=1}^{M} \boldsymbol{\pi}_i(m) = 1, \ \boldsymbol{\pi}_i(a) \geq 0. \end{aligned} \tag{8}$$

It follows from (7) that each feasible vector $\boldsymbol{g}_k$ for MDP k has to observe the constraints. Furthermore it has to be the minimal solution that observes the constraints. The outer maximum operator assures that the maximum of the weighted sum is computed. Vectors $\boldsymbol{\alpha}_k$ encode, by their definition in (6), the weights $\boldsymbol{w}(k)$. The optimization problem has a linear goal function but bilinear constraints which usually define a non-convex feasible region.

The NLP for the dual program can be formulated as

$$\begin{aligned}
&\max_{\boldsymbol{\Pi}\in\mathcal{P}}\left(\max_{\boldsymbol{h}_k\in\mathbb{R}^{N\times 1}}(\boldsymbol{h}_k)^T\boldsymbol{r}_k\right)\\
&\text{subject to } \forall i\in\mathcal{S}:\\
&\sum_{m=1}^{M}\sum_{l=1}^{N}\boldsymbol{\pi}_i(m)\boldsymbol{A}_k^l(m,i)\boldsymbol{h}_k(l)=\left(\boldsymbol{C}_k^{\boldsymbol{\Pi}}(\bullet i)\right)^T\boldsymbol{h}_k=\boldsymbol{\alpha}(i)\text{ and }\boldsymbol{h}_k(i)\geq 0.
\end{aligned}\tag{9}$$

To show the equivalence to (5) let $\boldsymbol{f}_k^*=\left((\boldsymbol{f}_k^1)^T,\ldots,(\boldsymbol{f}_k^M)^T\right)^T$ be an optimal solution of the dual problem. Then define $\boldsymbol{h}_k=\sum_{j=1}^M\boldsymbol{f}_k^j$ and $\boldsymbol{\pi}_i(j)=\boldsymbol{f}_k^j(i)/\boldsymbol{h}_k(i)$. Then $\boldsymbol{h}_k\geq\boldsymbol{0}$ and

$$\begin{aligned}
\left(\boldsymbol{C}_k^{\boldsymbol{\Pi}}\right)^T\boldsymbol{h}_k&=\sum_{m=1}^{M}\left(\begin{pmatrix}\boldsymbol{\pi}_1(m)&&\\&\ddots&\\&&\boldsymbol{\pi}_N(m)\end{pmatrix}\left(\boldsymbol{I}-\gamma\boldsymbol{P}_k^m\right)\right)^T\boldsymbol{h}_k\\
&=\sum_{m=1}^{M}\left(\boldsymbol{I}-\gamma\boldsymbol{P}_k^m\right)^T\begin{pmatrix}\boldsymbol{\pi}_1(m)&&\\&\ddots&\\&&\boldsymbol{\pi}_N(m)\end{pmatrix}\boldsymbol{h}_k\\
&=\sum_{m=1}^{M}\left(\boldsymbol{I}-\gamma\boldsymbol{P}_k^m\right)^T\boldsymbol{f}_k^m\\
&=\boldsymbol{\alpha}
\end{aligned}$$

which shows that $\boldsymbol{h}_k$ is a feasible solution for the NLP. To show that the solution is also maximal, we assume that a solution $\hat{\boldsymbol{h}}_k$ and a policy $\hat{\boldsymbol{\Pi}}$ exist such that the constraints of (9) are observed and $(\hat{\boldsymbol{h}}_k)^T\boldsymbol{r}_k^{\hat{\pi}}>(\boldsymbol{h}_k)^T\boldsymbol{r}_k$. Then define $\hat{\boldsymbol{f}}_k^j(i)=\hat{\boldsymbol{\pi}}_i(j)\hat{\boldsymbol{h}}_k(i)$, which observes the constraints of (5) and is therefore a feasible solution for the dual LP. Then $\sum_{j=1}^M(\hat{\boldsymbol{f}}_k^j)^T\boldsymbol{r}_k=(\hat{\boldsymbol{h}}_k)^T\boldsymbol{r}_k\leq\sum_{j=1}^M(\boldsymbol{f}_k^j)^T\boldsymbol{r}_k=(\boldsymbol{h}_k)^T\boldsymbol{r}_k$ because $\boldsymbol{f}_k^*$ is assumed to be optimal which implies that $\hat{\boldsymbol{h}}_k$ cannot exist and $\boldsymbol{h}_k$ is an optimal solution of (9).

The NLP for the combined MDPs becomes

$$\begin{aligned}
&\max_{\boldsymbol{\Pi}\in\mathcal{P}}\left(\max_{\boldsymbol{h}_k\in\mathbb{R}^{N\times 1}}\sum_{k=1}^{K}(\boldsymbol{h}_k)^T\boldsymbol{r}_k\right)\\
&\text{subject to } \forall k\in\mathcal{M},\forall i\in\mathcal{S},\forall a\in\mathcal{A}:\\
&\sum_{m=1}^{M}\sum_{l=1}^{N}\boldsymbol{\pi}_i(m)\boldsymbol{A}_k^l(m,i)\boldsymbol{h}_k(l)=\left(\boldsymbol{C}_k^{\boldsymbol{\Pi}}(\bullet i)\right)^T\boldsymbol{h}_k=\boldsymbol{\alpha}_k(i),\\
&\sum_{m=1}^{M}\boldsymbol{\pi}_i(m)=1,\ \boldsymbol{\pi}_i(a)\geq 0,\boldsymbol{h}_k(i)\geq 0.
\end{aligned}\tag{10}$$

Observe that $\boldsymbol{h}_k(i)$ reflects the mean discounted number of visits in state i of MDP k if the process starts with initial probability $\boldsymbol{\alpha}$ and MDP k is chosen with probability $\boldsymbol{w}(k)$.

Both problems (8) and (10), belong to the class of non-convex *quadratically constrained linear programs* (QCLP) (Amato et al. 2007) which are a subset of the more general class of *quadratically constrained quadratic programs* (QCQP) and have some specific properties that can be exploited in optimization algorithms.

3.1 Computational complexity

We now briefly show that the general problem is NP-hard which means that it is commonly agreed that no efficient algorithms exist for this kind of problems. First a decision variant of the optimization problem for concurrent MDPs is defined and then it is shown that this problem is NP-complete.

Definition 1 (Decision problem) Given a set of concurrent MDPs $\mathcal{M}$ and real (represented with $\mathcal{O}(NMK)$ bits) vectors and numbers $\boldsymbol{w} \in \mathbb{R}^K$, $g \in \mathbb{R}$, decide if there is a policy $\boldsymbol{\Pi} \in \mathcal{P}$ such that $\sum_{k=1}^{K} \boldsymbol{w}(k) G_k^{\boldsymbol{\Pi}} \geq g$.

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 policy $\boldsymbol{\Pi}$ and then verifies if $\boldsymbol{\Pi}$ fulfills the relation above.

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

Theorem 1 *The decision problem defined in Definition 1 is NP-complete.*

The proof can be found in "Appendix B". It is based on the reduction of 3-SAT (Garey and Johnson 1978) to the above decision problem with two concurrent MDPs.

4 Computation of optimal policies and rewards

Theorem 1 shows that algorithms to compute the optimal policy either require a potentially very long time or stop prematurely with a non-optimal and thus approximate solution, if a feasible solution has been found at all. We introduce five different approaches to approximate or compute the optimal policy and value vectors and analyze afterwards their performance.

4.1 A solution approach using a mixed integer linear program

If we restrict the policies to $\mathcal{P}_{pure}$, then the problem can be formulated as a Mixed Integer Linear Program (MILP). In this case $\boldsymbol{\Pi}$ is a $N \times M$ matrix in $\mathbb{B}$ with one element equal to 1 in each row. The MILP solution uses the dual version of the problem given in (5) and (10).

We first define some vectors and matrices. Let

$$\begin{aligned}
\boldsymbol{d}^T &= \left(\mathbb{1}_M^T \otimes (\boldsymbol{r}_1)^T, \ldots, \mathbb{1}_M^T \otimes (\boldsymbol{r}_K)^T\right) \in \mathbb{R}^{1\times KMN},\\
\boldsymbol{B}_k &= \left(\left(\boldsymbol{I} - \gamma \boldsymbol{P}_k^1\right)^T, \ldots, \left(\boldsymbol{I} - \gamma \boldsymbol{P}_k^M\right)^T\right) \in \mathbb{R}^{N\times NM},\\
\boldsymbol{B} &= \begin{pmatrix} \boldsymbol{B}_1 & & \\ & \ddots & \\ & & \boldsymbol{B}_K \end{pmatrix} \in \mathbb{R}^{KN\times KMN},\\
\boldsymbol{b} &= (\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_K)^T \in \mathbb{R}^{KN\times 1},\\
\boldsymbol{y}^T &= \left((\boldsymbol{y}_1^1)^T, \ldots, (\boldsymbol{y}_1^M)^T, (\boldsymbol{y}_2^1)^T, \ldots, (\boldsymbol{y}_K^M)^T\right) \in \mathbb{R}^{1\times KMN} \text{ where } \boldsymbol{y}_k^j \in \mathbb{R}^{N\times 1},\\
\boldsymbol{\pi} &= (\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_N) \text{ where } \boldsymbol{\pi}_n \in \mathbb{B}^{1\times M}.
\end{aligned} \tag{11}$$

The variable $\boldsymbol{y}_k^m(n)$ describes the discounted number of decisions m in state n of MDP k and characterizes the product $\boldsymbol{\pi}_n(m)\boldsymbol{h}_k(n)$. For the optimal policy $\boldsymbol{\Pi}^*$ the corresponding vectors are $\boldsymbol{y}_k^*$. The introduction of additional variables for products is a common approach for bilinear functions (Qualizza et al. 2012). The integer optimization problem for the computation of the optimal policy is then defined by

$$\begin{aligned}
&\max_{\boldsymbol{y}\in\mathbb{R}^{KMN\times 1}} \boldsymbol{d}^T \boldsymbol{y}\\
&\text{subject to } \forall m \in \{1, \ldots, M\}, \forall k \in \{1, \ldots, K\}, \forall n \in \{1, \ldots, N\} :\\
&\boldsymbol{B}\boldsymbol{y} \le \boldsymbol{b},\ \sum_{m=1}^{M} \boldsymbol{\pi}_n(m) = N - 1,\ \boldsymbol{\pi}_n(m)\boldsymbol{y}_k^m(n) = \boldsymbol{0},\ \boldsymbol{y} \ge \boldsymbol{0}.
\end{aligned} \tag{12}$$

The program is not an ILP since it contains products of Boolean and real variables $\boldsymbol{\pi}_n(m)\boldsymbol{y}_k^m(n) = 0$. However, using a standard trick from ILP modeling (Vielma 2015) each product of this form can be substituted by an additional constraint

$$\boldsymbol{y}_k^m(n) + \boldsymbol{\pi}_n(m)U \le U \tag{13}$$

where U is an upper bound for the value of $\boldsymbol{y}_k^m(n)$. The value of $\boldsymbol{y}_k^m(n)$ is the number of times the system is in state n and action m is chosen multiplied with the weight $\boldsymbol{w}(k)$. Thus, $\boldsymbol{y}_k^m(n) \le \boldsymbol{h}_k(n)$ has to hold. A simple upper bound for $\boldsymbol{h}_k(n)$ is $(1-\gamma)^{-1}$. For our ILP solution $U = (1 - \gamma)^{-1}$ is used; more sophisticated upper bounds will be introduced below.

The resulting ILP contains KMN real variables, MN Boolean variables, N equality constraints, $KMN+KN$ inequality constraints and KMN non-negativity constraints. Modern ILP solvers in principle allow one to compute a global maximum and a policy reaching the maximum. However, for larger problem instances, the effort to close the optimality gap between the best available solution and the computed upper bound might take a long time.

4.2 Using algorithms for non-convex programs

QCQPs have attracted some attention in the literature and until very recently new algorithms for local and global optimization have been published (Castillo et al. 2018;

Park and Boyd 2017; Qualizza et al. 2012). Algorithms for global optimization of QCQPs are based on spatial decomposition of the domain of the variables and apply mixed integer linear programs (MILPs) to compute bounds for the optimal solution (Castillo et al. 2018). Unfortunately, spatial decomposition results in an enormous number of new binary variables in the MILP such that the algorithms can only be applied for small problem instances as already argued in Park and Boyd (2017) and thus are not applicable in our context. However, below we adopt some of the ideas applied in the optimization of general QCQPs for the specific problems resulting from concurrent MDPs. Here we first consider the use of general optimization algorithms for non-convex problems which are at most able to compute local optima.

A large number of optimization algorithms for non-convex optimization problems exist. Usually these algorithms cannot exploit the very specific structure of our problem. We present here two generic formulations that can be used as input for different solvers for constrained non-convex problems.

The general representation equals

$$\begin{array}{l}\max_z f(z) \text{ with gradient } f'(z)\\ \text{and the constraints: } g(z) = \mathbf{0} \text{ and } z \geq \mathbf{0}.\end{array} \tag{14}$$

$f(\cdot)$ is a scalar function, the other functions are vector-valued functions. The above representation can be used as input for standard optimization algorithms like *fmincon* from MATLAB or *sqp* from Octave.

Again we solve (10) and define

$$z = \left(\boldsymbol{h}_1^T, \ldots, \boldsymbol{h}_K^T, \boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_N\right)^T \in \mathbb{R}^{N(K+M)\times 1}.$$

Then

$$f(z) = \sum_{k=1}^{K} \boldsymbol{h}_k^T \boldsymbol{r}_k, \;\; f'(z) = \left(\boldsymbol{r}_1^T, \ldots, \boldsymbol{r}_K^T, \mathbf{0}\right)$$

$$g(z) = \begin{pmatrix} \boldsymbol{\pi}_1 \mathbb{1} - 1 \\ \vdots \\ \boldsymbol{\pi}_N \mathbb{1} - 1 \\ \left(\boldsymbol{C}_1^{\boldsymbol{\Pi}}\right)^T \boldsymbol{h}_1 - \boldsymbol{\alpha}_1 \\ \vdots \\ \left(\boldsymbol{C}_K^{\boldsymbol{\Pi}}\right)^T \boldsymbol{h}_K - \boldsymbol{\alpha}_K \end{pmatrix} \text{ and } z \geq \mathbf{0}.$$

The problem contains $N(K+M)$ variables, $N(K+1)$ equality constraints, and $N(K+M)$ non-negativity constraints. This formulation of the problem results in a non-convex QCLP. Our observation is that general optimization algorithms often show a bad convergence behavior for this kind of problems. The main reason for the convergence problems of the algorithms seems to be the strong coupling between the variables in $\boldsymbol{\Pi}$ and $\boldsymbol{h}_k$ with the equality constraints which often hinders the algorithms to find a successful search direction for larger problems.

The above problem description contains redundant information because policy $\boldsymbol{\Pi}$ completely determines $\boldsymbol{h}_k$. This will now be exploited in a reformulation of the problem with a more complex objective function and less constraints using a problem-specific reformulation. As already mentioned, for each policy $\boldsymbol{\Pi}$, matrix $\boldsymbol{C}_k^{\boldsymbol{\Pi}}$ is a non-singular M-matrix which implies that it has a non-negative inverse. Thus, the gain for policy $\boldsymbol{\Pi}$ can be represented as a function f defined on the domain of policy matrices with

$$f(\boldsymbol{\Pi}) = \sum_{k=1}^{K} \boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k. \tag{15}$$

To compute the gradient, we have to consider the change of one element in $\boldsymbol{\Pi}$. Formally, $\boldsymbol{\Pi}$ can be interpreted as vector $\boldsymbol{\pi}$ of length MN. Changing $\boldsymbol{\pi}_n(m)$ by λ means to add $\lambda \boldsymbol{e}_n^T \boldsymbol{A}_k^n(m\bullet) = \lambda \boldsymbol{e}_n^T(\boldsymbol{e}_n - \gamma \boldsymbol{P}_k^m(n\bullet))$ to $\boldsymbol{C}_k^{\boldsymbol{\Pi}}$. This is a rank-one update such that the inverse matrix can be computed as in Hager (1989). If $\boldsymbol{\Pi}'$ results from $\boldsymbol{\Pi}$ by changing $\boldsymbol{\pi}_n(m) \mathrel{+}= \lambda$ (with $\lambda \leq \boldsymbol{\pi}_n(m)$), then

$$f(\boldsymbol{\Pi}') = f(\boldsymbol{\Pi}) - \lambda \sum_{k=1}^{K} \boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n) \left(\frac{\boldsymbol{A}_k^n(m\bullet)\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k}{1 + \lambda \boldsymbol{A}_k^n(m\bullet)\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)} \right) + o(\lambda^2) \tag{16}$$

for λ small enough such that the inverse matrix exists. The derivative is computed, as usual, with $\lim_{\lambda \to 0} \frac{f(\boldsymbol{\Pi}') - f(\boldsymbol{\Pi})}{\lambda}$ resulting in

$$\nabla f(\boldsymbol{\Pi}) = -\left(\sum_{k=1}^{K} \boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet 1) \boldsymbol{A}_k^1(1\bullet) \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k, \ldots, \sum_{k=1}^{K} \boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet N) \boldsymbol{A}_k^N(M\bullet) \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k \right) \tag{17}$$

and $g(\boldsymbol{\Pi}) = \boldsymbol{\Pi}\mathbb{1} - \mathbb{1}$. The functions $f(\boldsymbol{\Pi})$, $f'(\boldsymbol{\Pi})$ and $g(\boldsymbol{\Pi})$ can be used as input for a non-linear optimization algorithm, and the inequality constraints are no longer needed. The resulting optimization problem has N simple linear constraints, but a complex objective function. As it turns out, it usually shows a much better convergence than the original QCLP, because we can start with a feasible solution, but again, there is no guaranteed convergence of general optimization algorithms applied to the problem formulation.

4.3 Step-wise computation of increasing lower bounds

The previous approaches are standard techniques to solve the resulting optimization problem which reach their limits when problems become larger. The general problem is non-convex and, therefore, hard to solve. However, we now present a step-wise approach which computes increasing lower bounds resulting in a local optimum. Thus, we provide problem-specific local optimization heuristics for the problem.

We first introduce the theoretical base before the new algorithm is introduced. Based on (15) we develop an iterative algorithm. The objective function is linear in $\boldsymbol{h}_k$ and is uniquely determined by the policy $\boldsymbol{\Pi}$ since $\boldsymbol{h}_k^T = \boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}$. The set of policies $\mathcal{P}$ is a convex set built by N distributions of order M. Let $\mathcal{H}$ be the set of all vectors $\boldsymbol{h} = (\boldsymbol{h}_1, \ldots, \boldsymbol{h}_K)^T$ such that $\boldsymbol{h}_k^T = \boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}$ for some policy $\boldsymbol{\Pi}$. Unfortunately, the set $\mathcal{H}$ does not need to be convex for $K > 1$ because for two vectors $\boldsymbol{h}, \boldsymbol{h}' \in \mathcal{H}$ and some $\lambda \in (0, 1)$ the convex combination $\lambda \boldsymbol{h} + (1-\lambda)\boldsymbol{h}'$ does not have to belong to $\mathcal{H}$, even if it can be assured that in each scenario k, the convex combination $\lambda \boldsymbol{h}_k + (1-\lambda)\boldsymbol{h}_k'$ can be realized with some stationary policy. The reason is that matrix inversion that translates between matrices $\boldsymbol{C}_k$ and $\overline{\boldsymbol{C}}_k$ is non-linear in λ, hence, the required policies in the scenarios usually differ. We will clarify this property by means of small examples in Sect. 5.1. Thus, the best we can expect are local optima which will be computed next.

We now consider local modifications of a policy $\boldsymbol{\Pi}$. Let $\boldsymbol{\pi}_n(m) > 0, p \in \{1, \ldots, M\} \setminus \{m\}$ and $\lambda \in [0, \boldsymbol{\pi}_n(m)]$. $\boldsymbol{\Pi}'$ is the policy which results from $\boldsymbol{\Pi}$ by setting $\boldsymbol{\pi}_n(m)$ to 0 and adding $\boldsymbol{\pi}_n(m)$ to $\boldsymbol{\pi}_n(p)$. It is easy to show that $\boldsymbol{\Pi}'$ is a valid policy. Let $\boldsymbol{u}_k = \gamma(\boldsymbol{P}_k^m(n\bullet) - \boldsymbol{P}_k^p(n\bullet))$, then $\boldsymbol{C}_k^{\boldsymbol{\Pi}'} = \boldsymbol{C}_k^{\boldsymbol{\Pi}} - \boldsymbol{\pi}_n(m)\boldsymbol{e}_n^T \boldsymbol{u}_k$ results from a rank-1 update. For the value of the goal function the following relation holds based on the modifications due to rank-1 updates in the inverse matrices (Hager 1989):

$$\begin{aligned} G^{\boldsymbol{\Pi}',\boldsymbol{\Pi},\lambda}(\boldsymbol{w}) &= \sum_{k=1}^{K} \boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}',\boldsymbol{\Pi},\lambda} \boldsymbol{r}_k \\ &= G^{\boldsymbol{\Pi}}(\boldsymbol{w}) + \sum_{k=1}^{K} \underbrace{\lambda \frac{\boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)\boldsymbol{u}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k}{1 - \lambda \boldsymbol{u}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)}}_{g_k(\lambda)}, \end{aligned} \tag{18}$$

where the triple $\boldsymbol{\Pi}', \boldsymbol{\Pi}, \lambda$ in $G^{\boldsymbol{\Pi}',\boldsymbol{\Pi},\lambda}$ and $\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}',\boldsymbol{\Pi},\lambda}$ is a short notation for $\lambda\boldsymbol{\Pi}' + (1-\lambda)\boldsymbol{\Pi}$ (for $0 \le \lambda \le \boldsymbol{\pi}_n(m)$). Now define

$$\zeta_k = \boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)\boldsymbol{u}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k \text{ and } \eta_k = \boldsymbol{u}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n).$$

Both values can be negative and positive but $\eta_k < 1/\boldsymbol{\pi}_n(m)$ holds because it is known that the inverse matrix exists and $\boldsymbol{C}_k^{\boldsymbol{\Pi}'}$ is an M-matrix. Therefore

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

describes the change of the value vector in the kth component if the policy changes from $\boldsymbol{\Pi}$ to $\lambda\boldsymbol{\Pi}' + (1-\lambda)\boldsymbol{\Pi}$. To optimize the gain, the following optimization problem has to be solved.

$$\max_{\lambda \in [0, \boldsymbol{\pi}_n(m)]} \left(\sum_{k=1}^{K} g_k(\lambda) \right) \tag{19}$$

The first two derivatives of the functions are given by

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

The first derivative shows that each function $g_k(\lambda)$ is absolutely monotonic (or constant). The optimum of (19) is either in one of the endpoints, i.e. $\lambda \in \{0, \boldsymbol{\pi}_n(m)\}$, or

$$\sum_{k=1}^{K} \frac{\zeta_k}{(1-\lambda\eta_k)^2} = 0 \quad \text{and} \quad \sum_{k=1}^{K} \frac{2\zeta_k\eta_k}{(1-\lambda\eta_k)^3} < 0$$

has to hold. Since the first derivative is a polynomial of degree $2K-2$, the roots can be computed and checked for optimality.

Let λ^* be the maximum resulting from (19), then the new policy $\boldsymbol{\Pi}'$ results from changing $\boldsymbol{\pi}_n(m)$ to $\boldsymbol{\pi}_n(m) - \lambda^*$ and $\boldsymbol{\pi}_n(p)$ to $\boldsymbol{\pi}_m(p) + \lambda^*$. The resulting value of the objective function is given by (18).

For some policy $\boldsymbol{\Pi} \in \mathcal{P}$ we define

$$\mathcal{N}(\boldsymbol{\Pi}) = \left\{\boldsymbol{\Pi}' | \boldsymbol{\Pi}' \in \mathcal{P} \wedge \boldsymbol{\Pi}' \text{ results from } \boldsymbol{\Pi} \text{ by a rank-1 update}\right\}$$
and
$$\mathcal{N}_1(\boldsymbol{\Pi}) = \left\{\boldsymbol{\Pi}' | \boldsymbol{\Pi}' = \boldsymbol{\Pi} - \lambda \boldsymbol{e}_n^T(\boldsymbol{e}_m - \boldsymbol{e}_p) \text{ for } n \in \mathcal{S}, m, p \in \mathcal{A}, \lambda \in (0, \boldsymbol{\pi}_n(m)]\right\}.$$

Obviously $\mathcal{N}_1(\boldsymbol{\Pi}) \subseteq \mathcal{N}(\boldsymbol{\Pi})$. $\mathcal{N}_1(\boldsymbol{\Pi})$ contains all policies that are considered in (18). The sets $\mathcal{N}(\boldsymbol{\Pi})$ and $\mathcal{N}_1(\boldsymbol{\Pi})$ can also be defined for policies from $\mathcal{P}_{pure}$ rather than $\mathcal{P}$.

Theorem 2 *If $G^{\boldsymbol{\Pi}} < G^{\boldsymbol{\Pi}'}$ for some $\boldsymbol{\Pi}' \in \mathcal{N}(\boldsymbol{\Pi})$, then a policy $\boldsymbol{\Pi}'' \in \mathcal{N}_1(\boldsymbol{\Pi})$ exists such that $G^{\boldsymbol{\Pi}} < G^{\boldsymbol{\Pi}''}$.*

Proof Since $\boldsymbol{\Pi}'$ results from $\boldsymbol{\Pi}$ by a rank-one update and both matrices describe valid policies (i.e., they have unit row sums), they can only differ in one row. Assume that they differ in row n which implies that also $\boldsymbol{C}_k^{\boldsymbol{\Pi}}$ and $\boldsymbol{C}_k^{\boldsymbol{\Pi}'}$ differ also in row n. Let $\boldsymbol{c} = \boldsymbol{\Pi}(n\bullet) - \boldsymbol{\Pi}'(n\bullet)$, then it can be represented as $\boldsymbol{c} = \sum_{h=1}^{H} \boldsymbol{c}_h$ where $\boldsymbol{c}_h \mathbb{1} = 0$ and $\boldsymbol{c}_h$ contains only two non-zero elements such that $\boldsymbol{\Pi} - \boldsymbol{e}_n^T \boldsymbol{c}_h \in \mathcal{N}_1(\boldsymbol{\Pi})$. Define $\boldsymbol{u}_k^h = \boldsymbol{C}_k^{\boldsymbol{\Pi}} - \boldsymbol{C}_k^{\boldsymbol{\Pi} - \boldsymbol{e}_n^T \boldsymbol{c}_h}$, then $\boldsymbol{C}_k^{\boldsymbol{\Pi}'} = \boldsymbol{C}_k^{\boldsymbol{\Pi}} - \boldsymbol{e}_n^T \sum_{h=1}^{H} \boldsymbol{u}_k^h$. Then

$$\begin{aligned} G^{\boldsymbol{\Pi}'}(\boldsymbol{w}) &= G^{\boldsymbol{\Pi}}(\boldsymbol{w}) + \sum_{k=1}^{K} \frac{\boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)\left(\sum_{h=1}^{H} \boldsymbol{u}_k^h\right) \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k}{1 - \boldsymbol{u}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)} \\ &= G^{\boldsymbol{\Pi}}(\boldsymbol{w}) + \sum_{h=1}^{H} \sum_{k=1}^{K} \frac{\boldsymbol{\alpha}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n) \boldsymbol{u}_k^h \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}} \boldsymbol{r}_k}{1 - \boldsymbol{u}_k \overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)} \end{aligned}$$

The denominator is always positive because every matrix for a valid policy is an M-matrix with a non-negative inverse. Thus, it depends on the numerator whether the new policy has a smaller or larger gain. Since $G^{\boldsymbol{\Pi}'}(\boldsymbol{w}) > G^{\boldsymbol{\Pi}}(\boldsymbol{w})$, for at least one h the numerator has to be positive which implies

$$G^{\boldsymbol{\Pi}+\boldsymbol{e}_n^T\boldsymbol{c}_h}(\boldsymbol{w}) = G^{\boldsymbol{\Pi}}(\boldsymbol{w}) + \sum_{k=1}^{K} \frac{\boldsymbol{\alpha}_k\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)\boldsymbol{u}_k^h\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}\boldsymbol{r}_k}{1-\boldsymbol{u}_k^h\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}(\bullet n)} \geq G^{\boldsymbol{\Pi}}(\boldsymbol{w})$$

The inequality holds because the denominator is again positive and the numerator is by assumption positive. Since $\boldsymbol{c}_h$ contains only two non-zero elements $\boldsymbol{C}-\boldsymbol{e}_n^T\boldsymbol{c}_h \in \mathcal{N}_1(\boldsymbol{\Pi})$ and $G^{\boldsymbol{\Pi}+\boldsymbol{e}_n^T\boldsymbol{c}_h}(\boldsymbol{w}) > G^{\boldsymbol{\Pi}}(\boldsymbol{w})$ which completes the proof. □

Algorithm 1 Local optimization algorithm.

```
1: function POLICY_OPT($(\boldsymbol{\alpha}_k, \boldsymbol{A}_k^1, \ldots, \boldsymbol{A}_k^N, \boldsymbol{r}_k)_{k=1,\ldots,K}, \gamma$)
2:     initialize $\boldsymbol{\Pi}$ and compute $\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}$ for $k = 1, \ldots, K$;
3:     repeat
4:         found = false;
5:         for $n = 1 : N$ do
6:             for $m$ with $\boldsymbol{\pi}_n(m) > 0$ do
7:                 for all $p \in \mathcal{M}\setminus\{m\}$ do
8:                     solve (19) for $\lambda^*$;
9:                     if $\lambda^* > 0$ then
10:                        $\boldsymbol{\pi}_n(m)$ −= $\lambda^*$ and $\boldsymbol{\pi}_n(p)$ += $\lambda^*$;
11:                        update $\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}$ for $k = 1, \ldots, K$ using (18);
12:                        found = true;
13:                        exit all for-loops;
14:    until not found
```

The previous steps naturally define Algorithm 1, a local optimization algorithm which can be alternatively formulated as a policy iteration algorithm for the concurrent MDP. The algorithm computes general stationary policies. It will be shown by the following examples that it is often sufficient to compute pure policies. Algorithm 2 is an alternative version of the algorithm which searches for a locally optimal pure policy.

Theorem 3 *Algorithm 1 converges towards a policy $\boldsymbol{\Pi} \in \mathcal{P}$ which is locally optimal, i.e., $G^{\boldsymbol{\Pi}}(\boldsymbol{w}) \geq G^{\boldsymbol{\Pi}'}(\boldsymbol{w})$ for all $\boldsymbol{\Pi}' \in \mathcal{N}(\boldsymbol{\Pi})$.*

Algorithm 2 converges towards a policy $\boldsymbol{\Pi} \in \mathcal{P}_{pure}$ which is locally optimal, i.e., $G^{\boldsymbol{\Pi}}(\boldsymbol{w}) \geq G^{\boldsymbol{\Pi}'}(\boldsymbol{w})$ for all $\boldsymbol{\Pi}' \in \mathcal{N}(\boldsymbol{\Pi}) \cap \mathcal{P}_{pure}$.

Proof We consider Algorithm 1, the proof for Algorithm 2 is similar. The algorithm starts with a valid policy and transforms a valid policy into another valid policy (line 10) with a larger gain. In the nested *for-loops* (lines 7–9) all policies from the set $\mathcal{N}_1(\boldsymbol{\Pi})$ are analyzed, where $\boldsymbol{\Pi}$ is the current policy under investigation. If a policy with a larger gain is found, then this policy is selected as current policy. The nested *for loops* assure that all policies from $\mathcal{N}_1$ are checked until the first one that improves the gain is found and the algorithm stops if no policy in $\mathcal{N}_1(\boldsymbol{\Pi})$ with larger gain exists. By Theorem 2 this implies that no policy with a larger gain exists in $\mathcal{N}(\boldsymbol{\Pi})$ and $\boldsymbol{\Pi}$

Algorithm 2 Local optimization algorithm for pure policies.

```
1: function PURE_POLICY_OPT($(\boldsymbol{\alpha}_k, \boldsymbol{A}_k^1, \ldots, \boldsymbol{A}_k^N, \boldsymbol{r}_k)_{k=1,\ldots,K}, \gamma$)
2:     initialize $\boldsymbol{\Pi}$ with a pure policy and compute $\overline{\boldsymbol{C}}_k^{\boldsymbol{\Pi}}$ for $k = 1, \ldots, K$;
3:     repeat
4:         found = false;
5:         for $n = 1 : N$ do
6:             for $m$ with $\boldsymbol{\pi}_n(m) = 0$ and all $p \in \mathcal{M}\backslash\{m\}$ do
7:                 define $\boldsymbol{\Pi}'$ by setting $\boldsymbol{\pi}_n(m) = 0$ and $\boldsymbol{\pi}_n(p) = 1$;
8:                 Evaluate (18) for $\lambda = 1$;
9:                 if $G^{\boldsymbol{\Pi}'}(\boldsymbol{w}) > G^{\boldsymbol{\Pi}}(\boldsymbol{w})$ then
10:                    found = true;
11:                    exit all for-loops;
12:     until not found
```

is locally optimal. Thus, the algorithm generates an increasing sequence of gains and the corresponding policies and stops if a locally optimal policy is found. Since the optimal gain is bounded, convergence to a local optimum is guaranteed. □

The worst case effort of a single policy improvement step in Algorithm 1 is in $O(KN^3M^2)$. The nested *for-loops* can test up to NM^2 candidates for a new policy and each test has an effort in $O(KN^2)$ to evaluated (18). An improvement step in Algorithm 2 has a worst case effort of $O(KN^3M)$ because the number of pure policies in the neighborhood is restricted to $N(M-1)$ for pure policies. The required number of improvements to reach a local optimum is in the worst case exponential in the parameters N, M and K. However, experiments indicate that only a small number of steps is required to reach a local optimum which is a similar behavior to that of the simplex algorithm for the solution of LPs.

4.4 Computation of upper bounds

Apart from the MILP solver, all algorithms compute only local optima which are lower bounds for the global optimum. If a MILP solver stops prematurely after finding at least one feasible solution, then it provides a lower bound resulting from the best feasible solution and an upper bound resulting from a relaxation of the problem. Now we present a similar approach by introducing the computation of upper bounds. These bounds can be combined with the local optimization approaches to compute lower bounds. The difference between both bounds is denoted as the optimality gap.

Bounds are computed from a relaxation of the NLP (9). Let $G^*(\boldsymbol{w})$ be the optimal gain, $\boldsymbol{\Pi}^*$ an optimal policy for this program and $\boldsymbol{h}^* = (\boldsymbol{h}_1^*, \ldots, \boldsymbol{h}_K^*)$ the optimal value vector. As usual we describe policy $\boldsymbol{\Pi}^*$ by vectors $\boldsymbol{\pi}_i^*$ for $i \in \{1, \ldots, N\}$. Furthermore, we use the variables $y_k^m(n) = \boldsymbol{\pi}_n(m)\boldsymbol{h}_k(n)$. $\boldsymbol{y}_k^T = ((\boldsymbol{y}_k^1)^T, \ldots, (\boldsymbol{y}_k^M)^T)$ with $\boldsymbol{y}_k^m = (\boldsymbol{y}_k^m(1), \ldots, \boldsymbol{y}_k^m(N))^T$ which have already been defined before. For the optimal policy, value vectors are denoted as $\boldsymbol{y}_k^*$. The relaxed version of (9) becomes

$$\begin{array}{l} \max_{\boldsymbol{y}_1,\ldots \boldsymbol{y}_K} \left(\sum_{k=1}^{K} (\boldsymbol{r}_k)^T \left(\sum_{m=1}^{M} \boldsymbol{y}_k^m\right)\right) \\ s.t. \begin{pmatrix} \boldsymbol{B}_1 & & \\ & \ddots & \\ & & \boldsymbol{B}_K \end{pmatrix} \begin{pmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_K \end{pmatrix} = \begin{pmatrix} \boldsymbol{\alpha}_1 \\ \vdots \\ \boldsymbol{\alpha}_K \end{pmatrix}, \; \boldsymbol{y}_k \geq \boldsymbol{0} \text{ for } 1 \leq k \leq K. \end{array} \tag{20}$$

Matrices $\boldsymbol{B}_k$ are defined in (11). (20) defines an LP which is a relaxation of the NLP because the relation between $y_k^m(n)$, $\boldsymbol{h}_k(n)$ and $\boldsymbol{\pi}_n(m)$ is neglected. The optimal solution of (20) corresponds to the optimal solutions for the MDPs $1, \ldots, K$. To obtain stronger relations, additional constraints have to be added. Assume that bounds $h_k^-(n) \leq h_k^*(n) \leq h_k^+(n)$ are known. Then the following set of constraints can be added for $k \in \{1, \ldots, K\}, n \in \{1, \ldots, N\}, m \in \{1, \ldots, M\}$ (Qualizza et al. 2012):

$$\begin{array}{rl} \boldsymbol{h}_k^+(n)\boldsymbol{\pi}_n(m) - \boldsymbol{y}_k^m(n) \geq 0, & (1-\boldsymbol{\pi}_n(m))\boldsymbol{h}_k^+(n) - \sum\limits_{r=1, r\neq m}^{M} \boldsymbol{y}_k^r(n) \geq 0 \\ \boldsymbol{h}_k^-(n)\boldsymbol{\pi}_n(m) - \boldsymbol{y}_k^m(n) \leq 0, & (1-\boldsymbol{\pi}_n(m))\boldsymbol{h}_k^-(n) - \sum\limits_{r=1, r\neq m}^{M} \boldsymbol{y}_k^r(n) \leq 0 \\ \sum\limits_{m=1}^{M} \boldsymbol{\pi}_n(m) = 1, & \boldsymbol{\pi}_n(m) \geq 0 \end{array} \tag{21}$$

Experimental evaluations show that it is often sufficient to consider only the first bound in each of the first two rows. The other two bounds have no or only a minor effect.

Theorem 4 *If $\boldsymbol{h}_k^+(n) = \boldsymbol{h}_k^*(n)$ for all $k \in \{1, \ldots, K\}, n \in \{1, \ldots, N\}, m \in \{1, \ldots, M\}$, then the solution of the LP defined by (20) with the additional constraints (21) equals $G^*(\boldsymbol{w})$ and the vectors $\boldsymbol{y}_k^*$ define an optimal policy with $\boldsymbol{\pi}_n^*(m) = \boldsymbol{y}_k^{m*}(n) / \sum_{r=1}^{M} \boldsymbol{y}_k^{r*}(n)$.*

Proof $\boldsymbol{\pi}^*$ and $\boldsymbol{y}_k^{m*} = \boldsymbol{h}_k^*(n)\boldsymbol{\pi}_n(m)$ fulfill the constraints and define a feasible solution for the LP. Now assume that another feasible solution $\boldsymbol{\pi}_n, \boldsymbol{h}_k$ and $\boldsymbol{y}_k$ exists which results in a larger gain. Then

$$\sum_{k=1}^{K}\sum_{n=1}^{N} \boldsymbol{r}_k(n) \sum_{m=1}^{M} \boldsymbol{y}_k^m(n) > \sum_{k=1}^{K}\sum_{n=1}^{N} \boldsymbol{r}_k(n) \sum_{m=1}^{M} \boldsymbol{y}_k^{m*}(n)$$

which implies for some k, m

$$\sum_{m=1}^{M} \boldsymbol{y}_k^m(n) > \sum_{m=1}^{M} \boldsymbol{y}_k^{m*}(n) = \boldsymbol{h}_k^*.$$

However, due to the additional constraints we have

$$\sum_{m=1}^{M} \boldsymbol{h}_k^*(n)\boldsymbol{\pi}_n(m) - \sum_{m=1}^{M} \boldsymbol{y}_k^m(n) \geq 0 \;\Rightarrow\; \sum_{m=1}^{M} \boldsymbol{y}_k^{m*}(n) \geq \sum_{m=1}^{M} \boldsymbol{y}_k^m(n)$$

which implies that $\boldsymbol{\pi}_n$ cannot exist. □

Theorem 5 *If $\boldsymbol{h}_k^+(n) \geq \boldsymbol{h}_k^*(n)$ for all $k \in \{1,\ldots,K\}, n \in \{1,\ldots,N\}, m \in \{1,\ldots,M\}$, then the solution of the LP defined by* (20) *with the additional constraints* (21) *is an upper bound for $G^*(\boldsymbol{w})$.*

Proof Since $\boldsymbol{\pi}^*$ and $\boldsymbol{y}_k^{m*} = \boldsymbol{h}_k^*(n)\boldsymbol{\pi}_n(m)$ fulfill the constraints and define a feasible solution for the LP, the optimal solution of the LP has to be at least as large as the gain resulting from this feasible solution. □

Simple bounds for $\boldsymbol{h}_k^*(n)$ are $\boldsymbol{h}_k^-(n) = 0$ and $\boldsymbol{h}_k^+(n) = (1-\gamma)^{-1}$. To compute tighter bounds, we first notice that $\boldsymbol{h}_k(n)$ equals the discounted number of visits in state n of MDP k, if the process starts with distribution $\boldsymbol{\alpha}$. Bounds can then be computed from the following LP problem.

$$\begin{aligned} &\boldsymbol{h}_k^-(n) = \boldsymbol{w}(k) \min_{\boldsymbol{y}_k^1,\ldots,\boldsymbol{y}_k^M} \sum_{m=1}^{M} \boldsymbol{y}_k^m(n) \text{ and } \boldsymbol{h}_k^+(n) = \boldsymbol{w}(k) \max_{\boldsymbol{y}_k^1,\ldots,\boldsymbol{y}_k^M} \sum_{m=1}^{M} \boldsymbol{y}_k^m(n) \\ &\text{s.t.} \left(\left(\boldsymbol{I} - \gamma \boldsymbol{P}_k^1\right)^T \cdots \left(\boldsymbol{I} - \gamma \boldsymbol{P}_k^M\right)^T \right) \begin{pmatrix} \boldsymbol{y}_k^1 \\ \vdots \\ \boldsymbol{y}_k^M \end{pmatrix} \geq \boldsymbol{\alpha}_k \end{aligned} \tag{22}$$

The bounds might be further improved if a solution from one of the local optimization algorithms presented in the previous paragraphs is available. Let $\boldsymbol{\Pi}^L$ be the policy and $\boldsymbol{h}_k^L$ the value vector. Then define $G^L(\boldsymbol{w}) = \sum_{k=1}^{K} (\boldsymbol{r}_k)^T \boldsymbol{h}_k^L$. Since the upper bound is at least as large as the best known solution, the following constraint can be added to the LP problem (22).

$$(\boldsymbol{r}_k)^T \boldsymbol{h}_k \geq G^L(\boldsymbol{w}) - \sum_{l=1, l\neq k}^{K} G_k^*(\boldsymbol{w}) \tag{23}$$

where $G_k^*(\boldsymbol{w}) = \max_{\boldsymbol{\Pi}\in\mathcal{P}} \left(G_k^{\boldsymbol{\Pi}}(\boldsymbol{w})\right)$.

For the computation of an upper bound starting from an available locally optimal solution, first the bounds $\boldsymbol{h}_k^{\pm}$ have to be computed. This requires the solution of $2KN+1$ LP problems with NM inequality constraints. Alternatively, the trivial bounds, 0 and $(1-\gamma)^{-1}$, can be used. Finally, an LP with $KNM + NM$ variables, $KN + 4KNM$ inequality and N equality constraints has to be solved. Since the problems have a very regular structure, large instances can be solved with current LP solvers.

Let $G^U(\boldsymbol{w})$ be the solution of the LP problem (20) with the additional constraints (21). The difference $G^U(\boldsymbol{w}) - G^L(\boldsymbol{w})$ defines the optimality gap between the best known solution and the theoretical upper bound. To reduce the gap, additional cutting planes have to be computed which can be done from appropriate LP problems. Observe that the constraints in (20) and (21) are independent of the rewards and it is known that for any subsets $\widehat{\mathcal{M}} \subseteq \mathcal{M}$ and $\widehat{\mathcal{S}}_k \subseteq \mathcal{S}$ for $k \in \widehat{\mathcal{M}}$ the following relation holds.

$$\sum_{k\in\widehat{\mathcal{M}}}\sum_{i\in\widehat{\mathcal{S}}_k} \boldsymbol{h}_k^+(i) \geq \sum_{k\in\widehat{\mathcal{M}}}\sum_{i\in\widehat{\mathcal{S}}_k}\sum_{m\in\mathcal{A}} \boldsymbol{y}_k^{*m}(i) \geq \sum_{k\in\widehat{\mathcal{M}}}\sum_{i\in\widehat{\mathcal{S}}_k} \boldsymbol{h}_k^-(i) \tag{24}$$

We can define the following LP to obtain better bounds.

$$\begin{aligned} &v^-/v^+ = \min/\max \textstyle\sum_{k\in\widehat{\mathcal{M}}}\sum_{i\in\widehat{\mathcal{S}}_k}\sum_{m\in\mathcal{A}} \boldsymbol{y}_k^m(i) \\ &\text{s.t. the constraints defined in (20), (21) and } \sum_{k\in\mathcal{M}} (\boldsymbol{r}_k^T)^T \sum_{m\in\mathcal{A}} \boldsymbol{y}_k^m \geq G^L(\boldsymbol{w}) \end{aligned} \tag{25}$$

If $v^- > \sum_{k\in\widehat{\mathcal{M}}}\sum_{i\in\widehat{\mathcal{S}}_k} \boldsymbol{h}_k^-(i)$ or $v^+ < \sum_{k\in\widehat{\mathcal{M}}}\sum_{i\in\widehat{\mathcal{S}}_k} \boldsymbol{h}_k^+(i)$, the constraints

$$v^- \leq \sum_{k\in\widehat{\mathcal{M}}}\sum_{i\in\widehat{\mathcal{S}}_k}\sum_{m\in\mathcal{A}} \boldsymbol{y}_k^{*m}(i) \quad v^+ \geq \sum_{k\in\widehat{\mathcal{M}}}\sum_{i\in\widehat{\mathcal{S}}_k}\sum_{m\in\mathcal{A}} \boldsymbol{y}_k^{*m}(i)$$

can be added to the LP. If we choose in (24) a single MDP k and a single state i, then the objective function of (25) becomes

$$v^-/v^+ = \min/\max \sum_{m\in\mathcal{A}} \boldsymbol{y}_k^m(i). \tag{26}$$

Then $\boldsymbol{h}_k^+(i) = v^+$ and $\boldsymbol{h}_k^-(i) = v^-$ can be used as better bounds in the original LP. This step does not increase the size of the LP because only the spread of the bounds is reduced.

Algorithm 3 Iterative bound computation.

```
1: function ITERATIVE_OPT($(\boldsymbol{\alpha}_k, \boldsymbol{A}_k^1, \ldots, \boldsymbol{A}_k^N, \boldsymbol{r}_k)_{k=1,\ldots,K}, \gamma$)
2:   compute $\boldsymbol{\Pi}^L$ and $G^L(\boldsymbol{w})$ using Algorithm 1;
3:   repeat
4:     compute $\boldsymbol{y}_k$, $\boldsymbol{\Pi}^U$ and $G^U(\boldsymbol{w})$ by solving LP (20), (21);
5:     if $G^{\boldsymbol{\Pi}^U}(\boldsymbol{w}) > G^L(\boldsymbol{w})$ then
6:       $G^L(\boldsymbol{w}) = G^{\boldsymbol{\Pi}^U}, \boldsymbol{\Pi}^L = \boldsymbol{\Pi}^U$;
7:     if $|G^U(\boldsymbol{w}) - G^L(\boldsymbol{w})| > \epsilon$ then
8:       $(k,i) = \arg\max_{k\in\mathcal{M}, i\in\mathcal{S}} \boldsymbol{h}_k^+(i) - \sum_{m\in\mathcal{A}} \boldsymbol{y}_k^m$ for $(k,i)$ where $\boldsymbol{h}_k^+(i)$ has not been improved for $G^U(\boldsymbol{w})$;
9:       Solve the LP (25) with max-objective function (26) for $\boldsymbol{h}_k^+(i)$;
10:      $(k,i) = \arg\max_{k\in\mathcal{M}, i\in\mathcal{S}} \sum_{m\in\mathcal{A}} \boldsymbol{y}_k^m - \boldsymbol{h}_k^-(i)$ for $(k,i)$ where $\boldsymbol{h}_k^-(i)$ has not been improved for $G^U(\boldsymbol{w})$;
11:      Solve the LP (25) with min-objective function (26) for $\boldsymbol{h}_k^-(i)$;
12:  until iteration limit reached or $|G^U(\boldsymbol{w}) - G^L(\boldsymbol{w})| \leq \epsilon$
```

Algorithm 3 uses an iterative approach that tries to improve the bounds $\boldsymbol{h}_k^+(i)$ or $\boldsymbol{h}_k^-(i)$ with the largest difference to $\boldsymbol{h}_k^m(n)$. It is not guaranteed that the gap between upper and lower bound can be closed by reducing upper and increasing lower bounds of single variables only. Therefore, the algorithm should be stopped if no progress is made or if the number of iterations reaches a threshold. In this case more complex cutting planes as defined in (25) have to be used. However, the number of potential cuts grows then with $NK!$ and it is hard to decide in advance which of more complex cuts results in a successful reduction of the optimality gap. Experience with a large number of models shows that Algorithm 3 usually reduces the optimality gap, without completely closing it.

5 Examples

We analyze the algorithms on different examples. All experiments were run on a PC with an Intel 3.6 GHz processor and 16 GB main memory. Algorithms are implemented in *MATLAB* where the solvers *fmincon* and *intlinproc* are used.[2] We use the following abbreviations for the algorithms.

- *ILP* for the mixed integer linear programming approach presented in Sect. 4.1.
- *QCLP* for the non-linear programming approach presented in Sect. 4.2 that solves the quadratically constrained linear program.
- *NLP* for the non-linear programming approach presented in Sect. 4.2 solving the non-linear program with the general objective function.
- *IMGP* for Algorithm 1.
- *IMPP* for Algorithm 2.
- *Up-simple* for the upper bound with the local bound $(1-\gamma)^{-1}$.
- *Up-general* for the upper bound with local bound computed from (22).

We first consider two small non-convex problem instances, then randomly generated problem instances are evaluated and afterwards a control problem for queues is presented.

5.1 Small examples with an optimal random policies

Although the general problem is non-convex as shown, most instances of the problem have a fairly regular surface such that local search algorithms converge to global optimum and often the best pure policy is optimal or almost optimal. We show this in the following paragraph where randomly generated models are analyzed. Here, we present two small examples which have local optima.

[2] Alternatively, an *Octave* implementation with solvers *sqp* and *glpk* is also available and shows a similar performance.

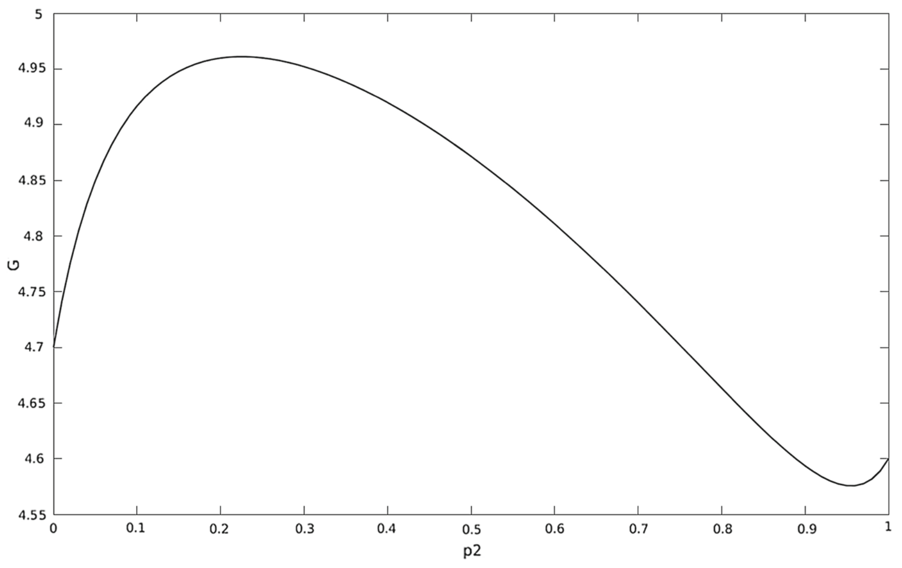

Fig. 1 Gain $G^{\boldsymbol{\Pi}}$ for different probabilities p_2 of choosing a in the second state

We begin with a set of deterministic MDPs with two states and two actions. The following matrices and vectors define the MDPs.

$$
\begin{aligned}
\boldsymbol{P}_1^a &= \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \boldsymbol{P}_1^b = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{r}_1 = \begin{pmatrix} 2/3 \\ 1/3 \end{pmatrix} \\
\boldsymbol{P}_2^a &= \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{P}_2^b = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad \boldsymbol{r}_2 = \begin{pmatrix} 4/5 \\ 1/5 \end{pmatrix} \\
\boldsymbol{P}_3^a &= \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{P}_3^b = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}, \quad \boldsymbol{r}_3 = \begin{pmatrix} 2/5 \\ 3/5 \end{pmatrix}
\end{aligned}
$$

For $\boldsymbol{p} = (0.5, 0.5)$, the optimal pure policy equals

$$\boldsymbol{\Pi}_{det} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \end{pmatrix}$$

with gain $G^{\boldsymbol{\Pi}_{det}} = 4.7$ and the optimal randomized policy is

$$\boldsymbol{\Pi}_{rand} = \begin{pmatrix} 0 & 1 \\ 0.2261 & 0.7739 \end{pmatrix}$$

with gain $G^{\boldsymbol{\Pi}_{rand}} = 4.9611$. Figure 1 shows the gain which is obtained if in state 1 action b is selected and in state 2 action a is selected with probability p_2 ranging from 0 through 1. It can be seen that two local optima exist, namely, the global optimum at $p_2 = 0.2261$ and another local optimum for $p_2 = 1$.

As a second example we consider a model with 4 MDPs with 3 states and 2 actions which is characterized by the following matrices and vectors.

$$\boldsymbol{P}_1^a = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \boldsymbol{P}_1^b = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \boldsymbol{r}_1 = \begin{pmatrix} 7/16 \\ 1/2 \\ 1/16 \end{pmatrix}$$
$$\boldsymbol{P}_2^a = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{pmatrix}, \boldsymbol{P}_2^b = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \boldsymbol{r}_2 = \begin{pmatrix} 1/4 \\ 1/4 \\ 1/2 \end{pmatrix}$$
$$\boldsymbol{P}_3^a = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}, \boldsymbol{P}_3^b = \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}, \boldsymbol{r}_3 = \begin{pmatrix} 1/9 \\ 2/3 \\ 2/9 \end{pmatrix}$$
$$\boldsymbol{P}_4^a = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{pmatrix}, \boldsymbol{P}_4^b = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 0 & 1 \end{pmatrix}, \boldsymbol{r}_4 = \begin{pmatrix} 3/8 \\ 3/8 \\ 1/4 \end{pmatrix}$$

The optimal deterministic policy equals for this example

$$\boldsymbol{\Pi}_{det} = \begin{pmatrix} 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

with gain $G^{\boldsymbol{\Pi}_{det}} = 3.5208$ and the optimal randomized policy is

$$\boldsymbol{\Pi}_{rand} = \begin{pmatrix} 0.1778 & 0.8223 \\ 0.056 & 0.944 \\ 0 & 1 \end{pmatrix}$$

and gain $G^{\boldsymbol{\Pi}_{rand}} = 3.623$. The surface of the gain function for different values p_1 (probability of choosing a in state 1) and p_2 (probability of choosing a in state 2), and selecting b in state 3 is shown in Fig. 2. The surface has four local optima. The deterministic policy with $p_1 = p_2 = 1$ with gain 3.227 is locally optimal, the randomized policies with $p_1 = 0.18$, $p_2 = 1$ and gain 3.4 and with $p_1 = 1$, $p_2 = 0.06$ and gain 3.451 are also locally optimal. Finally, the global optimum at $p_1 = 0.1778$ and $p_2 = 0.056$ is, of course, also locally optimal.

5.2 Random matrices

The five algorithms are compared on sets of randomly generated problems. After fixing K, M, N and γ, stochastic matrices $\boldsymbol{P}_k^m$ are generated from uniformly [0, 1] distributed random values, vectors $\boldsymbol{w}$ and $\boldsymbol{\alpha}$ are also generated from [0, 1] uniformly distributed numbers. To obtain distributions, the rows of $\boldsymbol{P}_k^m$ and the vectors $\boldsymbol{w}, \boldsymbol{\alpha}$ are normalized. Vectors $\boldsymbol{r}_k$ are generated from uniformly [0, 10] distributed numbers.

For the local optimization algorithms, we consider one initial policy without restarts; for *IMPP* the starting point is the policy which deterministically chooses the first action

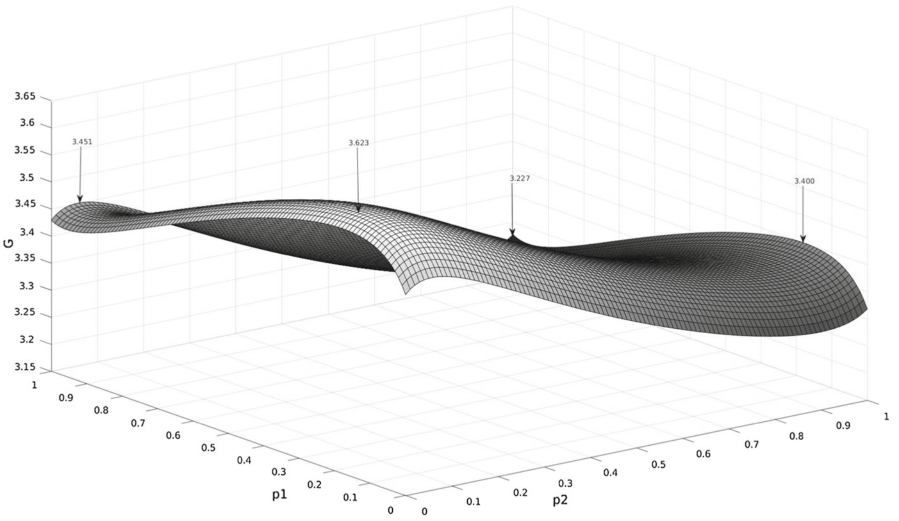

Fig. 2 Surface of the gain function depending on the probabilities p_1 and p_2 of choosing a in the states 1 and 2

in all states, and for *IMGP* we consider the optimal policy for the Kth MDP as the starting point.

The first set of experiments that is presented here is generated for $\gamma = 0.9$ and varying values of K, M, N. Results are presented in Table 1. For each configuration the five algorithms run on 100 randomly generated problem instances. *ti* is the average CPU time per instance, and *diff* is the average deviation from the best value considering all runs where the deviation is at least 0.1%. *div* contains the number of problem instances where the algorithm was not able to find a feasible solution. This cannot happen in the algorithms *IMPP* and *IMGP* where all solutions are feasible, the algorithms improve policies step by step. In *ILP* it could happen that no feasible solution is found by the ILP solver, if the algorithm does not find an integer solution. However, this did not happen in our examples and it also did not happen for *NLP*, therefore *div* is only shown for *QCLP*.

The algorithms *QCLP* and *NLP* perform at most 100,000 function evaluations which is much more than the predefined value in MATLAB, other solver parameters are not modified. There are three horizontal lines in the table. After the first line, the time for the ILP solver is limited to 600 s CPU time to avoid very long solution runs. This also implies that the solver stops prematurely with a non-optimal solution. However, the results indicate that the difference to the optimal policies is small in theses cases. After the second horizontal line we do not use *QCLP* any longer since it is too costly and not able to compute feasible solutions for large problem instances. After the third line also *NLP* was no longer used because it became too costly. For the largest configuration only *IMPP* and *IMGP* can be used. In this case even ILP solvers failed because they were not able to find an integer solution within the given time budget of 600 s, this holds for the ILP solver in MATLAB as well as for CPLEX, which is one of the most efficient available ILP solvers. Analyzing the different runs we can make the following observations:

- Even if the problem is not convex and the optimum needs not to be a pure policy, the optimal pure policy is usually very close to the computed optimum and in most cases the computed pure policy is optimal.
- If the problem instances become larger, then, in all cases we considered, the optimal policy that was found is pure.
- The local optimization algorithms *IMPP* and *IMGP* are fairly robust and fast. *ILP* is slightly less stable and requires much time for larger instances.
- Among the non-linear solvers *NLP* is much more efficient and reliable than *QCLP*. However, for large problem instances also *NLP* fails.

In a second series of experiments we increase γ to 0.999. The corresponding results are shown in Table 2. It can be seen that the results are very similar, the choice of γ seems to have only a minor effect on the behavior of the solvers. The only exception is the solver *QCLP* which almost completely fails because it was not able to compute feasible solutions most of the times.

A third series of experiments concerned itself with deterministic concurrent MDP models. The discount factor was set to $\gamma = 0.9$, and the instances that were generated had the property that the transition probabilities in the individual MDPs were either one or zero. We observe that deterministic models are harder to optimize with exact methods: the ILP solver exceeds the time budget on smaller models, *QCLP* has much more difficulties to converge, and the *NLP* solver yields more often only locally optimal policies. In contrast, performance of local optimization heuristics does not suffer much in comparison to exact methods. We believe that this effect is due to larger differences between deterministic models, that is, the difference between transition probability vectors is not only large across different actions in a state but also across the MDPs in the concurrent model. On larger instances where only local heuristics can be executed efficiently, we observe another interesting effect: if *IMPP* and *IMGP* both use the maximal time budget, the resulting policy of *IMPP* can be slightly better. We conjecture that this can be attributed to, first, smaller steps by *IMGP* in policy space, and second, larger structural differences between the MDPs in the concurrent model which introduce more local optima to the optimization problem (Table 3).

In another series of experiments we compare the upper bounds with the computed results for *IMPP* and *IMGP*. Again randomly generated examples are used. Table 4 shows the maximal and the average relative difference between the upper bounds and the policies found by the algorithms. For the iterative improvement, up to 100 iterations are allowed to improve lower and upper bounds for the values in vector $\boldsymbol{h}_k$. It can be seen that the average differences are small but in a few cases a larger gap occurs. However, these large gaps only occur for small problem instances and the difference between lower bound computed with IMGP and the upper bound is at most 3.1% in all cases and after iterative improvement, it is even smaller. For large models, the maximal deviation from the upper bound is below 1% which is almost negligible.

Table 1 Summary of results for randomly generated instances ($\gamma = 0.9$, 100 problem instances)

K	N	M	ILP			IMPP			IMGP			QCLP				NLP		
			Ti	Nopt	Diff (%)	Ti	Nopt	Diff (%)	Ti	Nopt	Diff (%)	Ti	Nopt	Diff (%)	Div	Ti	Nopt	Diff (%)
2	2	2	0.020	4	1.4	0.001	4	1.4	0.015	1	0.2	0.078	1	4.3	0	0.041	0	–
3	3	3	0.029	7	0.3	0.001	7	0.3	0.026	2	0.2	0.117	2	0.2	3	0.054	7	0.6
2	5	2	0.023	1	0.1	0.002	1	0.1	0.002	0	–	0.113	0	–	0	0.054	0	–
10	5	2	0.046	2	0.2	0.006	4	0.3	0.074	1	0.2	0.642	0	–	1	0.104	0	–
2	10	3	0.056	0	–	0.006	0	–	0.024	0	–	0.394	0	–	5	0.082	2	1.1
2	10	10	3.756	0	–	0.013	1	0.1	0.097	1	0.1	1.895	1	0.1	13	0.697	2	0.2
5	5	5	0.119	2	0.2	0.009	5	0.3	0.053	5	0.2	0.477	3	0.2	5	0.089	5	0.6
3	20	4	586.4	0	–	0.015	0	–	0.068	0	–	4.229	0	–	3	0.283	0	–
3	50	3	597.7	54	0.3	0.069	0	–	0.131	0	–	36.63	0	–	0	2.831	0	–
5	50	5	607.1	100	0.6	0.140	0	–	0.482	0	–	157.4	0	–	2	5.864	0	–
3	100	4	605.8	97	0.5	0.904	0	–	2.408	0	–	404.4	0	–	11	41.46	1	0.9
3	200	4	635.3	100	0.7	2.41	0	–	6.576	0	–	–	–	–	–	20.75	0	–
3	300	4	716.2	100	0.9	5.44	0	–	15.54	0	–	–	–	–	–	780.3	0	–
3	500	4	1095	100	0.7	19.4	0	–	64.0	0	–	–	–	–	–	2914	0	–
3	700	4	–	–	–	66.2	0	–	209.3	0	–	–	–	–	–	–	–	–
5	800	5	–	–	–	206.6	0	–	779.1	0	–	–	–	–	–	–	–	–
3	1000	4	–	–	–	213.8	0	–	755.9	0	–	–	–	–	–	–	–	–

Table 2 Summary of results for randomly generated instances ($\gamma = 0.999$, 100 problem instances)

K	N	M	ILP			IMPP			IMGP			QCLP				NLP		
			Ti	Nopt	Diff (%)	Ti	Nopt	Diff (%)	Ti	Nopt	Diff (%)	Ti	Nopt	Diff (%)	Div	Ti	Nopt	Diff (%)
2	2	2	0.015	5	1.8	0.001	5	2.9	0.018	1	0.2	0.791	0	–	14	0.034	0	–
3	3	3	0.021	5	0.3	0.001	5	0.3	0.021	2	0.7	2.538	1	1.5	45	0.037	5	0.5
2	5	2	0.022	2	0.1	0.001	2	0.1	0.019	0	–	3.104	0	–	52	0.043	0	–
10	5	2	0.041	3	0.2	0.06	5	0.3	0.082	1	0.2	15.47	1	0.2	96	0.097	3	0.1
2	10	3	0.062	0	–	0.007	0	–	0.025	0	–	10.87	0	–	72	0.077	7	0.7
2	10	10	4.903	0	–	0.014	1	0.2	0.102	0	–	36.39	0	–	100	0.457	10	1.3
5	5	5	0.107	1	0.3	0.011	3	0.5	0.060	2	0.3	10.70	0	–	59	0.09	7	0.4
3	20	4	263.0	0	–	0.016	0	–	0.065	0	–	53.50	0	–	92	0.187	2	0.6
3	50	3	521.5	75	0.2	0.057	0	–	0.146	0	–	112.6	0	–	82	1.071	0	–
5	50	5	609.1	100	0.6	0.166	0	–	0.545	0	–	–	–	–	–	1.972	0	–
3	100	4	498.4	96	0.5	0.602	0	–	2.015	0	–	–	–	–	–	8.219	0	–
3	200	4	640.9	99	1.0	2.394	0	–	7.309	0	–	–	–	–	–	47.53	0	–
3	300	4	723.2	100	1.0	5.321	0	–	17.87	0	–	–	–	–	–	142.7	0	–
3	500	4	–	–	–	25.04	0	–	87.30	0	–	–	–	–	–	850.5	3	0.4
3	700	4	–	–	–	73.01	0	–	274.4	0	–	–	–	–	–	–	–	–
5	800	5	–	–	–	247.7	0	–	924.8	0	–	–	–	–	–	–	–	–
3	1000	4	–	–	–	211.1	0	–	714.8	0	–	–	–	–	–	–	–	–

Table 3 Summary of results for randomly generated deterministic MDPs ($\gamma = 0.9$, 100 problem instances)

K	N	M	ILP		IMPP		IMGP		QCLP			NLP	
			Ti	Diff (%)	Ti	Diff (%)	Ti	Diff (%)	Ti	Diff (%)	Div	Ti	Diff (%)
2	2	2	0.017	3.3	0.001	3.3	0.012	0.3	0.053	9.1	1	0.031	9.1
3	3	3	0.080	3.5	0.002	3.7	0.018	0.8	0.166	5.5	4	0.048	1.6
2	5	2	0.069	1.0	0.002	2.0	0.012	1.5	0.151	1.3	7	0.058	3.2
10	5	2	0.173	1.5	0.004	2.0	0.089	1.4	1.608	1.4	25	0.097	0.9
2	10	3	0.164	0.7	0.004	2.0	0.023	1.1	1.141	2.2	44	0.104	1.9
2	10	10	0.478	0.5	0.012	1.1	0.106	0.5	7.270	0.0	88	0.822	1.7
5	5	5	0.268	1.8	0.007	3.3	0.155	0.9	1.030	1.6	36	0.105	2.9
3	20	4	9.681	0.3	0.036	1.3	0.251	0.8	15.54	0.1	97	0.602	2.2
3	50	3	2687	0.2	0.096	1.5	0.579	0.6	45.72	0.3	96	3.219	1.2
5	50	5	3693	0.3	0.181	2.1	4.259	0.7	97.68	1.6	96	5.060	1.6
3	700	4	–	–	193.5	0.8	393.1	0.0	–	–	–	–	–
5	800	5	–	–	605.6	1.0	676.0	0.7	–	–	–	–	–
3	1000	4	–	–	618.2	0.8	709.4	0.2	–	–	–	–	–

Table 4 Difference between the bounds and the optimal policies for general MDPs found by *IMPP* and *IMGP* ($\gamma = 0.9$, 100 problem instances)

K	N	M	IMPP (%)				IMGP (%)					
			$(1-\gamma)^{-1}$		Bound (22)		$(1-\gamma)^{-1}$		Bound (22)		Iterative	
			Avg	Max	Avg	Max	Avg	Max	Avg	Max	Avg	Max
2	2	2	1.2	16.0	0.1	3.1	1.2	16.0	0.1	3.1	0.0	1.2
3	3	3	3.8	11.0	0.5	3.7	3.8	10.4	0.4	3.1	0.1	2.5
5	5	5	6.3	11.3	1.3	3.0	6.3	11.3	1.3	3.0	0.5	2.4
3	10	3	2.6	7.2	0.4	2.0	2.6	7.2	0.4	1.9	0.1	1.3
3	50	3	1.2	2.0	0.2	0.5	1.2	2.0	0.2	0.5	0.1	0.3
3	100	4	1.1	1.4	0.3	0.4	1.1	1.4	0.3	0.4	0.2	0.4
3	200	5	0.8	1.2	0.3	0.4	0.8	1.2	0.3	0.4	0.3	0.3

5.3 A multi-server multi-queue model

To exemplify the differences between the individual scenarios, we provide a more involved example. We consider a simple queue with a capacity of m jobs and c servers. Inter-arrival and service times are exponentially distributed with rate λ and μ, respectively. Customers arriving at a system where the queue is full are lost. Each server can be in one of three states *on*, *starting*, and *off*. A server in state *on* can be *idle* or *working*, depending whether customers to serve are available or not. To reduce energy consumption, servers can be *switched off*, which implies that a server changes its state from *on* to *off*. To reduce the population in the system and the response time for customers, servers in state *off* can be *switched on*, which means that a server changes its state from *off* to *starting*. Afterwards it takes an exponentially distributed time with rate ν until it changes the state to *on* and is ready to serve customers. Decisions to switch servers *on* or *off* can be made upon the arrival or departure of jobs. Models like this one may be used as abstract models for server farms (Gandhi et al. 2010). The reward is defined by the sum of the population in the system and the required energy. This measure, which is often used to combine performance and energy consumption (Wierman et al. 2012), is minimized. We observe that the methods defined above can be used for minimization of the objective function if the reward vectors are negated and a constant is added to ensure non-negative rewards.

Usually, these models are analyzed under some heuristic control strategy by assuming that all parameters are completely known. Here, we consider the situation that some parameters are uncertain and we analyze an optimal strategy using MDPs. We assume that the uncertain parameters keep their values for some time which is too short to anticipate them in an adaptive control strategy.

The model can be naturally defined as an MDP with $(m+1)(c+2)(c+1)/2$ states. The number of decisions depends on the state. To restrict the number of decisions we assume that in a state at most 2 servers can be switched *on* or *off* which implies that up to 5 decisions are available in a state.

As an example we consider a system with $\mu = 1, \nu = 0.1, m = 9, c = 4$ and a uniformly distributed initial vector. The energy consumption in the states *off*, *starting* and *on* equals 0.1, 0.8 and 1.0, respectively. The resulting MDP has 150 states. γ is set to 0.999. We assume that the system may be used in low load $\lambda_{\text{low}} = 0.05$, medium load $\lambda_{\text{med}} = 0.5$ and high load $\lambda_{\text{high}} = 0.95$. The three rates define three different scenarios. The model describes a continuous time MDP. However, by using uniformization (Serfozo 1979) it can be transformed into an equivalent discrete time MDP.

We compute optimal policies for the three scenarios. These policies can be computed from a simple MDP and can be computed in less than a second. Furthermore, policies $\boldsymbol{\Pi}_{weight}$ are computed for different weight vectors. For $\boldsymbol{\Pi}_{weight}$ *IMPP* takes about 2 s, *IMGP* takes about 8 s. *ILP* requires the complete time budget of 600 s, *NLP* takes 200 through 300 s, if it converges, and *QCLP* does most times not converge. The policies for the single scenarios can be expressed by the weight vectors (1, 0, 0), (0, 1, 0) and (0, 0, 1), respectively.

The different policies are then analyzed for the three scenarios and the accumulated gains are computed. Results are summarized in Table 5. The first column contains the weight vector for which the policy is computed and the following three columns include the gain for the scenarios when the policy is applied. In the last column the sums of gains over all scenarios are shown.

Results in Table 5 have been computed with algorithm *IMGP*. To speed up convergence of the algorithm, policy computation for one weight vector is initialized with the optimal policy for the previous weight vector, where policies are computed according to the order in the table. Apart from the policies for the unit weight vectors most policies are not pure. E.g., for $\boldsymbol{w} = (1/3, 1/3, 1/3)$ the best pure policy results in the gains 2176, 6244, and 9507 for the three scenarios which sums up to 17,926. Thus, according to the sum of gains, a general policy, with a gain of 17,749, improves the result by 177 units or 1% compared to a pure policy. Yet the differences in the gains for the different scenarios are significant. It can be seen that the weighted policies are better if the scenario changes during policy execution and the policy is fixed. Of course, the optimal policy depends on the probability of observing each scenario. With equal probabilities, the weight vector (1/3, 1/3, 1/3) results in the best policy which improves the gain between 7.4 and 17.8% compared to the policies that have been computed for single scenarios.

6 Conclusions

In this paper, we discuss a stochastic multi-scenario optimization problem for MDPs, which amounts to optimizing a weighted sum of expected discounted rewards in several MDPs with shared action and state spaces under a common policy. It turns out that this specific problem is computationally hard, which motivates the usage of general tools from mathematical optimization and heuristics. A further property of our problem is that the set of deterministic policies may not contain the optimal policy.

In this work, we have designed and compared several optimization approaches to the aforementioned problem, namely, mixed-integer programming, general non-

Table 5 Rewards for different weight vectors and general policies in the three scenarios

$(w_{low}, w_{med}, w_{high})$	Scenario λ_{low}	Scenario λ_{med}	Scenario λ_{high}	Sum
(1.0, 0.0, 0.0)	1697	9172	10,042	20,911
(0.8, 0.2, 0.0)	1795	6706	9930	18,431
(0.8, 0.0, 0.2)	1745	7342	9508	18,594
(0.6, 0.4, 0.0)	1949	6097	9718	17,764
(0.6, 0.2, 0.2)	1899	6714	9310	17,923
(0.6, 0.0, 0.4)	1823	7180	9347	18,350
(0.4, 0.6, 0.0)	1995	6047	9711	17,753
(0.4, 0.4, 0.2)	2166	6146	9484	17,796
(0.4, 0.2, 0.4)	1989	6515	9289	17,793
(0.4, 0.0, 0.6)	2126	6002	9699	17,827
(1/3, 1/3, 1/3)	1978	6060	9711	17,749
(0.2, 0.8, 0.0)	2126	6002	9699	17,827
(0.2, 0.6, 0.2)	2278	6063	9607	17,948
(0.2, 0.4, 0.4)	2325	6194	9383	17,901
(0.2, 0.2, 0.6)	2121	6465	9273	17,859
(0.2, 0.0, 0.8)	1945	6756	9297	17,998
(0.0, 1.0, 0.0)	3410	5967	9694	19,070
(0.0, 0.8, 0.2)	3424	5979	9593	18,996
(0.0, 0.6, 0.4)	3437	6005	9425	18,867
(0.0, 0.4, 0.6)	3883	6175	9309	19,366
(0.0, 0.2, 0.8)	4245	6344	9247	19,836
(0.0, 0.0, 1.0)	4245	6344	9247	19,836

linear optimization, quadratically constrained linear programming, and local search heuristics. It turns out that from the exact methods used, mixed-integer programming and the NLP formulation yield good and often optimal results for small problem instances. However, the methods are limited in their performance and take a significant amount of time on instances with more than 300 states.

On the other hand, the local search heuristics that we have designed seem to perform well not only with respect to time complexity but also with respect to the quality of solutions. Specifically, the local search heuristics showed an acceptable performance even on instances with 1000 states. The deviation from the optimal solutions also has been small: the largest deviation was around 0.4%; for larger instances the difference dropped to almost zero.

A further investigation direction was the difference between randomized and deterministic policies. Here, a similar picture can be seen: the largest deviation occurred on small instances and was 3.7%, for larger instances we could observe smaller deviations of at most 0.6%.

This allows us to conclude that, while the general problem is computationally hard, heuristics seem to perform fairly well. Furthermore, even with stochastic policies being optimal in the general case, restricting oneself to deterministic policies does not seem to impose great limitations with respect to the policy performance.

Future work We think of several possible extensions of this work. First, one may extend the results from the expected discounted reward measure to the expected average reward measure. Furthermore, the model may be extended by replacing MDPs with more general formalisms such as BMDPs (Givan et al. 2000), generic MDPs with uncertainties (Satia and Lave 1973; White and Eldeib 1994), or stochastic games (Filar and Vrieze 1997). More specifically, our results can be applied as a discretization method for continuous uncertainty sets with a probability measure on them. Furthermore, it is possible to extend the methods to the finite horizon case and compare the results with those derived in Steimle et al. (2018).

Another possibility lies in considering more general, not necessarily stationary policies to reduce the uncertainty on the scenario. It might be possible to infer the transition probabilities after a limited amount of time steps and use then a policy that is better suited for the specific scenario.

Appendix A: Action dependent rewards

Assume that rewards of a concurrent MDP depend on actions and successor states. Then $\boldsymbol{R}_k^a(s, s')$ is the reward that is obtained in state s of MDP k if action a is chosen and s' is the successor state. Let

$$\left(\mathcal{S}, \boldsymbol{\alpha}, \left(\boldsymbol{P}_k^a\right)_{a\in\mathcal{A}}, \left(\boldsymbol{R}_k^a\right)_{a\in\mathcal{A}}\right) \tag{27}$$

be a concurrent MDP with action-dependent rewards. We assume that this MDP should be analyzed for the discounted reward with discount factor $\gamma \in (0, 1)$. The transformation into an MDP with rewards that do not depend on the action results in the following MDP which depends on γ.

$$\begin{aligned}
&\left(\tilde{\mathcal{S}}, \tilde{\boldsymbol{\alpha}}, \left(\tilde{\boldsymbol{P}}_k^a\right)_{a\in\mathcal{A}}, \tilde{\boldsymbol{r}}\right), \text{ where}\\
&\tilde{\mathcal{S}} = \mathcal{S} \cup \left\{(s, a, s') | s, s' \in \mathcal{S}, a \in \mathcal{A}, \boldsymbol{P}_k^a(s, a, s') > 0\right\}\\
&\tilde{\boldsymbol{\alpha}} \in \mathbb{R}_{\geq 0}^{|\tilde{\mathcal{S}}|\times 1}, \tilde{\boldsymbol{\alpha}}\mathbb{1} = 1 \text{ with } \tilde{\boldsymbol{\alpha}}(\boldsymbol{\sigma}) = \begin{cases} \boldsymbol{\alpha}(s) & \text{if } \boldsymbol{\sigma} = s,\\ 0 & \text{otherwise,} \end{cases}\\
&\tilde{\boldsymbol{P}}_k^a \in \mathbb{R}_{\geq 0}^{|\tilde{\mathcal{S}}|\times|\tilde{\mathcal{S}}|}, \tilde{\boldsymbol{P}}_k^a\mathbb{1} = \mathbb{1} \text{ with } \tilde{\boldsymbol{P}}_k^a(\boldsymbol{\sigma}, \boldsymbol{\sigma'}) = \begin{cases} \boldsymbol{P}_k^a(s, s') & \text{if } \boldsymbol{\sigma} = s \text{ and } \boldsymbol{\sigma'} = (s, a, s'),\\ 1.0 & \text{if } \boldsymbol{\sigma} = (s, a, s') \text{ and } \boldsymbol{\sigma'} = s',\\ 0 & \text{otherwise,} \end{cases}\\
&\tilde{\boldsymbol{r}} \in \mathbb{R}^{1\times|\tilde{\mathcal{S}}|} \text{ with } \tilde{\boldsymbol{r}}(\boldsymbol{\sigma}) = \begin{cases} \frac{\boldsymbol{R}_k^a(s,s')}{\sqrt{\gamma}} & \text{if } \boldsymbol{\sigma} = (s, a, s'),\\ 0 & \text{otherwise.} \end{cases}
\end{aligned} \tag{28}$$

Observe that the state space $\tilde{\mathcal{S}}$ consists of states $s \in \mathcal{S}$ and states (s, a, s'). The modified MDP alternates between states $s \in \tilde{\mathcal{S}} \cap \mathcal{S}$ and states $(s, a, s') \in \tilde{\mathcal{S}} \backslash \mathcal{S}$. Rewards are only gained in the latter states.

For some policy $\boldsymbol{\Pi}$ defined on $\mathcal{S}$ we define a policy $\tilde{\boldsymbol{\Pi}}$ on $\tilde{\mathcal{S}}$ as follows

$$\tilde{\boldsymbol{\pi}}_{\boldsymbol{\sigma}}(a) = \begin{cases} \boldsymbol{\pi}_s(a) & \text{if } \boldsymbol{\sigma} = s \in \mathcal{S}, \\ 1 & \text{if } \boldsymbol{\sigma} = (s, a, s'), \\ 0 & \text{otherwise.} \end{cases}$$

Thus, a policy for the MDP with action-dependent rewards can be uniquely translated into a policy for the MDP with action-independent rewards. Define the following sequences of vectors

$$\begin{aligned} &\boldsymbol{g}^0 = \boldsymbol{0} \in \mathbb{R}^{|\mathcal{S}| \times 1},\ \boldsymbol{g}^h(s) = \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \boldsymbol{\pi}_s(a) \boldsymbol{P}_k^a(s, s') \left(\boldsymbol{R}_k^a(s, s') + \gamma \boldsymbol{g}^{h-1}(s') \right) \\ &\tilde{\boldsymbol{g}}^0 = \boldsymbol{0} \in \mathbb{R}^{|\tilde{\mathcal{S}}| \times 1},\ \tilde{\boldsymbol{g}}^h(\boldsymbol{\sigma}) = \sum_{\boldsymbol{\sigma}' \in \tilde{\mathcal{S}}} \sum_{a \in \mathcal{A}} \tilde{\boldsymbol{\pi}}_s(a) \tilde{\boldsymbol{P}}_k^a(\boldsymbol{\sigma}, \boldsymbol{\sigma}') \left(\tilde{\boldsymbol{r}}(\boldsymbol{\sigma}) + \sqrt{\gamma} \tilde{\boldsymbol{g}}^{h-1}(\boldsymbol{\sigma}') \right) \end{aligned}$$

The vectors $\boldsymbol{g}^h$ and $\tilde{\boldsymbol{g}}^h$ contain the expected discounted rewards over the course of h transitions under the policy $\boldsymbol{\Pi}$ resp. $\tilde{\boldsymbol{\Pi}}$. Now we show the main relation between $\boldsymbol{g}^h$ and $\tilde{\boldsymbol{g}}^h$.

Theorem 6 *For all $s \in \mathcal{S}$ and all $h \in \mathbb{N}_0$ it is $\boldsymbol{g}^h(s) = \tilde{\boldsymbol{g}}^{2h}(s) = \tilde{\boldsymbol{g}}^{2h+1}(s)$.*

Proof We prove the correspondence of the values in the vectors for h and $2h$ by induction. For $h = 0$ we have by definition of the vectors $\boldsymbol{g}^0(s) = \tilde{\boldsymbol{g}}^0(s) = 0$. Now assume that the correspondence holds for $h \geq 0$, then

$$\begin{aligned} \tilde{\boldsymbol{g}}^{2h+2}(s) &= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \tilde{\boldsymbol{\pi}}_s(a) \tilde{\boldsymbol{P}}_k^a(s, (s, a, s')) \left(\tilde{\boldsymbol{r}}(s) + \sqrt{\gamma} \tilde{\boldsymbol{g}}^{2h+1}(s, a, s') \right) \\ &= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \boldsymbol{\pi}_s(a) \boldsymbol{P}_k^a(s, s') \left(\sum_{s'' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} \tilde{\boldsymbol{\pi}}_{(s,a,s')}(a') \tilde{\boldsymbol{P}}_k^{a'}((s, a, s'), s'') \right. \\ &\quad \left. \times \left(\sqrt{\gamma} \tilde{\boldsymbol{r}}(s, a, s') + \gamma \tilde{\boldsymbol{g}}^{2h}(s'') \right) \right) \\ &= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \boldsymbol{\pi}_s(a) \boldsymbol{P}_k^a(s, s') \left(\sqrt{\gamma} \tilde{\boldsymbol{r}}(s, a, s') + \gamma \tilde{\boldsymbol{g}}^{2h}(s') \right) \\ &= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \boldsymbol{\pi}_s(a) \boldsymbol{P}_k^a(s, s') \left(\sqrt{\gamma} \frac{\boldsymbol{R}_k^a(s,s')}{\sqrt{\gamma}} + \gamma \boldsymbol{g}^h(s') \right) \\ &= \boldsymbol{g}^{h+1}(s) \end{aligned}$$

which implies that it holds for $h + 1$. Now we consider $\tilde{\boldsymbol{g}}^{2h+1}(s)$. For $h = 0$ we have

$$\tilde{\boldsymbol{g}}^1(s) = \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \boldsymbol{\pi}_s(a) \tilde{\boldsymbol{P}}_k^a(s, (s, a, s')) \left(\tilde{\boldsymbol{r}}(s) + \sqrt{\gamma} \tilde{\boldsymbol{g}}^0(s, a, s') \right) = 0$$

because $\tilde{\boldsymbol{r}}(s) = 0$ and $\tilde{\boldsymbol{g}}^0 = \mathbf{0}$. Now assume that the result holds for $2h + 1$, we show that it holds for $2h + 3$.

$$\begin{aligned}
\tilde{\boldsymbol{g}}^{2h+3}(s) &= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \tilde{\boldsymbol{\pi}}_s(a) \tilde{\boldsymbol{P}}_k^a(s, (s, a, s')) \left(\tilde{\boldsymbol{r}}(s) + \sqrt{\gamma}\, \tilde{\boldsymbol{g}}^{2h+2}(s, a, s') \right) \\
&= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \boldsymbol{\pi}_s(a) \boldsymbol{P}_k^a(s, s') \left(\sum_{s'' \in \mathcal{S}} \sum_{a' \in \mathcal{A}} \tilde{\boldsymbol{\pi}}_{(s,a,s')}(a') \tilde{\boldsymbol{P}}_k^{a'}((s, a, s'), s'') \right. \\
&\quad \left. \times \left(\sqrt{\gamma}\, \tilde{\boldsymbol{r}}(s, a, s') + \gamma \tilde{\boldsymbol{g}}^{2h+1}(s'') \right) \right) \\
&= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \boldsymbol{\pi}_s(a) \boldsymbol{P}_k^a(s, s') \left(\sqrt{\gamma} \frac{\boldsymbol{R}_k^a(s, s')}{\sqrt{\gamma}} + \gamma \boldsymbol{g}^h(s') \right) \\
&= \boldsymbol{g}^{h+1}(s)
\end{aligned}$$

□

Let $G^{\boldsymbol{\Pi}}(\gamma) = \boldsymbol{\alpha} \lim_{h \to \infty} \boldsymbol{g}^h$ and $\tilde{G}^{\tilde{\boldsymbol{\Pi}}}(\sqrt{\gamma}) = \tilde{\boldsymbol{\alpha}} \lim_{h \to \infty} \tilde{\boldsymbol{g}}^h$ the discounted gains for the two MDPs with discount factors $\gamma \in (0, 1)$ and $\sqrt{\gamma}$, respectively.

Theorem 7 *For any policy $\boldsymbol{\Pi}$ and any discount factor $\gamma \in (0, 1)$ the relation*

$$G^{\boldsymbol{\Pi}}(\gamma) = \tilde{G}^{\tilde{\boldsymbol{\Pi}}}(\sqrt{\gamma})$$

holds.

Proof First, we show the existence of $G^{\boldsymbol{\Pi}}(\gamma)$. Since $\boldsymbol{R}_k^a(s, s')$ is finite for all $a \in \mathcal{A}$ and for all $s, s' \in \mathcal{S}$ and can be bounded by a real value $R \geq \left| \boldsymbol{R}_k^a(s, s') \right|$, the sequence $\left(\boldsymbol{g}^h \right)_{h \in \mathbb{N}_0}$ adheres to

$$\begin{aligned}
\left| \boldsymbol{g}^{h+i}(s) - \boldsymbol{g}^i(s) \right| &= \sum_{s' \in \mathcal{S}} \sum_{a \in \mathcal{A}} \boldsymbol{\pi}_s(a) \boldsymbol{P}_k^a(s, s') \left| \boldsymbol{R}_k^a(s, s') + \gamma \boldsymbol{g}^{h+i-1}(s') - \boldsymbol{R}_k^a(s, s') \right. \\
&\quad \left. - \gamma \boldsymbol{g}^{i-1}(s') \right| \\
&\leq \gamma \max_{s' \in \mathcal{S}} \left| \boldsymbol{g}^{h+i-1}(s') - \boldsymbol{g}^{i-1}(s') \right| .
\end{aligned}$$

By induction, it follows that

$$\begin{aligned}
\left| \boldsymbol{g}^{h+i}(s) - \boldsymbol{g}^i(s) \right| &\leq \gamma^i \max_{s' \in \mathcal{S}} \left| \boldsymbol{g}^h(s') \right| \\
&\leq \gamma^i R \sum_{j=0}^{h} \gamma^j \\
&= \gamma^i R \frac{1 - \gamma^h}{1 - \gamma} \\
&\leq \gamma^i \frac{R}{1 - \gamma}
\end{aligned}$$

for $i \in \mathbb{N}$ which implies the Cauchy convergence criterion. Then Theorem 6 implies that $\boldsymbol{g}(s) = \lim_{h\to\infty} \boldsymbol{g}^h(s) = \lim_{h\to\infty} \tilde{\boldsymbol{g}}^{2h}(s) = \lim_{h\to\infty} \tilde{\boldsymbol{g}}^h(s) = \tilde{\boldsymbol{g}}(s)$ for all $s \in \mathcal{S}$ and we have

$$\begin{aligned} G^{\boldsymbol{\Pi}}(\gamma) &= \sum_{s\in\mathcal{S}} \boldsymbol{\alpha}(s)\boldsymbol{g}(s) \\ &= \sum_{s\in\mathcal{S}} \tilde{\boldsymbol{\alpha}}(s)\tilde{\boldsymbol{g}}(s) && \text{since } \tilde{\boldsymbol{\alpha}}(s) = \boldsymbol{\alpha}(s) \text{ for } s \in \mathcal{S} \\ &= \sum_{\sigma\in\tilde{\mathcal{S}}} \tilde{\boldsymbol{\alpha}}(\sigma)\tilde{\boldsymbol{g}}(\sigma) && \text{since } \tilde{\boldsymbol{\alpha}}(\sigma) = 0 \text{ for } \sigma \notin \mathcal{S} \\ &= \tilde{G}^{\boldsymbol{\Pi}}(\sqrt{\gamma}). \end{aligned}$$

□

Analogously the following result can be shown.

Theorem 8 *For any policy $\tilde{\boldsymbol{\Pi}}$ in the action-independent MDP $\left(\tilde{\mathcal{S}}, \tilde{\boldsymbol{\alpha}}, \left(\tilde{\boldsymbol{P}}^a_k\right)_{a\in\mathcal{A}}, \tilde{\boldsymbol{r}}\right)$, the policy $\boldsymbol{\Pi}$ in the action-dependent MDP derived by projection of $\tilde{\boldsymbol{\Pi}}$ on $\mathcal{S}$ is subject to*

$$G^{\boldsymbol{\Pi}}(\gamma) = \tilde{G}^{\tilde{\boldsymbol{\Pi}}}(\sqrt{\gamma})$$

This implies that every policy (including the optimal policy) in the MDP with action-independent rewards yields the same expected discounted reward as its projection in the MDP with action-dependent rewards. Together with Theorem 7 this means that there exists a one-to-one correspondence between policies in both MDPs which yields the same expected discounted reward, and optimal policies in one MDP are also optimal in the other one. This completes the transformation from the action-dependent reward model.

Appendix B: Proof of Theorem 1

Theorem 1 *The decision problem defined in Definition 1 is NP-complete.*

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 $\mathcal{M} = \{1, 2\}$ consisting of two MDPs, a vector $\boldsymbol{w} \in \mathbb{R}^2$ and a real number $g \in \mathbb{R}$ such that the instance will be satisfiable if and only if there is a policy $\boldsymbol{\Pi}$ for the 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 a specially designated sink state s_0 which yields 0 reward in both MDPs.

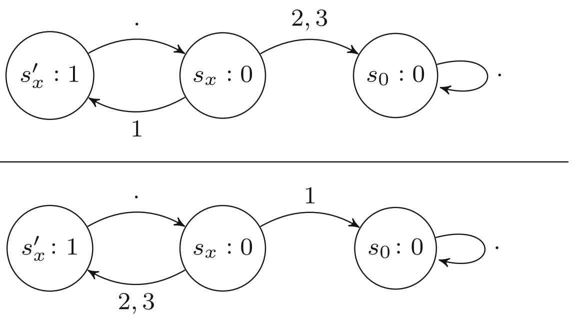

Fig. 3 The variable gadget

- 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 of the form s_x and 1 in states of the form 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 sink state s_0, it is $\boldsymbol{P}_k^a(s_0, s_0) = 1$ for all $a \in \mathcal{A}$ and $k \in \mathcal{M}$.
- In the variable states, we define $\boldsymbol{P}_1^1(s_x, s'_x) = \boldsymbol{P}_1^2(s_x, s_0) = \boldsymbol{P}_1^3(s_x, s_0) = 1$ and $\boldsymbol{P}_2^1(s_x, s_0) = \boldsymbol{P}_2^2(s_x, s'_x) = \boldsymbol{P}_2^3(s_x, s'_x) = 1$, 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 $\boldsymbol{P}_k^a(s'_x, s_x) = 1$ 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. 3 where the upper part corresponds to the first MDP and the lower part corresponds to the second MDP 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 $\boldsymbol{P}_k^a(C, s)$ by setting $\boldsymbol{P}_k^a(C, s) = 1$ 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. 4. 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 α with $\alpha(s_C) = 1/m$ for all clauses C for both MDPs and weights $\boldsymbol{w} = (1/2, 1/2)$. Concerning

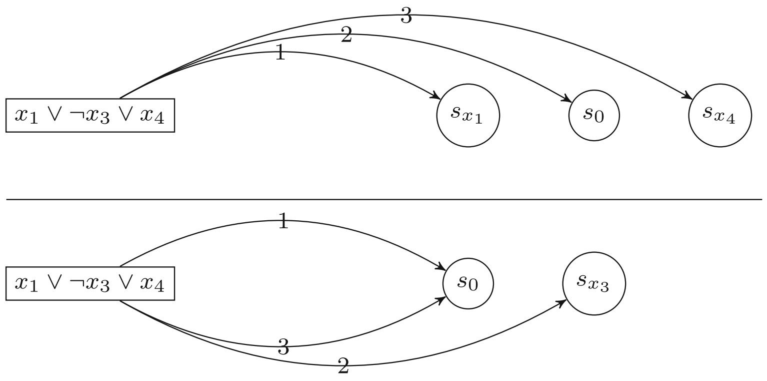

Fig. 4 The clause gadget

the value, we set an auxiliary constant $q := \frac{1}{1-\gamma^2}$ and the required value $g := \frac{\gamma^2 q}{2}$ where γ is a non-zero discount factor in the concurrent MDP.

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 policy π such that $\sum_{k=1}^{K} \boldsymbol{w}(k) G_k^{\boldsymbol{\Pi}} \geq g$. We construct the policy in two steps. In the first step, we set $\pi_{s_x}(1) = 1 \Leftrightarrow \beta(x) = 1$ 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 $\pi_{s_C}(a) = 1 \Leftrightarrow \nu(C) = a$.

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 $\gamma^2 q$ 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, summing over both MDPs, 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

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

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

$$\frac{1 \pm \sqrt{1 - 4\gamma^{-2} + 4\gamma^{-2}\sqrt{4 - \gamma^2}}}{2}. \tag{29}$$

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 (29) is thus greater than one. This means that the whole term (29) 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.

References

Amato C, Bernstein DS, Zilberstein S (2007) Solving POMDPs using quadratically constrained linear programs. In: Proceedings of the 20th international joint conference on artificial intelligence, IJCAI 2007. Hyderabad, India, January 6–12, 2007, pp 2418–2424

Berman A, Plemmons RJ (1994) Nonnegative matrices in the mathematical sciences. Classics in applied mathematics. SIAM, Philadelphia

Bertsimas D, Mišić VV (2017) Robust product line design. Oper Res 65(1):19–37

Bertsimas D, Silberholz J, Trikalinos T (2016) Optimal healthcare decision making under multiple mathematical models: application in prostate cancer screening. Health Care Manag Sci 21:105–118

Björklund H, Vorobyov S (2007) A combinatorial strongly subexponential strategy improvement algorithm for mean payoff games. Discrete Appl Math 155(2):210–229. https://doi.org/10.1016/j.dam.2006.04.029

Caro F, Das-Gupta A (2015) Robust control of the multi-armed bandit problem. Ann Oper Res. https://doi.org/10.1007/s10479-015-1965-7

Castillo AC, Castro PM, Mahalec V (2018) Global optimization of MIQCPs with dynamic piecewise relaxations. J Glob Optim 71(4):691–716. https://doi.org/10.1007/s10898-018-0612-7

Colvin M, Maravelias CT (2010) Modeling methods and a branch and cut algorithm for pharmaceutical clinical trial planning using stochastic programming. Eur J Oper Res 203(1):205–215

d'Epenoux F (1963) A probabilistic production and inventory problem. Manag Sci 10(1):98–108. https://doi.org/10.1287/mnsc.10.1.98

Dupacová J, Consigli G, Wallace SW (2000) Scenarios for multistage stochastic programs. Ann Oper Res 100(1–4):25–53. https://doi.org/10.1023/A:1019206915174

Ehrgott M (2005) Multicriteria optimization, 2nd edn. Springer, Berlin. https://doi.org/10.1007/3-540-27659-9

Feinberg EA, Schwartz A (eds) (2002) Handbook of Markov decision processes. Kluwer, Boston

Filar J, Vrieze K (1997) Competitive Markov decision processes. Springer, New York

Gandhi A, Gupta V, Harchol-Balter M, Kozuch MA (2010) Optimality analysis of energy-performance trade-off for server farm management. Perform Eval 67(11):1155–1171

Garey MR, Johnson DS (1978) Computers and intractability: a guide to the theory of NP-completeness. Freeman, San Francisco

Givan R, Leach SM, Dean TL (2000) Bounded-parameter Markov decision processes. Artif Intell 122(1–2):71–109

Hager WW (1989) Updating the inverse of a matrix. SIAM Rev 31(2):221–239

Iyengar GN (2005) Robust dynamic programming. Math Oper Res 30(2):257–280

Kaelbling LP, Littman ML, Cassandra AR (1998) Planning and acting in partially observable stochastic domains. Artif Intell 101(1–2):99–134

Klamroth K, Köbis E, Schöbel A, Tammer C (2013) A unified approach for different concepts of robustness and stochastic programming via non-linear scalarizing functionals. Optimization 62(5):649–671

Mercier L, Hentenryck PV (2008) Amsaa: a multistep anticipatory algorithm for online stochastic combinatorial optimization. In: Perron L, Trick MA (eds) Integration of AI and OR techniques in constraint programming for combinatorial optimization problems, 5th international conference, CPAIOR 2008, Paris, France, May 20–23, 2008, Proceedings. Lecture Notes in Computer Science, vol 5015, pp 173–187. Springer

Nesterov Y, Nemirovskii A (1994) Interior-point polynomial algorithms in convex programming. Society for Industrial and Applied Mathematics, Philadelphia

Nilim A, Ghaoui LE (2005) Robust control of Markov decision processes with uncertain transition matrices. Oper Res 53(5):780–798

Papadimitriou CH, Tsitsiklis JN (1987) The complexity of Markov decision processes. Math Oper Res 12(3):441–450

Park J, Boyd S (2017) Heuristics for nonconvex quadratically constrained quadratic programming. CoRR arXiv:1703.07870v2

Puterman ML (2005) Markov decision processes. Wiley, London

Qualizza A, Belotti P, Margot F (2012) Linear programming relaxations of quadratically constrained quadratic programs. In: Lee J, Leyffer S (eds) Mixed integer nonlinear programming, vol 154. Springer, New York

Raskin J, Sankur O (2014) Multiple-environment Markov decision processes. CoRR arXiv:1405.4733

Rockafellar RT, Wets RJ (1991) Scenarios and policy aggregation in optimization under uncertainty. Math Oper Res 16(1):119–147

Roijers DM, Scharpff J, Spaan MTJ, Oliehoek FA, de Weerdt M, Whiteson S (2014) Bounded approximations for linear multi-objective planning under uncertainty. In: Chien SA, Do MB, Fern A, Ruml W (eds) Proceedings of the twenty-fourth international conference on automated planning and scheduling, ICAPS 2014, Portsmouth, New Hampshire, USA, June 21–26, 2014. http://www.aaai.org/ocs/index.php/ICAPS/ICAPS14/paper/view/7929

Ruszczyński A, Shapiro A (2009) Lectures on stochastic programming. SIAM, Philadelphia. https://doi.org/10.1137/1.9780898718751

Satia JK, Lave RE (1973) Markovian decision processes with uncertain transition probabilities. Oper Res 21(3):728–740

Serfozo RF (1979) An equivalence between continuous and discrete time Markov decision processes. Oper Res 27(3):616–620

Sigaud O, Buffet O (eds) (2010) Markov decision processes in artificial intelligence. Wiley-ISTE, London

Singh SP, Cohn D (1997) How to dynamically merge Markov decision processes. In: Jordan MI, Kearns MJ, Solla SA(eds) Advances in neural information processing systems 10, [NIPS Conference, Denver, Colorado, USA, 1997]. The MIT Press, pp 1057–1063

Singh SP, Jaakkola TS, Jordan MI (1994) Learning without state-estimation in partially observable Markovian decision processes. In: Cohen WW, Hirsh H (eds) Machine learning, proceedings of the eleventh international conference, Rutgers University, New Brunswick, NJ, USA, July 10–13, 1994, pp 284–292

Steimle LN, Kaufman DL, Denton BT (2018) Multi-model Markov decision processes. Technical report, Optimization-online

Vielma JP (2015) Mixed integer linear programming formulation techniques. SIAM Rev 57(1):3–57

Walraven E, Spaan MTJ (2015) Planning under uncertainty with weighted state scenarios. In: Meila M, Heskes T (eds) Proceedings of the thirty-first conference on uncertainty in artificial intelligence, UAI 2015, July 12–16, 2015, Amsterdam, The Netherlands, pp 912–921. AUAI Press

White CC, Eldeib HK (1994) Markov decision processes with imprecise transition probabilities. Oper Res 42(4):739–749

White CC, White DJ (1989) Markov decision processes. Eur J Oper Res 39(6):1–16

Wierman A, Andrew LL, Tang A (2012) Power-aware speed scaling in processor sharing systems: optimality and robustness. Perform Eval 69(12):601–622

Wiesemann W, Kuhn D, Rustem B (2013) Robust Markov decision processes. Math Oper Res 38(1):153–183