LOCAL APPROXIMATION OF DISCOUNTED MARKOV DECISION PROBLEMS BY MATHEMATICAL PROGRAMMING METHODS

STEFAN HEINZ, JÖRG RAMBAU, AND ANDREAS TUCHSCHERER

ABSTRACT. We develop a method to approximate the value vector of discounted Markov decision problems (MDP) with guaranteed error bounds. It is based on the linear programming characterization of the optimal expected cost. The new idea is to use column generation to dynamically generate only such states that are most relevant for the bounds by incorporating the reduced cost information. The number of states that is sufficient in general and necessary in the worst case to prove such bounds is independent of the cardinality of the state space. Still, in many instances, the column generation algorithm can prove bounds using much fewer states. In this paper, we explain the foundations of the method. Moreover, the method is used to improve the well-known nearest-neighbor policy for the elevator control problem.

1. INTRODUCTION

For a number of Markov Decision Problems (MDP) coming from interesting dynamical optimization problems a classical computation of optimal policies is prevented by the *curses of dimensionality*.

Powell [Pow07] introduces the three curses of dimensionality that give rise to these intractable sizes. The first curse is as follows. The number of states in a Markov Decision Problem (details below) grows exponentially with the number of state parameters, where the base is the number of different values that a state parameter can take. A similar behavior appears often for the set of feasible actions at a state and the set of possible states the system can move due to using an action at some state. We will refer to these as the second and third curse of dimensionality, respectively.

In this paper we introduce a technique to overcome the first curse in some interesting cases. More specifically, we introduce a column-generation algorithm that computes in selected states lower and upper bounds for the expected cost for a prescribed policy, for an optimal policy, or for an action (assumed that in other states we decide optimally). Selected states might be such states in which we suspect that a given widely used policy performs badly. Or states in which we suspect that one policy acts better than another, in expectation.

Our algorithm employs the linear programming characterization of optimal policies in discounted MDPs. It starts with a small part of the state space and adds states driven by the reduced-cost criterion from linear programming. The reduced cost of state variables is the additional information that comes for free in the linear programming setting. Our tool exploits this extra-information.

1.1. **Related Work.**

Various propositions exist how the curses of dimensionality can be by-passed via approximations. We know of no method that can provide us with proven bounds on the gap between a computed policy and an optimal policy when the state space is too large to be handled in total. Moreover, automatically computed policies often lack an understandable structure, and one is interested how good a policy is that can be formulated as a logical decision rule. A prominent example is the common use of security stock policies in inventory control, even in cases where such policies are known to be suboptimal.

Key words and phrases. Markov Decision Problem, Linear Programming, Column Generation, Performance Guarantees.

In order to deal with the three curses of dimensionality arising in discounted and other MDPs, several approaches have been studied in the literature. A broad field of methods targeting large-scale MDPs (and generalizations) where exact methods become infeasible is *approximate dynamic programming* [Pow07, SB98, BT96], which evolved in the computer science community under the name *reinforcement learning*. Contrary to the classical computational methods described above, an advantage of many techniques in this area is that an explicit model of the environment, i. e., a precise specification of the MDP, is often not required. Instead, a simulator of the system can be employed. Similar to simulation, there is virtually no limit on the complexity of the state and transition structure. We refer to the books [Pow07, SB98, BT96] for details concerning approximate dynamic programming.

The main disadvantage we see in approximate dynamic programming is that very few methods provide performance guarantees, and those that do, e. g., [dFV03], only give worst-case and thus typically weak bounds. Therefore, the need for tools providing performance guarantees for policies is still there. In fact, policies stemming from approximate dynamic pogramming could very well be analyzed by our method to find bounds on their expected performance.

The approach described in the literature that yields results closest to ours is a *sparse sampling algorithm* proposed by Kearns et al. [KMN99]. The authors also give theoretical bounds on the necessary size of a subset of the state space that is needed by their approach in order to obtain an ε-approximation, see Remark 3.20 on Page 16. However, for the applications we aim at, their bounds are substantially weaker than ours.

Other approaches to locally explore the state space have been proposed by Dean et al. [DKKN93] and Barto et al. [BBS95]. The former employs policy iteration with a concept of locality similar to ours. This way, their method comes closest to our approach concerning the algorithm used. However, the method does not provide any approximation guarantees.

1.2. **Our contribution.**

In this paper, based on results from [Tuc10], we suggest a measurement tool that approximates the *expected total discounted cost* of a given policy starting in a given state, usually called *initial state*, relative to an unknown optimal policy (or another given policy) up to a prescribed error. Because this tool needs only a small part (depending on the discount factor) of the state space for its conclusions, it works in many cases where the size of the state space renders classical methods to compute the cost of an optimal policy infeasible. Since this cost criterion is the only one covered in this work, we call the expected total discounted cost of a policy simply the *cost* of a policy from now on.

Our tool can in many instances

- find out whether in a given state a policy produces a cost of no more than $(1+\varepsilon)$ times the cost of an unknown optimal policy;
- find out whether in a given state a policy produces a cost of at least $(1+\varepsilon)$ times the cost of an unknown optimal policy;
- prove that in a given state, one policy has a smaller cost than another one;
- prove that a policy can not be optimal;
- prove that a single action can not be optimal in a given state;
- use that knowledge to improve given policies in special situations, i.e., states with certain properties.

The results that can be obtained for concrete policies depend on the parameters and on the specific instances. By applying our tool to the elevator control problem, we find out that the nearest-neighbor policy NN is better than many other policies for elevator instances of *online dial a ride problems* with the goal to minimize average waiting times, but not optimal. This adds theoretical learnings to the simulation knowledge from [GHKR99]. Non-optimality is already implied by the property that NN never moves the elevator in an empty system. By evaluating this single action in the empty system state with our tool, we can guarantee that all policies that do not move in the empty system are suboptimal. We present a new policy NNPARK-f that positions the elevator optimally when no request is in the system. In a similar fashion, we improve NN to a better policy NNMAXPARK-f when the goal is to minimize the maximal waiting time among all requests. And for this objective, we can show with our tool that NN is one of the weakest policies.

All results reflect well our observations in simulations. This is no coincidence because we give bounds on expected costs, and, because of the law of large numbers, the same bounds should emerge in simulations with high probability.

1.3. **Outline of the Paper.** The paper is organized as follows: In Section 2 we phrase our mathematical goal more formally. Section 3 introduces the theoretical foundations of our method via induced MDPs. Our method itself is described in detail in Section 4. In Section 5, we present how the method can be applied to a benchmark application, an elementary elevator control problem. For this application, we were, e.g., able to design taylor-made improvements for the nearest-neighbor policy on the basis of the analysis with our tool. Simulation studies on larger systems have meanwhile shown that the key-learnings of our short-term dominated analysis are also valid for long-term experiments.

2. FORMAL PROBLEM STATEMENT

We briefly review Markov Decision Problems (MDP) in order to settle on the notation. A Markov decision process describes a discrete-time stochastic system of the following type. At each point in time the system is situated in some specific state. Each state defines a non-empty set of actions that represents the different possibilities to control or affect the process. Applying a particular action moves the system into another state according to a given probability distribution. Each state transition comes along with an immediately incurred cost.

More formally: A *Markov decision process* is a tuple $M = (\mathbb{S}, \mathbb{A}, p, c)$, where the components are defined as follows:

- $\mathbb{S}$ is a finite set of *states*.
- $\mathbb{A}$ is a mapping specifying for each state $i \in \mathbb{S}$ a non-empty and finite set $\mathbb{A}(i)$ of possible *actions* at state i.
- For all states $i, j \in \mathbb{S}$, the mapping $p_{ij}\colon \mathbb{A}(i) \to [0,1]$ gives the *transition probability* $p_{ij}(a)$ that the system moves from state i to state j when using action $a \in \mathbb{A}(i)$. For each state $i \in \mathbb{S}$ and each action $a \in \mathbb{A}(i)$, we have $\sum_{j\in\mathbb{S}} p_{ij}(a) = 1$.
- For all $i \in \mathbb{S}$, the mapping $c_i\colon \mathbb{A}(i) \times \mathbb{S} \to \mathbb{R}_+$ specifies the *stage cost* $c_i(a, j)$ when action $a \in \mathbb{A}(i)$ is chosen and the system moves to state $j \in \mathbb{S}$. The *expected stage cost* of using action $a \in \mathbb{A}(i)$ at state $i \in \mathbb{S}$ is denoted by $c_i(a) := \sum_{j\in\mathbb{S}} p_{ij}(a)c_i(a, j)$.

A *policy* for M is a mapping $\pi : \mathbb{S} \to \mathbb{A}(\mathbb{S})$. It is *feasible* if $\pi(i) \in \mathbb{A}(i)$. Let P_M denote the set of all feasible policies for M.

Note that the state space $\mathbb{S}$ is assumed to be finite. In contrast to the classical computational methods for the objective criterion of minimizing the total expected discounted cost, however, the approximation method proposed in this paper can cope with an infinite number of states. We will consider one Markov decision process with infinite state space in Section 5.

For each $t \in \mathbb{N}$, let the random variables X_t and Y_t denote the current state and the action used at stage t. Moreover, for all states $i, j \in \mathbb{S}$ and each action $a \in \mathbb{A}(j)$, let $\mathbb{P}_{i\pi}[X_t = j, Y_t = a]$ denote the probability that at stage t the state is j and the action is a, given that policy π is used and the initial state is i. The expectation operator w. r. t. this probability measure is denoted by $\mathbb{E}_{i\pi}$.

Let $M = (\mathbb{S}, \mathbb{A}, p, c)$ be a Markov decision process and let $\alpha \in [0,1)$. The *total expected α-discounted cost* of a policy π for M for an initial state $i \in \mathbb{S}$ is defined by

$$\begin{aligned} v_i^\alpha(\pi) &:= \sum_{t=0}^{\infty} \mathbb{E}_{i\pi}[\alpha^t \cdot c_{X_t}(Y_t)] \qquad (1) \\ &= \sum_{t=0}^{\infty} \alpha^t \sum_{j\in\mathbb{S}} \sum_{a\in\mathbb{A}(j)} \mathbb{P}_{i\pi}[X_t = j, Y_t = a] \cdot c_j(a). \end{aligned}$$

Let $V^\alpha : P_M \to \mathbb{R}^{\mathbb{S}}$ be the value vector function defined for each policy $\pi \in P_M$ by the value vector $v^\alpha(\pi)$ with elements $v_i^\alpha(\pi)$ for each $i \in \mathbb{S}$ as given above. The combination (M, V^α) of M and the value vector function V^α is called an *α-discounted cost Markov Decision Problem*, or short *discounted MDP*, and is denoted for short as (M, α). We denote with v^α the *optimal value vector* which is $v_i^\alpha = \min_{\pi\in P_M} v_i^\alpha(\pi)$ for all $i \in \mathbb{S}$. A policy π^* is *optimal* for (M, α) if $v_i^\alpha(\pi^*) = v^\alpha$

Originally the goal is to find an optimal policy. Our goal is the following: Given an α-discounted-cost MDP, a policy, and an $\varepsilon > 0$, find ε-exact performance guarantees for single start states, maybe relative to an unknown optimal policy or relative to some other policy. That is, more formally:

Problem 2.1. Given an α-discounted-cost MDP, a policy π, a state i_0 with $v_{i_0}^\alpha > 0$, and an $\varepsilon > 0$, find in state i_0 a lower bound $\underline{v}_{i_0}$ for the optimal cost and an upper bound $\overline{v}_{i_0}(\pi)$ for the cost of π such that

$$\frac{\overline{v}_{i_0}(\pi) - \underline{v}_{i_0}}{\underline{v}_{i_0}} \leq \varepsilon. \qquad \text{(Relative Performance Guarantee)}$$

Alternatively, find in state i_0 a lower bound $\underline{v}_{i_0}(\pi)$ for the cost of π and an upper bound $\overline{v}_{i_0}$ for the optimal cost such that

$$\underline{v}_{i_0}(\pi) > \overline{v}_{i_0}. \qquad \text{(Non-Optimality Certificate)}$$

In this paper, we present an algorithm that can provide such bounds and related data without necessarily touching all states. States used for the computation are selected dynamically, dependent on the individual data of the instance. The algorithm detects automatically when the desired guarantee can be given and stops with a proven result.

3. Induced MDPs and Bounds

In this section, we derive from a given MDP new MDPs whose value functions

- can be computed easier,
- yield bounds for the value function of the original MDP.

Let $c_{\max} := \max_{i\in\mathbb{S}, a\in\mathbb{A}(i)} c_i(a)$ be the maximum stage cost. Obviously, we have:

$$\left| \sum_{t=0}^{\infty} \alpha^t \cdot c_{X_t}(Y_t) \right| \leq \sum_{t=0}^{\infty} \alpha^t \cdot c_{\max} = \frac{c_{\max}}{1-\alpha}.$$

For discounted MDPs we have the nice property that there always exists an optimal deterministic policy. Recall that this implies optimality for each possible initial state. The following result can be found in the book of Bertsekas [Ber01].

Theorem 3.1 (See, e.g., [Ber01, Volume 1, Chapter 7.3])**.** *Let (M, α) be an α-discounted MDP with $\alpha \in [0,1)$. Then, we have the following:*

(1) Let π be a deterministic policy for M. Then the value vector $v^\alpha(\pi)$ equals the unique solution v of the system of linear equations:

$$v_i = c_i(\pi(i)) + \alpha \sum_{j\in\mathbb{S}} p_{ij}(\pi(i)) v_j, \quad i \in \mathbb{S}. \qquad (2)$$

(2) *The optimal value vector v^α equals the unique solution v of the system of equations:*

$$v_i = \min_{a \in \mathbb{A}(i)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j \right\}, \quad i \in \mathbb{S}. \tag{3}$$

(3) *There exists an optimal deterministic policy for M, and a deterministic policy π is optimal if and only if:*

$$\pi(i) \in \operatorname*{argmin}_{a \in \mathbb{A}(i)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j^\alpha(\pi) \right\}, \quad i \in \mathbb{S}. \tag{4}$$

The practical impact of Theorem 3.1 can be summarized as follows. The value vector of a deterministic policy can be computed by solving a system of linear equations. Moreover, the optimal value vector equals the unique solution of a system of equations incorporating a minimum term. One typically refers to the system of Equations (3) as the *optimality equations* or *Bellman equations*. Once the optimal value vector v^α is at hand, an optimal deterministic policy can easily be determined by computing $c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j^\alpha$ for each state $i \in \mathbb{S}$ and each action $a \in \mathbb{A}(i)$. Basically, all methods for computing an optimal deterministic policy first provide the optimal value vector, and then use Formula (4) to obtain the policy itself. Thus, the remaining task is to determine v^α.

Because of the reasons mentioned above, we will particularly deal with deterministic policies in the sequel. Moreover, the following definition of optimal actions will be used.

Definition 3.2 (Optimal actions)**.** Let (M, α) be an discounted MDP with $\alpha \in [0, 1)$. A possible action $a \in \mathbb{A}(i)$ at a state $i \in \mathbb{S}$ is called *optimal* if there exists an optimal deterministic policy π for M such that $\pi(i) = a$.

The classical methods for computing the optimal value vector v^α of a discounted MDP include *value iteration*, *policy iteration*, and *linear programming*. For details and possible variants and extensions of the methods, see [Put05, chapter 6], [FS02, chapter 2.3], or [Ber01, volume 2, chapter 1.3].

The central theorem concerning the linear programming method for computing the optimal value vector of a discounted MDP reads as follows.

Theorem 3.3 (See, e.g., [Ber01, Volume 2, Section 1.3.4])**.** *The optimal value vector $v^\alpha \in \mathbb{R}^{\mathbb{S}}$ of a discounted MDP (M, α) equals the unique optimal solution v of the following linear program:*

$$\begin{aligned} \max \quad & \sum_{i \in \mathbb{S}} v_i && && (\mathrm{P}^\Sigma) \\ \text{subject to} \quad & v_i - \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j \le c_i(a) && \forall i \in \mathbb{S}\ \forall a \in \mathbb{A}(i) \\ & v_i \in \mathbb{R} && \forall i \in \mathbb{S}. \end{aligned}$$

Therefore, one can obtain the optimal value vector by solving the linear program (P^Σ). This linear programming formulation was first proposed by d'Epenoux [d'E63] and has been the starting point for several approaches, e. g., see [SS85, dFV03, dFV04].

In the sequel we will deal with many linear programs similar to (P^Σ). To emphasize their specific distinctions, we will use a matrix-vector notation. Let (M, α) be an discounted MDP. Contrary to the usual Cartesian product, we define $S \times \mathbb{A}$ for any subset of states $S \subseteq \mathbb{S}$ as:

$$S \times \mathbb{A} := \{(i, a) \mid i \in S, a \in \mathbb{A}(i)\}.$$

That is, $S \times \mathbb{A}$ equals the set of all pairs of states in S and possible actions. Next we define the matrix $Q \in \mathbb{R}^{(\mathbb{S} \times \mathbb{A}) \times \mathbb{S}}$ for each $(i, a) \in \mathbb{S} \times \mathbb{A}$ and each state $j \in \mathbb{S}$ by:

$$Q_{(i,a),j} = \begin{cases} 1 - \alpha p_{ij}(a), & \text{if } i = j, \\ -\alpha p_{ij}(a), & \text{if } i \neq j. \end{cases}$$

Moreover, we make sloppy use of the symbol c and also denote by $c \in \mathbb{R}^{\mathbb{S}\times\mathbb{A}}$ the vector of the expected stage costs, i. e., the components of c are given by:

$$c_{ia} = c_i(a)$$

for each $(i,a) \in \mathbb{S}\times\mathbb{A}$. Now the linear program (P$^\Sigma$) can be written as:

$$\begin{aligned} \max\ & \mathbb{1}^t v \\ \text{subject to}\ & Qv \le c \\ & v \in \mathbb{R}^{\mathbb{S}}, \end{aligned} \qquad (\mathrm{P}^\Sigma)$$

where $\mathbb{1}^t = (1,1,\dots,1)$ denotes the all-ones vector.

The approximation algorithm to be proposed is motivated by the fact that for the huge state spaces arising in MDPs modeling practical problems, it is currently impossible to solve the associated linear program (P$^\Sigma$) in reasonable time. Our idea is to evaluate the value vector at one particular state $i_0 \in \mathbb{S}$ alone. Since we are only interested in $v^\alpha_{i_0}$, we can restrict the objective function of (P$^\Sigma$) by maximizing the value v_{i_0} only:

$$\begin{aligned} \max\ & v_{i_0} \\ \text{subject to}\ & Qv \le c \\ & v \in \mathbb{R}^{\mathbb{S}} \end{aligned} \qquad (\mathrm{P}^{i_0})$$

In contrast to (P$^\Sigma$), there does not exist a unique solution for the linear program (P^{i_0}) in general for the following reasons. On the one hand, there may be states in $\mathbb{S}$ that cannot be reached from i_0. On the other hand, there are typically some actions that are not optimal. Such a state $j \in \mathbb{S}$, that is either not reached at all or only reached via non-optimal actions, is not required to have a maximized value v_j in order to maximize v_{i_0}, i. e., the objective function of (P^{i_0}). The value v_j may even be negative in an optimal solution.

Similar to the original linear programming formulation, solving the linear program (P^{i_0}) is still infeasible considering the huge state spaces for practical applications. In order to obtain a linear program that is tractable independently of the size of the state space $\mathbb{S}$, we reduce the set of variables and constraints in the linear program (P^{i_0}) by taking into account only a restricted state space. Given a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, consider the submatrix $Q^S \in \mathbb{R}^{(S\times\mathbb{A})\times S}$ of the constraint matrix Q consisting of all rows (i,a) with $i \in S$ and all columns j with $j \in S$. Moreover, let $c^S \in \mathbb{R}^{S\times\mathbb{A}}$ be the subvector of vector c consisting of all the components with indices (i,a) satisfying $i \in S$. Now let us look at the following linear program:

$$\begin{aligned} \max\ & v_{i_0} \\ \text{subject to}\ & Q^S v \le c^S \\ & v \in \mathbb{R}^{S}. \end{aligned} \qquad (\mathrm{L}^{i_0}_S)$$

Sometimes we will also be interested in an optimal solution of this reduced linear program where the objective function is $\sum_{j\in S} v_j$:

$$\begin{aligned} \max\ & \mathbb{1}^t v \\ \text{subject to}\ & Q^S v \le c^S \\ & v \in \mathbb{R}^{S}, \end{aligned} \qquad (\mathrm{L}^{\Sigma}_S)$$

where again $\mathbb{1}^t = (1,1,\dots,1)$ denotes the all-ones vector.

Any feasible solution $\underline{v} \in \mathbb{R}^S$ of the linear program (L$^\Sigma_S$) and (L$^{i_0}_S$) can be extended to a feasible solution $\underline{v}^{\text{ext}} \in \mathbb{R}^{\mathbb{S}}$ of the linear program (P$^\Sigma$) and (P^{i_0}) with the same objective value, respectively, where

$$\underline{v}_i^{\text{ext}} = \begin{cases} \underline{v}_i, & \text{if } i \in S, \\ 0, & \text{if } i \in \mathbb{S} \setminus S. \end{cases} \tag{5}$$

The optimal value vector v^α is the componentwise largest vector satisfying the constraints of (P^Σ) and (P^{i_0}). Thus, each feasible solution of the linear programs (L_S^Σ) and ($\mathrm{L}_S^{i_0}$) provides a lower bound on the optimal value vector v^α at all states in S.

Lemma 3.4. *Given a discounted MDP (M, α), a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let $\underline{v}$ be any feasible solution of the linear programs (L_S^Σ) and ($\mathrm{L}_S^{i_0}$), respectively. Then, for each state $i \in S$, the component v_i^α of the optimal value vector v^α is at least $\underline{v}_i$, i. e.,*

$$\underline{v}_i \leq v_i^\alpha \quad \textit{for each } i \in S.$$

Particularly, the optimal value of the linear program ($\mathrm{L}_S^{i_0}$) is a lower bound on $v_{i_0}^\alpha$.

Although lower bounds on the optimal value vector are obtained for all states in the subset of states S, the approximation method proposed in this paper mainly aims at computing bounds on the component $v_{i_0}^\alpha$. The lower bounds on $v_{i_0}^\alpha$ are obtained as the optimal values of the linear programs ($\mathrm{L}_S^{i_0}$) for some $S \subseteq \mathbb{S}$ with $i_0 \in S$. These values can be obtained from the optimal solution of (L_S^Σ), too.

In the following we show that each subset $S \subseteq \mathbb{S}$ defines again an MDP. The idea is to add one additional state that models all transitions to states that are not included in S.

Definition 3.5 (Lower-bound induced MDP)**.** Let $M = (\mathbb{S}, \mathbb{A}, p, c)$ be an MDP and let $S \subseteq \mathbb{S}$ be any subset of states. Then, the *(lower-bound) S-induced MDP* $M(S) = (\mathbb{S}', \mathbb{A}', p', c')$ is defined as follows:

- If for all states $i \in S$ and all actions $a \in \mathbb{A}(i)$ we have $\sum_{j \in S} p_{ij}(a) = 1$, then the state space of $M(S)$ equals $\mathbb{S}' = S$. The mappings $\mathbb{A}'$, p', and c' are the corresponding restrictions of $\mathbb{A}$, p, and c to the possibly reduced state space $\mathbb{S}'$.
- Otherwise, the state space of the induced MDP equals $\mathbb{S}' = S \cup \{i_{\text{end}}\}$ with the following properties of state i_{end}. For each state $i \in S$ and each action $a \in \mathbb{A}(i)$ with $\sum_{j \in S} p_{ij}(a) < 1$, we set:

  $$p'_{ii_{\text{end}}}(a) := \sum_{j \in \mathbb{S} \setminus S} p_{ij}(a) = 1 - \sum_{j \in S} p_{ij}(a)$$

  and

  $$c'_i(a, i_{\text{end}}) := \frac{1}{p'_{ii_{\text{end}}}(a)} \sum_{j \in \mathbb{S} \setminus S} p_{ij}(a) c_i(a, j).$$

  That is, $c'_i(a, i_{\text{end}})$ equals the expected stage cost for using action a at state i, given that the successor state is not contained in S.

  Furthermore, there is only one feasible action at the state i_{end}, i. e., we have $\mathbb{A}'(i_{\text{end}}) = \{a_{\text{end}}\}$. Using action a_{end} the system always stays in state i_{end}, i. e., $p'_{i_{\text{end}} i_{\text{end}}}(a_{\text{end}}) = 1$, with a stage cost of $c'_{i_{\text{end}}}(a, i_{\text{end}}) = 0$. Except for the special cases described above, $\mathbb{A}'$, p', and c' are again the restrictions of $\mathbb{A}$, p, and c w. r. t. $\mathbb{S}'$.

In the literature a state with the properties of i_{end} is often called *absorbing terminal state*. A picture illustrating the Markov decision process of the induced MDP $M(S)$ for some proper subset of states $S \subset \mathbb{S}$ is given in Figure 1. Induced MDPs have the following properties.

Theorem 3.6. *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, we have for the lower-bound S-induced MDP $M(S) = (\mathbb{S}', \mathbb{A}', p', c')$:*

(1) $M(S) = M$ if and only if $S = \mathbb{S}$.

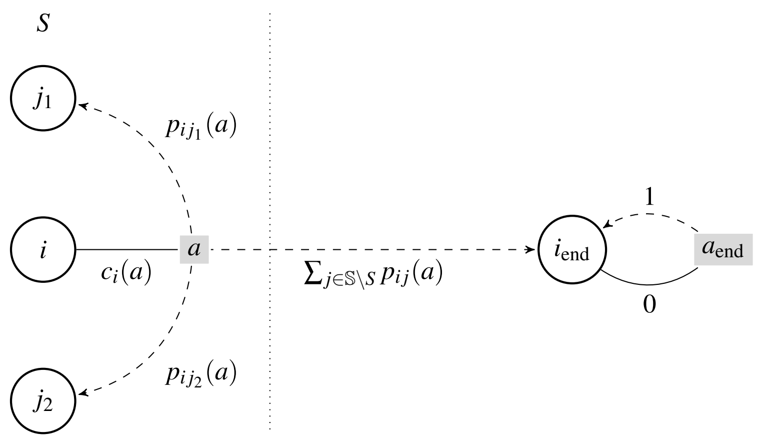


FIGURE 1. Illustration of the Markov decision process of the induced MDP $M(S)$ for some $S \subset \mathbb{S}$. Transitions within the reduced state space S are as in the original MDP M; transitions from S to $\mathbb{S}\setminus S$ in M are modeled via aggregated transitions to the absorbing terminal state i_{end}. The expected stage costs do not change, cf. Theorem 3.6.

(2) *The expected stage cost at state $i \in S$ for using action $a \in \mathbb{A}'(i) = \mathbb{A}(i)$ is the same for both MDPs M and $M(S)$, i. e., $c'_i(a) = c_i(a)$.*
(3) *The optimal value vector v of $M(S)$ for an $\alpha \in [0,1)$ is given by the unique optimal solution of the linear program (L_S^{Σ}) and $v_{i_{\text{end}}} = 0$.*

Proof. The first property is trivial. To prove the second one, let $i \in S$ and $a \in \mathbb{A}(i)$. If all possible successor states reached by using action a at state i are contained in S, i. e., $\sum_{j\in S} p_{ij}(a) = 1$, the statement is clear. Assume $\sum_{j\in S} p_{ij}(a) < 1$. Since $c'_i(a,j) = c_i(a,j)$ for each $j \in S$, we obtain by the definition of $c'_i(a,i_{\text{end}})$:

$$\begin{aligned} c_i(a) &= \sum_{j\in\mathbb{S}} p_{ij}(a)c_i(a,j) \\ &= \sum_{j\in S} p_{ij}(a)c_i(a,j) + \sum_{j\in\mathbb{S}\setminus S} p_{ij}(a)c_i(a,j) \\ &= \sum_{j\in S} p_{ij}(a)c'_i(a,j) + p'_{ii_{\text{end}}}(a)c'_i(a,i_{\text{end}}) \\ &= c'_i(a). \end{aligned}$$

Now the third property follows from the general linear programming result (see Theorem 3.3) and the observation that the optimal value vector of the MDP $M(S)$ is always zero at state i_{end}. □

Induced MDPs will play an important role in various parts of this paper.

Similarly to the reduced linear program $(\mathrm{L}_S^{i_0})$ providing a lower bound for the value $v_{i_0}^{\alpha}$ of an MDP, we propose the following approach to establish a linear program to obtain an upper bound on $v_{i_0}^{\alpha}$. Since there is only a finite number of states and actions, the maximum expected stage cost is attained by $c_{\max} := \max_{i\in\mathbb{S},a\in\mathbb{A}(i)} c_i(a)$. This implies an upper bound on the value vector of any policy: from (1) we easily get $v_i^{\alpha}(\pi) \le c_{\max}/(1-\alpha)$, for each policy π and each state $i \in \mathbb{S}$.

Now given a particular state $i_0 \in \mathbb{S}$ and a subset of states $S \subseteq \mathbb{S}$ such that $i_0 \in S$, we compute an upper bound on the value $v_{i_0}^{\alpha}$ as follows. Instead of just dropping the optimal value vector outside S, i. e., setting it to zero, we can set the corresponding variables to the general upper bound $v^\alpha_{\max} := c_{\max}/(1-\alpha)$. Therefore, the reduced linear program providing an upper bound reads:

$$\begin{aligned} \max\ & v_{i_0} && (\mathrm{U}^{i_0}_S)\\ \text{subject to}\ & Q^S v \le c^S + r^S \\ & v \in \mathbb{R}^S, \end{aligned}$$

where the vector $r^S \in \mathbb{R}^{S\times\mathbb{A}}$ is defined by:

$$r^S_{ia} = \alpha \cdot v^\alpha_{\max} \sum_{j\in\mathbb{S}\setminus S} p_{ij}(a), \tag{6}$$

for each $(i,a) \in S\times\mathbb{A}$. Obviously this linear problem is feasible and bounded.

Similar to the reduced linear program $(\mathrm{L}^{i_0}_S)$ for computing the lower bound, also $(\mathrm{U}^{i_0}_S)$ provides the optimal value vector at state i_0 for some adapted MDP. Here, the MDP is a slight modification of the lower-bound induced MDP introduced in Definition 3.5: the stage cost for the only transition at state i_{end} now equals the maximum expected stage cost $c_{\max}$ instead of the minimum stage cost 0.

Definition 3.7 (Upper-bound induced MDP)**.** Let $M = (\mathbb{S},\mathbb{A},p,c)$ be an MDP and let $S \subseteq \mathbb{S}$ be any subset of states. Then, the *upper-bound S-induced MDP $M'(S)$* is defined as the modified lower-bound S-induced MDP, where the stage cost at state i_{end} for using action a_{end} equals:

$$c'_{i_{\text{end}}}(a_{\text{end}}, i_{\text{end}}) = c_{\max} := \max_{i\in\mathbb{S}, a\in\mathbb{A}(i)} c_i(a).$$

Notice that the optimal value vector v^α restricted to the state subset S gives a feasible solution for the linear program $(\mathrm{U}^{i_0}_S)$. Therefore, the optimal value of $(\mathrm{U}^{i_0}_S)$ is indeed an upper bound on $v^\alpha_{i_0}$.

Lemma 3.8. *Given a discounted MDP (M,α), a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, the optimal value of the linear program $(\mathrm{U}^{i_0}_S)$ is an upper bound on $v^\alpha_{i_0}$.*

Remark 3.9. Similar to the lower bound case, one can also show the following. Given a discounted MDP (M,α), a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let $\overline{v}$ be the unique optimal solution of the linear program $(\mathrm{U}^{i_0}_S)$ with objective function $\max \sum_{j\in S} v_j$. Then, we have for the optimal value vector v of the upper-bound S-induced MDP $M'(S)$:

$$v = \begin{cases} \overline{v}_i, & \text{if } i \in S,\\ v^\alpha_{\max}, & \text{if } i = i_{\text{end}}. \end{cases}$$

Particularly, the component $\overline{v}_{i_0}$ equals the optimal value of $(\mathrm{U}^{i_0}_S)$.

Furthermore, the solution $\overline{v}$ provides an upper bound on each component v^α_i of the optimal value vector of the original MDP M for $i \in S$, i. e., we have $v^\alpha_i \le \overline{v}_i$.

The next results shows that by solving the linear program $(\mathrm{U}^{i_0}_S)$ one can also construct a policy for the original MDP whose value vector at state i_0 is bounded from above by the optimal value of $(\mathrm{U}^{i_0}_S)$. The policy is obtained by extending an optimal policy for the upper-bound S-induced MDP $M'(S)$ arbitrarily w. r. t. the states in $\mathbb{S}\setminus S$.

Theorem 3.10. *Consider a discounted MDP (M,α), a state $i_0 \in \mathbb{S}$, a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, and an optimal solution $\overline{v}_{i_0}$ for the linear program $(\mathrm{U}^{i_0}_S)$. For each state $i \in S$, let $a_i \in \mathbb{A}(i)$ be any action that satisfies the corresponding inequality in $(\mathrm{U}^{i_0}_S)$ with equality. Then, any policy π for M with $\pi(i) = a_i$ for each $i \in S$ satisfies:*

$$\underline{v}_{i_0} \le v^\alpha_{i_0} \le v^\alpha_{i_0}(\pi) \le \overline{v}_{i_0},$$

where $\underline{v}_{i_0}$ is the optimal value of $(\mathrm{L}^{i_0}_S)$.

Proof. The first inequality holds true due to Lemma 3.4 and the second one is clear anyway.

Since the value vector of policy π equals the solution of the system of linear equations (2) by Theorem 3.1, it can be shown that the value $v^{\alpha}_{i_0}(\pi)$ can also be computed as the optimal value of the following linear program:

$$\begin{aligned} \max\ & v_{i_0} \\ \text{subject to}\ & v_i - \alpha \sum_{j\in\mathbb{S}} p_{ij}(\pi(i)) v_j \le c_i(\pi(i)) && \forall i \in \mathbb{S} \\ & v_i \in \mathbb{R} && \forall i \in \mathbb{S}. \end{aligned}$$

Next this linear program is modified as follows. Firstly, constraints $v_i \le v^{\alpha}_{\max}$ for each $i \in \mathbb{S}\setminus S$ are added to the linear program. Since these constraints are redundant, this does not change the optimal value. Secondly, all original constraints for states in $\mathbb{S}\setminus S$ are removed. Thus, we obtain the following relaxation of the linear program above:

$$\begin{aligned} \max\ & v_{i_0} \\ \text{subject to}\ & v_i - \alpha \sum_{j\in\mathbb{S}} p_{ij}(\pi(i)) v_j \le c_i(\pi(i)) && \forall i \in S \\ & v_i \le v^{\alpha}_{\max} && \forall i \in \mathbb{S} \\ & v_i \in \mathbb{R} && \forall i \in \mathbb{S}. \end{aligned}$$

Note that this relaxation is equivalent to the linear program $(\mathrm{U}^{i_0}_S)$ restricted to the constraints defined by π, which itself has by definition of π the same objective value as $(\mathrm{U}^{i_0}_S)$, i. e., $\bar{v}_{i_0}$. Since we constructed a relaxation of the linear program for computing $v^{\alpha}_{i_0}(\pi)$, we obtain $v^{\alpha}_{i_0}(\pi) \le \bar{v}_{i_0}$. □

Furthermore, there is a second way to obtain an upper bound on the component $v^{\alpha}_{i_0}$ of the optimal value vector by using directly the unique optimal solution of the linear program (L^{Σ}_S) for computing the lower bound. The construction of this upper bound on $v^{\alpha}_{i_0}$ is as follows. For a given subset of states $S \subseteq \mathbb{S}$ and a particular state $i_0 \in S$, let π be an optimal policy for the S-induced MDP $M(S)$ as obtained from the optimal solution of the linear program (L^{Σ}_S). Let $Q^{S,\pi} \in \mathbb{R}^{S\times S}$ be the submatrix of Q^S consisting of all the rows (i,a) with $a = \pi(i)$, and let $c^{S,\pi}, r^{S,\pi} \in \mathbb{R}^S$ be corresponding subvectors of c^S and r^S, respectively, i. e.,

$$c^{S,\pi}_{i\pi(i)} = c_i(\pi(i)) \quad \text{and} \quad r^{S,\pi}_{i\pi(i)} = \alpha \cdot v^{\alpha}_{\max} \sum_{j\in\mathbb{S}\setminus S} p_{ij}(\pi(i)),$$

for each state $i \in S$. Consider the following system of linear equations:

$$Q^{S,\pi} v = c^{S,\pi} + r^{S,\pi}. \tag{7}$$

Note that the matrix $Q^{S,\pi}$ is strictly row diagonally dominant and therefore nonsingular. Thus, the system (7) has a unique solution $\bar{v}^{\pi} \in \mathbb{R}^S$. The next result shows that the value $\bar{v}^{\pi}_{i_0}$ is an upper bound on $v^{\alpha}_{i_0}$, too.

Theorem 3.11. *Given a discounted MDP (M,α), a state $i_0 \in \mathbb{S}$, a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, and an optimal policy π for the S-induced MDP $M(S)$, let $\bar{v}^{\pi}$ be the unique solution of system* (7)*, and let $\bar{v}_{i_0}$ be the optimal value of the linear program* $(\mathrm{U}^{i_0}_S)$*. Then,*

$$v^{\alpha}_{i_0} \le \bar{v}_{i_0} \le \bar{v}^{\pi}_{i_0}.$$

That is, $\bar{v}^{\pi}_{i_0}$ is an upper bound on the optimal value vector at state i_0, but a weaker one than $\bar{v}_{i_0}$. Moreover, the value $\bar{v}^{\pi}_{i_0}$ equals the optimal value of the following linear program:

$$\begin{aligned} \max\ & v_{i_0} && (\mathrm{U}^{i_0}_{S,\pi}) \\ \text{subject to}\ & Q^{S,\pi} v \le c^{S,\pi} + r^{S,\pi} \\ & v \in \mathbb{R}^S. \end{aligned}$$

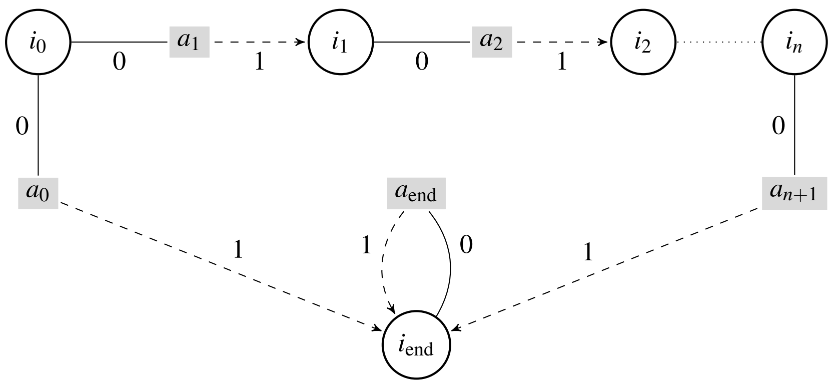


FIGURE 2. Markov decision process of an induced MDP $M(S)$ that yields different upper bounds $\overline{v}^{\pi_0}$ and $\overline{v}^{\pi_1}$ for the optimal policies π_0 and π_1 for $M(S)$ with $\pi_0(i_0) = a_0$ and $\pi_1(i_0) = a_1$.

Proof. The value $\overline{v}^{\pi}_{i_0}$ equals the optimal value of the linear program $(\mathrm{U}^{i_0}_{S,\pi})$. Since $(\mathrm{U}^{i_0}_{S,\pi})$ is a relaxation of the linear program $(\mathrm{U}^{i_0}_{S})$, we have $\overline{v}_{i_0} \leq \overline{v}^{\pi}_{i_0}$. □

Thus, by computing an optimal solution of the linear program $(\mathrm{L}^{\Sigma}_{S})$, which also yields an optimal policy π for the S-induced MDP $M(S)$, and by solving the corresponding system of linear equations (7) one can provide lower and upper bounds on $v^{\alpha}_{i_0}$.

Remark 3.12. The unique solution $\overline{v}^{\pi} \in \mathbb{R}^S$ of system (7) gives the value vector v of a policy π for the upper-bound S-induced MDP $M'(S)$:

$$v = \begin{cases} \overline{v}^{\pi}_i, & \text{if } i \in S, \\ v^{\alpha}_{\max}, & \text{if } i = i_{\text{end}}. \end{cases}$$

Recall that (7) is computed for policies that are optimal for $M(S)$. If such a policy π is optimal for $M'(S)$ as well, the two upper bounds on $v^{\alpha}_{i_0}$ compared in Theorem 3.11 coincide, i. e., we have $\overline{v}^{\pi}_{i_0} = \overline{v}_{i_0}$.

Under the assumptions of Theorem 3.11 one can show, similar to Theorem 3.10, that each policy π' for the original MDP M with $\pi'(i) = \pi(i)$ for each state $i \in S$ satisfies $v^{\alpha}_{i_0}(\pi') \leq \overline{v}^{\pi}_{i_0}$.

Obviously, several optimal policies may exist for an MDP in general. The following example shows that the upper bound $\overline{v}^{\pi}_{i_0}$ obtained by solving system (7) really depends on the chosen policy π. That is, different optimal policies may lead to different upper bounds.

Example 3.13. Let $S = \{i_0, i_1\}$ and consider the deterministic S-induced MDP $M(S)$ given by the Markov decision process shown in Figure 2 for $n = 1$. We assume that the maximum expected stage cost w. r. t. all states in $\mathbb{S}$ is positive, i. e., $c_{\max} = \max_{i \in \mathbb{S}, a \in \mathbb{A}(i)} c_i(a) > 0$. Since all stage costs for states in S equal 0, every policy for $M(S)$ is optimal. Note that there is only a choice to be made at state i_0. Consider the policies π_0 with $\pi_0(i_0) = a_0$ and π_1 with $\pi_1(i_0) = a_1$. Then, the solutions $\overline{v}^{\pi_0}$ and $\overline{v}^{\pi_1}$ of the corresponding systems (7) satisfy:

$$\begin{aligned} \overline{v}^{\pi_0}_{i_0} &= 0 + \alpha v^{\alpha}_{\max}, \\ \overline{v}^{\pi_0}_{i_1} &= 0 + \alpha v^{\alpha}_{\max}, \end{aligned}$$

and

$$\begin{aligned} \overline{v}^{\pi_1}_{i_0} - \alpha \overline{v}^{\pi_1}_{i_1} &= 0, \\ \overline{v}^{\pi_1}_{i_1} &= 0 + \alpha v^{\alpha}_{\max}, \end{aligned}$$

where again $v^\alpha_{\max} = c_{\max}/(1-\alpha)$ equals the general upper bound for each component of the value vector of any policy. Thus, we obtain:

$$\overline{v}^{\pi_0}_{i_0} = \alpha v^\alpha_{\max} \quad \text{and} \quad \overline{v}^{\pi_1}_{i_0} = \alpha^2 v^\alpha_{\max}.$$

Obviously, the policy π_1 provides a better upper bound than policy π_0.

The example can easily be extended such that the ratio between the two upper bounds becomes arbitrarily large. To this end, consider the S-induced MDP $M(S)$ shown in Figure 2 for an arbitrary integer $n \in \mathbb{N}$. There exists a sequence of states $i_1, \dots, i_n$ and actions $a_2, \dots, a_{n+1}$ with $\mathbb{A}(i_k) = \{a_{k+1}\}$ and $p_{i_k i_{k+1}(a_{k+1})} = 1$ for $k \in \{1, \dots, n-1\}$. Moreover, we have $p_{i_n i_{\text{end}}}(a_{n+1}) = 1$. Again all stage costs equal zero. Then, the optimal policy π_1 for $M(S)$ with $\pi_1(i_0) = a_1$ yields an upper bound of $\overline{v}^{\pi_1}_{i_0} = \alpha^{n+1} v^\alpha_{\max}$, while we still have $\overline{v}^{\pi_0}_{i_0} = \alpha v^\alpha_{\max}$ for the other optimal policy π_0 using action a_0 at state i_0. This results in a ratio of $\overline{v}^{\pi_0}_{i_0}/\overline{v}^{\pi_1}_{i_0} = 1/\alpha^n$, which goes to infinity for $n \to \infty$ since $\alpha < 1$.

Note that in the example the upper bound provided by policy π_1 equals the bound $\overline{v}_{i_0}$ obtained as the optimal value of the linear program ($\mathrm{U}^{i_0}_S$), i. e., $\overline{v}_{i_0} = \overline{v}^{\pi_1}_{i_0}$. In general, however, the upper bound $\overline{v}_{i_0}$ may be better than the bound $\overline{v}^{\pi}_{i_0}$ for each optimal policy π for $M(S)$. In other words, no optimal policy for $M(S)$ is optimal for $M'(S)$ as well.

Example 3.14. Consider again the example above for $n = 1$ except that we have a small stage cost for action a_1 of $c_{i_0}(a_1, i_1) = \varepsilon$, where $0 < \varepsilon < \alpha c_{\max}$. On the one hand, the policy π_1 is no longer optimal for $M(S)$, which leaves π_0 being the only optimal policy. On the other hand, the upper bound $\overline{v}_{i_0}$ equals:

$$\overline{v}_{i_0} = \min\left\{\alpha v^\alpha_{\max}, \varepsilon + \alpha^2 v^\alpha_{\max}\right\} = \varepsilon + \alpha^2 v^\alpha_{\max},$$

since $\varepsilon < \alpha c_{\max}$. Therefore, we obtain:

$$\overline{v}_{i_0} = \varepsilon + \alpha^2 v^\alpha_{\max} < \alpha v^\alpha_{\max} = \overline{v}^{\pi_0}_{i_0},$$

which shows that the upper bound $\overline{v}_{i_0}$ is predominant here.

Our approximation algorithm which we present in Section 4.1 is derived from the theory of this section. It generally employs the construction of upper bounds via solving the linear programs ($\mathrm{U}^{i_0}_S$) for subsets $S \subseteq \mathbb{S}$. However, it is also possible to incorporate the second type of upper bounds, especially since these bounds are more or less computed by the algorithm anyway.

Remark 3.15. The construction of lower and upper bounds for the component $v^\alpha_{i_0}$ of the optimal value vector can often be improved as follows. Let $S \subset \mathbb{S}$ be some restricted state space with $i_0 \in S$. Recall that for computing the bounds on $v^\alpha_{i_0}$ w. r. t. subset S our approach assumes for each component v^α_i of the optimal value vector with state $i \in \mathbb{S} \setminus S$ a lower and upper bound of 0 and $v^\alpha_{\max}$, respectively. Often, however, a better bound on individual components of v^α are known or can be determined.

It is easy to see that the upper bound constructions for $v^\alpha_{i_0}$ described in this section remain feasible if any available upper bounds $v^\alpha_{\max}(j) \geq v^\alpha_j$ for $j \in \mathbb{S}$ are used. That is, instead of the vector $r^S \in \mathbb{R}^S$ defined by Equation (6), we apply the vector $r^{\text{ub},S} \in \mathbb{R}^S$ where:

$$r^{\text{ub},S}_{ia} = \alpha \sum_{j \in \mathbb{S}\setminus S} p_{ij}(a) v^\alpha_{\max}(j),$$

for each $(i,a) \in S \times \mathbb{A}$. In doing so, both described ways to determine upper bounds on $v^\alpha_{i_0}$ can be improved.

Similarly, for given lower bounds $0 \leq v^\alpha_{\min}(j) \leq v^\alpha_j$ for $j \in \mathbb{S}$ on the components of the optimal value vector, a possibly improved lower bound on $v^\alpha_{i_0}$ can be obtained as the optimal value of the linear program:

$$\begin{aligned} \max \quad & v_{i_0} \\ \text{subject to} \quad & Q^S v \le c^S + r^{\mathrm{lb},S} \\ & v \in \mathbb{R}^S, \end{aligned}$$

where the vector $r^{\mathrm{lb},S} \in \mathbb{R}^S$ is defined by:

$$r^{\mathrm{lb},S}_{ia} = \alpha \sum_{j \in \mathbb{S} \setminus S} p_{ij}(a) v^{\alpha}_{\min}(j),$$

for each $(i,a) \in S \times \mathbb{A}$.

By incorporating such improved bounds in our algorithm the run-times can often be reduced significantly. We will make use of this technique in the computations in Section 5, e. g., for the considered elevator control MDPs. For this application, it is crucial to employ involved lower and upper bounds in order to obtain conclusive results at all.

In the following we present our structural approximation theorem which shows that an ε-approximation of one component of the optimal value vector can be obtained by taking into account only a small local part of the entire state space. We need the following definition.

Definition 3.16 (r-neighborhood)**.** For an MDP $(\mathbb{S},\mathbb{A},p,c)$, a particular state $i_0 \in \mathbb{S}$, and a number $r \in \mathbb{N}$, the *r–neighborhood $S(i_0,r)$ of i_0* is the subset of states that can be reached from i_0 within at most r transitions. That is, $S(i_0,0) := \{i_0\}$ and for $r > 0$ we define:

$$S(i_0,r) := S(i_0,r-1) \cup \left\{ j \in \mathbb{S} \mid \exists i \in S(i_0,r-1)\, \exists a \in \mathbb{A}(i) : p_{ij}(a) > 0 \right\}.$$

We will also call the set $S(i_0,r)$ *neighborhood of* i_0 *with radius* r.

Note that the stage costs accounted in the total expected discounted cost decrease geometrically. Thus, for a given approximation guarantee ε it is clear that the r-neighborhood $S(i_0,r)$ of i_0 for some radius $r = r(\varepsilon) \in \mathbb{N}$ will provide an ε-approximation for $v^{\alpha}_{i_0}$ via the associated linear programs. The following theorem provides a formula for the radius r required for a given approximation guarantee (we already documented a weaker version of this result in the preprint [HKP$^+$06]).

Theorem 3.17. *Let $M = (\mathbb{S},\mathbb{A},p,c)$ be an MDP, $\alpha \in [0,1)$ a discount factor, and $b,d \in \mathbb{N}$ such that:*

- *For each $i \in \mathbb{S}$, the number of possible actions $|\mathbb{A}(i)|$ at state i is bounded by $b \in \mathbb{N}$.*
- *For each $i \in \mathbb{S}$ and $a \in \mathbb{A}(i)$, the number of states $j \in \mathbb{S}$ with positive transition probabilities $p_{ij}(a)$ is bounded by $d \in \mathbb{N}$.*

Let $c_{\max} := \max_{i \in \mathbb{S}, a \in \mathbb{A}(i)} c_i(a)$ and $v^{\alpha}_{\max} := c_{\max}/(1-\alpha)$. Then, for each state $i_0 \in \mathbb{S}$ and for each $\varepsilon > 0$, the subset of states $S = S(i_0,r) \subseteq \mathbb{S}$ with

$$r = \max\left\{0, \left\lceil \log\left(\frac{\varepsilon}{v^{\alpha}_{\max}}\right) / \log \alpha \right\rceil - 1\right\}$$

satisfies the following properties:

(i) *$|S| \le \max\left\{(bd)^{r+1}, r+1\right\}$, in particular, the number of states in S does not depend on $|\mathbb{S}|$.*

(ii) *For state i_0, the unique optimal solution $\underline{v}$ of the linear program (L^{Σ}_S) (or any optimal solution $\underline{v}$ of $(\mathrm{L}^{i_0}_S)$, respectively) and the unique solution $\overline{v}^{\pi}$ of system (7) w. r. t. any optimal policy π for the S-induced MDP $M(S)$ satisfy:*

$$\overline{v}^{\pi}_{i_0} - \underline{v}_{i_0} \le \varepsilon.$$

In particular, $\underline{v}_{i_0}$ and $\overline{v}^{\pi}_{i_0}$ themselves are ε-close lower and upper bounds on the optimal value vector v^{α} at state i_0, i. e.,

$$\begin{aligned}0 &\le v^{\alpha}_{i_0} - \underline{v}_{i_0} \le \varepsilon,\\ 0 &\le \overline{v}^{\pi}_{i_0} - v^{\alpha}_{i_0} \le \varepsilon.\end{aligned}$$

Proof. Let $i_0 \in \mathbb{S}$ and $\varepsilon > 0$. Since the number of possible actions at each state and the number of successor states for any action are bounded by b and d, respectively, Property (i) follows directly from the construction of the set $S = S(i_0, r)$:

$$|S| \le \sum_{k=0}^{r} (bd)^k = \frac{(bd)^{r+1} - 1}{bd - 1} \le (bd)^{r+1},$$

if $bd \ge 2$. In the trivial case $bd = 1$ we obviously have $|S| = r + 1$.

The proof of Property (ii) is as follows. Consider the extension $\underline{v}^{\text{ext}} \in \mathbb{R}^{\mathbb{S}}$ of the solution $\underline{v}$ of the linear program (L^{Σ}_{S}) as defined in Equation (5):

$$\underline{v}^{\text{ext}}_i = \begin{cases} \underline{v}_i, & \text{if } i \in S,\\ 0, & \text{if } i \in \mathbb{S} \setminus S.\end{cases}$$

Moreover, let π be an optimal policy for $M(S)$ and construct an extension $\overline{v}^{\text{ext}} \in \mathbb{R}^{\mathbb{S}}$ of the solution $\overline{v}^{\pi}$ of system (7) w. r. t. policy π as follows:

$$\overline{v}^{\text{ext}}_i = \begin{cases} \overline{v}^{\pi}_i, & \text{if } i \in S,\\ v^{\alpha}_{\max}, & \text{if } i \in \mathbb{S} \setminus S.\end{cases}$$

Note that $\overline{v}^{\text{ext}}$ is in general not a feasible solution of the linear program (P^{Σ}).

By Theorem 3.6 the solution $\underline{v}$ of (L^{Σ}_{S}) equals the optimal value vector of the MDP $M(S)$. Since π is optimal for $M(S)$, Theorem 3.1 implies that the corresponding constraints in the linear program (L^{Σ}_{S}) are satisfied with equality by $\underline{v}$, i. e.,

$$\underline{v}_i = c_i(\pi(i)) + \alpha \sum_{j \in S} p_{ij}(\pi(i))\underline{v}_j \quad \forall i \in S,$$

which implies for the extension $\underline{v}^{\text{ext}}$:

$$\underline{v}^{\text{ext}}_i = c_i(\pi(i)) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i))\underline{v}^{\text{ext}}_j \quad \forall i \in S. \tag{8}$$

Note that in (8) we sum over the whole state space, which is feasible due to $\underline{v}^{\text{ext}}_j = 0$ for each $j \in \mathbb{S} \setminus S$.

On the other hand, since $\overline{v}^{\pi}$ satisfies the system of equations (7) we have the following relation for the extension $\overline{v}^{\text{ext}}$:

$$\overline{v}^{\text{ext}}_i = c_i(\pi(i)) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i))\overline{v}^{\text{ext}}_j \quad \forall i \in S. \tag{9}$$

From the Equations (8) and (9) we obtain:

$$\overline{v}^{\text{ext}}_i - \underline{v}^{\text{ext}}_i = \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i))(\overline{v}^{\text{ext}}_j - \underline{v}^{\text{ext}}_j) \quad \forall i \in S. \tag{10}$$

In the following, we show by reverse induction on $k = r, \dots, 0$ for each state $i \in S(i_0, k)$:

$$\overline{v}^{\text{ext}}_i - \underline{v}^{\text{ext}}_i \le \alpha^{r+1-k} v^{\alpha}_{\max}. \tag{11}$$

Note that all i to which (11) refers are contained in S because of $k \le r$.

For $k = r$ and for each state $i \in S(i_0, k)$, Inequality (11) follows from (10) due to $\overline{v}^{\text{ext}}_j \le v^{\alpha}_{\max}$ and $\underline{v}^{\text{ext}}_j \ge 0$ for each $j \in \mathbb{S}$:

$$\begin{aligned}\overline{v}^{\text{ext}}_i - \underline{v}^{\text{ext}}_i &\le \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i))\big(v^{\alpha}_{\max} - 0\big)\\ &= \alpha v^{\alpha}_{\max}.\end{aligned}$$

Here, the equality follows from the fact that $\sum_{j\in\mathbb{S}} p_{ij}(\pi(i)) = 1$ for each state $i \in \mathbb{S}$.

Now assume that Inequality (11) holds for each state $j \in S(i_0,k)$ with $0 < k \le r$. For each $i \in S(i_0,k-1)$, we again apply Equality (10):

$$\begin{aligned}\overline{v}_i^{\text{ext}} - \underline{v}_i^{\text{ext}} &= \alpha \sum_{j\in\mathbb{S}} p_{ij}(\pi(i))(\overline{v}_j^{\text{ext}} - \underline{v}_j^{\text{ext}}) \\ &= \alpha \sum_{j\in S(i_0,k)} p_{ij}(\pi(i))(\overline{v}_j^{\text{ext}} - \underline{v}_j^{\text{ext}}),\end{aligned}$$

where the second identity is due to the fact that each state $j \in \mathbb{S}$ with $p_{ij}(\pi(i)) > 0$ is contained in $S(i_0,k)$ since $i \in S(i_0,k-1)$. We can apply the induction hypothesis for each state $j \in S(i_0,k)$:

$$\begin{aligned}\overline{v}_i^{\text{ext}} - \underline{v}_i^{\text{ext}} &\le \alpha \sum_{j\in S(i_0,k)} p_{ij}(\pi(i))\alpha^{r+1-k} v_{\max}^{\alpha} \\ &= \alpha^{r+1-(k-1)} v_{\max}^{\alpha},\end{aligned}$$

which completes the inductive proof of (11).

For $i = i_0$ and $k = 0$, Inequality (11) implies:

$$\overline{v}_{i_0}^{\pi} - \underline{v}_{i_0} = \overline{v}_{i_0}^{\text{ext}} - \underline{v}_{i_0}^{\text{ext}} \le \alpha^{r+1} v_{\max}^{\alpha}.$$

Finally, we distinguish two cases to show Property (ii). If $\varepsilon \ge \alpha v_{\max}^{\alpha}$, we have $r = 0$, and thus $\overline{v}_{i_0}^{\pi} - \underline{v}_{i_0} \le \alpha v_{\max}^{\alpha} \le \varepsilon$. Otherwise, if $\varepsilon < \alpha v_{\max}^{\alpha}$, it follows that $\log(\varepsilon/v_{\max}^{\alpha}) < \log\alpha < 0$ and $r = \lceil \log(\varepsilon/v_{\max}^{\alpha})/\log\alpha \rceil - 1$ which implies:

$$\begin{aligned}\overline{v}_{i_0}^{\pi} - \underline{v}_{i_0} &\le \alpha^{\lceil \log(\varepsilon/v_{\max}^{\alpha})/\log\alpha \rceil} v_{\max}^{\alpha} \\ &\le \alpha^{\log(\varepsilon/v_{\max}^{\alpha})/\log\alpha} v_{\max}^{\alpha} \\ &= \varepsilon.\end{aligned}$$

It remains to be proven that $\underline{v}_{i_0}$ and $\overline{v}_{i_0}^{\pi}$ are ε-close lower and upper bounds for the component $v_{i_0}^{\alpha}$. From Lemmas 3.4 and 3.8 it is already known that $v_{i_0}^{\alpha} \ge \underline{v}_{i_0}$ and $v_{i_0}^{\alpha} \le \overline{v}_{i_0}^{\pi}$. By these inequalities we obtain:

$$\begin{aligned}\overline{v}_{i_0}^{\pi} - v_{i_0}^{\alpha} &\le \overline{v}_{i_0}^{\pi} - \underline{v}_{i_0} \le \varepsilon, \\ v_{i_0}^{\alpha} - \underline{v}_{i_0} &\le \overline{v}_{i_0}^{\pi} - \underline{v}_{i_0} \le \varepsilon.\end{aligned}$$

□

We mention that Theorem 3.17 is still true in the case of an infinite state space $\mathbb{S}$ if there exists a finite upper bound for the expected stage costs, i. e., $\sup_{i\in\mathbb{S},a\in\mathbb{A}(i)} c_i(a) < \infty$. Since the optimal value of the linear program $(\mathrm{U}_S^{i_0})$ is a stronger upper bound on $v_{i_0}^{\alpha}$ than $\overline{v}_{i_0}^{\pi}$ (see Theorem 3.11), we also have the following result.

Corollary 3.18. *Under the same assumptions as used in Theorem 3.17, let $\overline{v}_{i_0}$ be the optimal value of the linear program $(\mathrm{U}_S^{i_0})$ for the subset of states $S = S(i_0,r)$. Then, we have:*

$$\overline{v}_{i_0} - \underline{v}_{i_0} \le \varepsilon.$$

Particularly, $\overline{v}_{i_0}$ is also an ε-close upper bound on $v_{i_0}^{\alpha}$, i. e., $\overline{v}_{i_0} - v_{i_0}^{\alpha} \le \varepsilon$.

Remark 3.19. The size of the restricted state space is optimal in some sense, as can be seen from the example of a “tree like” MDP, in which every state has exactly b different controls, that, with uniform transition probabilities, lead to exactly d “new states” (that can be reached only via this control). In this case, one can show that $S = S(i_0,r)$ as above is the smallest restricted state space to obtain the desired approximation. Of course, incorporating additional parameters of the MDP might give better results in special cases.

Remark 3.20. Of all the approaches from the literature the random sampling algorithm of Kearns et al. [KMN99] gives the results most comparable to Theorem 3.17. However, the size of the restricted state space in our construction is significantly smaller than that for random sampling. This algorithm samples states within the neighborhood of the considered state i_0 up to a radius r_s with:

$$r_s = \left\lceil \frac{\log x}{\log \alpha} \right\rceil, \quad \text{where } x := \frac{\varepsilon(1-\alpha)^3}{4c_{\max}}.$$

Obviously, this gives a considerably larger subset of states since r_s is greater than the radius $r = \lceil \log(\varepsilon(1-\alpha)/c_{\max})/\log\alpha \rceil - 1$ used in Theorem 3.17. For instance, if $c_{\max} = 1$, $\alpha = 0.7$, and $\varepsilon = 0.1$, the radius r_s equals $r_s = 21$, while the radius in our construction equals $r = 10$.

However, the setting considered in [KMN99] is quite different as the authors assume the maximum number of successor states d for an action to be very large or even infinite. Indeed, the number of states sampled by their algorithm is independent of d. This way, their approach deals with the third curse of dimensionality also, i. e., a huge number of possible successors. They sample for each considered state in radius smaller than r_s, at most

$$T = x^{-2}\left[\ln\left(\frac{1-\alpha}{x}\right) + 2r\ln\left(x^{-2}br\ln\left(\frac{1-\alpha}{x}\right)\right)\right]$$

consecutive states if $T < d$. Note that this restriction only makes a difference when d is really large: even fairly simple situations imply huge values for T, e. g., if $c_{\max} = 1$, $b = 4$, $\alpha = 0.7$, and $\varepsilon = 0.1$, we obtain for T a value greater than 1.9 billion.

Our proposal is not to use the state space restricted by the bound on the necessary radius but a state space dynamically computed by column generation techniques. This will be the topic of the next section.

4. A Column Generation Algorithm for Finding Good Induced MDPs

In order to compute local approximations of the optimal value vector $v^{\alpha}_{i_0}$ around a particular state of a given MDP, it is usually inappropriate to apply the construction of Theorem 3.17 directly. In this section, we propose our algorithmic approach to approximate v^{α} locally which is based on the theory developed so far. Further applications of our algorithm include the approximation for a concrete policy or a specific action at a single state. Details are given in Section 5. The algorithmic approach presented below is the basis of our computational tool that is applied in Section 5 to the elevator control problem. Finally we compare our method to the approach of Dean et al. [DKKN93].

4.1. **Algorithm.** The general idea of our approximation algorithm is to start with a small subset of states $S_1 \subset \mathbb{S}$ containing the considered state $i_0 \in \mathbb{S}$. The state space S_1 provides initial lower and upper bounds on $v^{\alpha}_{i_0}$ via the solution of the corresponding linear programs $(\mathrm{L}^{i_0}_{S_1})$ and $(\mathrm{U}^{i_0}_{S_1})$. Then, in order to improve the approximation on $v^{\alpha}_{i_0}$, the state space S_1 is successively extended by adding new states. Note that each newly added state $i \in \mathbb{S} \setminus S_1$ results in one additional variable and $|\mathbb{A}(i)|$ additional constraints in both linear programs $(\mathrm{L}^{i_0}_{S_1\cup\{i\}})$ and $(\mathrm{U}^{i_0}_{S_1\cup\{i\}})$. This way, the algorithm constructs a finite sequence of subsets $S_1 \subset S_2 \subset \cdots \subset S_n \subseteq \mathbb{S}$ for some $n \in \mathbb{N}$ together with a sequence of improving lower and upper bounds on $v^{\alpha}_{i_0}$ obtained as the optimal values of the corresponding linear programs. Using policy iteration instead of linear programming a similar algorithmic approach has already been proposed by Dean et al. [DKKN93]. However, our approach has several advantages as we will see later.

Recall that the theoretical approximation results given in Theorem 3.17 and Corollary 3.18 provide an approximation in terms of the absolute difference between upper and lower bounds. In practice, however, a relative guarantee is typically more suitable when

Algorithm 1 Generic approximation algorithm

```
1: Input: an MDP $(\mathbb{S},\mathbb{A},p,c)$ (given implicitly), a discount factor $\alpha \in [0,1)$, a state $i_0 \in \mathbb{S}$, a subset of states $S \subseteq \mathbb{S}$
   with $i_0 \in S$, $\varepsilon > 0$
2: Output: lower and upper bounds $\underline{v}_{i_0}, \overline{v}_{i_0}$ on $v^{\alpha}_{i_0}$ with $(\overline{v}_{i_0} - \underline{v}_{i_0})/\underline{v}_{i_0} \leq \varepsilon$
3: compute $\underline{v}_{i_0}$ and $\overline{v}_{i_0}$ as the optimal values of the linear programs $(\mathrm{L}^{i_0}_S)$ and $(\mathrm{U}^{i_0}_S)$
4: if $(\overline{v}_{i_0} - \underline{v}_{i_0})/\underline{v}_{i_0} \leq \varepsilon$ then
5:     return $\underline{v}_{i_0}, \overline{v}_{i_0}$
6: else
7:     $S \leftarrow S \cup S_{\text{new}}$ for some $S_{\text{new}} \subseteq \mathbb{S} \setminus S$
8:     go to step 3
9: end if
```

$\underline{v}_{i_0} > 0$. Therefore, the usual goal of our algorithm is to obtain an approximation on $v^{\alpha}_{i_0}$, where the relative difference between the upper and lower bounds is less than a desired guarantee $\varepsilon > 0$, i. e.,

$$\frac{\overline{v}_{i_0} - \underline{v}_{i_0}}{\underline{v}_{i_0}} \leq \varepsilon \text{ for } \underline{v}_{i_0} > 0.$$

Once this approximation guarantee is obtained, the algorithm terminates. In the following, we tacitly assume that $\underline{v}_{i_0} > 0$ whenever the relative performance guarantee is referred to. The generic approximation algorithm is summarized in Algorithm 1. Clearly, Algorithm 1 terminates after a finite number of iterations since the state space $\mathbb{S}$ is finite and we have $\underline{v}_{i_0} = \overline{v}_{i_0} = v^{\alpha}_{i_0}$ for the optimal values of the linear programs $(\mathrm{L}^{i_0}_{\mathbb{S}})$ and $(\mathrm{U}^{i_0}_{\mathbb{S}})$.

Remark 4.1. It has been shown in Theorem 3.10 that by solving the linear program $(\mathrm{U}^{i_0}_S)$ for some state space $S \subseteq \mathbb{S}$ with $i_0 \in S$, one can easily derive a policy π for the original MDP with the property $v^{\alpha}_{i_0}(\pi) \leq \overline{v}_{i_0}$. Consequently, our approximation algorithm also determines a *near-optimal* action a_0 at state i_0 in the sense that there exists a policy π with $\pi(i_0) = a_0$ such that $(v^{\alpha}_{i_0}(\pi) - v^{\alpha}_{i_0})/v^{\alpha}_{i_0} \leq \varepsilon$.

4.2. **Column Generation.** Our implementation of Algorithm 1 is based on the idea to extend the considered state space dynamically by means of column generation, which is a standard technique for solving large-scale linear programs. We refer to the book of Desaulniers et al. [DDS05] for details about column generation. The original problem we aim to solve (approximately) here is $(\mathrm{L}^{i_0}_{\mathbb{S}})$, which equals the linear program (P^{i_0}). Consequently, the master problem that is to be solved in each iteration of the column generation is $(\mathrm{L}^{i_0}_S)$ for some subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$. Thus, for computing the sequence of state spaces $S_1 \subset S_2 \subset \cdots \subset S_n \subseteq \mathbb{S}$ we solely consider the linear programs providing the lower bounds on $v^{\alpha}_{i_0}$. The linear programs $(\mathrm{U}^{i_0}_S)$ only contribute in terms of the computed upper bounds. We mention that it is not straight-forward to apply column generation w. r. t. some linear program $(\mathrm{U}^{i_0}_S)$ with $S \subset \mathbb{S}$ since an associated feasible solution cannot be extended trivially to one for $(\mathrm{L}^{i_0}_{\mathbb{S}})$. Using our column generation algorithm good approximations on $v^{\alpha}_{i_0}$ can be provided by proper subsets of $\mathbb{S}$ that are substantially smaller than the original state space $\mathbb{S}$, as we will see later.

In order to specify our column generation method in detail, consider the dual linear program of $(\mathrm{L}^{i_0}_S)$, which reads as follows in scalar notation:

$$\begin{aligned}
\min \quad & \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} c_i(a) u_{ia} && && (\mathrm{DL}^{i_0}_S) \\
\text{subject to} \quad & \sum_{a \in \mathbb{A}(i_0)} u_{i_0 a} - \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{i i_0}(a) u_{ia} = 1 && && \\
& \sum_{a \in \mathbb{A}(j)} u_{ja} - \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u_{ia} = 0 && \forall j \in S \setminus \{i_0\} && \\
& u_{ia} \geq 0 && \forall i \in S \, \forall a \in \mathbb{A}(i). &&
\end{aligned}$$

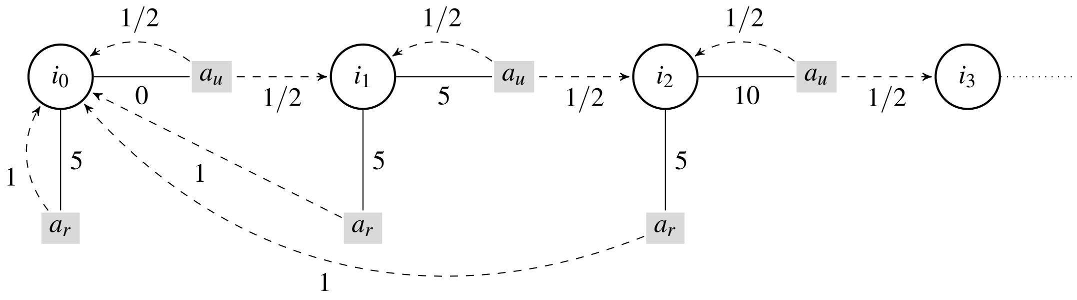


FIGURE 3. Part of the Markov decision process for the machine replacement problem.

Given an optimal solution $\underline{u}$ of the dual linear program $(\mathrm{DL}_S^{i_0})$ for some subset $S \subset \mathbb{S}$, the reduced profit $\bar{p}_j$ of a state $j \in \mathbb{S} \setminus S$ equals:

$$\bar{p}_j = \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia}. \tag{12}$$

Depending on the structure of the MDP it can happen that a possibly small and proper subset of states S suffices to compute $v_{i_0}^{\alpha}$ exactly. This is the case in situations where many states are not reachable from i_0 or only via actions that can be classified by the algorithm to be non-optimal. Such a situation is shown in the following example.

Example 4.2. Consider the following machine replacement problem, e. g., see [Ber01, volume 1, chapter 1]. A single machine is to be operated in an efficient way for a given number of periods. In the course of time, the machine may fall off in quality. That is, at the beginning of each period the machine is in any of $n \in \mathbb{N}$ states, denoted by $i_0, i_1, \dots, i_n$, where states become worse with increasing index. State i_0 corresponds to a machine in perfect condition. Operating the machine for one period may cause the current state to degrade or to stay unchanged. Consider a state i_k for some $k \in \{0, \dots, n\}$. Then there exist two possible actions:

- a_u: use the machine as it is for one period at a cost of $c_k \in \mathbb{R}_+$ which brings the machine to state i_l with probability $p_{i_k i_l}$ for $l \in \{k, \dots, n\}$.
- a_r: repair the machine at a cost of $c^r \in \mathbb{R}_+$ which brings it to the perfect state i_0 and allows for operating the machine for one period at zero cost without a possible degeneration of the machine.

Assuming to operate the machine for an infinite number of periods, an MDP $(\mathbb{S}, \mathbb{A}, p, c)$ for the machine replacement problem is naturally given by:

$$\begin{aligned} \mathbb{S} &= \{i_0, \dots, i_n\}, && \\ \mathbb{A}(i_k) &= \{a_u, a_r\} && \forall k \in \{0, \dots, n\}, \\ p_{i_k i_l}(a_u) &= p_{i_k i_l} && \forall k, l \in \{0, \dots, n\}, \\ p_{i_k i_l}(a_r) &= \begin{cases} 1, & \text{if } l = 0 \\ 0, & \text{if } l \neq 0 \end{cases} && \forall k, l \in \{0, \dots, n\}, \\ c_{i_k}(a_u) &= c_k && \forall k \in \{0, \dots, n\}, \\ c_{i_k}(a_r) &= c^r && \forall k \in \{0, \dots, n\}. \end{aligned}$$

In the example we consider 10 possible machine states, i. e., $n = 9$, and make the (quite unrealistic but possible) assumption that $c_k = 5k$ for each $k \in \{0, \dots, n\}$ and $c^r = 5$. The transition probabilities equal $p_{i_k i_k}(a_u) = p_{i_k i_{k+1}}(a_u) = 1/2$ for $k \in \{0, \dots, n-1\}$ and $p_{i_n i_n}(a_u) = 1$. The associate Markov decision process is partially illustrated in Figure 3.

It is easy to see that using the machine as it is at state i_0 and repairing the machine in each other state defines an optimal policy for the MDP independently of the used discount factor $\alpha \in [0,1)$.

Let $S_1 = \{i_0\}$ be the initial subset of states. The associated dual linear program $(\mathrm{DL}_{S_1}^{i_0})$ reads:

$$\begin{array}{rlll} \min & 0u_{i_0a_u} & + 5u_{i_0a_r} & \\ \text{subject to} & (1-\frac{\alpha}{2})u_{i_0a_u} + & (1-\alpha)u_{i_0a_r} & = 1 \\ & & u_{i_0a_u}, u_{i_0a_r} & \geq 0. \end{array}$$

The optimal solution is unique and given by $\underline{u}_{i_0a_u} = 2/(2-\alpha)$ and $\underline{u}_{i_0a_r} = 0$. Consequently, the reduced profit of state i_1 equals:

$$\bar{p}_{i_1} = \frac{\alpha}{2}\underline{u}_{i_0a_u} = \frac{\alpha}{2-\alpha} > 0,$$

while each other state in $\mathbb{S} \setminus S_1$ has reduced profit 0. Adding state i_1, we obtain the subset of states $S_2 = \{i_0, i_1\}$ and the associated dual linear program $(\mathrm{DL}_{S_2}^{i_0})$:

$$\begin{array}{rlllll} \min & 0u_{i_0a_u} & + 5u_{i_0a_r} & + 5u_{i_1a_u} & + 5u_{i_1a_r} & \\ \text{subject to} & (1-\frac{\alpha}{2})u_{i_0a_u} + & (1-\alpha)u_{i_0a_r} & & - \alpha u_{i_1a_r} & = 1 \\ & -\frac{\alpha}{2}u_{i_0a_u} & & + (1-\frac{\alpha}{2})u_{i_1a_u} & + u_{i_1a_r} & = 0 \\ & & & u_{i_0a_u}, u_{i_0a_r}, u_{i_1a_u}, & u_{i_1a_r} & \geq 0. \end{array}$$

One can show that for $\alpha \leq 2/3$ the optimal dual solution is given by:

$$\begin{aligned} \underline{u}_{i_0a_u} &= 2/(2-\alpha-\alpha^2), \\ \underline{u}_{i_1a_r} &= \alpha/(2-\alpha-\alpha^2), \\ \underline{u}_{i_0a_r} &= \underline{u}_{i_1a_u} = 0. \end{aligned}$$

Therefore, the reduced profit of each state in $\mathbb{S} \setminus S_2$ equals zero and the column generation terminates having computed the component $v_{i_0}^{\alpha}$ of the optimal value vector exactly. Due to linear programming duality, we have $v_{i_0}^{\alpha} = 5\alpha/(2-\alpha-\alpha^2)$.

On the other hand, if we have $\alpha > 2/3$, the unique optimal solution of $(\mathrm{DL}_{S_2}^{i_0})$ satisfies $\underline{u}_{i_1a_u} > 0$ and $\underline{u}_{i_1a_r} = 0$. Thus, the column generation continues: the greater α, the more states are generated until the algorithms terminates. For instance, computational results showed that for this example the algorithm generates all states in the case $\alpha = 0.99$.

4.3. **Approximation for Policies.** Since we will consider different MDPs, the value vector of an MDP M w. r. t. a given discount factor α will be denoted by v_M^{α} in this section. Recall that the value vector $v_M^{\alpha}(\pi)$ of a concrete policy π for a given discount MDP (M,α) can be computed by solving the system of linear equations (2). However, the usual methods to solve a system in $|\mathbb{S}|$ variables and linear equations do not work for us because of the huge state spaces we face in our context.

In this section, we address the local approximation of the value vector $v_M^{\alpha}(\pi)$. The following observation shows that this goal can be accomplished by the same method as before, used for the local approximation of the optimal value vector.

Theorem 4.3. *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c)$ and a policy π, define the* policy induced *MDP $M(\pi) = (\mathbb{S}, \mathbb{A}', p', c')$ by $\mathbb{A}'(i) = \{\pi(i)\}$ for each state $i \in \mathbb{S}$ and suitable restrictions p' and c' of the transition probabilities and stage costs. Then, we have $v_M^{\alpha}(\pi) = v_{M(\pi)}^{\alpha}$ for any discount factor $\alpha \in [0,1)$.*

Proof. $v_M^{\alpha}(\pi) = v_{M(\pi)}^{\alpha}$ follows from Theorem 3.1 on page 4 since the equations (2) for policy π to compute $v_M^{\alpha}(\pi)$ and the optimality equations (3) for $M(\pi)$ are identical. □

By Theorem 4.3 we can approximate the value vector of a concrete policy π at a given state in the same way as we did for the optimal value vector but using a different MDP. The linear programs providing lower bounds on $v^{\alpha}_{M,i_0}(\pi)$ that are to be solved here are of the following type, where $S \subseteq \mathbb{S}$:

$$\begin{aligned} \max\ & v_{i_0} \\ \text{subject to}\ & Q^{S,\pi} v \leq c^{S,\pi} \\ & v \in \mathbb{R}^S. \end{aligned} \tag{$\mathrm{L}^{i_0}_{S,\pi}$}$$

The linear program ($\mathrm{L}^{i_0}_{S,\pi}$) as well as the associated linear program ($\mathrm{U}^{i_0}_{S,\pi}$) providing an upper bound on $v^{\alpha}_{M,i_0}(\pi)$, which we already introduced in Theorem 3.11, have as many variables as constraints. Adding one state in the column generation method only results in one new variable and one new constraint. Since it is not necessary to explore the state space for several actions at one state, approximating $v^{\alpha}_{M,i_0}(\pi)$ is usually easier than approximating v^{α}_{M,i_0}. Therefore, the required state space for the former will often contain fewer states than needed for the latter, especially if the number of possible actions at the states is large. Of course, this difference applies for the neighborhood construction with fixed radius as used in Corollary 3.18 and the approximation algorithm as well.

An optimal solution for the linear program ($\mathrm{L}^{i_0}_{S,\pi}$) can obviously be obtained by solving the corresponding system of linear equations.

Corollary 4.4. *Given an MDP $(\mathbb{S},\mathbb{A},p,c)$, a discount factor $\alpha \in [0,1)$, a policy π, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, an optimal solution to the linear program ($\mathrm{L}^{i_0}_{S,\pi}$) is given by the unique solution of the following system of linear equations:*

$$Q^{S,\pi} v = c^{S,\pi}. \tag{13}$$

Solving systems of linear equations instead of linear programs is another advantage in the approximation of $v^{\alpha}_{M,i_0}(\pi)$ compared to that of v^{α}_{M,i_0}.

4.4. Approximation for Actions.

Note that for a given policy π and a state $i_0 \in \mathbb{S}$, the component $v^{\alpha}_{i_0}(\pi)$ of the value vector of policy π does not only depend on the action $\pi(i_0)$, but on many further decisions made by the policy as well. Thus, by comparing $v^{\alpha}_{i_0}(\pi)$ and the component of the optimal value vector $v^{\alpha}_{i_0}$, the entire policy π is evaluated when the initial state is i_0.

Often it is more desirable to evaluate only a single action at a particular state (not an entire policy), given that the decisions at other states are made w. r. t. an optimal policy. To the best of our knowledge, this type of evaluation has not been proposed in the context of MDPs before. Using another restriction of the original MDP, our method can be applied for this purpose, too. We define the optimal total expected α-discounted cost w. r. t. a fixed action as follows.

Definition 4.5. Given an MDP $M = (\mathbb{S},\mathbb{A},p,c)$, a state $i_0 \in \mathbb{S}$, and an action $a_0 \in \mathbb{A}(i_0)$, let $M(i_0,a_0) = (\mathbb{S},\mathbb{A}',p',c')$ be the MDP with $\mathbb{A}'(i_0) = \{a_0\}$ and $\mathbb{A}'(i) = \mathbb{A}(i)$ for each state $i \in \mathbb{S} \setminus \{i_0\}$ and suitable restrictions p' and c' of p and c, respectively. The *optimal total expected α-discounted cost $v^{\alpha}_{M,i_0}(a_0)$ w. r. t. action* a_0 for a given discount factor $\alpha \in [0,1)$ is defined by:

$$v^{\alpha}_{M,i_0}(a_0) = v^{\alpha}_{M(i_0,a_0),i_0}.$$

The value $v^{\alpha}_{M,i_0}(a_0)$ can be seen as the value vector at state i_0 of a policy that is optimal among all policies that apply action a_0 at state i_0. Therefore, the difference between the values $v^{\alpha}_{M,i_0}(a_0)$ and v^{α}_{M,i_0} reflects the impact of using action a_0 at state i_0 instead of an optimal action.

Theorem 4.6. *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c)$ and a state $i_0 \in \mathbb{S}$, an action $a_0 \in \mathbb{A}(i_0)$ is optimal (as defined in Definition 3.2 on page 5) for a discount factor α if and only if $v^\alpha_{M,i_0}(a_0) = v^\alpha_{M,i_0}$.*

Proof. Let action a_0 be optimal, i. e., there exists an optimal policy π for M with $\pi(i_0) = a_0$. Since π is also a policy for the MDP $M(i_0, a_0)$, we obtain:

$$v^\alpha_M = v^\alpha_M(\pi) = v^\alpha_{M(i_0,a_0)}(\pi) \geq v^\alpha_{M(i_0,a_0)}.$$

Therefore, $v^\alpha_M = v^\alpha_{M(i_0,a_0)}$ since $v^\alpha_M \leq v^\alpha_{M(i,a)}$ holds for each state $i \in \mathbb{S}$ and each action $a \in \mathbb{A}(i)$. In particular, we have $v^\alpha_{M,i_0}(a_0) = v^\alpha_{M(i_0,a_0),i_0} = v^\alpha_{M,i_0}$.

Now assume control a_0 is not optimal. Hence, each policy π for M with $\pi(i_0) = a_0$ is not optimal either, which gives:

$$v^\alpha_M < \min_{\pi:\pi(i_0)=a_0} v^\alpha_M(\pi) = v^\alpha_{M(i_0,a_0)},$$

where again $x < y$ for vectors $x, y \in \mathbb{R}^m$ means $x_i \leq y_i$ for each $i \in \{1, \dots, m\}$ and $x_i < y_i$ for at least one $i \in \{1, \dots, m\}$.

Since the optimality equations (2) for computing v^α_M and $v^\alpha_{M(i_0,a_0)}$ only differ for state i_0:

$$v^\alpha_{M,i_0} = \min_{a \in \mathbb{A}(i_0)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{i_0 j}(a) v^\alpha_{M,j} \right\},$$

$$v^\alpha_{M(i_0,a_0),i_0} = c_{i_0}(a_0) + \alpha \sum_{j \in \mathbb{S}} p_{i_0 j}(a_0) v^\alpha_{M(i_0,a_0),j},$$

$v^\alpha_{M,i_0} = v^\alpha_{M(i_0,a_0),i_0}$ would imply $v^\alpha_M = v^\alpha_{M(i_0,a_0)}$, which is a contradiction. Thus, we also have $v^\alpha_{M,i_0} < v^\alpha_{M(i_0,a_0),i_0} = v^\alpha_{M,i_0}(a_0)$. $\square$

For an arbitrary subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let Q^{S,i_0,a_0} be the submatrix of Q^S, where exactly the rows (i_0, a) with $a \neq a_0$ are removed. Similarly, let c^{S,i_0,a_0} be the subvector obtained from c^S by removing the components with index (i_0, a) for $a \neq a_0$. Now consider the following linear program providing a lower bound on $v^\alpha_{M,i_0}(a_0)$:

$$\begin{aligned} \max \quad & v_{i_0} \qquad\qquad (\mathrm{L}^{i_0}_{S,a_0}) \\ \text{subject to} \quad & Q^{S,i_0,a_0} v \leq c^{S,i_0,a_0} \\ & v \in \mathbb{R}^S. \end{aligned}$$

It is clear how the corresponding linear programs for computing upper bounds for $v^\alpha_{M,i_0}(a_0)$ would look like. It follows from the definition that $v^\alpha_{M,i_0}(a_0)$ equals the optimal value of the linear program for $S = \mathbb{S}$. Since this linear program equals (P^{i_0}) except for the constraints for state i_0, the computational effort for approximating $v^\alpha_{M,i_0}(a_0)$ via our column generation algorithm is expected to be similar to that required for the component v^α_{M,i_0} of the optimal value vector. In the approximation process linear programs of the type $(\mathrm{L}^{i_0}_{S,a_0})$ restricted to some subset of states $S \subseteq \mathbb{S}$ are to be solved.

Obviously, Theorem 4.6 directly implies the following result.

Corollary 4.7. *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c)$, a discount factor α, a state $i_0 \in \mathbb{S}$, and an action $a_0 \in \mathbb{A}(i_0)$, assume that we have $v^\alpha_{M,i_0}(a) \geq v^\alpha_{M,i_0}(a_0)$ for each action $a \in \mathbb{A}(i_0)$. Then, the action a_0 is optimal.*

The corollary implies that our approximation algorithm may be employed to determine an optimal action at a particular state i_0. Assume that the algorithm has computed an upper bound $\overline{v}_{M,i_0}(a_0)$ on $v^\alpha_{M,i_0}(a_0)$ for some action $a_0 \in \mathbb{A}(i_0)$ and lower bounds $\underline{v}_{M,i_0}(a) \leq v^\alpha_{M,i_0}(a)$ for each different action $a \in \mathbb{A}(i_0) \setminus \{a_0\}$. Then, if $\overline{v}_{M,i_0}(a_0) \leq \underline{v}_{M,i_0}(a)$ for each $a \in \mathbb{A}(i_0) \setminus \{a_0\}$, the action a_0 is optimal. In Section 5 we will exploit this observation in the analysis of policies a for elevator control MDP.

4.5. **Comparison.** Finally, we briefly discuss our approximation algorithm compared to the approach of Dean et al. [DKKN93]. The aim of their method is to find an optimal policy for a state space restricted to those states which are likely to be encountered within a smaller number of transitions. Similar to our approach, their algorithm computes an optimal policy for the induced MDP in each iteration and extends the restricted state space dynamically depending on the obtained policy. Instead of linear programming, policy iteration is used to compute the optimal policies. The main advantages of Algorithm 1 compared to their method are the following. Firstly, in the approximation process we are able to monitor the current approximation guarantee, while the approach of Dean et al. only provides lower bounds on $v^{\alpha}_{i_0}$. Thus, they cannot determine how good the current approximation really is. Secondly, we are able to properly guide the expansion of the restricted state space as the reduced profits of the candidate states are available. This way, our approximation algorithm benefits substantially (see [Tuc10] for computational results). The method of Dean et al. must use heuristic ideas to increase S, in particular, one strategy aims to estimate the reduced profits. Probably, both algorithms have a similar run-time per iteration since the policy iteration method and linear programming method for computing the optimal value vector are in some sense equivalent. Our algorithm may be a bit slower per iteration when a second linear program is solved.

5. Application and Computational Results

In this section we present some seletected results on the outcomes of our tool on a prominent example: the elevator control problem. It is one example out of three that have been investigated with our tool so far. The other two computational studies have an additional twist because they are about online algorithms, for which competitive analysis showed counter-intuitive outcomes. We have chosen the elevator example because:

(1) the state space even of a very basic model is too large to be handled explicitly;
(2) the problem of finding performance guarantees of any kind was open;
(3) some new results could be obtained by using our method;
(4) the limitations of our method are emphasized as well.

For a more detailed, extensive computational study we refer to [Tuc10].

In the following, we introduce a Markov decision process formulation of the elevator problem. We then briefly introduce the policies under investigation. Then, we introduce the improved bounds for this problem which are essential to achieve interesting insights. Finally, we state the instantiation of the elevator problem and the results.

5.1. **Markov Decision Process Model.** In order to formulate a Markov decision process model, we deal with the following situation. The system operates a set of elevators $E = \{1,\dots,n_E\}$ in a building with a set of floors $F = \{1,\dots,n_F\}$. Each elevator can load at most one request. At each floor there is a waiting area that accommodates at most $q \in \mathbb{N} \cup \{\infty\}$ transport requests. We limit our considerations to a discrete time model. At each time slot the current situation is described by the following data:

- Each elevator $e \in E$ is situated at one floor $f_e \in E$ and is either loaded or empty.
- For each floor $f \in F$, there exists a sequence $\sigma_f = r_1,\dots,r_{n_f}$ of waiting requests, where $n_f \in \{0,\dots,q\}$ is their number. Moreover, each request r_k for $k \in \{1,\dots,n_f\}$ is of the form $r_k = (f, f_k, w_k)$, where $f_k \in F \setminus \{f\}$ is its destination floor and $w_k \in \mathbb{N}_0$ is the waiting time of request r_k so far. Denote by $w_{\sigma_f} := w_1$ the maximum waiting time of a request in sequence σ_f if it is non-empty, and let Σ_f be the set of all possible sequences at floor f.

Feasible Actions. If elevator $e \in E$ is loaded, let $d_e \in F$ be the destination floor of the request being transported, and let $d_e = 0$ otherwise. In one time unit an elevator $e \in E$ can execute exactly one of the following operations:

WAIT: at its current floor f_e,

MOVE_UP: one floor if $f_e < n_F$,
MOVE_DOWN: one floor if $f_e > 1$,
LOAD: the next request at the current floor f_e if $d_e = 0$ and $\sigma_{f_e} \neq \emptyset$, i. e., the elevator is empty and there is at least one request waiting at floor f_e, or
DROP: the loaded request if $f_e = d_e$, i. e., the elevator is loaded and its current floor equals the destination floor of the loaded request.

State Space. A state $i \in \mathbb{S}$ in the Markov decision process model $(\mathbb{S}, \mathbb{A}, p, c)$ is of the following form:

$$i = (w_{\max}, (\sigma_f)_{f \in F}, (f_e, d_e)_{e \in E}),$$

where $w_{\max} \in \mathbb{N}_0$ specifies the maximum waiting time of a request so far. Moreover, a state captures all data concerning waiting requests and possibly loaded requests as well as the positions of the elevators. We will also denote the parameters of a state i by $w_{\max}(i)$, $\sigma_f(i)$ for each $f \in F$, and $f_e(i), d_e(i)$ for each $e \in E$. The resulting state space $\mathbb{S}$ is given by:

$$\begin{aligned}\mathbb{S} = \{(w_{\max}, (\sigma_f)_{f \in F}, ((f_e, d_e)_{e \in E}) \mid\ & w_{\max} \in \mathbb{N}_0, w_{\max} \geq w_{\sigma_f}\ \forall f \in F\colon \sigma_f \neq \emptyset, \\ & \sigma_f \in \Sigma_f\ \forall f \in F, \\ & (f_e, d_e) \in F \times (\{0\} \cup F)\ \forall e \in E\}.\end{aligned}$$

As the stored waiting times in a state may become arbitrarily large even if the waiting queue length q is bounded, the state space $\mathbb{S}$ is infinite.

Each action in $\mathbb{A}(i)$ for a state $i \in \mathbb{S}$ is composed of one control decision $a(e)$ for each elevator $e \in E$, i. e., an action $a \in \mathbb{A}(i)$ is of the form $a = (a(e_1), \dots, a(e_{n_E}))$. The control decision of an elevator may be any one of the operations mentioned above: WAIT, MOVE_UP, MOVE_DOWN, LOAD, DROP. However, we assume that a loaded elevator $e \in E$ immediately serves the request being transported: if $f_e < d_e$ or $f_e > d_e$, the elevator e will move up or down, respectively, and if $f_e = d_e$, the request will be dropped.

Transitions. In our model each transition between two states is assumed to last exactly one time step, moving from one time slot to the next one. Moreover, we assume that at most one new request is released at each time slot. We describe possible state transitions only for the case of a single elevator since the general case is obtained by handling the control decisions of all elevators consecutively. If no new request arrives, the deterministic successor $j \in \mathbb{S}$ of a state $i \in \mathbb{S}$ when using action $a = (a(e)) \in \mathbb{A}(i)$ is given by:

- The maximum waiting time at state j equals:
$$w_{\max}(j) = \max\{w_{\max}(i), \max_{f \in F\colon \sigma_f(j) \neq \emptyset} w_{\sigma_f(j)}\}.$$
- For each floor $f \in F \setminus \{f_e\}$, we have $\sigma_f(j) = \sigma_f(i)$. If $a(e) = \text{LOAD}$, the update for the waiting queue at floor f_e is $\sigma_{f_e}(j) = r_2, \dots, r_{n_{f_e}}$, where $\sigma_{f_e}(i) = r_1, \dots, r_{n_{f_e}}$. Otherwise, we have $\sigma_{f_e}(j) = \sigma_{f_e}(i)$.
- The current floor and load of elevator e are updated by:
$$(f_e(j), d_e(j)) = \begin{cases} (f_e(i), d_e(i)), & \text{if } a(e) = \text{WAIT}, \\ (f_e(i)+1, d_e(i)), & \text{if } a(e) = \text{MOVE\_UP}, \\ (f_e(i)-1, d_e(i)), & \text{if } a(e) = \text{MOVE\_DOWN}, \\ (f_e(i), f_1), & \text{if } a(e) = \text{LOAD}, \\ (f_e(i), 0), & \text{if } a(e) = \text{DROP}, \end{cases}$$
where $r_1 = (f_e, f_1, w_1)$ denotes the first request in the sequence $\sigma_{f_e}(i)$ in the loading case.

When a new request $r = (a, b, 0)$ is released at a floor $a \in F$ with destination floor $b \in F \setminus \{a\}$, we obtain the successor $(w_{\max}(j), (\sigma'_f)_{f \in F}, (f_e(j), d_e(j)))$ of state i. In this state, we have $\sigma'_f = \sigma_f(j)$ for each floor $f \in F \setminus \{a\}$ and

$$\sigma'_a = \begin{cases} \sigma_a(j) + r, & \text{if } |\sigma_a(j)| < q, \\ \sigma_a(j), & \text{if } |\sigma_a(j)| = q, \end{cases}$$

where $\sigma_a(j) + r$ denotes the sequence with request r added to $\sigma_a(j)$.

The transition probabilities p are defined by a two step process. Firstly, we have a fixed probability that a new request is released at a state transition (Bernoulli distribution). If that is the case, the start and destination floor of the new request are determined according to some probability distribution in the second step.

Depending on the used objective function, the stage costs are given as follows. If we focus on minimizing the maximum waiting time of a request, it is always assumed that the waiting queues are unbounded, i. e., $q = \infty$. In this case, the stage cost $c_i(a, j) = c_i^{\max}(a, j)$ associated with states $i, j \in \mathbb{S}$ and action $a \in \mathbb{A}(i)$ equals the increase of the maximum waiting time due to action a:

$$c_i^{\max}(a, j) = w_{\max}(j) - w_{\max}(i).$$

Notice that the total sum of stage costs for the transitions of an (i, j)-path equals the total increase of the maximum waiting time in this sequence of states.

For minimizing the average waiting time, we assume the waiting queue length to be bounded, i. e., $q < \infty$. Whenever a request is released at a floor $f \in F$ where the waiting queue is full, i. e., $|\sigma_f| = q$, the request is rejected from the system at a penalty cost of $c_p \geq 1$. For each floor $f \in F$, let $0 \leq p_f \leq 1$ be the probability that a request is released at some time slot at floor f. Given states $i, j \in \mathbb{S}$ and an action $a \in \mathbb{A}(i)$, let $j' \in \mathbb{S}$ be the successor of i using action a if no new request arrives. Then, the stage cost $c_i(a, j) = c_i^{\text{avg}}(a, j)$ is defined as the sum of all requests waiting at state i that are not loaded by action a plus the expected penalty cost:

$$c_i^{\text{avg}}(a, j) = \sum_{f \in F} |\sigma_f(i)| - |\{e \in E \mid a(e) = \text{LOAD}\}| \\ + \begin{cases} c_p \cdot \sum_{f \in F\colon |\sigma_f(j')| = q} p_f, & \text{if } \sigma_f(j) = \sigma_f(j') \text{ for all } f \in F, \\ 0, & \text{otherwise.} \end{cases}$$

In the case the waiting queues of the states j and j' differ, a new request has been released at a floor where the waiting queue was not full w. r. t. state j'. Thus, the transition does not involve a penalty cost.

Notice that $c_i^{\text{avg}}(a, j)$ equals the increase of the sum of all waiting times plus the expected penalty cost. Thus the sum of the expected stage costs for all transitions of an (i, j)-path equals the sum of all accumulated waiting times and expected penalty costs during the associated time period. Minimizing this objective for a finite sequence of requests is equivalent to minimizing the average waiting time.

We want to point out, that the basic Markov decision process model we consider here differs substantially from the one used by Crites and Barto [CB98].

5.2. **Policies.** Later in this section we present some selected results for the policies described below. Again for extensive computational results we refer to [Tuc10].

The considered policies work as follows:

FIRSTINFIRSTOUT (FIFO): Serve the request with the smallest current waiting time next (this request is unique by our assumption that at most one request is released at each time slot). For the policy FIFO, we additionally store in each state the order of arrival for the waiting requests.

NEARESTNEIGHBOR (NN): Determine a waiting request whose start floor is located nearest to the current floor of the elevator. If there exists a unique request with this property, serve it next. Otherwise, such a request exists in both directions. Then, serve the one with smaller floor number next.

REPLAN: Compute a schedule minimizing the makespan (without returning to some origin), i. e., the time needed to serve all waiting requests, and serve the requests according to this schedule. We implemented a branch-and-bound method to compute these schedules.

IGNORE: As long as a schedule is available, serve the waiting requests accordingly. If no schedule is available, do the same as the policy REPLAN and store the schedule. The policy IGNORE requires a modified MDP where each state encodes a schedule containing a (possibly empty) subset of the waiting requests. Moreover, if for some state this schedule is empty and a request is waiting, each associated action has a second component that sets the schedule for all waiting requests.

5.3. **Improved Bounds.** We exploit involved lower and upper bounds on the components v_i^α of the optimal value vector. Recall that we consider two different elevator control MDPs, one for analyzing the average waiting time and another for dealing with the maximum waiting time.

5.3.1. *Average waiting time.* The construction of state-specific bounds for the MDP modeling the average waiting time is as follows. For each state $i \in \mathbb{S}$, we employ a lower bound $v_{\min}^\alpha(i) \le v_i^\alpha$ consisting of two parts, i. e., $v_{\min}^\alpha(i) = v_{\min}^{\alpha,1}(i) + v_{\min}^{\alpha,2}(i)$.

The first lower bound $v_{\min}^{\alpha,1}(i)$ takes into account future requests arriving in the system. It is based on a lower bound for the probability $p^{\text{no\_elevator}}$ that a request arrives at a floor, where no elevator is located. Let again $0 \le p_f \le 1$ be the probability that a request with start floor $f \in F$ is released at a time slot. Consider a permutation $f_1, \dots, f_{|F|} \in F$ of the floors such that the probabilities are non-decreasing w. r. t. the permutation: $p_{f_1} \le \dots \le p_{f_{|F|}}$. Since in each state there exist at least $|F| - |E|$ floors where no elevator is located, the probability $p^{\text{no\_elevator}}$ is at least the sum of the $|F| - |E|$ smallest arrival probabilities $p_{f_1}, \dots, p_{f_{|F|-|E|}}$, i. e., we have:

$$p^{\text{no\_elevator}} \ge \sum_{k=1}^{|F|-|E|} p_{f_k}.$$

Since each request arriving at a floor where no elevator is located will have a waiting time greater or equal 1 and such a request can arrive at each time slot, we obtain:

$$v_i^\alpha \ge \frac{p^{\text{no\_elevator}}}{1-\alpha} \ge \frac{\sum_{k=1}^{|F|-|E|} p_{f_k}}{1-\alpha} =: v_{\min}^{\alpha,1}(i).$$

Note that the first inequality above is only valid since the penalty cost satisfies by assumption $c_p \ge 1 \ge p^{\text{no\_elevator}}$. This gives the first part of the lower bound.

The second part $v_{\min}^{\alpha,2}(i)$ of the lower bound on v_i^α for a state $i \in \mathbb{S}$ captures the total α-discounted cost resulting from the requests waiting in state i. To this end, we consider a relaxation of the elevator control problem where each elevator requires no time for moving empty and all requests waiting at the same floor can be served in arbitrary order. Note that the resulting problem is equivalent to a scheduling problem where the machines correspond to the elevators and the jobs correspond to the waiting requests. In the following, the current time slot at state i will be denoted by 0 and the consecutive time slots by $1, 2, \dots$. Algorithm 2 determines a feasible schedule under the assumptions made and returns the associated number of waiting requests for each future time slot. We claim that the obtained schedule is optimal w. r. t. the resulting total α-discounted cost.

Theorem 5.1. *Under the assumptions made, Algorithm 2 determines a schedule that serves all waiting requests in state i at a minimum total α-discounted cost for each* $0 \le \alpha < 1$.

Algorithm 2 Algorithm processing all waiting requests assuming that each elevator requires no time for moving empty and that all requests at the same floor can be served in arbitrary order. Note that in case of minimizing the average waiting time the information about the maximum waiting time $w_{\max}$ is irrelevant.

```
1: Input: a state $i = (w_{\max}, (\sigma_f)_{f\in F}, (f_e, d_e)_{e\in E})$ in a Markov decision process with elevator set $E$ and floor set $F$
2: Output: a sequence of numbers $n_0^{\text{wait}}, \dots, n_t^{\text{wait}} \in \mathbb{N}_0$ for some $t \in \mathbb{N}_0$, where $n_k^{\text{wait}}$ is the number of requests still waiting at time slot $k$ for each $k \in \{0,\dots,t\}$
3: let $n \leftarrow |\bigcup_{f\in F} \sigma_f(i)|$ and re-index so that $\Delta_1 \le \dots \le \Delta_n$ is a non-decreasing sequence of distances $|a-b|$ of all waiting requests $(w,a,b) \in \bigcup_{f\in F} \sigma_f(i)$
4: for each $e \in E$ do                                        ▷ get minimum loading time slot for elevator $e$
5:     if $d_e = 0$ then                                        ▷ elevator empty
6:         $f \leftarrow f_e$ and $t_e \leftarrow 0$
7:     else                                                     ▷ elevator loaded
8:         $f \leftarrow d_e$ and $t_e \leftarrow |f_e - d_e| + 1$          ▷ driving and dropping time
9:     end if
10:    $t_e \leftarrow t_e + \min_{(a,b,w)\in\bigcup_{f\in F}\sigma_f(i)} |f-a|$      ▷ add time to reach request
11: end for
12: $t \leftarrow 0$
13: $n_t^{\text{wait}} \leftarrow n$                                  ▷ no request served yet
14: for $k = 1$ to $n$ do
15:    let $e' \in \operatorname{argmin}_{e\in E} t_e$
16:    $n_{t+1}^{\text{wait}}, \dots, n_{t_{e'}-1}^{\text{wait}} \leftarrow n_t^{\text{wait}}$          ▷ no more requests loaded before time $t_{e'}$
17:    $n_{t_{e'}}^{\text{wait}} \leftarrow n_t^{\text{wait}} - 1$                    ▷ one request loaded at time $t_{e'}$
18:    $t \leftarrow t_{e'}$
19:    $t_{e'} \leftarrow t_{e'} + \Delta_i + 2$                        ▷ add driving and loading/dropping time
20: end for
21: return $n_0^{\text{wait}}, \dots, n_t^{\text{wait}}$
```

This cost equals:

$$\sum_{k=0}^{t} \alpha^k n_k^{\text{wait}}$$

Proof. We refrain from giving a rigorous proof here. Algorithm 2 is essentially the Shortest Processing Time First rule, which is known to be optimal for the sum of completion times. Since all requests have already been released, a schedule with optimal total completion time has optimal total flow time as well. Discounting does not affect optimality here. □

The result above implies the second lower bound for state $i \in \mathbb{S}$:

$$v_{\min}^{\alpha,2}(i) := \sum_{k=0}^{t} \alpha^k n_k^{\text{wait}} \le v_i^{\alpha}.$$

Notice that $v_{\min}^{\alpha,2}(i)$ takes into account the costs incurred from currently waiting requests only, while $v_{\min}^{\alpha,1}(i)$ solely considers costs due to future requests. Therefore, their sum $v_{\min}^{\alpha}(i) := v_{\min}^{\alpha,1}(i) + v_{\min}^{\alpha,2}(i)$ is a valid lower bound for the component v_i^{α} of the optimal value vector, too.

Obviously, the trivial upper bound $v_{\max}^{\alpha} = c_{\max}/(1-\alpha)$ is very weak in the considered elevator control MDP for most states since the maximum expected stage cost equals $c_{\max} = |F|q + c_p \sum_{f\in F} p_f$. The approach to determine a suitable upper bound $v_{\max}^{\alpha}(i) \ge v_i^{\alpha}$ for each state $i \in \mathbb{S}$ is to compute the expected total number of waiting requests and the expected penalty for each future time slot t up to some limit assuming that no requests are served.

Let $N_t^{\text{wait}} \in \mathbb{N}_0$ and $N_{t,f}^{\text{wait}}$ denote the random variables for the total number of waiting requests and the number of requests waiting at floor $f \in F$ for time slot $t \in \mathbb{N}_0$, respectively.

By the linearity of the expectation we have:

$$\mathbb{E}[N_t^{\text{wait}}] = \mathbb{E}[\sum_{f \in F} N_{t,f}^{\text{wait}}] = \sum_{f \in F} \mathbb{E}[N_{t,f}^{\text{wait}}].$$

For each $f \in F$, the expected value $\mathbb{E}[N_{t,f}^{\text{wait}}]$ can be computed according to the arrival probability p_f at floor f by:

$$\mathbb{E}[N_{t+1,f}^{\text{wait}}] = \min\{\mathbb{E}[N_{t,f}^{\text{wait}}] + p_f, q\}.$$

Moreover, let $P_t \geq 0$ denote the random penalty cost for a stage $t \in \mathbb{N}_0$. In order to determine the expected penalty $\mathbb{E}[P_t]$, we compute the probability $p_{t,f}^{\text{full}}$ that the waiting queue at a floor $f \in F$ is full at time slot t. Let $c_f := q - |\sigma_f(i)|$ be the remaining capacity at each floor $f \in F$ in state i. Note that we always have $p_{t,f}^{\text{full}} = 0$ as long as $t < c_f$ since at most one request is released each stage. Generally, $p_{t,f}^{\text{full}}$ equals the probability that at least c_f new requests have arrived at floor f by time t. Therefore, we obtain for each $t \in \mathbb{N}_0$:

$$p_{t,f}^{\text{full}} = \sum_{k=c_f}^{t} \binom{t}{k} p_f^k (1-p_f)^{t-k}.$$

Again by the linearity of the expectation, the expected penalty $\mathbb{E}[P_t]$ at time $t \in \mathbb{N}_0$ equals:

$$\mathbb{E}[P_t] = c_p \sum_{f \in F} p_{t,f}^{\text{full}} \cdot p_f.$$

Given the expected number of waiting requests $\mathbb{E}[N_t^{\text{wait}}]$ and the expected penalty cost $\mathbb{E}[P_t]$ under the assumption that no requests are served, for each time slot $t \in \mathbb{N}_0$, we have:

$$v_i^\alpha \leq \sum_{t=0}^{\infty} \alpha^t (\mathbb{E}[N_t^{\text{wait}}] + \mathbb{E}[P_t]).$$

Clearly, it is impossible to determine an infinite number of expectations. We stop the expensive computation described above at a time slot $t_{\max}$ when the maximum total α-discounted expected cost $\alpha^{t_{\max}} v_{\max}^\alpha$ after time $t_{\max}$ falls below some threshold value (e. g., 0.1) and add $\alpha^{t_{\max}} v_{\max}^\alpha$. Thus, we obtain the upper bound:

$$v_i^\alpha \leq \alpha^{t_{\max}} v_{\max}^\alpha + \sum_{t=0}^{t_{\max}} \alpha^t (\mathbb{E}[N_t^{\text{wait}}] + \mathbb{E}[P_t]) =: v_{\max}^\alpha(i).$$

Notice that the construction above assumes that none of the requests currently waiting in a state are ever served. We mention that for approximating the component v_i^α of the optimal value vector for some $i \in \mathbb{S}$, it is possible to take into account the processing of the requests waiting in state i according to any feasible schedule. In doing so, the expected stage costs $\mathbb{E}[N_t^{\text{wait}}]$ and $\mathbb{E}[P_t]$ reduce for some time slots t due to serving requests in state i. Consequently, we obtain an improved upper bound $v_{\max}^\alpha(i)$. Our implementation applies the feasible schedule obtained by the policy NN. Note that this construction is generally infeasible when the goal is to approximate the value $v_{i_0}^\alpha(\pi)$ for a policy π since the schedule of π may change when additional requests arrive. Obviously, this is not the case for FIFO, i. e., we can employ the improved bound according to the schedule obtained by FIFO. For all other policies under consideration we have to assume that no requests are served.

5.3.2. *Maximum waiting time.* In the elevator control MDP modeling the maximum waiting time, a lower bound $v_{\min}^\alpha(i) \leq v_i^\alpha$ for a state $i \in \mathbb{S}$ is obtained as follows. Let $F_{\text{wait}}(i) \subseteq F$ be the subset of floors where at least one request is waiting in state i, i. e., $F_{\text{wait}}(i) = \{f \in F \mid \sigma_f(i) \neq \emptyset\}$. Moreover, for each floor $f \in F_{\text{wait}}(i)$, let r_f denote the first request in the waiting queue $\sigma_f(i)$ in state i.

The idea for constructing the lower bound on v_i^α is to determine for each floor $f \in F_{\text{wait}}(i)$ the smallest time t_f by which an elevator can reach floor f, after possibly having served a loaded request. That is, each request r_f for a floor $f \in F_{\text{wait}}(i)$ cannot be loaded before

Algorithm 3 Algorithm for computing for a given state, a set of time slots, where the maximum waiting time will increase.

```
 1: **Input:** a set of non-empty floors $F_{\text{wait}}(i)$, current waiting times $w_f(i)$ and final waiting times $w_f^{\text{final}}$ for each
    $f \in F_{\text{wait}}$, the maximum current waiting time $w_{\max}(i)$
 2: **Output:** a finite set of time slots $T \subset \mathbb{N}_0$, where the maximum waiting time will increase
 3: $T \leftarrow \emptyset$, $F_{\text{wait}} \leftarrow F_{\text{wait}}(i)$, and $w_{\max} \leftarrow w_{\max}(i)$
 4: **while** $F_{\text{wait}} \neq \emptyset$ **do**
 5:     let $f' \in \operatorname{argmax}_{f \in F_{\text{wait}}} w_f(i)$                    ▷ Get floor with oldest request
 6:     **if** $w_{f'}^{\text{final}} > w_{\max}$ **then**
 7:         $T \leftarrow T \cup \{w_{\max} - w_{f'}(i), \dots, w_{f'}^{\text{final}} - w_{f'}(i) - 1\}$
 8:         $w_{\max} \leftarrow w_{f'}^{\text{final}}$
 9:     **end if**
10:     $F_{\text{wait}} \leftarrow F_{\text{wait}} \setminus \{f'\}$
11: **end while**
12: **return** $T$
```

time t_f. Consequently, the smallest possible final waiting time of request r_f equals $w_f^{\text{final}} := w_f(i) + t_f$, where $w_f(i)$ denotes the present waiting time of request r_f in state i. Note that the current maximum waiting time $w_{\max}(i)$ will increase if we have $\max_{f \in F_{\text{wait}}(i)} w_f^{\text{final}} > w_{\max}(i)$. By considering all floors $f \in F_{\text{wait}}(i)$ in order of decreasing current waiting times $w_f(i)$, one can determine a subset of time slots $T \subset \mathbb{N}_0$, where the maximum waiting time will increase, i. e., the associated stage cost equals 1. Algorithm 3 describes in detail how to compute these time slots. It is easy to see that Algorithm 3 correctly determines the set of time slots T at which the maximum waiting time will increase. The set T implies the following lower bound on v_i^α:

$$v_i^\alpha \geq \sum_{t \in T} \alpha^t =: v_{\min}^\alpha(i).$$

Clearly, an upper bound $v_{\max}^\alpha(i)$ for a state $i \in \mathbb{S}$ can be derived by arbitrarily serving the requests waiting in state i and assuming that another request is released at time slot 1 and never served. Consider any policy to compute a feasible schedule for all waiting requests. In our implementation we use FIFO. According to the schedule we obtain the subset of time slots $T \subset \mathbb{N}_0$, where the maximum waiting time will increase (the method to determine T is similar to Algorithm 3). For constructing the second part of the bound let $w_{\max}$ be the maximum final waiting time of a request in state i w. r. t. the used schedule. Note that the waiting time of a request released at time slot 1 will be bounded by $w_{\max}$ until time $w_{\max} + 1$. Therefore, never serving this request implies a stage cost of 1 for the time slots $w_{\max} + 1, w_{\max} + 2, \dots$. Putting the two parts of the construction together, we obtain the following upper bound on v_i^α:

$$v_{\max}^\alpha(i) := \sum_{t \in T} \alpha^t + \sum_{t = w_{\max}+1}^{\infty} \alpha^t = \sum_{t \in T} \alpha^t + \frac{\alpha^{w_{\max}+1}}{1-\alpha} \geq v_i^\alpha.$$

Similar as before, this upper bound is in general not valid if a component $v_{i_0}^\alpha(\pi)$ of the value vector of a given policy π is to be approximated, although the bound is again valid for FIFO. For other policies, we use a simple upper bound for $v_i^\alpha(\pi)$ with $i \in \mathbb{S}$ obtained by computing the maximum current waiting time $w(i)$ of a request in state i. It is clear that the stage cost will equal 0 for the time slots $0, \dots, w_{\max}(i) - w(i) - 1$. This implies the upper bound:

$$v_{\max}^\alpha(i) := \frac{\alpha^{w_{\max}(i) - w(i)}}{1-\alpha} \geq v_i^\alpha(\pi).$$

5.4. **Studied Instances.** Finally, we introduce the instances of the two described Markov decision processes for online elevator control that are studied in the sequel. Recall that the

TABLE 1. Considered instances of the elevator control Markov decision processes. The considered start-to-destination floor probability distributions, denoted by `ud` and `sp` are given in Table 2 and Table 3, respectively.

instance	n_F	n_E	q	c_p	p^r	p^{sd}	$\lvert\mathbb{S}\rvert$
`ela-1-4-10-02-sp`	8	1	4	10	0.2	`sp`	2 086 898 858
`elm-1-02-ud`	8	1	∞	–	0.2	`ud`	∞

TABLE 2. The start-to-destination floor probability distribution `ud` representing combined up and down traffic of equal intensities. f_1 and f_2 denote arbitrary start and destination floors with $f_1, f_2 \in \{2, \dots, 8\}$.

start floor	destination floor	
	1	f_2
1	–	1/14
f_1	1/14	–

TABLE 3. The start-to-destination floor probability distribution `sp` representing a special situation.

start floor	destination floor							
	1	2	3	4	5	6	7	8
1	–	–	–	1/20	–	3/20	–	2/20
2	–	–	–	–	–	–	–	–
3	–	–	–	–	–	–	–	–
4	2/20	–	–	–	–	1/20	–	1/20
5	–	–	–	–	–	–	–	–
6	3/20	–	–	–	–	–	2/20	1/20
7	–	–	–	–	–	–	–	–
8	2/20	–	–	–	–	2/20	–	–

two models differ only in the stage costs. In each case we can specify an instance by the following data:

- a number of floors $n_F \in \mathbb{N}$ defining the set of floors $F := \{1, \dots, n_F\}$,
- a number of elevators $n_E \in \mathbb{N}$ defining the elevator set $E := \{1, \dots, n_E\}$,
- a waiting queue length $q \in \mathbb{N} \cup \{\infty\}$,
- a penalty cost $c_p \geq 1$,
- a probability $0 \leq p^r \leq 1$ that exactly one new request is released at a time slot, and
- a probability distribution for the start and destination floor of a new request given by a function $p^{sd}\colon F \times F \to \mathbb{R}$ with $p^{sd}(f, f) = 0$ for each floor $f \in F$ and $\sum_{f_1 \in F} \sum_{f_2 \in F} p^{sd}(f_1, f_2) = 1$, i. e., the probability that a new request has start floor $f_1 \in F$ and destination floor $f_2 \in F$ equals $p^{sd}(s, d)$.

The two instances we consider are given in Table 1. We keep the same names as in [Tuc10]. The instance `ela-1-4-10-02-sp` is a Markov decision process for the case of minimizing the average waiting time, while the one for minimizing the maximum waiting time is called `elm-1-02-ud`. Here, we only look at problems featuring a single elevator (see [Tuc10] for more tests). We focus on two different distributions for the start and destination floors of new requests. On the one hand, we look at combined *up and down traffic*, i. e., for each transport request, Floor 1 is either its start floor or its destination floor (see Table 2). This setting is natural for a cargo elevator system in an automated warehouse, where goods are placed into storage and retrieved over time. On the other hand, Table 3 shows a special traffic situation that may be representative for some time in the course of a day. One can think of this situation as follows. Still, there are some requests arriving at Floor 1 to be placed into storage and some requests are retrieved, but only a subset of floors are currently utilized. Moreover, there is some *interfloor traffic*, i. e., requests have start and destination floors that are different from floor 1. This may be due to production processes taking place or required relocations of the stored goods.

5.5. **Reading the Charts.** We aim at monitoring during the computation for a selected state $i_0 \in \mathbb{S}$ that is reached while running a simulation or real-world system the following quantity:

$$\varepsilon_{i_0}^{\alpha}(\pi) := \frac{v_{i_0}^{\alpha}(\pi) - v_{i_0}^{\alpha}}{v_{i_0}^{\alpha}} \qquad \text{whenever } v_{i_0}^{\alpha} > 0, \tag{14}$$

where π is a particular policy for the considered MDP. The value $\varepsilon_{i_0}^{\alpha}(\pi)$ gives the relative increase of the total α-discounted expected cost for the initial state $i_0 \in \mathbb{S}$ when using policy π or action $\pi(i_0)$ instead of an optimal policy. Since it is generally impossible to compute the quantities defined in Equation (14) exactly, we aim at providing lower bounds. This requires an upper bound on the component $v_{i_0}^{\alpha}$ of the optimal value vector and a lower bound on $v_{i_0}^{\alpha}(\pi)$ or $v_{i_0}^{\alpha}(\pi(i_0))$, respectively, which are all obtained by our approximation algorithm.

The evaluation figures are arranged as follows. One chart may show for one particular state i_0, the approximation progress of

- an optimal policy: $v_{i_0}^{\alpha}$ and
- a concrete policy π: $v_{i_0}^{\alpha}(\pi)$.

In the following we will refer to the values $v_{i_0}^{\alpha}$ and $v_{i_0}^{\alpha}(\pi)$ simply as the *optimal cost* and the *cost of policy* π, respectively.

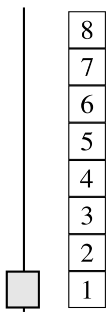
For each cost value reported, we depict the progress of lower and upper bounds computed in the approximation process depending on the number of explored states and generated variables, respectively. Additionally, we will provide the best obtained lower bounds on the value $\varepsilon_{i_0}^{\alpha}(\pi)$ for each analyzed policy π.

5.6. **Approximation Results for the Average Waiting Time.** For this test, we selected a discount factor of $\alpha = 0.8$ and the trivial initial state $i_2 = i_{\text{elv}}$[1] where no transport request is waiting and the elevator is situated at floor 1. The associated approximation results for the MDP are depicted in Figure 4. Obviously, none of the considered policies is really close to an optimal policy for the initial state i_2.

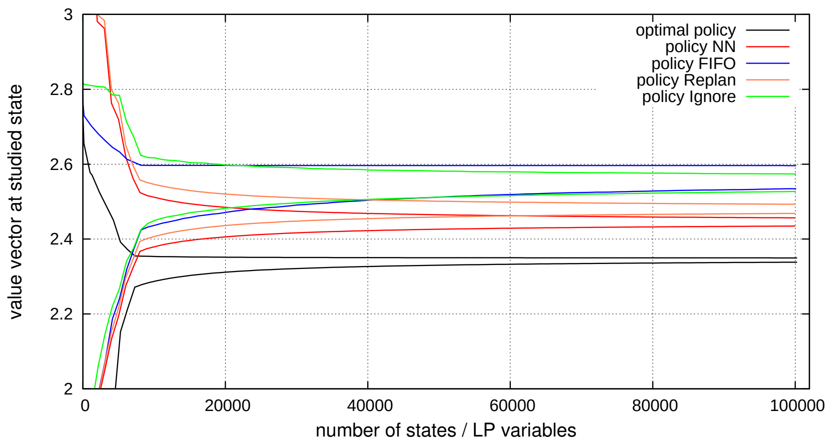
Observe the effectivity of the column generation: Our method proves that NN is not optimal using less than 10 000 states. It proves that NN is better than REPLAN using around 60 000 states. The proof that IGNORE and FIFO are worse than REPLAN takes around 50 000 states. After the generation of no more than 10 000 states, we know the cost of an optimal policy up to approximately 0.1, i.e., by then we have reached an accuracy of better than 5 %. Compared to this, the size of a static set of states determined by the formula of Theorem 3.17(i) for an approximation guarantee of 0.1 would be larger than the whole state space with 2 086 898 858 states.

The most interesting constructive observation we made relating to these results is the following: an optimal action at state i_2 is to move the elevator upwards. In the case no request is to be served by an elevator we face the task to position it such that future requests can be handled well. This issue is often referred to as the *parking policy* in the literature. Obviously, all of the considered policies do trivial parking, i. e., an elevator that is not dedicated to serve a request simply waits at its current floor. Our approximation method proves that this parking policy is not optimal for the state i_2. This result motivates to compare the actions WAIT, MOVE_DOWN, and MOVE_UP also for each state, where no request

[1]We chose the notation i_2 instead of i_0 to be consistent with the states considered in [Tuc10]



(a)



(b)

FIGURE 4. (a) Trivial state i_2 in an elevator control MDP w. r. t. the average waiting time for a single elevator and 8 floors. The initial state i_2 has no waiting requests and the elevator at Floor 1 (b) Approximation results for the elevator control MDP with 8 floors, one elevator, 4 waiting slots per floor and the special request distribution. We have $\varepsilon_{i_2}^{\alpha}(\mathsf{NN}) \geq 3.6\,\%$, $\varepsilon_{i_2}^{\alpha}(\mathsf{FIFO}) \geq 7.9\,\%$, $\varepsilon_{i_2}^{\alpha}(\mathsf{REPLAN}) \geq 5.1\,\%$, and $\varepsilon_{i_2}^{\alpha}(\mathsf{IGNORE}) \geq 7.5\,\%$.

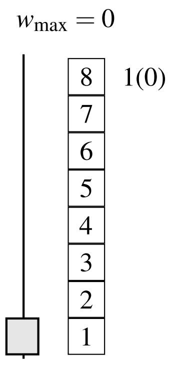
is waiting and the elevator is located at an arbitrary floor in $F \setminus \{1\}$. That is, for each state $i = ((\sigma_f)_{f \in F}, f_e, d_e)$ with $\sigma_f = \emptyset$ for each $f \in F$, $d_e = 0$, and arbitrary floor $f_e \in F$, we evaluate the total expected 0.8-discounted costs of all feasible actions. This way, we could determine a unique optimal action for each of these states according to the approach due to Corollary 4.7 on page 21. It turned out that the action WAIT is only optimal if the elevator is located at floor 6. Otherwise, moving the elevator closer to floor 6 can be proven to be optimal. Thus, we obtained an optimal parking policy for the corresponding MDP.

5.6.1. *Approximation Results for the Maximum Waiting Time.* Next, we analyze the performance of the policies NN, FIFO, REPLAN, and IGNORE when the objective is to minimize the maximum waiting time of a request. That is, we study the proposed elevator control MDP w. r. t. the maximum waiting time. We report on the results for the MDP with 8 floors, 1 elevator, infinity queuing capacity and the up-down traffic distribution from Table 2. The initial state i_1 has one waiting requests at Floor 8, a maximal waiting time of 0 so far, and the elevator in Floor 1. The discount factor α is set to 0.8 again.

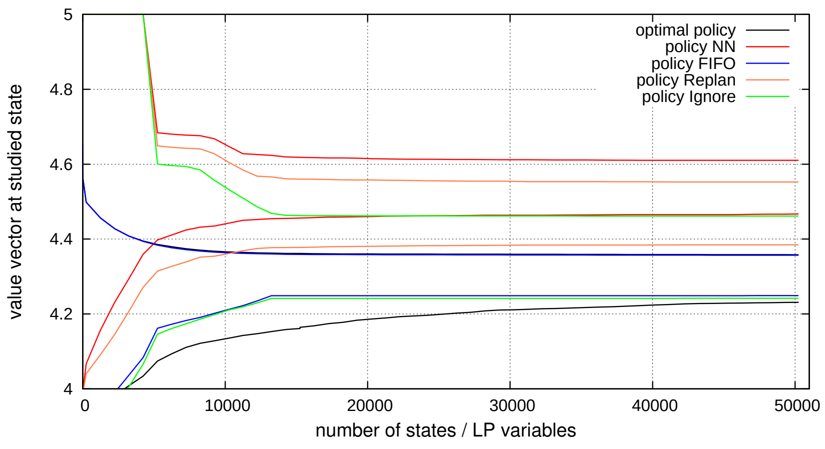
Figure 5 shows the associated results obtained by our approximation algorithm for the initial state i_1. Obviously, the policy NN performs badly, and is provably worse than FIFO and IGNORE. Moreover, the cost of REPLAN is shown to be greater than the cost of FIFO and the optimal cost.

Although the studies for average and maximum waiting time elevator control MDPs can only partitially reflect the behavior observed in simulations, we want to point out that our analysis provided useful information to improve existing online algorithms. For instance, let us consider the policy NN. Our results revealed that NN has the following weaknesses:

- NN does not employ a parking policy,
- its tie-breaking rule may lead to bad decisions, and
- the maximum waiting times achieved by NN are quite bad.



(a)



(b)

FIGURE 5. (a) State i_1 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. The initial state i_1 has one waiting requests at Floor 8 with destination at Floor 1, zero waiting time so far, and the elevator in Floor 1. (b) Approximation results for the elevator control MDP with 8 floors, 1 elevator, infinity queuing capacity, and up-down traffic. We have $\varepsilon_{i_1}^{\alpha}(\mathsf{NN}) \geq 2.5\,\%$, $\varepsilon_{i_1}^{\alpha}(\mathsf{FIFO}) \geq 0\,\%$, $\varepsilon_{i_1}^{\alpha}(\mathsf{REPLAN}) \geq 0.6\,\%$, and $\varepsilon_{i_1}^{\alpha}(\mathsf{IGNORE}) \geq 0\,\%$.

Due to these observations, we define the policy NNPARK-f as the following modification of NN:

- If the elevator is empty and there does not exist a waiting request, move the elevator towards floor f.
- If the elevator is empty and the nearest waiting request is not unique, serve that request with the greater waiting time first.

In order to focus even more on good maximum waiting times, we propose the following extension of NNPARK-f: if the elevator is empty and there exists a waiting request whose current waiting time equals the maximum waiting so far, this request is served next ignoring all other requests. We denote this online algorithm by NNMAXPARK-f.

We assess by simulation whether these modifications of NN are advantageous for the long-term behavior of the policy. The system defined by the Markov decision process cconsists of one elevator, eight floors, and the objective is to minimize the maximum waiting time, i. e., the queue length is infinite: We simulated for 10 000 time steps and compute average values for the observed average and maximum waiting times for 100 simulation runs. Table 4 shows simulation results for two Markov decision processes featuring a probability of $p^r = 0.1$ for the arrival of a new request at a time slot (this generates quite a high load). Obviously, NNPARK-f improves over NN for both, average and the maximum waiting times. Moreover, the NNMAXPARK-f achieves by far the best maximum waiting times, while the average waiting times are similar to those of the originial online algorithm NN, but inferior compared to NNPARK-f.

5.7. **Limitations.** The weakness of our tool in the elevator control problem is the following: since a relatively small discounting factor of 0.8 was necessary to reach conclusive results in the computations, long-term effects that, e.g., would rule out FIFO as an efficient policy

TABLE 4. Average value for the average and maximum waiting times achieved by the online algorithms NN and its variants NNPARK-f and NN-MAXPARK-f according to 100 simulation runs for 10 000 time steps. The parking floor is chosen to be $f = 6$ for the MDP with start-to-destination probability distributions `sp` (see Table 3) and $f = 1$ for `ud` (see Table 2).

Probability	NN		NNPARK-f		NNMAXPARK-f	
distribution	avg.	max.	avg.	max.	avg.	max.
`sp`	13.66	116.15	13.34	113.38	13.69	98.76
`ud`	12.26	139.33	11.93	128.59	12.26	97.94

(compare [FR06]) cannot be detected. Maybe, the computation starting in a different start state (full system) can yield more information, but the general problem persists. In other words, further research is needed to capture long-term effects. So far, our method is only suited to assess the short-term performance issues of policies.

6. CONCLUSION

In this paper we presented a method to obtain performance guarantees for the expected total cost of policies in discounted Markov decision processes. We introduced a technique which is able to approximate for a given state of an MDP the discounted cost for a given policy, for an optimal policy, and for individual actions (assuming that in all other states an optimal action is chosen). We computed a tight bound on the number of states that is sufficient to achieve a prescribed approximation guarantee; a number independent of the size of the state space. To automatically exploit the special structures of individual instances we introduced a column generation algorithm that can in many cases obtain a performance guarantee using significantly fewer states than guaranteed by the general bound. In order to illustrate strengths (provable performance guarantees for specific instances) and weaknesses (assessment of long-term behaviour) of our approach, we thouroughly analyzed the elevator control problem. The key-learnings led to the design of two new policies with provable better performance in the considered states. This result was further confirmed by a simulation study that is independent of our tool.

We believe, that the ability of the tool to reveal weak spots (i.e., states in which decisions are far from optimal) of otherwise not-so-bad policies can help to selectively modify widely accepted policies in states in which they fail. And this without completely replacing a plausible and easy-to-implement decision rule by rules solely based on numerical calculations.

The tool introduce in this paper was also successfully applied in the analysis of online algorithms for the online target-date assignment problem and the online bincoloring problem, where the performance assessment of competitive analysis yields counter-intuitive results [Tuc10]. There, our tool could provide a performance analysis of online algorithms embedded in the MDP framework that are much more in line with intuition and experience from simulation.

In future research we plan to perform further significant performance assessments and policy improvements for problems in dynamic optimization and online optimization for which existing results are unsatisfactory.

REFERENCES

[BBS95] Andrew G. Barto, Steven J. Bradtke, and Satinder P. Singh. Learning to act using real-time dynamic programming. *Artificial Intelligence*, 72:81–138, 1995.

[Ber01] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1 and 2. Athena Scientific, Belmont, 2nd edition, 2001.

[BT96] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*, volume 1. Athena Scientific, Belmont, 1st edition, 1996.

[CB98] Robert H. Crites and Andrew G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2–3):235–262, 1998.

[DDS05] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors. *Column generation*. GERAD 25th anniversary series. Springer, 2005.

[d'E63] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963.

[dFV03] Daniela P. de Farias and Benjamin Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003.

[dFV04] Daniela P. de Farias and Benjamin Van Roy. On constraint sampling in the linear programming approach to approximate dynamic programming. *Mathematics of Operations Research*, 29(3):462–478, 2004.

[DKKN93] Thomas L. Dean, Leslie Pack Kaelbling, Jak Kirman, and Ann E. Nicholson. Planning with deadlines in stochastic domains. In *AAAI*, pages 574–579, 1993.

[FR06] Philipp Friese and Jörg Rambau. Online-optimization of a multi-elevator transport system with reoptimization algorithms based on set-partitioning models. *Discrete Appl. Math.*, 154(13):1908–1931, 2006. Also available as ZIB Report 05-03.

[FS02] Eugene A. Feinberg and Adam Shwartz, editors. *Handbook of Markov Decision Processes: Methods and Applications*. Kluwer Academic Publishers, 2002.

[GHKR99] Martin Grötschel, Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau. Simulation studies for the online dial-a-ride problem. Report 99–09, ZIB, 1999. `opus.kobv.de/zib/volltexte/1999/398/`.

[HKP+06] Stefan Heinz, Volker Kaibel, Matthias Peinhardt, Jörg Rambau, and Andreas Tuchscherer. LP-based local approximation for Markov decision problems. Report 06–20, ZIB, 2006. `opus.kobv.de/zib/volltexte/2006/914/`.

[KMN99] Michael J. Kearns, Yishay Mansour, and Andrew J. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. In *International Joint Conferences on Artificial Intelligence*, pages 1324–1331, 1999.

[Pow07] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley and Sons, Inc., Hoboken, New Jersey, 1st edition, 2007.

[Put05] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., Hoboken, New Jersey, 2nd edition, 2005.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1st edition, 1998.

[SS85] Paul J. Schweitzer and Abraham Seidmann. Generalized polynomial approximations in Markov decision processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985.

[Tuc10] Andreas Tuchscherer. *Local Evaluation of Policies for Discounted Markov Decision Problems*. PhD thesis, Technische Universität Berlin, 2010.