Local Evaluation of Policies for Discounted Markov Decision Problems

vorgelegt von

Dipl.-Math. oec. Andreas Tuchscherer

Von der Fakultät II – Mathematik und Naturwissenschaften
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften
– Dr. rer. nat. –

genehmigte Dissertation

Berichter:	Prof. Dr. Dr. h.c. mult. Martin Grötschel
	Prof. Dr. Jörg Rambau
Vorsitzender:	Prof. Dr. Fredi Tröltzsch

Tag der wissenschaftlichen Aussprache: 8. Dezember 2010

Berlin 2010
D 83

Zusammenfassung

Das Bestimmen realistischer Indikatoren für die Güte von Online-Algorithmen für gegebene Online-Optimierungsprobleme ist eine schwierige Aufgabe. Bisher übliche Ansätze, wie die kompetitive Analyse, weisen erhebliche Nachteile auf. Sofern stochastische Informationen über zukünftige Anfragen zur Verfügung stehen, könnten Markov-Entscheidungs-Probleme (MPDs) eine geeignete Alternative darstellen. Da jedoch die Anzahl an Zuständen in MDPs, die aus praktischen Anwendungen hervorgehen, üblicherweise exponentiell in den ursprünglichen Eingabeparametern ist, sind Standardverfahren zur Analyse von Strategien bzw. Online-Algorithmen nicht anwendbar.

In dieser Arbeit wird ein neues algorithmisches Verfahren zur lokalen Evaluierung der Güte von Strategien für diskontierte MDPs vorgestellt. Der Ansatz basiert auf einem Spaltengenerierungsalgorithums zur Approximation der gesamten erwarteten diskontierten Kosten einer unbekannten optimalen Strategie, einer gegebenen Strategie oder einer einzelnen Entscheidung. Der Algorithmus bestimmt eine ε-Approximation, indem lediglich ein relativ kleiner lokaler Teil des gesamten Zustandsraumes untersucht wird. Es wird gezeigt, dass die Anzahl der zur Approximation erforderlichen Zustände unabhängig von der Gesamtzahl an Zuständen ist. Die berechneten Approximationen sind normalerweise deutlich besser als die theoretische Schranken, die andere Ansätze liefern.

Neben dem Pricing-Problem wird die Struktur der linearen Programme untersucht, die in der Spaltengenerierung auftreten. Darüber hinaus werden verschiedene Erweiterungen des Algorithmus vorgeschlagen und analysiert. Diese zielen darauf ab, gute Approximationen schnell berechnen zu können.

Das Potential des Verfahrens wird beispielhaft anhand von diskontierten MDPs untersucht, die aus Online-Optimierungsproblemen hervorgehen. Bei diesen handelt es sich um das Bin-Coloring-Problem, ein Terminvergabe-Problem und ein Aufzugssteuerungs-Problem. Das Verfahren ist zumeist in der Lage, Indikatoren für die Güte von Online-Algorithmen zu bestimmen, die Beobachtungen aus Simulationen deutlich besser widerspiegeln als die kompetitive Analyse. Außerdem lassen sich Schwachstellen der betrachteten Online-Algorithmen aufdecken. Dadurch konnte ein neuer Online-Algorithmus für das Bin-Coloring-Problem entwickelt werden, der auch in Simulationen besser abschneidet als bisher existierende Algorithmen.

Abstract

Providing realistic performance indicators of online algorithms for a given online optimization problem is a difficult task in general. Due to significant drawbacks of other concepts like competitive analysis, Markov decision problems (MDPs) may yield an attractive alternative whenever reasonable stochastic information about future requests is available. However, the number of states in MDPs emerging from real applications is usually exponential in the original input parameters. Therefore, the standard methods for analyzing policies, i. e., online algorithms in our context, are infeasible.

In this thesis we propose a new computational tool to evaluate the behavior of policies for discounted MDPs locally, i. e., depending on a particular initial state. The method is based on a column generation algorithm for approximating the total expected discounted cost of an unknown optimal policy, a concrete policy, or a single action (which assumes actions at other states to be made according to an optimal policy). The algorithm determines an ε-approximation by inspecting only relatively small local parts of the total state space. We prove that the number of states required for providing the approximation is independent of the total number of states, which underlines the practicability of the algorithm. The approximations obtained by our algorithm are typically much better than the theoretical bounds obtained by other approaches.

We investigate the pricing problem and the structure of the linear programs encountered in the column generation. Moreover, we propose and analyze different extensions of the basic algorithm in order to achieve good approximations fast.

The potential of our analysis tool is exemplified for discounted MDPs emerging from different online optimization problems, namely online bin coloring, online target date assignment, and online elevator control. The results of the experiments are quite encouraging: our method is mostly capable to provide performance indicators for online algorithms that much better reflect observations made in simulations than competitive analysis does. Moreover, the analysis allows to reveal weaknesses of the considered online algorithms. This way, we developed a new online algorithm for the online bin coloring problem that outperforms existing ones in our analyses and simulations.

Acknowledgments

This thesis emerged from the projects "Combinatorial aspects of logistics" and "Stability, sensitivity, and robustness in combinatorial online-optimization" of the DFG Research Center Matheon. Firstly, I want to thank Martin Grötschel for giving me the opportunity to collaborate in these projects, for his support, and for the freedom he gave me concerning the main focus of my research. My thanks also go to Ralf Borndörfer for providing the general conditions for me at the Zuse Institute Berlin to finish my thesis.

Related to the considered subject, I especially thank Jörg Rambau for coming up with this exciting topic, his valuable advice, his encouragement, and much support with regard to contents. The cooperation with him was very pleasant, unfortunately it was intensive only for a short time since he moved to Bayreuth (and I was not willing to do so). First studies concerning *the local evaluation of policies for discounted Markov decision problems* have been done in collaboration with Jörg Rambau, Stefan Heinz, Volker Kaibel, and Matthias Peinhardt. I am grateful for the nice and fruitful joint research. Particularly, I thank Benjamin Hiller for many inspiring discussions, valuable feedback to my preliminary results, and some support concerning the use of LaTeX. I also thank Siarhei Makarevich who was a great help by implementing parts of the code for the proposed method and by providing many of the computational results and associated illustrations.

Moreover, I am indebted to my proofreaders Benjamin Hiller, Siarhei Makarevich, and Cornelia Tuchscherer. In particular, Benjamin's feedback and suggestions were very fruitful and helped me a lot to improve the presentation.

I thank my parents Cornelia and Wolfgang and my parents-in-law Brigitte and Karl-Heinz for all their support in the last months by often taking care of my son Rafael. At last, I have to mention two very important persons. Special thanks go to my wife Susanne for her patience and bearing the stressful time we recently had. And, I exceedingly thank Jesus Christ for making all of this possible.

Contents

Chapter 1

Introduction

Classical optimization assumes that all input data of the problem to be solved are at hand. In many real-life problems, however, some data may naturally not be available at the moments when decisions have to be made, but become known stepwise over time. This observation has prompted the research in *online optimization*.

In an *online optimization problem*, data arrive in a sequence called *request sequence*. Each time a new request appears, an irrevocable decision has to be made by an *online algorithm* based on the data of the past, but precise information about the future is unavailable. The goal is to make "good" decisions, where the quality of the decision depends on the information available so far, but also on data arriving in the future.

We focus on *combinatorial* online optimization problems which arise in many different settings, e. g., logistics and transportation, and are of high practical importance. Each decision comes along with a cost and we would like to design an online algorithm whose decisions lead to an overall cost which is as small as possible. Frequently, so-called *reoptimization algorithms* are employed: whenever a new request becomes known, an auxiliary offline optimization problem is solved, based on the data currently available. The solution that may be obtained by an exact or heuristic method is then used to control the system.

Often online algorithms, e. g., reoptimization algorithms, seem to perform well in practice or simulations and one would like to have a tool that provides theoretical evidence for these observations. A variety of performance measures for analyzing the quality of online algorithms has been proposed and applied in the literature. In the following, we will briefly sketch the most important ones.

Competitive Analysis The most popular concept called *competitive analysis* [ST85, BEY98, FW98] compares the considered online algorithm ALG with a hypothetical optimal offline algorithm OPT that knows the request sequence in advance and can process it at a minimum cost. For each request sequence σ, denote by $\mathsf{ALG}(\sigma)$ and $\mathsf{OPT}(\sigma)$ the cost of ALG and OPT, re-

spectively. An online algorithm ALG is called *c-competitive* for $c \geq 1$ if we have:

$$\mathsf{ALG}(\sigma) \leq c \cdot \mathsf{OPT}(\sigma) + b \tag{1.0.1}$$

for each request sequence σ and some $b \geq 0$. The *competitive ratio* of an online algorithm ALG is defined as the infimum of all $c \geq 1$ such that ALG is c-competitive. Moreover, ALG is called *competitive* if it is c-competitive for some $c \geq 1$ that is independent of the considered request sequence σ. One often considers the case $b = 0$ only.

Competitive analysis investigates how much the quality of an online algorithm degrades compared to an optimal offline algorithm in the worst-case, due to the lack of information about the future. Consequently, the competitive ratio is a very pessimistic measure for the performance of an online algorithm.

Due to the worst-case nature of competitive analysis different undesirable phenomena occur in the analysis for various online optimization problems, especially for such that are of practical relevance:

1. There does not exist a competitive online algorithm at all.
2. Many online algorithms that are apparently of different quality have the same competitive ratio.
3. A reasonable online algorithm that performs well in simulation experiments has a worse competitive ratio than an algorithm that obviously performs badly.

These issues show that competitive analysis can be very inappropriate to evaluate the typical performance of online algorithms.

Other Performance Measures Competitive analysis can be seen as a game between an *online player* and a *malicious adversary*. The online player applies a particular online algorithm in order to achieve a good competitive ratio, whereas the adversary aims to construct a request sequence to be processed by the online player such that the resulting competitive ratio is bad. In other words, the mentioned disadvantages of competitive analysis are due to the fact that the malicious adversary is too powerful.

Many concepts to reduce the power of the adversary have been proposed in the literature. The most direct approach is to somehow restrict the request sequences the adversary can produce, e. g., see [BIRS95, AFG05, Tor98]. Another possibility is to consider *randomized online algorithms* that are allowed to use random decisions for processing requests, proposed by Borodin et al. [BLS92]. The cost of a randomized algorithm for a given

sequence is a random variable, and we are interested in its expectation. Normally, the adversary knows the probability distribution used by the online player but cannot observe the actual outcome of the random experiments. As a consequence, he must choose the complete input sequence in advance. This type of adversary is called *oblivious adversary*, see [BEY98] for different types of adversaries. Replacing $\mathsf{ALG}(\sigma)$ by the expectation $\mathbb{E}[\mathsf{ALG}(\sigma)]$ in Equation (1.0.1) defines the competitive ratio of a randomized online algorithm ALG. Analyzing randomized online algorithms is typically more complicated than deterministic ones.

Another alternative is to consider request sequences generated randomly according to some probability distribution and to analyze the expected performance of an online algorithm. Obviously, such an *average-case analysis* requires some probabilistic assumptions on how the inputs look like. Typically, requests are chosen independently and identically distributed. In this context, the optimal cost OPT_n for a request sequence of length $n \in \mathbb{N}$ is a random variable, just as the cost ALG_n obtained by an online algorithm ALG. The expected performance of ALG can be evaluated by each of the ratios:

$$\mathbb{E}\left[\frac{\mathsf{ALG}_n}{\mathsf{OPT}_n}\right] \quad \text{or} \quad \frac{\mathbb{E}[\mathsf{ALG}_n]}{\mathbb{E}[\mathsf{OPT}_n]}.$$

See [SSS06, SS06] for discussions on the difference between the two measures.

Moreover, there are a lot of different approaches in this area, e. g., analyzing the asymptotic expected performance [CSHY80] and smoothed competitive analysis [BLMS$^+$06] that represents a compromise between worst-case and average-case performance. For more details on performance measures for online algorithms we refer to [Alb03, DLO05, BF07].

To summarize, there are many methods that allow to provide more realistic performance indicators for online algorithms than obtained by competitive analysis. On the negative side, these approaches are usually more complicated to apply and each type of average-case analysis requires probabilistic assumptions concerning future requests. That is, such methods are more restrictive concerning their application, which again justifies using competitive analysis. Generally, it is reasonable to claim that the theory developed for online optimization rarely provides good guidelines for the choice of online algorithms to be employed in practice. In order to realistically evaluate how an algorithm performs in practice it is often necessary to resort to a simulation of the system.

Markov Decision Processes In the presence of a probability distribution for future requests *Markov decision processes* [Put05, FS02, Ber01] can be

used as a model for online optimization problems (see Section 4.1 for different examples). A Markov decision process, also referred to as stochastic dynamic program or discrete-time stochastic control problem, is a model for sequential decision making when outcomes are uncertain but obey given probabilities. Such a system consists of *states*, *actions*, *stage costs*, and *transition probabilities*. At some point in time, a specific state of a system is observed. Based on this state, a decision must be made by choosing an action. This causes the system to randomly move into another state according to given transition probabilities and an associated stage cost is incurred.

Markov decision processes are theoretically well understood and for many suitable *objective criteria*[1] e. g., the total expected discounted cost or the average expected cost for an infinite horizon, an optimal policy can in principle be computed. However, Markov decision processes arising from real applications are usually of astronomic size, which in turn makes the standard computational methods infeasible.

Powell [Pow07] introduces the *three curses of dimensionality* that give rise to these intractable sizes. The first curse is as follows. A state i in a Markov decision process is usually represented by a vector of different state parameters, i. e., $i = (s_1, \ldots, s_n)$ for some $n \in \mathbb{N}$. If each component s_k for $k \in \{1, \ldots, n\}$ can take $L \in \mathbb{N}$ possible values, then we can have up to L^n states, which is exponential in n. Thus, the state space grows very quickly with growing number of state parameters. A similar behavior is often the case for the set of actions at a state and the set of possible states the system can move to due to using an action at some state. We will refer to these as the second and third curse of dimensionality, respectively. In Section 2.2.7, we will describe some approaches from the literature to cope with the curses of dimensionality.

Following the notation of Puterman [Put05], we will call a Markov decision process together with an objective criterion *Markov decision problem (MDP)*. We will only consider *discounted MDPs* where the objective criterion is to minimize the *total expected α-discounted cost* for some $\alpha \in [0, 1)$ over an infinite horizon, properly introduced in the next chapter.

[1] Most publications related to Markov decision processes speak of *optimality criteria*, whereas this term is used in mathematical programming to characterize optimal solutions w. r. t. a given *objective function*. Since we favor the latter notion but do necessarily consider functions, we will speak of objective criteria.

Contribution

In this thesis we propose a new computational tool for analyzing the expected quality of online algorithms based on discounted MDPs. The method aims at providing evidence for the performance of algorithms that brings theory more in line with the behavior observed in practice or simulation.

Note that the analysis concepts mentioned above indicate the quality of an online algorithm only by a single value. In contrast, our method computes an evaluation depending on a particular situation of the considered system. Translated into the language of discounted MDPs, we evaluate a concrete policy or a single action w. r. t. a given initial state by estimating how much the associated total expected discounted cost degrades compared to that caused by an optimal policy. By restricting to this local evaluation of policies our method is able to cope with an arbitrarily large or even infinite state space, eluding the first curse of dimensionality. As the approach is unable to properly deal with the other curses, we assume the number of actions at each state and the number of possible successor states to be small.

The backbone of our analysis tool is an approximation algorithm that determines continuously improving lower and upper bounds for the total expected discounted cost of an unknown optimal policy, a concrete policy, or a single action (which assumes actions at other states to be made according to an optimal policy). The algorithm is based on the classical linear programming formulation for computing the *optimal value vector* and an optimal policy for a discounted MDP. We modify the formulation by taking into account only a small local part S of the total state space $\mathbb{S}$. Our algorithm dynamically extends the considered subset of states S by means of a column generation procedure in order to improve the achieved approximation guarantee. The main reason for the practical efficiency of the algorithm seems to be the feature that the dynamic extension of the subset S is guided by the reduced profit structure of the current reduced linear program. We point out that the approximation algorithm fully relies on evaluating a discounted cost function: for analyzing other objective criteria, e. g., the *average expected cost per stage*, considering only local parts of the state space is completely useless.

On the theoretical side, we prove that for a given $\varepsilon > 0$ and an initial state $i_0 \in \mathbb{S}$ there always exists some subset of states $S \subseteq \mathbb{S}$ such that the associated restricted linear programs yield an ε-approximation of the optimal value vector at the state i_0, while the size of S is independent of the total number of states[2]. Instead, the size of S depends only on the local structure

[2]We already published a first version of this result in [HKP+06].

of the discounted MDP at state i_0. This result provides a kind of evidence for the practicability of our approximation algorithm. We give an involved example proving that the algorithm may generate more states than required by the mentioned theoretical construction. In practice, however, the opposite is true: the approximation algorithm typically inspects substantially fewer states.

Besides a thorough investigation of the structure of the linear programs encountered in the column generation algorithm, we propose different pricing strategies, approximation heuristics, and a method for constructing initial bases, that may be employed in the algorithm. The practicability of the approximation algorithm strongly depends on these techniques. Among other things, we also derive a combinatorial interpretation and formula for the pricing problem and prove another upper bound construction based on the reduced profits.

The functionality offered by our approximation algorithm allows to

- provide computationally based approximations depending on the considered discounted MDP that tend to be much better than theoretical and thus worst-case bounds,
- assess the relative cost increase caused by using a concrete policy or action instead of an optimal policy for a given initial state,
- compute near-optimal actions,
- identify non-optimal actions and possibly an optimal one by excluding the former, and
- utilize the evidence about the impact of single decisions / actions in the design of online algorithm, e. g., a reoptimization algorithm.

The introduced analysis tool may be seen as a first step towards a method that can be incorporated in a simulation or real system in order to evaluate online each decision made. If a decision can be shown to be bad, an approach like ours might even propose a better one to the user. Taking into account the complex structures of practical environments and the typical real-time requirements, our approximation algorithm allows such an application so far only for very simple online optimization problems.

We apply our method to three online optimization problems, namely the online bin coloring problem [KdPSR08], an online elevator control problem [GHKR99], and an online target date assignment problem [HKM+05]. In each case, the analysis tool provides performance indicators for particular online algorithms that reflect much better the behavior observed in simulation

than competitive analysis does. Our results for the three online optimization problems include the following:

1. For the online bin coloring problem, we prove that the reasonable online algorithm GreedyFit clearly outperforms the totally stupid algorithm OneBin, whereas competitive analysis suggests the opposite. Based on the analysis for GreedyFit, we were able to develop the new online algorithm SafeBin that improves over GreedyFit. This could also be observed in simulations.

2. We mainly focus on scheduling a single elevator in the considered online elevator control problem. Although our tool is unable to assess the long-term behavior of online algorithms, we obtain some enlightening performance results for the simple online algorithms NearestNeighbor (NN) and FirstInFirstOut (FIFO) as well as different versions of the reoptimization-based algorithms Replan and Ignore. In particular, it turns out that NN performs very well concerning the average waiting time, while Ignore and FIFO give good results w. r. t. the maximum waiting time. Replan based on minimizing the sum of squared waiting times seems to be a good compromise to achieve a suitable behavior for both objectives simultaneously. The insight gained by our analysis allowed to design improved versions of NN that include a provably optimal strategy to position an idle elevator. Moreover, we provide some results for the case of a group of two elevators.

   In order to provide appropriate approximations for this problem, we had to develop involved theoretical bounds to estimate the impact of states not yet contained in the considered local subset.

3. For an online target date assignment problem we evaluate a slightly different cost function assuming a more reasonable date-wise discounting. Also in this case the reasonable online algorithm PFD which seems to be hard to analyze outperforms the 2-competitive algorithm PTD. For the standard version of the method using a discounted MDP this has already been shown in the diploma thesis of Heinz [Hei05] based on joint work.

Outline

This thesis is arranged as follows. Chapter 2 introduces the precise definition of a Markov decision process and Markov decision problem (MDP) together with related notations. For discounted MDPs, we highlight the main theoretical results and computational methods and describe the state-of-the-art

approaches to deal with the three curses of dimensionality. Moreover, we briefly sketch other objective criteria different from minimizing the total expected discounted cost and discuss their assets and drawbacks.

Chapter 3 is devoted to our approach for approximating the total expected discounted cost of an unknown optimal policy, a concrete policy, or a single action for a given initial state in a discounted MDP. The chapter develops the approximation algorithm and the mentioned local approximation theorem. Furthermore, we give various theoretical insights that are closely related to our method.

We describe in Chapter 4 how to generically formulate a Markov decision process model for a given online optimization problem in order to apply the approximation algorithm to evaluate online algorithms. Then we introduce the particular online optimization problems to be studied together with known results focusing on competitive analysis and the used Markov decision process models. After a computational analysis of the approximation algorithm and its possible components, we present our results for the considered online optimization problems. Finally, Chapter 5 is devoted to conclusions and an outlook.

Chapter 2

Markov Decision Problems

In this chapter, we give the formal definition of a Markov decision problem, for short MDP, and describe parts of the basic theory and computational methods that are relevant for the approximation algorithm we propose in this thesis. Section 2.1 defines Markov decision processes and Markov decision problems and introduces most of the notation used in the sequel. In Section 2.2 we present important theoretical results including the so-called optimality equations and describe the standard computational methods for discounted MDPs. Moreover, we give an overview about approaches for dealing with problems of large scale, due to suffering from the curses of dimensionality. Finally, Section 2.3 summarizes other objective criteria for MDPs, discusses briefly their advantages and disadvantages, and points out why our approach works only for the total expected discounted cost criterion.

For details about MDPs, we recommend the books of Puterman [Put05], Feinberg and Shwartz [FS02], and Bertsekas [Ber01].

2.1 Problem Definition

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.

As already mentioned in Chapter 1, we will use the term Markov decision problem for a Markov decision process together with an objective criterion. By this differentiation we follow the conception used by Puterman [Put05]. The notation in this thesis leans on the book of Feinberg and Shwartz [FS02]. Formally, a Markov decision process is defined as follows.

Definition 2.1.1 (Markov decision process) A *Markov decision process* is a tuple $(\mathbb{S}, \mathbb{A}, p, c)$, 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 $c_i(a) := \sum_{j \in \mathbb{S}} p_{ij}(a) c_i(a, j)$. △

The above definition of a Markov decision process reflects all the features relevant for this thesis. In the literature, often a more general definition is given. For instance, the possible actions, transition probabilities, and stage costs may depend on the *decision time point* t also called *stage index*. In our case, when these structures are independent of the stage index, the Markov decision process is often called *stationary*.

We mention that Puterman as well as Feinberg and Shwartz consider Markov decision processes with stage rewards instead of costs. Generally, both rewards and costs can be considered simultaneously by allowing negative values for the stage costs or rewards, respectively. In our context, however, it will be crucial that all expected stage costs are non-negative. Moreover, it is possible to generalize Definition 2.1.1 to infinite state spaces $\mathbb{S}$. The classical computational methods (described in Section 2.2 for the objective criterion of minimizing the total expected discounted cost), however, are infeasible for infinite state spaces. In contrast, the approximation method proposed in this thesis can cope with an infinite number of states. We will consider one Markov decision process with infinite state space in Sections 4.1.5 and 4.3.4.

Let us consider one of the classical examples for a Markov decision process, cf. [Ber01, volume 1, chapter 1].

Example 2.1.2 (Inventory control) This problem deals with managing the inventory of a certain item. At the start of each period the inventory manager observes the current stock and decides how many units to order. Orders are assumed to be delivered immediately, and there is a finite inventory capacity of C units. We also assume that the demands for each period are independent and identically distributed random variables with non-negative integer values. For each period, let $p(d)$ be the probability that

the demand has an amount of d units. We assume a maximum demand of D, i. e., $p(d) = 0$ for each demand $d > D$. If the demand exceeds the current inventory, then the shortage is backlogged in the next period and must be satisfied in that period also.

Let i be the inventory at the start of the current period, and let j be the inventory at the end of the period which equals the inventory at the start of the next period. If j is positive, then an inventory cost of $c_{\text{inv}}(j) > 0$ is incurred. If j is negative, then we have a backlogging cost of $c_{\text{back}}(j) > 0$. The cost for ordering a units equals $c_{\text{ord}}(a) \geq 0$.

The inventory control can be modeled by a Markov decision process as follows:

$$
\begin{aligned}
\mathbb{S} &= \{i \in \mathbb{Z} \mid -D \leq i \leq C\}, \\
\mathbb{A}(i) &= \{a \in \mathbb{N} \mid \max\{-i, 0\} \leq a \leq C - i\} && \forall i \in \mathbb{S}, \\
p_{ij}(a) &= \begin{cases} p(i + a - j), & \text{if } j \leq i + a \\ 0, & \text{if } j > i + a \end{cases} && \forall i, j \in \mathbb{S}\ \forall a \in \mathbb{A}(i), \\
c_i(a, j) &= c_{\text{ord}}(a) + \begin{cases} c_{\text{inv}}(j), & \text{if } j > 0 \\ c_{\text{back}}(j), & \text{if } j < 0 \\ 0, & \text{if } j = 0 \end{cases} && \forall i, j \in \mathbb{S}\ \forall a \in \mathbb{A}(i). \quad \triangle
\end{aligned}
$$

Further examples for Markov decision processes studied in this thesis are described in Section 4.1.

The following definition of a path is unusual in the literature but will come in handy later.

Definition 2.1.3 (Path) Given a Markov decision process $M = (\mathbb{S}, \mathbb{A}, p, c)$, a sequence $P = (i_1, a_1, i_2, a_2, \ldots, a_{n-1}, i_n)$ for $n \in \mathbb{N}$ of states $i_1, \ldots, i_n \in \mathbb{S}$ and actions with $a_k \in \mathbb{A}(i_k)$ and $p_{i_k i_{k+1}}(a_k) > 0$ for $k \in \{1, \ldots, n-1\}$ is called (i_1, i_n)-*path* or *path from* i_1 *to* i_n in M. The *length* $|P|$ of path P is defined as the number of state transitions in P, i. e., $|P| = n - 1$. Moreover, for states $i, j \in \mathbb{S}$, we say that

- j *is reachable* or *can be reached* from i if there exists an (i, j)-path in M, and
- j is a *successor (state)* of i and i is a *predecessor (state)* of j if j is reachable from i by a path of length 1.

For a particular state $i \in \mathbb{S}$, we will refer to the minimum length of a path from i to a state $j \in \mathbb{S}$ as the *depth* of j w. r. t. state i. $\triangle$

2.1.1 Policies

Controlling a Markov decision process is considered over a certain *planning horizon* which may be finite, infinite, or of random length. The method developed in this thesis assumes an infinite planning horizon. However, taking into account a finite planning horizon can easily be reduced to the case of an infinite horizon: the idea is to add a state with only one possible action leading to itself again with probability 1 at zero cost.

The goal is to determine a *policy* that achieves a best possible performance of the system w. r. t. some objective to be defined. To give the general definition of a policy, we need the notion of the *history* which incorporates all the traversed states and used actions up to some stage index.

Definition 2.1.4 (History) Given a Markov decision process $(\mathbb{S}, \mathbb{A}, p, c)$ and a stage index $t \in \mathbb{N}$, the *set of histories* H_t of the system up to stage index t is given by:

$$H_t := \{(i_1, a_1, \ldots, i_{t-1}, a_{t-1}, i_t) \mid (i_k, a_k) \in \mathbb{S} \times \mathbb{A}(i_k), 1 \leq k \leq t-1, i_t \in \mathbb{S}\}.$$

We will also refer to each element of H_t as a *history*. △

Note that in the definition of a history $(i_1, a_1, \ldots, i_{t-1}, a_{t-1}, i_t)$ it is not assumed that the sequence is a path. That is, the states in the sequence are not required to be reachable from their predecessors using the corresponding action, i. e., for each $k \in \{1, \ldots, t-1\}$, we may have $p_{i_k i_{k+1}}(a_k) = 0$.

A randomized policy is given by a collection of probability distributions for the actions to be used.

Definition 2.1.5 (Policy) Let $M = (\mathbb{S}, \mathbb{A}, p, c)$ be a Markov decision process. A *(randomized) policy* for M is a sequence $(\pi^1, \pi^2, \ldots, \pi^t, \ldots)$ with the following properties for each stage index $t \in \mathbb{N}$:

1. $\pi^t_{h_t a_t} \geq 0$ for each $h_t \in H_t$ and each $a_t \in \mathbb{A}(i_t)$.
2. $\sum_{a_t \in \mathbb{A}(i_t)} \pi^t_{h_t a_t} = 1$ for each $h_t \in H_t$.

$\pi^t_{h_t a_t}$ gives the probability that action a_t is chosen at stage index t when the history is h_t. We denote the set of policies for M by P_M. △

By definition a randomized policy is a decision rule that may depend on the current stage index as well as the complete history up to that stage. In the sequel we will mostly consider policies with special properties. A memoryless policy does not depend on the history except for the current state and the stage index. Moreover, a memoryless policy is called stationary if it only

depends on the current state, but is independent of the stage index. Finally, a stationary policy that is not randomized is called deterministic, i. e., it assigns to each state exactly one possible action.

Definition 2.1.6 Let $M = (\mathbb{S}, \mathbb{A}, p, c)$ be a Markov decision process, and let $\Pi = (\pi^1, \pi^2, \dots, \pi^t, \dots)$ be a policy for M.

1. Π is called *memoryless* if π^t is independent of $(i_1, a_1, \dots, i_{t-1}, a_{t-1})$ for each stage index $t \in \mathbb{N}$ and each history $(i_1, a_1, \dots, i_{t-1}, a_{t-1}, i_t) \in H_t$.

2. Π is called *stationary* if Π is memoryless and independent of the stage index t, i. e., $\pi^{t_1} = \pi^{t_2}$ for all $t_1, t_2 \in \mathbb{N}$. We denote a stationary policy $(\pi, \pi, \dots)$ simply by π and use the notation π_{ia} instead of $\pi^t_{h_t a_t}$.

3. A stationary policy π for M is called *deterministic* if for each $i \in \mathbb{S}$ there exists an action $a_i \in \mathbb{A}(i)$ with $\pi_{ia_i} = 1$. That is, we have $\pi_{ia} = 0$ for each $a \in \mathbb{A}(i) \setminus \{a_i\}$. We will use the notation $\pi(i) = a_i$. △

Note that since the sets $\mathbb{S}$ and $\mathbb{A}(i)$ for each $i \in \mathbb{S}$ of a given Markov decision process $(\mathbb{S}, \mathbb{A}, p, c)$ are finite by definition, the set of deterministic policies is always finite, too.

The performance of a policy is measured using a so-called *value vector function* that assigns to each policy a value, given the initial state of the process. Based on the value vector function an associated *objective criterion* can be defined. Naturally, there are various reasonable objective criteria that can be used to evaluate the performance of a given policy. Some of the these criteria featuring an infinite planning horizon, e. g., the total discounted expect cost, the total (undiscounted) expected cost, the expected average cost per stage, and Blackwell optimality will be introduced later in this chapter.

Definition 2.1.7 (Value vector) Let $M = (\mathbb{S}, \mathbb{A}, p, c)$ be a Markov decision process. A *value vector function* for M is a mapping $V \colon P_M \to \mathbb{R}^{\mathbb{S}}$ with $\Pi \mapsto v(\Pi)$ that maps a policy Π to a vector $v(\Pi) \in \mathbb{R}^{\mathbb{S}}$ which is called *value vector* of policy Π. The *optimal value vector* $v \in \mathbb{R}^{\mathbb{S}}$ w. r. t. V is defined by:

$$v_i := \inf_{\Pi \in P_M} v_i(\Pi), \quad \text{for each } i \in \mathbb{S}.$$

Any policy Π with $v(\Pi) = v$ is called *optimal* for V. △

For each state $i \in \mathbb{S}$, the number $v_i(\Pi)$ defines a valuation for the policy Π, given that the initial state is i. Of course, it is not clear under which conditions an optimal policy exists: the infimum has to be attained simultaneously for all initial states.

2.1.2 Markov Decision Problems

Having a value vector function at hand, we are now ready to define a Markov decision problem.

Definition 2.1.8 (Markov decision problem) Let $M = (\mathbb{S}, \mathbb{A}, p, c)$ be a Markov decision process and let $V\colon P_M \to \mathbb{R}^{\mathbb{S}}$ be a value vector function for M. Then the pair (M, V) is called *Markov decision problem*, MDP for short. We will also denote the corresponding MDP by $(\mathbb{S}, \mathbb{A}, p, c, V)$. △

In our context, we will only look at so-called discounted MDPs, where the value vector of a policy is defined in terms of the total expected discounted cost. To define this value, we use the following notation. 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 $\mathbb{E}_{i\Pi}$.

Definition 2.1.9 (Discounted MDP) 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

$$\begin{aligned} v_i^\alpha(\Pi) &:= \sum_{t=0}^{\infty} \mathbb{E}_{i\Pi}[\alpha^t \cdot c_{X_t}(Y_t)] \\ &= \sum_{t=0}^{\infty} \alpha^t \sum_{j \in \mathbb{S}} \sum_{a \in \mathbb{A}(j)} \mathbb{P}_{i\Pi}[X_t = j, Y_t = a] \cdot c_j(a). \end{aligned} \tag{2.1.1}$$

Let V^α 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 MDP $(M, \alpha) := (M, V^\alpha)$ is called *discounted MDP* with *discount factor* α or *α-discounted MDP*. To prevent ambiguity when we consider several discounted MDPs at the same time, we will sometimes denote the value vector of a policy $\Pi \in P_M$ also by $v_M^\alpha(\Pi)$. △

For the total expected α-discounted cost of a policy, the incurred stage cost after t steps are accounted with factor α^t only. Thus, the discount factor α adjusts to what extend possible future costs are taken into account. One could also say that α determines the tradeoff between now and later. On the one hand, a discount factor of $\alpha = 0$ only takes into account the expected stage cost of the first transition and completely ignores any cost incurred by

later transitions. On the other hand, a discount factor close to one substantially considers possible costs in the remote future, too.

Puterman [Put05, chapter 5.3] mentions another way to interpret discounting. Consider a deterministic policy π and some initial state $i \in \mathbb{S}$. One can show that the component $v_i^\alpha(\pi)$ of the value vector of π equals the total (undiscounted) expected cost of policy π for the initial state i w. r. t. a planning horizon of random length that is geometrically distributed with parameter α. For a short survey on objective criteria different from the total expected discounted cost see Section 2.3.

2.2 Results and Computational Methods for Discounted MDPs

In this section we summarize some of the most important results for discounted MDPs and present the classical computational algorithms for their solution. All of these are highly relevant in our context. With a few exceptions we refrain from giving proofs, but provide references to the literature.

First let us argue that the total expected α-discounted cost of a policy equals the *expected total α-discounted cost*. Let $c_{\max} := \max_{i\in\mathbb{S}, a\in\mathbb{A}(i)} c_i(a)$ be the maximum stage cost. Obviously, we have:

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

Hence, the theorem of dominated convergence, e. g., see [Bau01, chapter 2], implies:

$$\mathbb{E}_{i\Pi}\left[\sum_{t=0}^{\infty} \alpha^t \cdot c_{X_t}(Y_t)\right] = \sum_{t=0}^{\infty} \mathbb{E}_{i\Pi}[\alpha^t \cdot c_{X_t}(Y_t)] = v_i^\alpha(\Pi),$$

i. e., the total expected α-discounted cost criterion and the expected total α-discounted cost criterion are equivalent.

2.2.1 Optimality Equations

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.

Theorem 2.2.1 ([Ber01, volume 1, chapter 7.3]) *Let $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ 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 of the system of linear equations:*

$$x_i = c_i(\pi(i)) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i)) x_j, \quad i \in \mathbb{S}. \tag{2.2.1}$$

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

$$x_i = \min_{a \in \mathbb{A}(i)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) x_j \right\}, \quad i \in \mathbb{S}. \tag{2.2.2}$$

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

$$\pi(i) \in \operatorname*{argmin}_{a \in \mathbb{A}(i)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j^\alpha \right\}, \quad i \in \mathbb{S}. \tag{2.2.3}$$

The existence of an optimal deterministic policy motivates the following optimization problem.

Problem 2.2.2 (Discounted MDP) Given a discounted Markov decision problem $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, find an optimal deterministic policy. △

The practical impact of Theorem 2.2.1 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 (2.2.2) as the *optimality equations* or *Bellman equations*. Once the optimal value vector 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 v^α, and then use Equation (2.2.3) 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 2.2.3 (Optimal actions) Let $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ be an α-discounted MDP with $\alpha \in (0, 1)$. Each 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 $\pi(i) = a$. △

In the following, we will briefly outline the classical methods for computing the optimal value vector v^α of a discounted MDP including *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]. We will also look at the question concerning the complexity of Problem 2.2.2. Since we will only deal with deterministic policies in the following, we will simply speak of policies, meaning deterministic ones implicitly.

2.2.2 Value Iteration

The value iteration method successively approximates the optimal value vector v^α of a discounted MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ by a sequence $v^k \in \mathbb{R}^\mathbb{S}$ for $k = 1, 2, \ldots$ that converges to v^α. In doing so, the vector $v^1 \in \mathbb{R}^\mathbb{S}$ can be chosen arbitrarily. The sequence is constructed by applying the following mapping $T\colon \mathbb{R}^\mathbb{S} \to \mathbb{R}^\mathbb{S}$ with $x \mapsto Tx$ defined by:

$$(Tx)_i = \min_{a \in \mathbb{A}(i)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) x_j \right\}, \quad \text{for each } i \in \mathbb{S}.$$

Note that the optimality equations (2.2.2) which characterize the optimal value vector v^α given in Theorem 2.2.1 can be rewritten as follows:

$$x = Tx, \quad x \in \mathbb{R}^\mathbb{S}. \tag{2.2.4}$$

Thus, the optimal value vector v^α is the unique solution of Equation (2.2.4). In other words, v^α is the unique fixing point of the mapping T. Moreover, it can be shown that applying T onto an arbitrary start vector repeatingly, the resulting sequence converges to the optimal value vector v^α (T is a so-called *contraction mapping*).

Theorem 2.2.4 ([FS02, chapter 2.3]) *Given a discounted Markov decision problem $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ and any vector $x \in \mathbb{R}^\mathbb{S}$, we have $\lim_{k\to\infty} T^k x = v^\alpha$.*

Given an arbitrary vector $v^1 \in \mathbb{R}^\mathbb{S}$, the value iteration methods computes the sequence of vectors $v^k \in \mathbb{R}^\mathbb{S}$ for $k = 2, 3, \ldots$ as follows:

$$v^{k+1} = Tv^k, \quad \text{for } k = 1, 2, \ldots.$$

At the same time a sequence of policies converging to an optimal policy is constructed. Since this procedure is not finite in general, the value iteration method only guarantees for each $\varepsilon > 0$, the computation of an *ε-optimal*

Algorithm 1 Value iteration

1: **Input:** a discounted MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, $x \in \mathbb{R}^{\mathbb{S}}$, $\varepsilon > 0$
2: **Output:** an ε-optimal policy π for M, i. e., $\|v^\alpha - v^\alpha(\pi)\|_\infty \leq \varepsilon$, an $\varepsilon/2$-approximation y for v^α, i. e., $\|v^\alpha - y\|_\infty \leq \varepsilon/2$
3: compute vector $y \in \mathbb{R}^{\mathbb{S}}$ and policy π for each $i \in \mathbb{S}$ by:

$$y_i = \min_{a \in \mathbb{A}(i)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) x_j \right\}$$

$$\pi(i) = \operatorname*{argmin}_{a \in \mathbb{A}(i)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) x_j \right\}$$

4: **if** $\|y - x\|_\infty \leq \frac{(1-\alpha)\varepsilon}{2\alpha}$ **then**
5: **return** π, y
6: **else**
7: $x \leftarrow y$ and go to step 3
8: **end if**

policy π, i. e., $\|v^\alpha - v^\alpha(\pi)\|_\infty \leq \varepsilon$, and an $\varepsilon/2$-*approximation* y for v^α, i. e., $\|v^\alpha - y\|_\infty \leq \varepsilon/2$. The value iteration method is given in detail in Algorithm 1. By Theorem 2.2.4 it is clear that the value iteration method terminates after a finite number of iterations. We will not prove the correctness of the threshold value to terminate the algorithm in Step 4, which is a bit technical.

Theorem 2.2.5 ([FS02, chapter 2.3]) *The value iteration method given in Algorithm 1 is finite and correct.*

There are different types of improvements for the basic version of the value iteration method described above. For instance, suboptimality tests for excluding non-optimal actions can be incorporated. Moreover, there are variants of the standard value iteration method that are based on slightly modified contraction mappings called *Pre-Gauss-Seidel* and *Gauss-Seidel* method. These variants may be considered as an acceleration of the basic algorithm. For details, we refer to the mentioned books.

2.2.3 Policy Iteration

The policy iteration method constructs a finite sequence of deterministic policies $\pi^1, \pi^2, \ldots, \pi^n$, for some $n \in \mathbb{N}$, such that

$$v^\alpha(\pi^{k+1}) < v^\alpha(\pi^k), \quad \text{for } k = 1, 2, \ldots, n-1,$$

where $x < y$ for vectors $x, y \in \mathbb{R}^m$ means $x_i \leq y_i$ for each $i \in \{1, \ldots, m\}$ and $x_i < y_i$ for at least one $i \in \{1, \ldots, m\}$. Moreover, π^n is optimal. Thus, the generated policies improve step by step, finally reaching an optimal policy.

The way policies are constructed by the policy iteration algorithm is based on the following observation. For each state $i \in \mathbb{S}$ and each policy π, we define a subset of actions $A(i, \pi) \subseteq \mathbb{A}(i)$ by

$$A(i,\pi) = \left\{ a \in \mathbb{A}(i) \mid v_i^\alpha(\pi) > c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j^\alpha(\pi) \right\}. \tag{2.2.5}$$

It follows from Theorem 2.2.1, part 1 and 2, that π is optimal if and only if $A(i, \pi) = \emptyset$ for all states $i \in \mathbb{S}$. The idea of the policy iteration algorithm is to replace action $\pi(i)$ by some action $a \in A(i, \pi)$ to obtain an improved policy. Notice that again by Theorem 2.2.1, part 1, we always have $\pi(i) \notin A(i, \pi)$ for each state $i \in \mathbb{S}$. The following theorem shows the correctness of this approach.

Theorem 2.2.6 ([FS02, chapter 2.3]) *Let $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ be a discounted MDP and let π be a policy for M. Then, the following statements hold true.*

(i) If $A(i, \pi) = \emptyset$ for each state $i \in \mathbb{S}$, then π is optimal.

(ii) If $A(i, \pi) \neq \emptyset$ for some state $i \in \mathbb{S}$, let μ be any deterministic policy with $\mu \neq \pi$ such that

- *for each $i \in \mathbb{S}$ with $A(i, \pi) = \emptyset$, we have $\mu(i) = \pi(i)$ and*
- *for each $i \in \mathbb{S}$ with $A(i, \pi) \neq \emptyset$, we have either $\mu(i) = \pi(i)$ or $\mu(i) \in A(i, \pi)$.*

Then, we have $v^\alpha(\mu) < v^\alpha(\pi)$.

The policy iteration algorithm consists mainly of two steps. First, in the *policy evaluation step* the value vector $v^\alpha(\pi)$ for the current policy π is computed by solving the associated system of linear equations (2.2.1). Note that $v^\alpha(\pi)$ is required to determine the sets $A(i, \pi)$ for each state $i \in \mathbb{S}$. Then, an improved policy is constructed, or it is observed that the current policy is optimal. This operation is usually called *policy improvement step*. The policy iteration method in detail is given in Algorithm 2.

Note that Algorithm 2 terminates after a finite number of steps since by the second part of Theorem 2.2.6 the constructed policies improve in each iteration and the number of deterministic policies is finite. Moreover, the first part of Theorem 2.2.6 implies that the returned policy is optimal, which proves the correctness of the policy iteration method.

Algorithm 2 Policy iteration

1: **Input:** a discounted MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a policy π for M
2: **Output:** an optimal policy π for M
3: compute $v^\alpha(\pi)$ as the unique solution of the system of linear equations:

$$x_i = c_i(\pi(i)) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i)) x_j, \quad i \in \mathbb{S}$$

4: compute $A(i,\pi) = \{a \in \mathbb{A}(i) \mid v_i^\alpha(\pi) > c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j^\alpha(\pi)\}$
5: **if** $A(i,\pi) = \emptyset$ for each $i \in \mathbb{S}$ **then**
6: **return** π
7: **else**
8: take any policy $\mu \neq \pi$ with $\mu(i) \in A(i,\pi)$ for all $i \in \mathbb{S}$ with $\mu(i) \neq \pi(i)$
9: **end if**
10: $\pi \leftarrow \mu$ and go to step 3

Theorem 2.2.7 *The policy iteration method, given in Algorithm 2, is finite and correct.*

Similar to the value iteration method, one can also include a suboptimality test to get rid of some non-optimal actions in the policy iteration method. Moreover, there is a modified version of the standard algorithm that features a faster rate of convergence. For details we refer particularly to the book of Puterman [Put05, chapter 6].

2.2.4 Linear Programming

Since the approximation approach proposed in this thesis is based on the classical linear programming formulation for computing the optimal value vector, we will look at this method in more detail.

The definition of the mapping T in Section 2.2.2 implies that T is monotone: for given vectors $x, y \in \mathbb{R}^{\mathbb{S}}$ satisfying $x \leq y$, i. e., $x_i \leq y_i$ for each index $i \in \{1, \ldots, |\mathbb{S}|\}$, we have $Tx \leq Ty$. Let $v \in \mathbb{R}^{\mathbb{S}}$ be an arbitrary vector with the property $v \leq Tv$, i. e.,

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

This property is particularly satisfied for the zero vector $(0, \ldots, 0)^T$ since we have $c_i(a) \geq 0$ for each $i \in \mathbb{S}$ and each $a \in \mathbb{A}(i)$. The monotonicity of T implies $Tv \leq T^2 v$, which itself implies $T^2 v \leq T^3 v$, and so on. Combining

these inequalities we obtain:

$$v \leq Tv \leq T^2v \leq T^3v \leq \dots.$$

Since $\lim_{k\to\infty} T^k x = v^\alpha$ for each $x \in \mathbb{R}^{\mathbb{S}}$ by Theorem 2.2.4, altogether we have shown for any vector $v \in \mathbb{R}^{\mathbb{S}}$:

$$v \leq Tv \implies v \leq \lim_{k\to\infty} T^k v = v^\alpha.$$

Due to $v^\alpha = Tv^\alpha$, this gives the following result.

Lemma 2.2.8 *Let $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ be an α-discounted MDP. Then, for each vector $v \in \mathbb{R}^{\mathbb{S}}$ with $v \leq Tv$, we have $v \leq v^\alpha$. Thus, the optimal value vector v^α is the componentwise largest vector satisfying the inequality $v \leq Tv$.*

Of course, $v \leq Tv$ is equivalent to the system of inequalities:

$$v_i \leq c_i(a) + \alpha \sum_{j\in\mathbb{S}} p_{ij}(a) v_j \quad \forall i \in \mathbb{S}\ \forall a \in \mathbb{A}(i).$$

This leads to the central theorem concerning the linear programming method for computing the optimal value vector of a discounted MDP.

Theorem 2.2.9 *The optimal value vector $v^\alpha \in \mathbb{R}^{\mathbb{S}}$ of a discounted MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ equals the unique optimal solution of the following linear program:*

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

Proof. It is clear that the optimal value vector v^α is a feasible solution for the linear program (P). Moreover, by Lemma 2.2.8 we have $v^\alpha \geq v$ for each feasible solution $v \in \mathbb{R}^{\mathbb{S}}$. Thus, v^α is the unique optimal solution of (P). □

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

Although it is not relevant in this thesis, we mention that the unique optimal solution of the linear program (P) is in some sense independent of the objective function: Theorem 2.2.9 can obviously be generalized as follows.

Corollary 2.2.10 *Given a discounted MDP* $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$*, the optimal value vector* v^α *equals the unique optimal solution of the linear program:*

$$\begin{aligned} \max \ & \sum_{j \in \mathbb{S}} \beta_j v_j & & \\ \text{subject to } & v_i - \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j \leq c_i(a) & & \forall i \in \mathbb{S} \, \forall a \in \mathbb{A}(i) \\ & v_j \in \mathbb{R} & & \forall j \in \mathbb{S}, \end{aligned}$$

for any positive numbers $\beta_j > 0, j \in \mathbb{S}$*.*

Moreover, an optimal policy can be obtained from the optimal solution of the dual linear program to (P). Having introduced a dual price of u_{ia} for each constraint in the linear program (P), the associated dual linear program reads:

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

It is quite easy to prove the following properties of the linear program (D).

Theorem 2.2.11 ([FS02, chapter 2.3]) *Let* $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ *be an* α*-discounted MDP. Then, we have:*

1. *Any feasible solution* u *of* (D) *satisfies:*

$$\sum_{a \in \mathbb{A}(i)} u_{ia} \geq 1 \quad \text{for each } i \in \mathbb{S}.$$

2. *The linear program* (D) *is bounded and feasible.*

3. *Let* u^* *be an optimal solution of* (D)*, then any deterministic policy* π *for* M *with* $u^*_{i\pi(i)} > 0$ *for each* $i \in \mathbb{S}$ *is optimal and there exists at least one such policy.*

By the Theorems 2.2.9 and 2.2.11 the optimal value vector and an optimal policy for a discounted MDP can be computed using any linear programming solver (see Algorithm 3).

Moreover, it can also be shown that there is a one-to-one correspondence between the set of feasible solutions of the dual linear program (D) and the

Algorithm 3 Linear programming

1: **Input:** a discounted MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$
2: **Output:** the optimal value vector v^α and an optimal policy π for M
3: compute optimal solutions v^* and u^* of the linear programs (P) and (D) using a linear programming solver
4: determine any deterministic policy π with the property $u^*_{i\pi(i)} > 0$ for each $i \in \mathbb{S}$
5: **return** v^* and π

set of (randomized) stationary policies for the discounted MDP. This result is due to de Ghellinck and Eppen [dGE67]. In particular, each feasible basic solution of (D) corresponds to a deterministic policy and vice versa. This second part can be shown as follows. By Theorem 2.2.11 we know that each feasible solution u of (D) satisfies $\sum_{a\in\mathbb{A}(i)} u_{ia} \geq 1$ for each $i \in \mathbb{S}$, i. e., for each state at least one dual variable must be positive. Since each feasible basic solution of (D) consists of at most $|\mathbb{S}|$ non-zero values, there cannot exist two distinct actions $a_1, a_2 \in \mathbb{A}(i)$ such that $u_{ia_1}, u_{ia_2} > 0$ for some state $i \in \mathbb{S}$. Thus, for each $i \in \mathbb{S}$, there exists exactly one action $a_i \in A(i)$ such that $u_{ia} = 0$ for every $a \in \mathbb{A}(i) \setminus \{a_i\}$. This shows that each feasible basic solution corresponds to a deterministic policy.

The mapping now is as follows. On the one hand, given a feasible basic solution, the corresponding deterministic policy π is uniquely defined by $u_{i\pi(i)} > 0$. On the other hand, for a given deterministic policy the associated basic solution is defined analogously by the basis $\{(i, \pi(i)) \mid i \in \mathbb{S}\}$. These columns define indeed a basis of (D) for the following reason. The matrix of the resulting system of linear equations is *strictly column diagonally dominant*, i. e., the absolute value of each diagonal entry is greater than the sum of the absolute values of the other entries in the associated column. It is well known that such a matrix is nonsingular, e. g., see [Pla06]. The feasibility of the basic solution follows from the fact that all entries of the inverse of this matrix are non-negative (this general result is not shown here, see the proof of Theorem 3.5.3 for a similar argumentation). We summarize the results mentioned above in the following theorem.

Theorem 2.2.12 ([FS02, chapter 2.3]) *Let $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ be an α-discounted MDP. Then, there is a one-to-one correspondence between the set of feasible solution of the dual linear program* (D) *and the set of (randomized) stationary policies. Particularly, the set of feasible basic solutions of* (D) *corresponds to the set of deterministic policies where the mapping is*

as follows:

u feasible basic solution $\mapsto$ deterministic policy π_u with $u_{i\pi(i)} > 0$.
π deterministic policy $\mapsto$ basic solution u^π with basis $\{(i, \pi(i)) \mid i \in \mathbb{S}\}$.

Note that if there is only one optimal deterministic policy for a discounted MDP, then there exists also a unique optimal solution to (D) and vice versa. Moreover, it is clear from the observations above that there do not exist degenerate basic solutions of the dual linear program.

Corollary 2.2.13 *Given a discounted MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, each feasible basic solution u of the associated dual linear program* (D) *has the following property: for each state $i \in \mathbb{S}$, there exists an action $a_i \in \mathbb{A}(i)$ such that $u_{ia_i} > 0$ and $u_{ia} = 0$ for each $a \in \mathbb{A}(i) \setminus \{a_i\}$. That is, each feasible basic solution is non-degenerate.*

Another interesting point in this context is that the linear programming method using the simplex algorithm is in some way equivalent to the policy iteration method, which was first mentioned by de Ghellinck [dG60]. This can be seen as follows. Let us assume that we solve the dual linear program (D) by the simplex method and that we are given a feasible basic solution u for (D). Let π_u be the policy corresponding to solution u according to the mapping defined in Theorem 2.2.12 and let v be the associated primal solution to u. Note that we have $v_j = v_j^\alpha(\pi_u)$ for each $j \in \mathbb{S}$ since $u_{ia} = 0$ for each $i \in \mathbb{S}$ and each $a \in \mathbb{A}(i) \setminus \{\pi_u(i)\}$ implies by the complementary slackness theorem that v satisfies the system of linear equations (2.2.1). Therefore, the reduced cost $\bar{c}_{ia}$ of u_{ia} equals:

$$\bar{c}_{ia} = c_i(a) - v_i^\alpha(\pi_u) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) v_j^\alpha(\pi_u) \quad \text{for each } i \in \mathbb{S}, a \in \mathbb{A}(i).$$

Thus, a variable u_{ia} has a negative reduced cost if and only if a is contained in the set $A(i, \pi_u)$ defined by (2.2.5). This observation implies the equivalence between both methods.

On the one hand, consider one iteration of the standard simplex method where exactly one pivot step is executed. Let π_u be the associated policy before the pivot step. Making a non-basic variable $u_{i'a'}$ with $\bar{c}_{i'a'} < 0$ entering the basis, corresponds to changing the policy π_u into policy μ with $\mu(i') = a'$ and $\mu(i) = \pi_u(i)$ for every $i \in \mathbb{S} \setminus \{i'\}$. This policy update is of course also possible in the policy iteration algorithm as we have $a' \in A(i', \pi_u)$ since the reduced cost of $u_{i'a'}$ is negative. The observations above show that this specific version of the policy iteration algorithm resembles some kind of network simplex method.

On the other hand, in the update step of the policy iteration algorithm the action at several states may be changed. This corresponds to a linear programming algorithm where more than one pivot step is done in each iteration, which is called *block-pivoting simplex algorithm*, e. g., see [Pad99].

2.2.5 Directed Hypergraph Model Formulation

In this section, we give a graph theoretic interpretation of the problem to determine an optimal deterministic policy for a discounted MDP (Problem 2.2.2). The task of computing an optimal policy is formulated as a minimum cost flow problem in a directed hypergraph. Using a directed hypergraph model for a Markov decision process was firstly proposed by Nielsen and Kristensen [NK06] in the case of a finite planning horizon.

Definition 2.2.14 (Directed hypergraph) A *(directed) hypergraph* is a pair $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_1, v_2, \ldots, v_n\}$ for some $n \in \mathbb{N}$ is the set of nodes, and $\mathcal{E} = \{e_1, e_2, \ldots, e_m\}$ for some $m \in \mathbb{N}$ is the set of hyperarcs. A hyperarc $e \in \mathcal{E}$ is a pair $e = (t_e, H_e)$, where $t_e \in \mathcal{V}$ is the *tail* of e and $H_e \subseteq \mathcal{V}$ is the *head set* of e. △

We mention that the directed hypergraphs considered here present a special case of the general definition, where the tail of a hyperarc is also given by a set of nodes, e. g., see [GLNP93].

It is quite obvious that a Markov decision process can be represented by a weighted hypergraph. The following table shows the relations between the components of a Markov decision process and the components of the corresponding hypergraph:

Markov decision process	hypergraph
state	node
action	hyperarc
expected stage cost	hyperarc weight at tail node
transition probability	hyperarc weight at head node

In the sequel we will often use this hypergraph representation of a Markov decision process. For instance, consider the following simple Markov decision process $(\mathbb{S}, \mathbb{A}, p, c)$, where:

$$\begin{aligned}
&\mathbb{S} = \{i_1, i_2\},\\
&\mathbb{A}(i_1) = \{a_1, a_2\},\ \mathbb{A}(i_2) = \{a_3\},\\
&c_{i_1}(a_1) = 1,\ c_{i_1}(a_2) = 2,\ c_{i_2}(a_3) = 0,\\
&p_{i_1 i_1}(a_1) = 1,\ p_{i_1 i_1}(a_2) = \tfrac{1}{2},\ p_{i_1 i_2}(a_2) = \tfrac{1}{2},\ p_{i_2 i_2}(a_3) = 1.
\end{aligned}$$

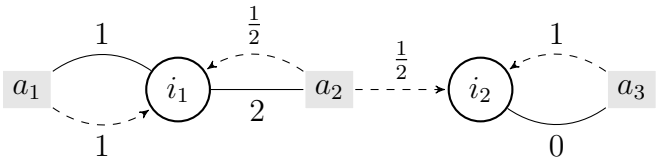


Figure 2.1: Directed hypergraph model of a simple Markov decision process. The states i_1 and i_2 are represented by nodes and the actions a_1, a_2, and a_3 are represented by hyperarcs also indicating expected stage costs and possible state transitions.

The corresponding weighted hypergraph is shown in Figure 2.1. Each hyperarc is depicted having a solid part leaving the tail node and a dashed part entering the head set. The former shows the expected stage cost of the associated action, while the latter represents the possible state transitions with corresponding probabilities.

Next we give the graph theoretic formulation of Problem 2.2.2 via a hypergraph minimum cost flow problem, which is generally defined as follows.

Problem 2.2.15 (Hypergraph minimum cost flow problem) Given a directed hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$, a positive *flow multiplier* $\mu_v(e) \in \mathbb{R}_+$ associated with each node $v \in H_e$ for each hyperarc $e \in \mathcal{E}$, and a real *demand / supply* $b(v) \in \mathbb{R}$ for each node $v \in \mathcal{V}$, a *flow* in $\mathcal{H}$ is a function $f\colon \mathcal{E} \to \mathbb{R}$ that satisfies the following *flow conservation constraints*:

$$\sum_{\substack{e\in\mathcal{E}:\\ v=t_e}} f(e) - \sum_{\substack{e\in\mathcal{E}:\\ v\in H_e}} \mu_v(e) f(e) = b(v) \quad \text{for each } v \in \mathcal{V}. \tag{2.2.6}$$

A flow is called *feasible* if $f(e) \geq 0$ for each hyperarc $e \in \mathcal{E}$. Let $\kappa(e)$ be a *hyperarc cost* associated with e for each $e \in \mathcal{E}$. Then the *hypergraph minimum cost flow problem* is to find a feasible flow in $\mathcal{H}$ that minimizes the function $\sum_{e\in\mathcal{E}} \kappa(e) f(e)$. △

It is straightforward to check that the linear program (D) describes the following hypergraph minimum cost flow problem.

Theorem 2.2.16 *Given a discounted MDP* $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$*, the linear program* (D) *describes the following instance of the hypergraph minimum cost flow problem with hypergraph* $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ *and parameters* μ*,* b*, and* κ*:*

- $\mathcal{V} = \mathbb{S}$*,*
- $\mathcal{E} = \{(i, N_i(a)) \in \mathbb{S} \times 2^{\mathcal{V}} \mid i \in \mathbb{S}, a \in \mathbb{A}(i)\}$*, where we define the set* $N_i(a) = \{j \in \mathbb{S} \mid p_{ij}(a) > 0\}$ *for each* $i \in \mathbb{S}$ *and each* $a \in \mathbb{A}(i)$*,*

- $\mu_j(i, N_i(a)) = \alpha p_{ij}(a)$ *for each* $(i, N_i(a)) \in \mathcal{E}$ *and each* $j \in N_i(a)$,
- $b(i) = 1$ *for each* $i \in \mathcal{V}$,
- $\kappa(i, N_i(a)) = c_i(a)$ *for each hyperarc* $(i, N_i(a)) \in \mathcal{E}$.

Thus, an optimal policy can also be computed by solving the hypergraph minimum cost flow problem constructed in Theorem 2.2.16. To the best of our knowledge, no combinatorial algorithms for solving Problem 2.2.15 have yet been proposed. In contrast to our situation, most publications in the literature consider the case where the head of each hyperarc consists of a single node only, whereas the tail may be an arbitrary set of nodes, e. g., see [CGS97, JMRW92].

2.2.6 Complexity

For an overview of the running times of the classical methods, i. e., value iteration, policy iteration, and linear programming, we refer to Ye [Ye05] (recall that value iteration determines only an ε-optimal policy). Moreover, this paper proposed the first strongly polynomial-time algorithm for computing an optimal policy in a discounted MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$. It requires at most $O(|\mathbb{S}|^{1.5}(\log 1/(1-\alpha) + \log|\mathbb{S}|))$ iterations of the classical interior-point method and at most $O(|\mathbb{S}|^4(\log 1/(1-\alpha) + \log|\mathbb{S}|))$ arithmetic operations.

What is most important in our context is the following. Independently of the used method, the complexity for computing an optimal policy always depends at least linearly on the number of states. However, by the first curse of dimensionality the size of the state space $\mathbb{S}$ is typically exponential in the common input parameters. Therefore, it is mostly infeasible in practice to determine the optimal value vector and an optimal policy, respectively, for all states in the discounted MDP.

This observation motivates the development of alternative approaches that in a certain way concentrate on reduced state spaces. The new approximation algorithm proposed in this thesis falls in this category, too.

2.2.7 Alternative Methods

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.

The main goal is to determine an effective (non-optimal) policy for, e. g., a discounted MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ with huge state space $\mathbb{S}$. This policy is typically obtained via an approximation of the optimal value vector v^α. Often the following *regression model* is utilized. For approximating v^α, one considers a parameterized class of vectors $\tilde{v}^\alpha \colon \mathbb{R}^k \to \mathbb{R}^\mathbb{S}$ for some $k \ll |\mathbb{S}|$ and aims to compute a parameter vector $r \in \mathbb{R}^k$ to fit v^α, i. e., $\tilde{v}^\alpha(r) \approx v^\alpha$. In doing so, the dimension is reduced from $|\mathbb{S}|$ down to k. Having determined the vector r, the *greedy policy* π w. r. t. $\tilde{v}^\alpha(r)$ is defined by:

$$\pi(i) \in \operatorname*{argmin}_{a \in \mathbb{A}(i)} \left\{ c_i(a) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(a) \tilde{v}^\alpha_j(r) \right\} \quad \text{for each } i \in \mathbb{S}.$$

The hope is that policy π performs well. In some special cases, bounds on the quality of π can be established [dFV03].

Most approximate dynamic programming methods are based on two elements. The first one is an *approximation architecture* that aims to be simple and to represent the optimal value vector well at the same time. For the described regression model the architecture is given by the definition of the function $\tilde{v}^\alpha$. A typical example is to fit the optimal value vector by a linear combination

$$\tilde{v}^\alpha(r) = \sum_{i=1}^{k} r_i \phi_i$$

of k so-called *base vectors*[1] $\phi_1, \ldots, \phi_k \in \mathbb{R}^\mathbb{S}$. We refer to the book of Bertsekas and Tsitsiklis [BT96, chapter 3] for an overview of common architectures.

The second element is a *training algorithm* also referred to as "optimizing simulator" to obtain a good or even optimal approximation within the underlying architecture. This algorithm is mostly similar to a classical simulation extended by some regression, adaption, or other learning mechanism. For instance, in the described regression model the training algorithm is used to update the parameter vector r. Most training algorithms are based on massive sampling of possible realizations and estimating the expectation (Monte Carlo simulation). Moreover, adapted versions of classical

[1] Contrary to our notation, value functions instead of value vectors are usually considered in approximate dynamic programming. Consequently, *base functions* are used in the mentioned architecture.

dynamic programming methods [SS85] and techniques including temporal-difference learning and Q-learning are employed. For instance, de Farias and Van Roy [dFV03, dFV04] employ a modification of the standard linear programming method to compute $\tilde{v}^\alpha(r)$ using the approximation architecture introduced above.

Additionally to the sketched approach, state aggregation [Van06, BC89] and decomposition techniques [MHK$^+$98] are often very useful. We refer to the books [Pow07, SB98, BT96] for details concerning approximate dynamic programming.

Note that the described approximate dynamic programming approach suffers from two types of errors even if a best possible approximation for v^α offered by the architecture can be computed. On the one hand, an approximation like $\tilde{v}^\alpha(r)$ may often be far away from the optimal value vector v^α due to the architecture applied. For instance, in the case of the described regression model, one generally cannot hope that v^α is contained in the span of the considered base vectors. We believe that for MDPs arising from combinatorial online optimization problems this error should often be significant as the structure of v^α will typically be quite unclear. On the other hand, taking the greedy policy w. r. t. the approximation causes another error. De Farias and Van Roy [dFV03] and Singh and Yee [SY94] show that a close approximation of v^α implies that the value vector of the associated greedy policy is also in some sense close to v^α, i. e., the policy performs close to an optimal one.

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, these techniques seem to be inappropriate for analyzing the quality of online algorithms.

Other methods slightly related to approximate dynamic programming that are more interesting in our context are the following. 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 3.2.7 on page 57. However, for the applications we aim at their bounds are weaker than ours. Furthermore, and maybe more importantly, their method does not seem to allow a modification that in practice visits substantially fewer states than guaranteed by the theoretical analysis, yet maintaining the approximation guarantee. In our view, this is a main advantage of the method that we propose (see the results in Section 4.2.1).

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.

2.3 Further Objective Criteria

In this section, we shortly outline further objective criteria featuring an infinite planning horizon that are different from the total expected discounted cost. Three important ones are the following:

1. the *total (undiscounted) expected cost*,
2. the *average expected cost per stage*, and
3. the *Blackwell optimality*.

In the following, let $(\mathbb{S}, \mathbb{A}, p, c)$ be a Markov decision process. We will again use the notation of Section 2.1.2: $\mathbb{P}_{i\Pi}[X_t = j, Y_t = a]$ denotes the probability that at stage t the state is j and the action is a, given that a (randomized) policy Π is used and the initial state is i. Moreover, the expectation operator w. r. t. this probability measure is denoted by $\mathbb{E}_{i\Pi}$.

Total Expected Cost

The total expected cost of a policy Π for an initial state $i \in \mathbb{S}$ is denoted by $v_i^1(\Pi)$ and given by Equation (2.1.1) for $\alpha = 1$, i. e.,

$$v_i^1(\Pi) := \lim_{T \to \infty} \sum_{t=0}^{T} \mathbb{E}_{i\Pi}[c_{X_t}(Y_t)]. \tag{2.3.1}$$

That is, the value $v_i^1(\Pi)$ may be seen as the total expected 1-discounted cost. However, it is clear that without further assumptions the limit in Equation (2.3.1) may be infinite or the upper limit may be different from the lower limit. The conditions to guarantee that the limit exists at least for one policy are quite restrictive. For instance, with one exception the total expected costs are not well-defined for the Markov decision processes considered in Chapter 4. For details on the required assumptions to apply the total expect cost criterion, we recommend the book of Puterman [Put05, chapter 7].

Average Expected Cost

In the average expected cost criterion, the limiting behavior of the term $1/T \sum_{t=0}^{T-1} \mathbb{E}_{i\Pi}[c_{X_t}(Y_t)]$ for $T \to \infty$ is considered for an initial state $i \in \mathbb{S}$ and a policy Π. In general, however, $\lim_{T\to\infty} 1/T \sum_{t=0}^{T-1} \mathbb{E}_{i\Pi}[c_{X_t}(Y_t)]$ may not exist and interchanging limit and expectation may not be allowed. Therefore, four different evaluation measures can be considered:

- The lower limit of the average expected cost per stage:

$$\phi_i(\Pi) = \liminf_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}_{i\Pi}[c_{X_t}(Y_t)].$$

- The upper limit of the average expected cost per stage:

$$\Phi_i(\Pi) = \limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} \mathbb{E}_{i\Pi}[c_{X_t}(Y_t)].$$

- The expectation of the lower limit of the average cost per stage:

$$\psi_i(\Pi) = \mathbb{E}_{i\Pi}\left[\liminf_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} c_{X_t}(Y_t)\right].$$

- The expectation of the upper limit of the average cost per stage:

$$\Psi_i(\Pi) = \mathbb{E}_{i\Pi}\left[\limsup_{T\to\infty} \frac{1}{T} \sum_{t=0}^{T-1} c_{X_t}(Y_t)\right].$$

For each of these criteria, the associated value vector and the concept of an optimal policy are defined in the same way as for the total expected discounted cost. It can be shown that for each of the four criteria, there exists an optimal deterministic policy. Interestingly, Bierth [Bie87] proved that all criteria are equivalent for stationary policies:

$$\phi(\pi) = \Phi(\pi) = \psi(\pi) = \Psi(\pi) \quad \text{for each stationary policy } \pi.$$

The major disadvantage of the average cost criteria is that all costs incurred in a finite number of stages are completely ignored. For instance, the two sequences of stage costs $0, 0, 0, \ldots$ and $100, 0, 0, \ldots$ both give an average expected cost of 0 and are thus measured the same although the first one should be preferred. Thus, high stage costs for a finite number of steps do not disqualify a policy of being optimal. Consequently, more sensitive criteria should be considered. One example is given in the next paragraph.

Blackwell Optimality

A policy Π is called *Blackwell optimal* if

$$v^{\alpha}(\Pi) = v^{\alpha} \quad \text{for each } \alpha \in [\alpha_0, 1) \text{ for some } 0 < \alpha_0 < 1.$$

That is, Π is Blackwell optimal if it is optimal w. r. t. the total expected discounted cost criterion for each α sufficiently close to 1. Blackwell [Bla62] proved the existence of a Blackwell optimal policy. Moreover, it can be shown that each Blackwell optimal policy is also optimal w. r. t. the average expected cost criteria. For more details concerning the different objective criteria related to the average expected cost, we refer to the books [Put05, chapter 8] and [FS02, chapter 2].

Practicability of Different Objective Criteria

In this thesis we concentrate on the total expected discounted cost criterion only. This is due to the following reasons.

Firstly, most of the alternative objective criteria have practical disadvantages. The total expected cost criterion is often simply not well-defined and thus cannot be used. Furthermore, the average expected cost criterion is unselective in the sense that there are usually a lot of optimal policies, which may differ substantially from the practical point of view. For instance, in the Markov decision process model we consider for the online bin coloring problem, see Section 4.1.3, the totally stupid online algorithm / policy OneBin is optimal w. r. t. the average expected cost. This drawback can be overcome by using more sensitive criteria like the concept of Blackwell optimality. Of course, the total expected discounted cost criterion may not completely satisfy the goals users have, either. However, this objective criterion offers a significant degree of flexibility due to the choice of the discount factor. For instance, by using a discount factor very close to 1, one approaches the Blackwell optimality criterion. In most practical cases one will usually be interested in a large discount factor, which accounts for decisions in later stages of the process, too. For most computational methods, however, the required running time increases with increasing discount factor. This in especially true for the approximation algorithm proposed in this thesis.

Secondly, the computational methods to determine an optimal policy w. r. t. the total expected discounted cost criterion are usually simpler and more efficient than those for alternative objective criteria. In particular, it is much harder to compute a Blackwell optimal policy. We do not go into details here.

Finally, we already mentioned that the approximation algorithm proposed in this thesis is essentially based on exploring smaller local parts of the total state space. For all presented objective criteria different from the total expected discounted cost, considering small subsets of states is in general completely useless since possible costs in the far future are just as relevant as those incurred very soon. Consequently, our approach is inappropriate for analyzing one of these objective criteria.

Chapter 3

LP-Based Local Approximation for Discounted Markov Decision Problems

In this chapter, we propose a new method for approximating the optimal value vector of a discounted MDP at single states. As we will only focus on discounted MDPs in the sequel, the term discounted will be omitted from now on, speaking simply of MDPs. Moreover, since we deal exclusively with deterministic policies in this thesis, we will only speak of policies in the following, always meaning deterministic ones. All results presented in this chapter are new.

The outline of the chapter is as follows. The construction of lower and upper bounds on one component of the optimal value vector is described in Section 3.1. In Section 3.2 we present our approximation theorem showing that an ε-approximation for the component can be obtained by taking into account only a local part of the state space whose size is independent of the total number of states in the MDP. Section 3.3 introduces the foundation of our approximation algorithm based on column generation, which is the main contribution of this thesis. In Section 3.4 we describe how the algorithm can be utilized to obtain also approximations for a concrete policy or a single action. Finally, Section 3.5 is devoted to several theoretical and practical issues related to the column generation algorithm. In particular, we investigate the structure of the encountered dual linear programs, derive a combinatorial formula for the reduced profits, and propose various pricing strategies and approximation heuristics. Moreover, we develop an equivalent variant of the approximation algorithm that refrains from using linear programming techniques, but employs the policy iteration method instead.

3.1 Approach

Our approximation approach for MDPs is based on the classical linear programming formulation as described in detail in Section 2.2.4. It has been shown that the optimal value vector v^α of an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ can be

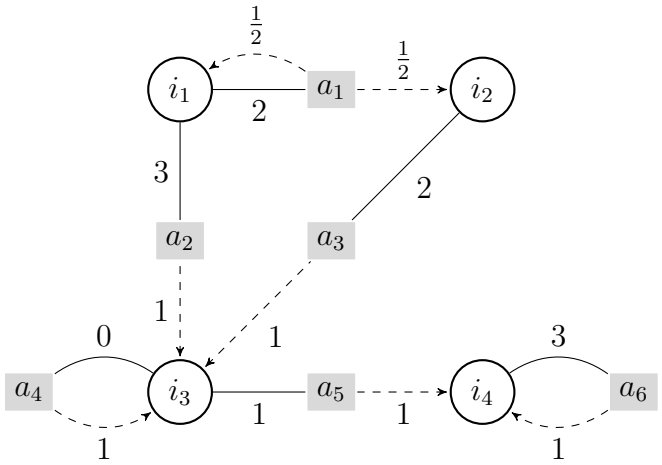


Figure 3.1: A Markov decision process represented by the directed hypergraph model as introduced in Section 2.2.5.

computed as the optimal solution of the linear program:

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

To achieve a consistent notation in this chapter, this linear program and its dual will be denoted by (P^{Σ}) and (D^{Σ}) from now on.

Example 3.1.1 As an example, consider the MDP given by the Markov decision process shown in Figure 3.1 and some discount factor $\alpha \in [0, 1)$. The corresponding linear program to compute the optimal value vector reads:

$$\begin{aligned} \max \quad & v_{i_1} + v_{i_2} + v_{i_3} + v_{i_4} && \\ \text{subject to} \quad & (1 - \tfrac{\alpha}{2}) v_{i_1} - \tfrac{\alpha}{2} v_{i_2} \leq 2 && (i_1, a_1) \\ & v_{i_1} - \alpha v_{i_3} \leq 3 && (i_1, a_2) \\ & v_{i_2} - \alpha v_{i_3} \leq 2 && (i_2, a_3) \\ & (1 - \alpha) v_{i_3} \leq 0 && (i_3, a_4) \\ & v_{i_3} - \alpha v_{i_4} \leq 1 && (i_3, a_5) \\ & (1 - \alpha) v_{i_4} \leq 3 && (i_4, a_6) \\ & v_{i_1}, v_{i_2}, v_{i_3}, v_{i_4} \in \mathbb{R}. && \end{aligned}$$

The unique optimal solution of this linear program is given by:

$$\begin{aligned} v_{i_1} &= \begin{cases} 3, & \text{if } \alpha > 2/5, \\ \frac{2(2+\alpha)}{2-\alpha}, & \text{if } \alpha \leq 2/5, \end{cases} \\ v_{i_2} &= 2, \\ v_{i_3} &= 0, \\ v_{i_4} &= \tfrac{3}{1-\alpha}. \end{aligned}$$ △

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 $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ be an 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^{Σ}) can be written as:

$$\begin{aligned} \max \ & \mathbb{1}^t v \\ \text{subject to } & Qv \leq c \\ & v \in \mathbb{R}^{\mathbb{S}}, \end{aligned} \tag{P^{Σ}}$$

where $\mathbb{1}^t = (1, 1, \ldots, 1)$ 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 impossible to solve the associated linear program (P^{Σ}). 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^{Σ}) by maximizing the value v_{i_0} only:

$$\begin{aligned} \max \ & v_{i_0} \\ \text{subject to } & Qv \leq c \\ & v \in \mathbb{R}^{\mathbb{S}} \end{aligned} \tag{P^{i_0}}$$

In contrast to (P^{Σ}), 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.

Example 3.1.2 As example, consider again the MDP given by the Markov decision process in Figure 3.1 and some $\alpha \in [0, 1)$. For the choice $i_0 = i_2$, the set of optimal solutions for the associated linear program (P^{i_0}) reads:

$$\left\{ (v_{i_1}, v_{i_2}, v_{i_3}, v_{i_4}) \in \mathbb{R}^4 \mid -\infty \leq v_{i_1} \leq \begin{cases} 3, & \text{if } \alpha > 2/5, \\ \frac{2(2+\alpha)}{2-\alpha}, & \text{if } \alpha \leq 2/5, \end{cases} \right.$$
$$v_{i_2} = 2,$$
$$v_{i_3} = 0,$$
$$\left. -\tfrac{1}{\alpha} \leq v_{i_4} \leq \tfrac{3}{1-\alpha} \right\}.$$

In this example, the optimal solution values for v_{i_1} and v_{i_4} are not unique since state i_1 cannot be reached from i_2, while state i_4 can only be reached from i_2 via i_3 using action a_5 which is not optimal. △

Moreover, notice that for the MDP considered in the examples above, the optimal solution of the linear program (P^{Σ}) is optimal and feasible for (P^{i_0}), too. We show that this fact is true in general by proving the following lemma which covers results about generic linear programs of the type (P^{Σ}) and (P^{i_0}). These issues are handled in detail as they will be applied to different cases later. To see the analogy between the linear programs above and those considered in Lemma 3.1.3, the values λ_{ija} and d_{ia} are to be interpreted as $\lambda_{ija} = p_{ij}(a)$ and $d_{ia} = c_i(a)$.

Lemma 3.1.3 *Let $n \in \mathbb{N}$ and let $a_i \in \mathbb{N}$ and $A(i) := \{1, \ldots, a_i\}$ for each integer $i \in [n] := \{1, \ldots, n\}$. Moreover, for all $i, j \in [n]$ and each $a \in A(i)$, let $\lambda_{ija} \geq 0$ with the property:*

$$\sum_{j=1}^{n} \lambda_{ija} \leq 1 \quad \textit{for each } i \in [n] \textit{ and } a \in A(i).$$

Let $m = |\{(i,a) \mid i \in [n], a \in A(i)\}|$ *be the number of possible pairs* (i,a) *and let* $\alpha \in [0,1)$. *Define the matrix* $B \in \mathbb{R}^{m \times n}$ *by:*

$$B_{(i,a),j} = \begin{cases} 1 - \alpha\lambda_{ija}, & \text{if } i = j, \\ -\alpha\lambda_{ija}, & \text{if } i \neq j, \end{cases}$$

For a given vector $d \in \mathbb{R}^m_{\geq 0}$ *with components* d_{ia} *for* $i \in [n], a \in A(i)$ *and any integer* $i_0 \in [n]$, *consider the following linear programs:*

$$\begin{aligned} \max \ & \mathbb{1}^t x \\ \text{subject to } & Bx \leq d \\ & x \in \mathbb{R}^n \end{aligned} \tag{LP$^\Sigma$}$$

and

$$\begin{aligned} \max \ & x_{i_0} \\ \text{subject to } & Bx \leq d \\ & x \in \mathbb{R}^n \end{aligned} \tag{LP^{i_0}}$$

that only differ in the objective function. Then, the following hold true:

1. *Both linear programs are feasible and bounded.*
2. *The optimal solution of* (LP$^\Sigma$) *is unique and also optimal for* (LP^{i_0}).
3. *If* $a_i = 1$ *for each* $i \in [n]$, *the optimal solution of* (LP$^\Sigma$) *can be computed as the unique solution of the system of linear equations:*

$$Bx = d. \tag{3.1.1}$$

Proof. Both linear programs are feasible since the right hand side of the linear programs is non-negative, i. e., $d_{ia} \geq 0$ for each $i \in [n]$ and each $a \in A(i)$. Hence, the zero vector is always a feasible solution. To prove that they are bounded, let $x \in \mathbb{R}^n$ be any feasible solution of the linear programs and let $R := \max\{x_j \mid j \in [n]\}$ be the maximum component of x. Moreover, denote the maximum of all values d_{ia} by $d_{\max} := \max\{d_{ia} \mid i \in [n], a \in A(i)\}$. For each $i \in [n]$ and each $a \in A(i)$, the constraints imply:

$$x_i \leq d_{ia} + \alpha \sum_{j=1}^{n} \lambda_{ija} x_j \leq d_{\max} + \alpha \sum_{j=1}^{n} \lambda_{ija} R \leq d_{\max} + \alpha R,$$

since $\lambda_{ija} \geq 0$ and $\sum_{j=1}^{n} \lambda_{ija} \leq 1$ for all $i, j \in [n]$ and each $a \in A(i)$. Therefore, we obtain:

$$R \leq d_{\max} + \alpha R \implies R \leq \frac{d_{\max}}{1-\alpha},$$

i. e., each component of x is bounded by $d_{\max}/(1-\alpha)$. Since x is an arbitrary feasible solution, this implies that both linear programs are bounded.

For the rest of the proof we need following claim.

Claim *Given two feasible solutions $x', x'' \in \mathbb{R}^n$ of (LP^{Σ}) or (LP^{i_0}), respectively, their componentwise maximum $x \in \mathbb{R}^n$, i. e., $x_j = \max\{x'_j, x''_j\}$ for each $j \in [n]$, is a feasible solution for both linear programs as well.*

The claim can be shown as follows. For each $i \in [n]$ and each $a \in A(i)$, we have:

$$\begin{aligned}
x_i &= \max\{x'_i, x''_i\} \\
&\leq \max\left\{ d_{ia} + \alpha \sum_{j=1}^{n} \lambda_{ija} x'_j, d_{ia} + \alpha \sum_{j=1}^{n} \lambda_{ija} x''_j \right\} \\
&= d_{ia} + \max\left\{ \underbrace{\alpha \sum_{j=1}^{n} \lambda_{ija} x'_j}_{\leq \alpha \sum_{j=1}^{n} \lambda_{ija} x_j}, \underbrace{\alpha \sum_{j=1}^{n} \lambda_{ija} x''_j}_{\leq \alpha \sum_{j=1}^{n} \lambda_{ija} x_j} \right\} \\
&\leq d_{ia} + \alpha \sum_{j=1}^{n} \lambda_{ija} x_j,
\end{aligned}$$

which proves the claim. Now it is easy to show that there exists a unique optimal solution for the linear program (LP^{Σ}). Assume that there exist two different optimal solutions $x^*, y^* \in \mathbb{R}^n$ for (LP^{Σ}). Then, their componentwise maximum is a better solution that is also feasible due to the claim, which yields a contradiction.

Next assume that the optimal solution x^* for (LP^{Σ}) is not optimal for the linear program (LP^{i_0}), and let $y^* \in \mathbb{R}^n$ be an optimal solution for (LP^{i_0}). Thus, we have $y^*_{i_0} > x^*_{i_0}$. Then again, the componentwise maximum of x^* and y^* has a better objective value than x^* w. r. t. (LP^{Σ}) and is feasible, too. This contradiction shows that the optimal solution for (LP^{Σ}) is also optimal for (LP^{i_0}).

Finally, consider the case $a_i = 1$ for each $i \in [n]$. On the one hand, the constraint matrix B is strictly row diagonally dominant and therefore nonsingular, e. g., see [Pla06]. Thus, the system of linear equations (3.1.1) has

exactly one solution. On the other hand, note that the optimal solution for the linear program (LP^{Σ}) satisfies all constraints with equality since otherwise the solution can be improved. Hence, both solutions are identical. □

Clearly, Lemma 3.1.3 implies that the above observation concerning the linear programs (P^{Σ}) and (P^{i_0}) is true in general.

Corollary 3.1.4 *Given an MDP* $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ *and a state* $i_0 \in \mathbb{S}$*, the optimal solution for* (P^{Σ}) *is also an optimal solution for* (P^{i_0})*. That is, the optimal value of* (P^{i_0}) *equals* $v_{i_0}^{\alpha}$*.*

For a study concerning the structure of the feasible basic solutions of the dual linear program to (P^{i_0}), see Section 3.5.2. In the following we describe our approach to compute lower bounds on the value $v_{i_0}^{\alpha}$.

3.1.1 Lower Bounds

Similar to the original linear programming formulation, solving the linear program:

$$\begin{aligned} \max \ & v_{i_0} \\ \text{subject to } & Qv \leq c \\ & v \in \mathbb{R}^{\mathbb{S}} \end{aligned} \tag{P^{i_0}}$$

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:

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

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

$$\begin{aligned} \max \ & \mathbb{1}^t v \\ \text{subject to } & Q^S v \leq c^S \\ & v \in \mathbb{R}^S, \end{aligned} \tag{L_S^{Σ}}$$

where again $\mathbb{1}^t = (1, 1, \ldots, 1)$ denotes the all-ones vector. Similar as before, Lemma 3.1.3 implies the following results.

Corollary 3.1.5 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, then both linear programs $(\mathrm{L}_S^{i_0})$ and (L_S^{Σ}) are feasible and bounded. Moreover, the optimal solution of (L_S^{Σ}) is unique and also an optimal solution to $(\mathrm{L}_S^{i_0})$.*

The more important observation, however, is as follows. Any feasible solution $\underline{v} \in \mathbb{R}^S$ of the linear program (L_S^{Σ}) and $(\mathrm{L}_S^{i_0})$ can be extended to a feasible solution $\underline{v}^{\mathrm{ext}} \in \mathbb{R}^{\mathbb{S}}$ of the linear program (P^{Σ}) and (P^{i_0}) with the same objective value, respectively, where

$$\underline{v}_j^{\mathrm{ext}} = \begin{cases} \underline{v}_j, & \text{if } j \in S, \\ 0, & \text{if } j \in \mathbb{S} \setminus S. \end{cases} \tag{3.1.2}$$

Recall that by Lemma 2.2.8 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 3.1.6 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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 is a 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 thesis 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 $i_0 \in S$. Obviously, by Corollary 3.1.5 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 3.1.7 (Lower-bound induced MDP) Let $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ be an MDP and let $S \subseteq \mathbb{S}$ be any subset of states. Then, the *(lower-bound) S-induced MDP* $M(S) = (\mathbb{S}', \mathbb{A}', p', c', \alpha)$ 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 3.2. Induced MDPs have the following properties.

Theorem 3.1.8 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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 $M(S) = (\mathbb{S}', \mathbb{A}', p', c', \alpha)$:*

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

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 of $M(S)$ is given by the unique optimal solution of the linear program (L_S^{Σ}) and $v^{\alpha}_{i_{\text{end}}} = 0$.*

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

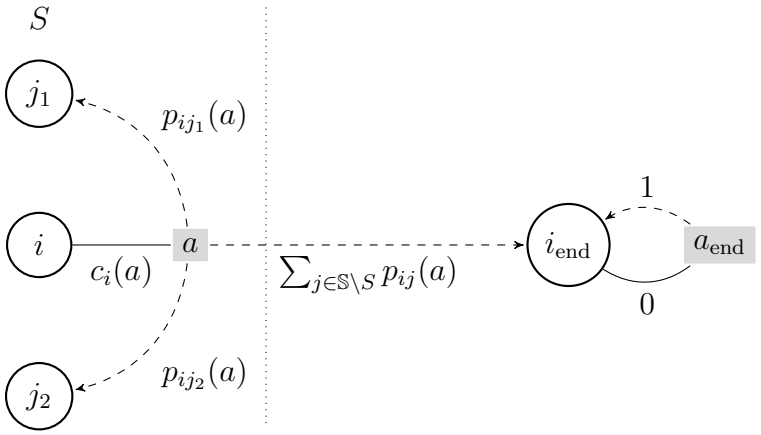


Figure 3.2: Illustration of the Markov decision process of the induced MDP $M(S)$ for some $S \subset \mathbb{S}$. Transitions within the reduced state space S are as in the original MDP M, e. g., (i, a, j_1) and (i, a, j_2); transitions from S to $\mathbb{S} \setminus S$ in M are modeled via aggregated transitions to the absorbing terminal state i_{end}. The expected stage costs do not change, cf. Theorem 3.1.8.

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 2.2.9) 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 chapter.

3.1.2 Upper Bounds

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 $v^\alpha_{i_0}$. Since there is only a finite number of states and actions, the maximum expected stage cost is attained, let $c_{\max} := \max_{i\in\mathbb{S},a\in\mathbb{A}(i)} c_i(a)$. This implies an upper bound on the value vector of any policy: from (2.1.1) we easily get $v^\alpha_i(\pi) \leq c_{\max}/(1-\alpha)$, 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 $v^\alpha_{i_0}$ 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:

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

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

$$r^S_{ia} = \alpha \cdot v^\alpha_{\max} \sum_{j\in\mathbb{S}\setminus S} p_{ij}(a), \tag{3.1.3}$$

for each $(i,a) \in S \times \mathbb{A}$. Obviously, by Lemma 3.1.3 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 3.1.7: 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 3.1.9 (Upper-bound induced MDP) Let $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ 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:

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

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 3.1.10 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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 3.1.11 Similar to the lower bound case, one can also show the following. Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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 $(\mathrm{U}_S^{i_0})$ with objective function $\max \sum_{j \in S} v_j$. Then, we have for the optimal value vector $v^\alpha_{M'(S)}$ of the upper-bound S-induced MDP $M'(S)$:

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

Particularly, the component $v^\alpha_{M'(S),i_0}$ equals the optimal value of $(\mathrm{U}_S^{i_0})$.

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

The next results shows that by solving the linear program $(\mathrm{U}_S^{i_0})$ 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 $(\mathrm{U}_S^{i_0})$. 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 3.1.12 *Consider an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, and an optimal solution $\overline{v}$ for the linear program $(\mathrm{U}_S^{i_0})$. For each state $i \in S$, let $a_i \in \mathbb{A}(i)$ be any action that satisfies the corresponding inequality in $(\mathrm{U}_S^{i_0})$ with equality. Then, any policy π for M with $\pi(i) = a_i$ for each $i \in S$ satisfies:*

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

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

Proof. The first inequality holds true due to Lemma 3.1.6 and the second one is clear anyway.

Since the value vector of policy π equals the solution of the system of linear equations (2.2.1) by Theorem 2.2.1, it follows from Lemma 3.1.3 that the value $v^\alpha_{i_0}(\pi)$ can also be computed as the optimal value of the following linear program:

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

Next this linear program is modified as follows. Firstly, constraints $v_i \leq v^{\alpha}_{\max}$ 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:

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

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., $\overline{v}_{i_0}$. Since we constructed a relaxation of the linear program for computing $v^{\alpha}_{i_0}(\pi)$, we obtain $v^{\alpha}_{i_0}(\pi) \leq \overline{v}_{i_0}$. □

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:

$$Q^{S,\pi} v = c^{S,\pi} + r^{S,\pi} \tag{3.1.4}$$

Note that the matrix $Q^{S,\pi}$ is strictly row diagonally dominant and therefore nonsingular. Thus, the system (3.1.4) has a unique solution $\overline{v}^{\pi} \in \mathbb{R}^S$. The next result shows that the value $\overline{v}^{\pi}_{i_0}$ is an upper bound on $v^{\alpha}_{i_0}$, too.

Theorem 3.1.13 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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 $\overline{v}^{\pi}$ be the unique solution of system* (3.1.4)*, and let $\overline{v}_{i_0}$ be the optimal value of the linear program $(\mathrm{U}^{i_0}_S)$. Then, we have:*

$$v^{\alpha}_{i_0} \leq \overline{v}_{i_0} \leq \overline{v}^{\pi}_{i_0}.$$

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

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

Proof. It follows directly from Lemma 3.1.3, part 3, that 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 (3.1.4) one can provide lower and upper bounds on $v^{\alpha}_{i_0}$.

Remark 3.1.14 The unique solution $\overline{v}^{\pi} \in \mathbb{R}^S$ of system (3.1.4) gives the value vector of a policy π for the upper-bound S-induced MDP $M'(S)$:

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

Recall that (3.1.4) 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^{α}_{M,i_0} compared in Theorem 3.1.13 coincide, i. e., we have $\overline{v}^{\pi}_{i_0} = \overline{v}_{i_0}$.

Under the assumptions of Theorem 3.1.13 one can show, similar to Theorem 3.1.12, that each policy π' for the original MDP M with $\pi'(i) = \pi(i)$ for each state $i \in S$ satisfies $v^{\alpha}_{M,i_0}(\pi') \leq \overline{v}^{\pi}_{i_0}$. △

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 (3.1.4) really depends on the chosen policy π. That is, different optimal policies may lead to different upper bounds.

Example 3.1.15 Let $S = \{i_0, i_1\}$ and consider the deterministic S-induced MDP $M(S)$ given by the Markov decision process shown in Figure 3.3 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

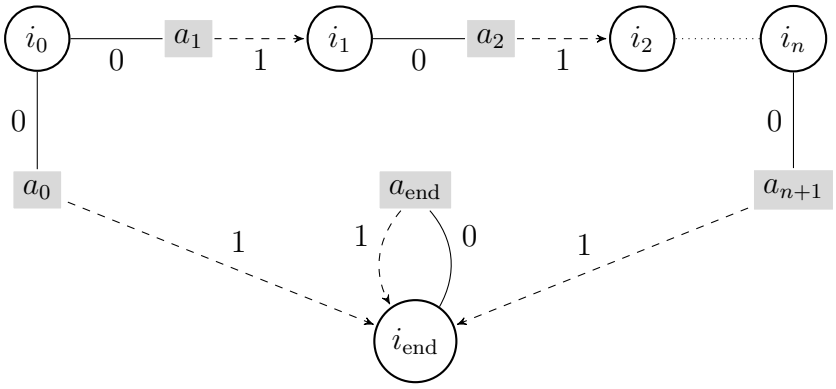


Figure 3.3: Markov decision process of an induced MDP $M(S)$ that yields different upper bounds $\overline{v}^{\pi_0}$ and $\overline{v}^{\pi_1}$ for the optimal policies π_0 and π_1 for $M(S)$ with $\pi_0(i_0) = a_0$ and $\pi_1(i_0) = a_1$.

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 (3.1.4) satisfy:

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

and

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

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

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

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 3.3 for an arbitrary integer $n \in \mathbb{N}$. Now there exists a sequence of states $i_1, \ldots, i_n$ and actions $a_2, \ldots, a_{n+1}$ with $\mathbb{A}(i_k) = \{a_{k+1}\}$ and $p_{i_k i_{k+1}}(a_{k+1}) = 1$ for $k \in \{1, \ldots, n-1\}$. 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 $\overline{v}_{i_0}^{\pi_1} = \alpha^{n+1} v_{\max}^{\alpha}$, while we still have $\overline{v}_{i_0}^{\pi_0} = \alpha v_{\max}^{\alpha}$ for the other optimal policy π_0 using action a_0 at state i_0. This results in a ratio of $\overline{v}_{i_0}^{\pi_0} / \overline{v}_{i_0}^{\pi_1} = 1/\alpha^n$, which goes to infinity for $n \to \infty$ since $\alpha < 1$. $\triangle$

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 $(\mathrm{U}_S^{i_0})$, i. e., $\overline{v}_{i_0} = \overline{v}_{i_0}^{\pi_1}$. In general, however, the upper bound $\overline{v}_{i_0}$ may be better than the bound $\overline{v}_{i_0}^{\pi}$ 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 3.1.16 For instance, consider again the example above for $n = 1$ except that we now 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:

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

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

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

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

Our approximation algorithm presented in Section 3.3 is derived from the theory of this section. It generally employs the construction of upper bounds via solving the linear programs $(\mathrm{U}_S^{i_0})$ 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, as we will see later from the results of Theorem 3.5.12 and Corollary 3.5.14.

Remark 3.1.17 The construction of lower and upper bounds for the component $v_{i_0}^{\alpha}$ 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 $v_{i_0}^{\alpha}$ w. r. t. subset S our approach assumes for each component v_i^{α} of the optimal value vector with state $i \in \mathbb{S} \setminus S$ a lower and upper bound of 0 and $v_{\max}^{\alpha}$, respectively. However, often better bounds on individual components of v^{α} are known or can be determined.

It is easy to see that the upper bound constructions for $v_{i_0}^{\alpha}$ described in this section remain feasible if any available upper bounds $v_{\max}^{\alpha}(j) \geq v_j^{\alpha}$ for $j \in \mathbb{S}$ are used. That is, instead of the vector $r^S \in \mathbb{R}^S$ defined by Equation (3.1.3), we apply the vector $r^{\mathrm{ub},S} \in \mathbb{R}^S$ where:

$$r_{ia}^{\mathrm{ub},S} = \alpha \sum_{j \in \mathbb{S} \setminus S} p_{ij}(a) v_{\max}^{\alpha}(j),$$

for each $(i, a) \in S \times \mathbb{A}$. In doing so, both described ways to determine upper bounds on $v_{i_0}^{\alpha}$ 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:

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

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 Chapter 4, e. g., for the considered elevator control MDPs, it is crucial to employ involved lower and upper bounds in order to obtain reasonable results at all.

3.2 Structural Approximation Result

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 3.2.1 (r–neighborhood) For an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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:

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

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

Our approximation theorem is as follows (we already published a weaker version of this result in [HKP+06]).

Theorem 3.2.2 *Let $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ be an MDP 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) $|S| \leq \max\{(bd)^{r+1}, r+1\}$, 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 (3.1.4) w. r. t. any optimal policy π for the S-induced MDP $M(S)$ satisfy:

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

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.,

$$0 \leq v^{\alpha}_{i_0} - \underline{v}_{i_0} \leq \varepsilon,$$
$$0 \leq \overline{v}^{\pi}_{i_0} - v^{\alpha}_{i_0} \leq \varepsilon.$$

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 $S = S(i_0, r)$:

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

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

The proof of Property (ii) is as follows. Consider the extension $\underline{v}^{\mathrm{ext}} \in \mathbb{R}^{\mathbb{S}}$ of the solution $\underline{v}$ of the linear program (L^{Σ}_S) as defined in Equation (3.1.2):

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

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 (3.1.4) w. r. t. policy π as follows:

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

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

By Theorem 3.1.8 the solution $\underline{v}$ of (L_S^{Σ}) equals the optimal value vector of the MDP $M(S)$. Since π is optimal for $M(S)$, Theorem 2.2.1 implies that the corresponding constraints in the linear program (L_S^{Σ}) 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 $\underline{v}^{\text{ext}}$:

$$\underline{v}_i^{\text{ext}} = c_i(\pi(i)) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i)) \underline{v}_j^{\text{ext}} \quad \forall i \in S. \tag{3.2.1}$$

Note that in (3.2.1) we sum over the whole state space, which is feasible due to $\underline{v}_j^{\text{ext}} = 0$ for each $j \in \mathbb{S} \setminus S$.

On the other hand, since $\overline{v}^{\pi}$ satisfies the system of equations (3.1.4) we have the following relation for the extension $\overline{v}^{\text{ext}}$:

$$\overline{v}_i^{\text{ext}} = c_i(\pi(i)) + \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i)) \overline{v}_j^{\text{ext}} \quad \forall i \in S. \tag{3.2.2}$$

From the Equations (3.2.1) and (3.2.2) we obtain:

$$\overline{v}_i^{\text{ext}} - \underline{v}_i^{\text{ext}} = \alpha \sum_{j \in \mathbb{S}} p_{ij}(\pi(i)) (\overline{v}_j^{\text{ext}} - \underline{v}_j^{\text{ext}}) \quad \forall i \in S. \tag{3.2.3}$$

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

$$\overline{v}_i^{\text{ext}} - \underline{v}_i^{\text{ext}} \leq \alpha^{r+1-k} v_{\max}^{\alpha}. \tag{3.2.4}$$

Note that all i to which (3.2.4) refers are contained in S because of $k \leq r$.

For $k = r$ and for each state $i \in S(i_0, k)$, Inequality (3.2.4) follows from (3.2.3) due to $\overline{v}_j^{\text{ext}} \leq v_{\max}^{\alpha}$ and $\underline{v}_j^{\text{ext}} \geq 0$ for each $j \in \mathbb{S}$:

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

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 (3.2.4) holds for each state $j \in S(i_0, k)$ with $0 < k \leq r$. For each $i \in S(i_0, k-1)$, we again apply Equality (3.2.3):

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

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 $j \in S(i_0, k)$:

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

which completes the inductive proof of (3.2.4).

For $i = i_0$ and $k = 0$, Inequality (3.2.4) implies:

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

Finally, we distinguish two cases to show Property (ii). If $\varepsilon \geq \alpha v_{\max}^{\alpha}$, we have $r = 0$, and thus $\overline{v}_{i_0}^{\pi} - \underline{v}_{i_0} \leq \alpha v_{\max}^{\alpha} \leq \varepsilon$. 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:

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

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 3.1.6 and 3.1.10 it is already known that $v_{i_0}^{\alpha} \geq \underline{v}_{i_0}$ and $v_{i_0}^{\alpha} \leq \overline{v}_{i_0}^{\pi}$. By these inequalities we obtain:

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

□

We mention that the construction of Theorem 3.2.2 does not only provide an approximation for the component $v_{i_0}^{\alpha}$ of the optimal value vector, but a larger local approximation of v^{α} regarding all states in the subset $S = S(i_0, r)$: Inequality (3.2.4) in the proof of Theorem 3.2.2 gives the following bounds for the states in S.

Corollary 3.2.3 *Under the same assumptions as used in Theorem 3.2.2 with the restriction that $\underline{v}$ is necessarily the optimal solution of the linear program (L_S^{Σ}), let π be any optimal policy for the induced MDP $M(S)$. Then, for each $k \in \{1, \ldots, r\}$ and each state $i \in S(i_0, k)$, we have $\underline{v}_i \leq v_i^{\alpha} \leq \overline{v}_i^{\pi}$ with the approximation guarantee:*

$$\overline{v}_i^{\pi} - \underline{v}_i \leq \alpha^{r+1-k} v_{\max}^{\alpha}.$$

By Corollary 3.2.3 it is justified to speak of local approximations of the optimal value vector around state i_0. Note that the approximation guarantee for a state in S is the better, the fewer the number of transitions is, needed to reach the state from i_0.

Although this is not important in our context, we mention that Theorem 3.2.2 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 3.1.13), we also have the following result.

Corollary 3.2.4 *Under the same assumptions as used in Theorem 3.2.2, 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:*

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

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

The following example shows that the construction of the state space used in Theorem 3.2.2 and Corollary 3.2.4 is optimal in the sense that for an arbitrary MDP, the state space cannot be reduced in order to achieve the desired approximation guarantee. However, this may be possible if additional assumptions on the structure of the MDP are made.

Example 3.2.5 Let $0 < \varepsilon \leq \alpha/(1-\alpha)$. By Corollary 3.2.4 the r–neighborhood of a state i_0 for $r = \lceil \log(\varepsilon / v_{\max}^{\alpha}) / \log \alpha \rceil - 1$ suffices to obtain an approximation guarantee of $\overline{v}_{i_0} - \underline{v}_{i_0} \leq \varepsilon$ for optimal solutions $\underline{v}$ and $\overline{v}$ of the linear programs (L_S^{Σ}) and $(\mathrm{U}_S^{i_0})$, respectively.

In the following, we construct for arbitrary integer numbers $b, d \in \mathbb{N}$ an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ such that for the $(r-1)$–neighborhood $S(i_0, r-1)$ the corresponding solutions $\underline{v}$ and $\overline{v}$ yield $\overline{v}_{i_0} - \underline{v}_{i_0} > \varepsilon$. The MDP has a tree-like structure rooted at state $i_0 = i_1^{(0)} \in \mathbb{S}$. The state space $\mathbb{S}$ is given by:

$$S_n = \left\{ i_k^{(n)} \mid 1 \leq k \leq (bd)^n \right\} \quad \text{for each } n \in \{0, \ldots, r\},$$

$$\mathbb{S} = \bigcup_{n=0}^{r} S_n.$$

Moreover, we have identical sets of actions $\mathbb{A}(i) = \{a_1, \ldots, a_b\}$ for every state $i \in \mathbb{S}$. For each state in S_n with $n \in \{0, \ldots, r-1\}$, applying any action moves the system with uniform transition probabilities to one of exactly d "new states" in S_{n+1} that can be reached only via this action. That is, for any $m \in \{1, \ldots, b\}, n \in \{0, \ldots, r-1\}$, and $k \in \{1, \ldots, (bd)^n\}$, we have:

$$p_{i_k^{(n)} i_l^{(n+1)}}(a_m) = \frac{1}{d}$$

for every $l \in \{(k-1)bd+(m-1)d+1, \ldots, (k-1)bd+md\}$. The expected stage costs are $c_i(a) = 0$ for every state $i \in \mathbb{S} \setminus S_r$ and each possible action $a \in \mathbb{A}(i)$. At each state $i \in S_r$ the systems remains in that state with an additional stage cost of 1 in each transition, i. e., for each action $a \in \mathbb{A}(i)$, we have $p_{ii}(a) = 1$ and $c_i(a) = 1$.

Now let us consider the solution $\underline{v}$ of the linear program (L_S^{Σ}) for the subset of states $S = S(i_0, r-1) = \mathbb{S} \setminus S_r$. On the one hand, since the expected stage costs for all states in S are zero, we have $\underline{v} \equiv 0$. On the other hand, since the stage cost for each transition always equals 1 as soon as a state in S_r has been reached, the value of the solution $\overline{v}$ of $(\mathrm{U}_S^{i_0})$ at state i_0 equals:

$$\overline{v}_{i_0} = \sum_{k=r}^{\infty} \alpha^k = \frac{\alpha^r}{1-\alpha},$$

which gives $\overline{v}_{i_0} - \underline{v}_{i_0} = \alpha^r/(1-\alpha)$. Note that due to $v_{\max}^{\alpha} = 1/(1-\alpha)$ we have $r = \lceil \log(\varepsilon(1-\alpha))/\log\alpha \rceil - 1$. This implies:

$$\alpha^r = \alpha^{\lceil \log(\varepsilon(1-\alpha))/\log\alpha \rceil - 1} > \alpha^{\log(\varepsilon(1-\alpha))/\log\alpha} = \varepsilon(1-\alpha),$$

Thus we obtain $\overline{v}_{i_0} - \underline{v}_{i_0} > \varepsilon$. △

The following remark generalizes the situation considered in the example.

Remark 3.2.6 Assume that the MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ does not contain cycles in $S = S(i_0, r)$, i. e., for each state $i \in S$, there does not exist an (i,i)-path P with length $|P| > 0$. Then, each optimal policy for the lower-bound S-induced MDP $M(S)$ is also optimal for the upper-bound S-induced MDP $M'(S)$. Hence, for each optimal policy π for $M(S)$, Remark 3.1.14 implies $\overline{v}_{i_0}^{\pi} = \overline{v}_{i_0}$ for the optimal value $\overline{v}_{i_0}$ of $(\mathrm{U}_S^{i_0})$. Moreover, the approximation guarantee of Corollary 3.2.3 is satisfied with equality for policy π:

$$\overline{v}_i^{\pi} - \underline{v}_i = \alpha^{r+1-k} v_{\max}^{\alpha},$$

for each $i \in S(i_0, k)$ and $k \in \{0, \ldots, r\}$. △

Remark 3.2.7 Of all the approaches from the literature the random sampling algorithm of Kearns et al. [KMN99] gives the results most comparable to Theorem 3.2.2. 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 3.2.2. 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. △

3.3 Algorithm

In order to compute local approximations of the optimal value vector v^α around a particular state of a given MDP, it is usually inappropriate to apply the construction of Theorem 3.2.2 directly (see Section 4.2.1 for computational results). 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 3.4. The algorithmic approach presented below is the basis of our computational tool that is applied for various MDPs in Chapter 4.

Algorithm 4 Generic approximation algorithm

1: **Input:** an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ (given implicitly), 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**

3.3.1 Approach

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_{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 3.2.2 and Corollary 3.2.4 provide an approximation in terms of the absolute difference between upper and lower bounds. In practice, however, a relative guarantee is typically more suitable. 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.,

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

Once this approximation guarantee is obtained, the algorithm terminates. This generic approximation algorithm is summarized in Algorithm 4. Clearly,

Algorithm 4 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 3.3.1 It has been shown in Theorem 3.1.12 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 $(v^\alpha_{i_0}(\pi) - v^\alpha_{i_0})/v^\alpha_{i_0} \leq \varepsilon$. △

3.3.2 Column Generation

Our implementation of Algorithm 4 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 $(\mathrm{L}^{i_0}_{\mathbb{S}})$ that 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 $v^\alpha_{i_0}$, and totally neglect the linear programs $(\mathrm{U}^{i_0}_S)$ for $S \subseteq \mathbb{S}$. These contribute only in terms of the computed upper bounds. We mention that it is impossible 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 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 without using matrix-vector notation:

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

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:

$$\bar{p}_j = \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia}. \tag{3.3.1}$$

The theory of linear programming implies the following result.

Theorem 3.3.2 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subsetneq \mathbb{S}$ with $i_0 \in S$, let $\underline{u}$ be an optimal solution of the linear program $(\mathrm{DL}_S^{i_0})$. If we have $\bar{p}_j = 0$ for all states $j \in \mathbb{S} \setminus S$, the lower bound $\underline{v}_{i_0}$ computed by $(\mathrm{L}_S^{i_0})$ satisfies $\underline{v}_{i_0} = v_{i_0}^{\alpha}$.*

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 3.3.3 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 $i_0, i_1, \ldots, i_n$, 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 $k \in \{0, \ldots, n\}$. 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 $l \in \{k, \ldots, n\}$.

- 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, a discounted MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ for the machine replacement problem is naturally

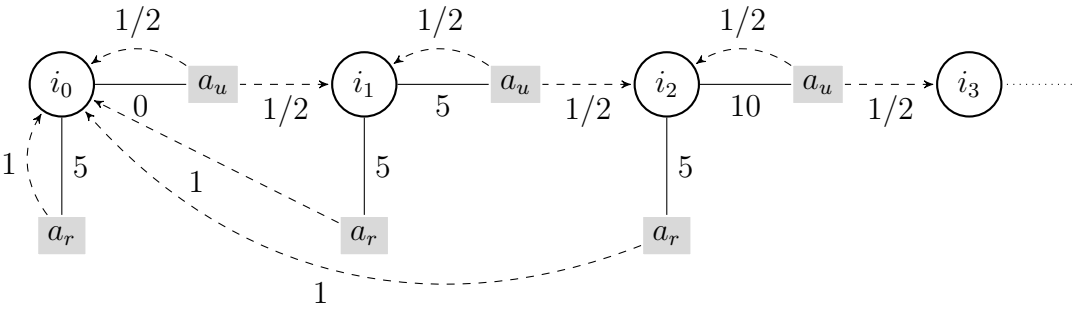


Figure 3.4: Part of the Markov decision process for the machine replacement problem.

given by:

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

In the example we consider 10 possible machine states, i. e., $n = 9$, and make the quite unrealistic assumption that $c_k = 5k$ for each $k \in \{0, \ldots, n\}$ 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 $k \in \{0, \ldots, n-1\}$ and $p_{i_n i_n}(a_u) = 1$. The associate Markov decision process is partially illustrated in Figure 3.4. 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 α.

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

$$
\begin{aligned}
\min \quad & 0u_{i_0 a_u} + 5u_{i_0 a_r} \\
\text{subject to } & (1 - \tfrac{\alpha}{2})u_{i_0 a_u} + (1 - \alpha)u_{i_0 a_r} = 1 \\
& u_{i_0 a_u}, u_{i_0 a_r} \geq 0.
\end{aligned}
$$

The optimal solution is unique and given by $\underline{u}_{i_0 a_u} = 2/(2 - \alpha)$ and $\underline{u}_{i_0 a_r} = 0$.

Consequently, the reduced profit of state i_1 equals:

$$\bar{p}_{i_1} = \frac{\alpha}{2}\underline{u}_{i_0 a_u} = \frac{\alpha}{2-\alpha} > 0,$$

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 $(\mathrm{DL}^{i_0}_{S_2})$:

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

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 $v^\alpha_{i_0}$ of the optimal value vector exactly. Due to linear programming duality, we have $v^\alpha_{i_0} = 5\alpha/(2-\alpha-\alpha^2)$.

On the other hand, if we have $\alpha > 2/3$, the unique optimal solution of $(\mathrm{DL}^{i_0}_{S_2})$ 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 the algorithm generates all states in the case $\alpha = 0.99$. △

We see that the column generation algorithm is sometimes able to detect non-optimal actions. This way, it may compute exactly a local part of the optimal value vector and an optimal policy restricted to the states reached by the policy from i_0. Moreover, the example points out that the non-optimality of actions will be proven by the algorithm more likely and already for smaller state spaces $S \subset \mathbb{S}$ when the discount factor is small. For discount factors close to 1, this seems only to be possible for states where the process more or less "ends", i. e., the system can be controlled in such a way that the sum of all future stage costs is very small. For instance, this situation occurs in the bin coloring MDP introduced in Section 4.1.3. Consider a state where the maximum colorfulness χ is such that it cannot be increased more than once in all subsequent stages. If for such a state there exists an action that causes

an increase of χ although this not necessary, i. e., there exits an action that does not increase χ also, the former action is non-optimal.

Next we consider the pricing problem arising in the column generation algorithm. The goal of the pricing problem is to find a state with maximum reduced profit. Note that by Equation (3.3.1) the reduced profit $\bar{p}_j$ of a state $j \in \mathbb{S} \setminus S$ can only be positive if there exists a state $i \in S$ and an action $a \in \mathbb{A}(i)$ such that $p_{ij}(a) > 0$. Thus, only successors of states in S are candidates to be added to the state space S in one pricing step. We denote this set of candidate states by S_{cand}, i. e.,

$$S_{\text{cand}} := \{j \in \mathbb{S} \setminus S \mid \exists i \in S \, \exists a \in \mathbb{A}(i) \colon p_{ij}(a) > 0\}. \tag{3.3.2}$$

In order to determine a state with maximum reduced profit, one can simply compute the reduced profit for each state in S_{cand} by Equation (3.3.1). Thus, the pricing problem is in some sense trivial. The total number of arithmetic operations for this straight approach equals $\Theta(|S_{\text{cand}}||S \times \mathbb{A}|)$ in general since each state in S_{cand} may be a successor of all states in S via all possible actions.

The pricing problem can be seen as the problem of finding a star with maximum weight in the following weighted digraph $D = (V_1 \cup V_2, E)$, where

$$\begin{aligned} V_1 &= S, \\ V_2 &= \mathbb{S} \setminus S. \end{aligned}$$

The set of arcs is defined by the possible transitions between states in V_1 and V_2, and the arc weights are defined suitably:

$$\begin{aligned} E &= \{(i,j) \in V_1 \times V_2 \mid \exists a \in \mathbb{A}(i) : p_{ij}(a) > 0\}, \\ c_{ij} &= \alpha \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia} \quad \forall (i,j) \in E. \end{aligned}$$

The graph theoretic interpretation of the pricing problem is as follows.

Theorem 3.3.4 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let $\underline{u}$ be an optimal solution of the linear program $(\mathrm{DL}_S^{i_0})$ and let $D = (V_1 \cup V_2, E)$ be the corresponding digraph as defined above. Then, for each state $j \in \mathbb{S} \setminus S$, its reduced profit $\bar{p}_j$ equals the weight $w(j)$ of the maximal star in D that includes j. Particularly, the maximum weight of a star containing exactly one node in V_2 equals the maximum reduced profit.*

Proof. For a state $j \in \mathbb{S} \setminus S$, let $S(j) := \{i \in S \mid \exists a \in \mathbb{A}(i) : p_{ij}(a) > 0\}$ be the set of the predecessor states of j that are in S. The maximal star

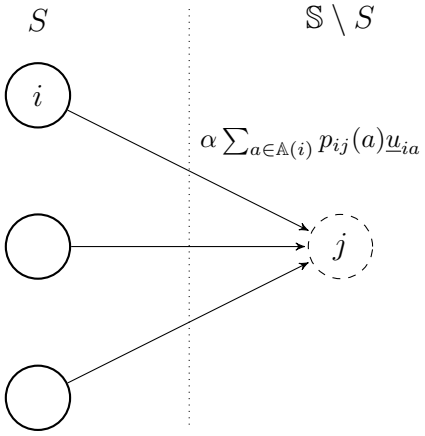


Figure 3.5: Illustration that the reduced profit of a state $j \in \mathbb{S} \setminus S$ can be computed as the weight of a maximal star that includes j.

that includes j also contains all the nodes in $S(j)$. Therefore, its weight $w(j)$ equals:

$$w(j) = \sum_{i \in S(j)} c_{ij} = \sum_{i \in S(j)} \alpha \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia} = \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia} = \bar{p}_j. \qquad \square$$

An illustration of Theorem 3.3.4 is given in Figure 3.5. Of course it is possible to formulate the pricing problem as an equivalent optimization problem in a directed hypergraph, too.

We give a short overview about details of our approximation algorithms that are considered later on. In each iteration of the column generation algorithm usually several states with positive reduced profit are added to the previous state space S. We will study different pricing strategies for the algorithm to fast compute suitable subsets of states to be added. Moreover, we will look at approximation heuristics that aim to determine a good initial state space $S_1 \subseteq \mathbb{S}$. These ingredients of our algorithm are described later in the Sections 3.5.3 and 3.5.4 after their theoretical basis has been established.

To achieve a given approximation guarantee, the column generation algorithm usually requires substantially fewer states compared to the neighborhood construction of Theorem 3.2.2 and Corollary 3.2.4. See Section 4.2.1 for computational results that give evidence for this behavior. An example where the opposite is the case independently of the used pricing strategy is presented in Section 3.5.3.

3.4 Applications

We showed in Theorem 3.2.2 and Corollary 3.2.4 that an ε-approximation of the optimal value vector v^α at a particular state can be computed by considering only a local subset of states whose size does not depend on the total number of states. Moreover, we proposed a column generation algorithm that is more suitable in practice to approximate v^α locally. In the following we highlight different applications for both the approximation theorem and the approximation algorithm. The applications are obtained by considering appropriate restrictions of the original MDP. Since we will consider different MDPs, the value vector of an MDP M will be denoted by v^α_M in this section.

3.4.1 Approximation for Policies

Recall that the value vector $v^\alpha_M(\pi)$ of a concrete policy π for a given MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ can be computed by solving the system of linear equations (2.2.1). However, the usual methods to solve a system in $|\mathbb{S}|$ variables and linear equations does 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 $v^\alpha_M(\pi)$. 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 3.4.1 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ and a policy π, define the MDP $M(\pi) = (\mathbb{S}, \mathbb{A}', p', c', \alpha)$ 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 $v^\alpha_M(\pi) = v^\alpha_{M(\pi)}$.*

Proof. $v^\alpha_M(\pi) = v^\alpha_{M(\pi)}$ follows from Theorem 2.2.1 on page 15 since the equations (2.2.1) for policy π to compute $v^\alpha_M(\pi)$ and the optimality equations (2.2.2) for $M(\pi)$ are identical. □

By Theorem 3.4.1 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 $S \subseteq \mathbb{S}$:

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

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 $v^{\alpha}_{M,i_0}(\pi)$ already introduced in Theorem 3.1.13, both 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 3.2.4 and the approximation algorithm as well.

By Lemma 3.1.3, part 3, 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 3.4.2 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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:*

$$Q^{S,\pi} v = c^{S,\pi}. \tag{3.4.1}$$

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}.

3.4.2 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 3.4.3 Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and an action $a_0 \in \mathbb{A}(i_0)$, let $M(i_0, a_0) = (\mathbb{S}, \mathbb{A}', p', c', \alpha)$ 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* is defined by:

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

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 3.4.4 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ and a state $i_0 \in \mathbb{S}$, an action $a_0 \in \mathbb{A}(i_0)$ is optimal (as defined in Definition 2.2.3 on page 16) if and only if we have $v^{\alpha}_{M,i_0}(a_0) = v^{\alpha}_{M,i_0}$.*

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 $i \in \{1, \ldots, m\}$ and $x_i < y_i$ for at least one $i \in \{1, \ldots, m\}$.

Since the optimality equations (2.2.1) 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 $v^{\alpha}_{M,i_0} < v^{\alpha}_{M(i_0,a_0),i_0} = v^{\alpha}_{M,i_0}(a_0)$. □

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

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

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 3.4.4 directly implies the following result.

Corollary 3.4.5 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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 $a \in \mathbb{A}(i_0)\backslash\{a_0\}$. Then, if $\overline{v}_{M,i_0}(a_0) \leq \underline{v}_{M,i_0}(a)$ for each $a \in \mathbb{A}(i_0)\backslash\{a_0\}$, the action a_0 is optimal. In Chapter 4 we will exploit this observation in the analysis of policies for MDPs emerging from online optimization problems. Particularly, we will determine an optimal parking policy for the considered elevator control MDP in Section 4.3.4.

3.5 Details of the Column Generation Method

In this section, we give further theoretical results relating to our approximation algorithm via column generation. These results give additional insight in the structure of the encountered linear programs and are used to derive valuable components for the approximation algorithm. The outline is as follows. In Section 3.5.1 we develop a hypergraph minimum cost flow interpretation of the reduced dual linear programs, similar to that for the standard problem introduced in Section 2.2.5. Then, in Section 3.5.2 we analyze the structure

of the basic solutions of the restricted dual linear programs, prove an explicit formula for computing the dual prices for a given basis, and propose a method to construct initial bases for solving the linear programs encountered in the column generation. A formula to determine the reduced profit of a candidate state is derived in Section 3.5.3. Moreover, we establish an upper bound construction based on the reduced profits and propose different pricing strategies. In Section 3.5.4 we develop practical approximation heuristics that can be incorporated within the column generation algorithm. Finally, we derive an algorithm based on policy iteration that is equivalent to our approximation method but does not feature any technique from linear programming, see 3.5.6.

3.5.1 Dual Problem Interpretation

It has been shown in Section 2.2.5 that the standard dual linear program:

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

can be interpreted as a hypergraph minimum cost flow problem. In this paragraph, we establish a similar result for the reduced dual linear program $(\mathrm{DL}_S^{i_0})$ for any $i_0 \in \mathbb{S}$ and any subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$:

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

The differences between both linear programs are as follows. On the one hand, only the variables and constraints for the reduced state space S are considered in $(\mathrm{DL}_S^{i_0})$. On the other hand, in the reduced linear program the right hand side equals 1 only for the constraint w. r. t. state i_0 and equals 0, otherwise. In order to construct an instance of the hypergraph minimum cost flow problem, we rely on the S-induced MDP for the following reason. Since

there may exist states in S whose successors are contained in $\mathbb{S} \setminus S$, flow may "leave" the set S. Thus, the additional state i_{end} is of particular significance here: the corresponding node in the hypergraph absorbs all flow to states in $\mathbb{S} \setminus S$. The construction in detail is as follows.

Theorem 3.5.1 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a subset of states $S \subseteq \mathbb{S}$, and a state $i_0 \in S$, let $M(S) = (\mathbb{S}', \mathbb{A}', p', c', \alpha)$ be the S-induced MDP. The dual linear program $(\mathrm{DL}_S^{i_0})$ describes the following instance of the hypergraph minimum cost flow problem with hypergraph $\mathcal{H} = (\mathcal{V}, \mathcal{E})$ and parameters μ, b, and κ:*

- $\mathcal{V} = \mathbb{S}'$,
- $\mathcal{E} = \{(i, N_i(a)) \in \mathbb{S}' \times 2^{\mathcal{V}} \mid i \in \mathbb{S}', a \in \mathbb{A}'(i)\}$, *where we define the set $N_i(a) = \{j \in \mathbb{S}' \mid p'_{ij}(a) > 0\}$ for each $i \in \mathbb{S}'$ and each $a \in \mathbb{A}'(i)$,*
- $\mu_j(i, N_i(a)) = \alpha p'_{ij}(a)$ *for each $(i, N_i(a)) \in \mathcal{E}$ and each $j \in N_i(a)$,*
- $b(i_0) = 1$ *and $b(i) = 0$ for each $i \in \mathcal{V} \setminus \{i_0\}$,*
- $\kappa(i, N_i(a)) = c'_i(a)$ *for each hyperarc $(i, N_i(a)) \in \mathcal{E}$.*

Proof. In the case $\mathbb{S}' = S$, i. e., the induced MDP $M(S)$ does not contain an additional state i_{end}, we have $\mathbb{A}' = \mathbb{A}$, $p' = p$, and $c' = c$. Thus, the linear program $(\mathrm{DL}_S^{i_0})$ can be rewritten as:

$$\begin{aligned}
\min \quad & \sum_{i \in \mathbb{S}'} \sum_{a \in \mathbb{A}'(i)} c'_i(a) u_{ia} && && (3.5.1)\\
\text{subject to} \quad & \sum_{a \in \mathbb{A}'(i_0)} u_{i_0 a} - \alpha \sum_{i \in \mathbb{S}'} \sum_{a \in \mathbb{A}'(i)} p'_{i i_0}(a) u_{ia} = 1 && &&\\
& \sum_{a \in \mathbb{A}'(j)} u_{ja} - \alpha \sum_{i \in \mathbb{S}'} \sum_{a \in \mathbb{A}'(i)} p'_{ij}(a) u_{ia} = 0 && \forall j \in \mathbb{S}' \setminus \{i_0\} &\\
& u_{ia} \geq 0 && \forall i \in \mathbb{S}'\ \forall a \in \mathbb{A}'(i). &
\end{aligned}$$

Obviously, the linear program (3.5.1) describes the instance of the hypergraph minimum cost flow specified above, proving the theorem for $\mathbb{S}' = S$.

In the case $\mathbb{S}' = S \cup \{i_{\text{end}}\}$, the linear program $(\mathrm{DL}_S^{i_0})$ is equivalent to:

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

Recall that by Theorem 3.1.8 and by the construction of the induced MDP we have $\mathbb{A}'(i_{\text{end}}) = \{a_{\text{end}}\}$, $c'_{i_{\text{end}}}(a_{\text{end}}) = 0$, and $p'_{i_{\text{end}} i_{\text{end}}}(a_{\text{end}}) = 1$. Moreover, for all states $i, j \in S$ and each $a \in \mathbb{A}(i)$, the components of $M(S)$ satisfy $\mathbb{A}'(i) = \mathbb{A}(i)$, $p'_{ij}(a) = p_{ij}(a)$, $p'_{i_{\text{end}} j}(a_{\text{end}}) = 0$, and $c'_i(a) = c_i(a)$. Thus, the linear program above again equals (3.5.1), which completes the proof. □

Thus, the hypergraph minimum cost flow problem encountered here is a single-source percolation problem, where one unit of flow leaves i_0 and the task is to trickle this flow away such that the resulting cost is minimum. In doing so, traversing each hyperarc reduces the amount of flow since the discount factor satisfies $\alpha < 1$. To make a flow vanish completely, it must be send along cycles. For instance, once flow has entered i_{end}, it is consumed without any further incurred cost.

Incorporating a parametric search method Oldham [Old01] proposed a combinatorial algorithm for solving the deterministic version of this percolation problem in $O(mn^2 \log n)$. Note that in the deterministic case, the hypergraph reduces to a usual graph.

3.5.2 Dual Basic Solutions

In this section we address the structure of the basic solutions for the reduced dual linear program $(\mathrm{DL}_S^{i_0})$ for some $S \subseteq \mathbb{S}$ and $i_0 \in S$. Clearly, this linear program equals the dual of the linear program (P^{i_0}) defined on page 37 in the case $S = \mathbb{S}$. Note further that the dual of the reduced linear program $(\mathrm{U}_S^{i_0})$, that provides an upper bound on $v_{i_0}^{\alpha}$, has exactly the same constraints as $(\mathrm{DL}_S^{i_0})$. Therefore, all results concerning the basic solutions for $(\mathrm{DL}_S^{i_0})$ also apply for the dual linear programs to (P^{i_0}) and $(\mathrm{U}_S^{i_0})$.

Recall that by Theorem 2.2.12 and Corollary 2.2.13, all feasible basic solutions of the standard dual program (D^{Σ}) correspond one-to-one to deterministic policies and are non-degenerate. As we will see, these properties do not hold true in general for the feasible basic solutions of the linear program ($\mathrm{DL}_S^{i_0}$).

In the sequel, we will refer to a basic solution that corresponds to a policy as a policy basic solution.

Definition 3.5.2 (Policy basic solution) Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, and a policy π for the S-induced MDP $M(S)$, the basic solution u^π for the linear program ($\mathrm{DL}_S^{i_0}$) with basis $\{(i, \pi(i)) \mid i \in S\}$ is called *policy basic solution of* π. △

The policy basic solution of a policy π is well-defined for the following reason. The matrix consisting of the columns of the constraint matrix of ($\mathrm{DL}_S^{i_0}$) with indices $(i, \pi(i))$ for each $i \in S$ is strictly column diagonally dominant, and thus nonsingular. Therefore, u^π is indeed a basic solution of the linear program ($\mathrm{DL}_S^{i_0}$).

First we show that each policy basic solution of the linear program ($\mathrm{DL}_S^{i_0}$) is feasible and that its objective value equals the value vector of the policy in the MDP $M(S)$ at state i_0.

Theorem 3.5.3 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, the policy basic solution u^π of any policy π for $M(S)$ satisfies the following properties:*

1. *u^π is feasible for the linear program ($\mathrm{DL}_S^{i_0}$).*
2. *The objective value of u^π in ($\mathrm{DL}_S^{i_0}$) equals the component $v^\alpha_{M(S),i_0}(\pi)$ of the value vector of π, in particular the policy basic solution of an optimal policy for $M(S)$ is optimal for ($\mathrm{DL}_S^{i_0}$).*

Proof. Let π be a policy for $M(S)$. Define the matrix $P(\pi) \in \mathbb{R}^{S \times S}$ as follows:

$$P(\pi)_{ij} = p_{ij}(\pi(i)) \quad \text{for each } i, j \in S.$$

By definition the policy basic solution u^π in the dual linear program ($\mathrm{DL}_S^{i_0}$) is given by $u^\pi_{ia} = 0$ for each $i \in S$ and $a \in \mathbb{A}(i) \setminus \{\pi(i)\}$ and the unique solution of the following system of linear equations:

$$u_{j\pi(j)} - \alpha \sum_{i \in S} p_{ij}(\pi(i)) u_{i\pi(i)} = \begin{cases} 1, & \text{if } j = i_0, \\ 0, & \text{if } j \neq i_0. \end{cases}$$

Denoting the column vector of variables by $u := (u_{j\pi(j)})_{j\in S} \in \mathbb{R}^S$, the system of equations can be rewritten in matrix-vector notation as:

$$(I - \alpha P(\pi))^t u = (1, 0, \ldots, 0)^t, \tag{3.5.2}$$

where $I \in \mathbb{R}^{S\times S}$ is the identity matrix. It is well known that the inverse of the matrix in (3.5.2) equals $(I - \alpha P(\pi))^{-1} = \sum_{k=0}^{\infty}(\alpha P(\pi))^k$ since:

$$(I - \alpha P(\pi)) \cdot (I + \alpha P(\pi) + \cdots + (\alpha P(\pi))^{k-1}) = I - (\alpha P(\pi))^k$$

for each $k \in \mathbb{N}$ and $(\alpha P(\pi))^k \to 0$ for $k \to \infty$. In particular, all entries of the matrix $(I - \alpha P(\pi))^{-1}$ are non-negative. Together with Equation (3.5.2) this implies $u^\pi_{j\pi(j)} \geq 0$ for each state $j \in S$, i. e., the basic solution u^π is feasible for the linear program $(\mathrm{DL}^{i_0}_S)$.

For the second part, consider the corresponding primal solution v^π for the dual basis $\{(i, \pi(i)) \mid i \in S\}$. By definition v^π is given as the unique solution of the system of linear equations (2.2.1) on page 16 for $M = M(S)$. Therefore, the solution v^π equals the value vector of π in the S-induced MDP, in particular, we have $v^\pi_{i_0} = v^\alpha_{M(S),i_0}(\pi)$. By linear programming duality the objective value of v^π in $(\mathrm{L}^{i_0}_S)$, i. e., $v^\pi_{i_0} = v^\alpha_{M(S),i_0}(\pi)$, equals that of u^π in $(\mathrm{DL}^{i_0}_S)$, which completes the proof. □

In contrast to the standard dual program (D^Σ), the basic solutions that correspond to policies, i. e., policy basic solutions, may be degenerate for the linear program $(\mathrm{DL}^{i_0}_S)$.

Example 3.5.4 Consider once more the MDP shown in Figure 3.1, and let $i_0 = i_2$. The associated dual linear program $(\mathrm{DL}^{i_0}_S)$ for $S = \mathbb{S}$ reads:

$$\begin{array}{lrrrrrrl}
\min & 2u_{i_1a_1} & +3u_{i_1a_2} & +2u_{i_2a_3} & & +u_{i_3a_5} & +3u_{i_4a_6} & \\
\text{s. t. } & (1-\frac{\alpha}{2})u_{i_1a_1} & +u_{i_1a_2} & & & & & = 0 \\
& -\frac{\alpha}{2}u_{i_1a_1} & & +u_{i_2a_3} & & & & = 1 \\
& & -\alpha u_{i_1a_2} & -\alpha u_{i_2a_3} & +(1-\alpha)u_{i_3a_4} & +u_{i_3a_5} & & = 0 \\
& & & & & -\alpha u_{i_3a_5} & +(1-\alpha)u_{i_4a_6} & = 0 \\
& & & & & & u_{i_1a_1}, u_{i_1a_2}, u_{i_2a_3}, u_{i_3a_4}, u_{i_3a_5}, u_{i_4a_6} & \geq 0.
\end{array}$$

Consider the policy π with

$$\pi(i_1) = a_2, \pi(i_2) = a_3, \pi(i_3) = a_4, \pi(i_4) = a_6.$$

One can show that π is optimal if $\alpha \geq 2/5$. The policy basic solution of π reads:

$$u_{i_1a_1} = 0, u_{i_1a_2} = 0, u_{i_2a_3} = 1, u_{i_3a_4} = \tfrac{\alpha}{1-\alpha}, u_{i_3a_5} = 0, u_{i_4a_6} = 0.$$

Thus, the basic solution is degenerate. △

It should also be mentioned that different policies π_1 and π_2 do not necessarily define different policy basic solutions. In the example above, the policy μ with $\mu(i_1) = a_1$ and $\mu(i) = \pi(i)$ for each $i \in S \setminus \{i_1\}$ gives the same policy basic solution as π.

Moreover, there may exist additional feasible bases for the dual program $(\mathrm{DL}_S^{i_0})$ that do not correspond to policies. In Example 3.5.4, the four columns $(i_1, a_2), (i_2, a_3), (i_3, a_4), (i_3, a_5)$ apparently define a basis but do not correspond to a policy since there are two columns covering state i_3 and no column covering state i_4. This basis again gives the same basic solution as in the example.

Despite the fact that feasible bases not corresponding to policies may exist, one can show that each feasible basic solution for $(\mathrm{DL}_S^{i_0})$ has at least one associated basis that corresponds to a policy, i. e., each basic solution is a policy basic solution. In order to show this property, we will exploit the following lemma.

Lemma 3.5.5 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let u be any feasible solution of the linear program $(\mathrm{DL}_S^{i_0})$. Assume there exists a partition of S into sets $S_1, S_2 \subset S$, i. e., $S_1 \cup S_2 = S$ and $S_1 \cap S_2 = \emptyset$, with the following properties:*

1. *$i_0 \in S_1$.*

2. *For each $j \in S_2$, we have $u_{ia} = 0$ for each $i \in S_1$ and each $a \in \mathbb{A}(i)$ such that $p_{ij}(a) > 0$. That is, for each possible transition from S_1 to S_2, the associated dual variable equals zero.*

Then, the solution u satisfies $u_{ja} = 0$ for each $j \in S_2$ and each $a \in \mathbb{A}(j)$.

Using the hypergraph flow interpretation of the linear program $(\mathrm{DL}_S^{i_0})$ given in Section 3.5.1, the statement of Lemma 3.5.5 is quite clear. Note that the partition of the state space S into S_1 and S_2 defines a cut in the associated hypergraph. Moreover, due to the assumptions made, there does not exist flow crossing the cut. Therefore, the flow vanishes completely in S_2 since there exists only a single source node in the hypergraph which is $i_0 \in S_1$. A rigorous proof of the lemma is given below.

Proof (Lemma 3.5.5). For each state $j \in S_2$, we have:

$$\sum_{a \in \mathbb{A}(j)} u_{ja} = \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u_{ia} = \alpha \sum_{i \in S_2} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u_{ia},$$

where the last equality is valid since $p_{ij}(a)u_{ia} > 0$ implies $i \in S_2$. Adding the equations above for each $j \in S_2$ gives:

$$\begin{aligned}\sum_{j\in S_2}\sum_{a\in\mathbb{A}(j)} u_{ja} &= \alpha \sum_{j\in S_2}\sum_{i\in S_2}\sum_{a\in\mathbb{A}(i)} p_{ij}(a)u_{ia} \\ &= \alpha \sum_{i\in S_2}\sum_{a\in\mathbb{A}(i)} u_{ia}\sum_{j\in S_2} p_{ij}(a)\end{aligned}$$

Note that $\sum_{j\in S_2} p_{ij}(a) \leq \sum_{j\in\mathbb{S}} p_{ij}(a) = 1$ for each $i \in S_2$. Therefore, we obtain:

$$\sum_{j\in S_2}\sum_{a\in\mathbb{A}(j)} u_{ja} \leq \alpha \sum_{i\in S_2}\sum_{a\in\mathbb{A}(i)} u_{ia},$$

which implies:

$$\sum_{j\in S_2}\sum_{a\in\mathbb{A}(j)} u_{ja} \leq 0.$$

This proves the lemma since the solution u is non-negative. □

Theorem 3.5.6 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, each feasible basic solution of the dual linear program $(\mathrm{DL}_S^{i_0})$ is a policy basic solution for some policy π for $M(S)$.*

Proof. First we prove that each feasible basic solution u of $(\mathrm{DL}_S^{i_0})$ that has at most one positive variable for each state, i. e.,

$$|\{(i,a) \in S \times \mathbb{A} \mid u_{ia} > 0\}| \leq 1 \quad \text{for each } i \in S, \tag{3.5.3}$$

is a policy basic solution. Let u' be such a feasible basic solution and let $S_1 \subseteq S$ be the set of states $i \in S$ that satisfy Property (3.5.3) with equality for u'. Moreover, for each state $i \in S_1$, define $a_i \in \mathbb{A}(i)$ to be the unique action at state i such that $u'_{ia_i} > 0$. Now consider any policy π for $M(S)$ with $\pi(i) = a_i$ for each $i \in S_1$, and let u^π be the policy basic solution of π. We will show that $u^\pi = u'$.

By definition of S_1 the basic solution u' is uniquely determined by the system:

$$u_{ja_j} - \alpha \sum_{i\in S_1} p_{ij}(a_i)u_{ia_i} = \begin{cases} 1, & \text{if } j = i_0, \\ 0, & \text{if } j \neq i_0, \end{cases} \tag{3.5.4}$$

$$u_{ja} = 0 \quad \text{for each } a \in \mathbb{A}(j) \setminus \{a_j\}, \tag{3.5.5}$$

for each $j \in S_1$. Obviously, by definition the policy basic solution u^π also satisfies the Equations (3.5.4) and (3.5.5) for each $j \in S_1$, implying that u^π

equals u' for all states in S_1. It remains to show that $u^{\pi}_{ia} = 0$ for each state $i \in S_2 := S \setminus S_1$ and each $a \in \mathbb{A}(i)$.

Because of the first constraint of $(\mathrm{DL}^{i_0}_S)$, we have $i_0 \in S_1$. Due to the remaining constraints of the linear program, the basic solution u' also satisfies:

$$\sum_{a \in \mathbb{A}(j)} u'_{ja} - \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u'_{ia} = 0,$$

for each $j \in S_2$. Thus, we obtain:

$$\sum_{i \in S_1} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u'_{ia} \leq \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u'_{ia} = 0,$$

since $u'_{ja} = 0$ for each $a \in \mathbb{A}(j)$. Particularly, for each $i \in S_1$ and $a \in \mathbb{A}(i)$, we have $u'_{ia} = 0$ if $p_{ij}(a) > 0$. Since u' equals u^{π} for each state in S_1, this property is also true for u^{π}. As the assumptions of Lemma 3.5.5 are satisfied for the policy basic solution u^{π} and the partition of S into S_1 and S_2, we conclude $u^{\pi}_{ia} = 0$ for each $i \in S_2$ and each $a \in \mathbb{A}(i)$. This completes the first part of the proof.

It remains to be proven that each feasible basic solution u of the linear program $(\mathrm{DL}^{i_0}_S)$ satisfies Property (3.5.3). Assume that there exists a basis $B \subseteq S \times \mathbb{A}$ such that the associated basic solution u^B has the following property:

$$\exists i' \in S \, \exists a_1, a_2 \in \mathbb{A}(i') : a_1 \neq a_2, (i', a_1), (i', a_2) \in B, u^B_{i'a_1}, u^B_{i'a_2} > 0. \quad (3.5.6)$$

Since B is a basis, there cannot exist a row of zeros in the matrix B. That is, for each state $j \in S$, at least one of the following conditions is true:

1. There exists an action $a(j) \in \mathbb{A}(j)$ such that $(j, a(j)) \in B$.
2. There exists $i(j) \in S$ and $a(j) \in \mathbb{A}(i(j))$ with $p_{i(j)j}(a(j)) > 0$ such that $(i(j), a(j)) \in B$.

We will also say that the basic columns *cover* all states in S by Condition 1 or Condition 2.

Due to Property (3.5.6), two basic columns cover state i' by the first condition. Therefore, it is impossible that for each state $j \in S$, there exists a basic column covering j by Condition 1 since $|B| = |S|$. Consequently, there exists a state $j' \in S$ that satisfies Condition 2, but not Condition 1.

Now consider the system of linear equations defining the vector of basic variables $\hat{u}^B$ of the basic solution u^B:

$$B\hat{u}^B = (1, 0, \ldots, 0)^t.$$

Note that $j' \neq i_0$, since otherwise the first equation of the system would imply $u^B_{i(j')a(j')} < 0$. Therefore, the equation for state j' in the system implies $u^B_{i(j')a(j')} = 0$ for each basic column $(i(j'), a(j'))$ that covers j' by Condition 2. Consequently, the system of linear equations defining $\hat{u}^B$ can be reduced as follows. For each basic column $(i(j'), a(j'))$ covering state j' by Condition 2, the corresponding basic variable is fixed to $u^B_{i(j')a(j')} = 0$ and column $(i(j'), a(j'))$ is removed from B.

In doing so, all non-zero entries in the row representing state j' vanish, creating at least one zero row. Let $k \geq 1$ be the number of removed columns, and let $B' \in \mathbb{R}^{S \times n}$ with $n = |S| - k$ be the obtained submatrix. Since B is nonsingular, the matrix B' has full column rank and the number of zero rows in matrix B' is at most k. Thus, by removing all zero rows from B', we obtain a matrix $B_1 \in \mathbb{R}^{m \times n}$ with $n \leq m < |S|$. Let $S_1 \subset S$ be the set of states represented by the rows of B_1. Due to $j' \neq i_0$ and Property (3.5.6), we have $i_0, i' \in S_1$.

Then, for each state $j \in S_1$ one of the Conditions 1 and 2 w. r. t. S_1 and B_1 instead of S and B are again satisfied. Due to $i' \in S_1$, Property (3.5.6), and the fact that the matrix B_1 has at least the same number of rows as columns, the construction above can be repeated. This can be done arbitrarily often, which contradicts the fact that the state space S is finite, completing the proof. □

Theorem 3.5.6 implies that similar as described in Section 2.2.4 for the original linear programming formulation, each basic solution of $(\mathrm{DL}^{i_0}_S)$ corresponds to (at least) one policy for the S-induced MDP $M(S)$.

Next we derive an explicit formula to compute the policy basic solution u^π of a given policy π for $M(S)$ for the linear program $(\mathrm{DL}^{i_0}_S)$, given any subset $S \subseteq \mathbb{S}$ and state $i_0 \in S$. The formula is given in terms of so-called π-induced paths as introduced in the following definition together with further types of paths that are used later.

Definition 3.5.7 (Induced path) Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a subset of states $S \subseteq \mathbb{S}$, a policy π for $M(S)$, and an integer $n \in \mathbb{N}$, let $P = (i_1, a_1, i_2, a_2, \ldots, a_{n-1}, i_n)$ be an (i_1, i_n)-path in M (cf. Definition 2.1.3). The *weight* of path P is defined by:

$$w(P) = \alpha^{n-1} \prod_{k=1}^{n-1} p_{i_k i_{k+1}}(a_k).$$

1. If $i_1, i_2, \ldots, i_n \in S$ and $a_k = \pi(i_k)$ for $k \in \{1, \ldots, n-1\}$, the path P is called π-*induced* w. r. t. S.

2. If $i_1, i_2, \ldots, i_{n-1} \in S$, $i_n \in \mathbb{S} \setminus S$, and $a_k = \pi(i_k)$ for $k \in \{1, \ldots, n-1\}$, the path P is called *extended π-induced* w. r. t. S.

3. If $i_1, i_2, \ldots, i_{n-1} \in S$, $i_n \in \mathbb{S}$, and $\underline{u}$ is an optimal solution of the dual linear program $(\mathrm{DL}_S^{i_0})$ with $\underline{u}_{i_k a_k} > 0$ for $k \in \{1, \ldots, n-1\}$, then the path P is called *$\underline{u}$-induced* w. r. t. S. △

Note that the weight of a path $(i_1, a_1, \ldots, a_{n-1}, i_n)$ equals α^{n-1} times the probability of the path, i. e., the probability that for each $k \in \{1, \ldots, n\}$, the state i_k is reached after $k-1$ steps starting from i_1, when the actions $a_1, \ldots, a_{n-1}$ are used at the state $i_1, \ldots, i_{n-1}$, respectively. Moreover, by Definition 2.1.3 this weight is always positive.

The explicit formula to determine a policy basic solution is as follows.

Theorem 3.5.8 *Consider an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, and a policy π for $M(S)$. Then, the policy basic solution u^π of π for the linear program $(\mathrm{DL}_S^{i_0})$ equals:*

$$u_{ia}^\pi = \begin{cases} \bar{u}_i^\pi, & \text{if } a = \pi(i), \\ 0, & \text{if } a \neq \pi(i), \end{cases} \tag{3.5.7}$$

for each $i \in S$ and each $a \in \mathbb{A}(i)$, where

$$\bar{u}_j^\pi = \sum_{\substack{\pi\text{-induced} \\ (i_0, j)\text{-path } P}} w(P) \qquad \forall j \in S. \tag{3.5.8}$$

Proof. Since u^π is the basic solution of $(\mathrm{DL}_S^{i_0})$ with the basic columns $(i, \pi(i))$ for each $i \in S$, it is clear that u^π must satisfy the Equation (3.5.7) for each $i \in S$ and each $a \in \mathbb{A}(i)$, where $\bar{u}^\pi$ is the unique solution of the following system of linear equations:

$$\bar{u}_j - \alpha \sum_{i \in S} p_{ij}(\pi(i)) \bar{u}_i = \begin{cases} 1, & \text{if } j = i_0, \\ 0, & \text{if } j \neq i_0, \end{cases} \quad \forall j \in S. \tag{3.5.9}$$

Denote the right hand side of the Equation (3.5.8) by x_j for each state $j \in S$. To prove the theorem, it is sufficient to show that $(x_j)_{j \in S}$ satisfies the system of linear equations (3.5.9). Let $j \in S$ be an arbitrary state. We distinguish two cases.

First consider the case $j \neq i_0$. Then, all π-induced paths from i_0 to j consist of at least two states. Thus, we have:

$$\begin{aligned} x_j &= \sum_{\substack{\pi\text{-induced} \\ (i_0, j)\text{-path } P}} w(P) \\ &= \sum_{i \in S} \left(\sum_{\substack{\pi\text{-induced } (i_0, j)\text{-path} \\ P = (i_0, \ldots, i, \pi(i), j)}} w(P) \right), \end{aligned}$$

where the states i_0 and i of $P = (i_0, \ldots, i, \pi(i), j)$ may coincide with each other. Obviously, the weight of path P equals $w(P) = \alpha p_{ij}(\pi(i)) \cdot w(P')$, where $P' = (i_0, \ldots, i)$. Thus, we obtain:

$$\begin{aligned} x_j &= \sum_{i \in S} \left(\alpha p_{ij}(\pi(i)) \underbrace{\sum_{\substack{\pi\text{-induced} \\ (i_0, i)\text{-path } P'}} w(P')}_{x_i} \right) \\ &= \alpha \sum_{i \in S} p_{ij}(\pi(i)) x_i. \end{aligned}$$

It remains to prove that $(x_j)_{j \in S}$ satisfies the system of equations (3.5.9) for $j = i_0$. In this case we have to distinguish between the trivial π-induced path (i_0) and longer paths. For the latter we can use the same construction as in the first case:

$$\begin{aligned} x_{i_0} &= \sum_{\substack{\pi\text{-induced} \\ (i_0, i_0)\text{-path } P}} w(P) \\ &= 1 + \sum_{\substack{\pi\text{-induced } (i_0, i_0)\text{-path } P \\ P \neq (i_0)}} w(P) \\ &= 1 + \sum_{i \in S} \left(\alpha p_{ii_0}(\pi(i)) \underbrace{\sum_{\substack{\pi\text{-induced} \\ (i_0, i)\text{-path } P'}} w(P')}_{x_i} \right) \\ &= 1 + \alpha \sum_{i \in S} p_{ii_0}(\pi(i)) x_i. \end{aligned}$$

This completes the proof since the solution of the system (3.5.9) is unique and $(x_j)_{j \in S}$ is a solution. □

We will use Theorem 3.5.8 in the next section, in order to prove an alternative formula for computing the reduced profits, which itself is employed in one

of the approximation heuristics and required for the proof of an alternative approach to obtain upper bounds for a component of the optimal value vector.

In the following, we look at general upper bounds on the variables of the dual linear program $\mathrm{DL}_S^{i_0}$ for any $S \subseteq \mathbb{S}$. The next theorem should be seen as a pure theoretical result as the bounds are not used in the column generation algorithm or otherwise.

Theorem 3.5.9 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let $S_0 := S$ and define $S_r \subseteq S$ for $r \in \mathbb{N}$ to be set of all states in S that cannot be reached from i_0 by at most $r-1$ transitions:*

$$S_r := S \setminus S(i_0, r-1) \quad \text{for } r \in \mathbb{N},$$

where $S(i_0, r)$ denotes the r–neighborhood of i_0 (cf. Definition 3.2.1). Then, for each feasible solution u of the dual linear program $(\mathrm{DL}_S^{i_0})$ and each $r \in \mathbb{N}_0$, we have:

$$\sum_{j \in S_r} \sum_{a \in \mathbb{A}(j)} u_{ja} \leq \frac{\alpha^r}{1-\alpha}. \tag{3.5.10}$$

Proof. The proof goes by induction on r. By adding all the constraints of the linear program $(\mathrm{DL}_S^{i_0})$ we obtain:

$$\begin{aligned}
\sum_{j \in S} \sum_{a \in \mathbb{A}(j)} u_{ja} &= 1 + \sum_{j \in S} \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u_{ia} \\
&= 1 + \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} u_{ia} \sum_{j \in S} p_{ij}(a) \\
&\leq 1 + \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} u_{ia}.
\end{aligned}$$

Due to $S_0 = S$ this implies:

$$\sum_{j \in S_0} \sum_{a \in \mathbb{A}(j)} u_{ja} \leq \frac{1}{1-\alpha}.$$

Now assume $r > 0$. Note that $i_0 \notin S_r$. Thus, similar as before, the constraints of $(\mathrm{DL}_S^{i_0})$ imply:

$$\begin{aligned}
\sum_{j \in S_r} \sum_{a \in \mathbb{A}(j)} u_{ja} &= \sum_{j \in S_r} \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u_{ia} \\
&= \alpha \sum_{j \in S_r} \sum_{i \in S_{r-1}} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u_{ia},
\end{aligned}$$

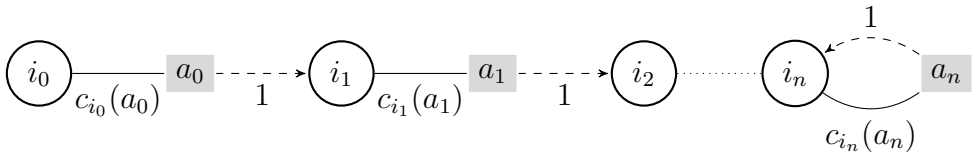


Figure 3.6: Markov decision process where the bounds for the dual variables of Theorem 3.5.9 are tight.

where the last equality is valid since for each $j \in S_r$, we have $p_{ij}(a) = 0$ for each $i \in S \setminus S_{r-1}$ and each $a \in \mathbb{A}(i)$. This implies by induction:

$$\begin{aligned}\sum_{j \in S_r} \sum_{a \in \mathbb{A}(j)} u_{ja} &= \alpha \sum_{i \in S_{r-1}} \sum_{a \in \mathbb{A}(i)} u_{ia} \sum_{j \in S_r} p_{ij}(a) \\ &\leq \alpha \sum_{i \in S_{r-1}} \sum_{a \in \mathbb{A}(i)} u_{ia} \\ &\leq \alpha \frac{\alpha^{r-1}}{1-\alpha} \\ &= \frac{\alpha^r}{1-\alpha},\end{aligned}$$

which completes the proof. □

It is easy to see that the bounds for the dual variables given in Theorem 3.5.9 cannot be improved in general. Consider the MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ given by the Markov decision process shown in Figure 3.6 for some integer $n \in \mathbb{N}_0$ and any discount factor $\alpha \in [0, 1)$. The unique solution u of the dual linear program $(\mathrm{DL}_S^{i_0})$ for $S = \mathbb{S}$ satisfies: $u_{i_k a_k} = \alpha^k$ for $k \in \{0, \ldots, n-1\}$ and $u_{i_n a_n} = \alpha u_{i_{n-1} a_{n-1}} + \alpha u_{i_n a_n}$, which gives $u_{i_n a_n} = \alpha^n/(1-\alpha)$. Therefore, we obtain:

$$\sum_{j \in S_m} \sum_{a \in \mathbb{A}(j)} u_{ja} = \sum_{k=m}^{n} u_{i_k a_k} = \frac{\alpha^m}{1-\alpha} \quad \text{for each } m \in \{0, \ldots, n-1\}.$$

Construction of Initial Bases

Recall that several linear programs have to be solved during the column generation process. Providing the solver with a good initial basis for each encountered linear program can obviously help to speed up the solution process.

We propose the following way to construct suitable initial bases. We only describe the approach for the linear programs $(\mathrm{L}_S^{i_0})$ with $S \subseteq \mathbb{S}$ providing the

lower bounds on $v_{i_0}^{\alpha}$. Constructing initial bases for the linear programs $(\mathrm{U}_S^{i_0})$ for $S \subseteq \mathbb{S}$ is done analogously. Firstly, having solved a linear program $(\mathrm{L}_{S'}^{i_0})$ for some subset $S' \subset \mathbb{S}$ in the previous iteration of the column generation, we maintain the obtained optimal basis B'. That is, a basis status value is stored for each variable and each constraint, i. e., the associated slack variables. Note that the size of the basis is $|S' \times \mathbb{A}|$. Now for solving the linear program $(\mathrm{L}_S^{i_0})$ for subset $S \subseteq \mathbb{S}$ with $S' \subset S$ in the next iteration, the stored optimal basis B' for $(\mathrm{L}_{S'}^{i_0})$ is extended as follows. For each state $i \in S \setminus S'$, the variable v_i is set to basic and we determine an action $a_i \in \mathbb{A}(i)$ such that:

$$a_i \in \operatorname*{argmin}_{a \in \mathbb{A}(i)} \left\{ \frac{c_i(a) + \alpha \sum_{j \in S'} p_{ij}(a) \underline{v}_j}{1 - \alpha p_{ii}(a)} \right\},$$

where $\underline{v}$ is the computed optimal solution of $(\mathrm{L}_{S'}^{i_0})$. Then, all constraints with indices $(i, a) \in (S \setminus S') \times \mathbb{A}$ for $a \neq a_i$ are set to basic. The idea of this construction is to guess for each state $i \in S \setminus S'$, one constraint (i, a) with $a \in \mathbb{A}(i)$ that is satisfied with equality in some optimal solution for $(\mathrm{L}_S^{i_0})$. Based on the obtained optimal solution $\underline{v}$ of $(\mathrm{L}_{S'}^{i_0})$, it is reasonable to choose the constraint (i, a_i) for each $i \in S \setminus S'$.

Note that the resulting number of variables and constraints set to basic equals:

$$\underbrace{|S' \times \mathbb{A}|}_{\text{old basis}} + \underbrace{|S \setminus S'|}_{\text{added variables}} + \underbrace{|(S \setminus S') \times \mathbb{A}| - |S \setminus S'|}_{\text{added constraints}} = |S \times \mathbb{A}|.$$

Moreover, the construction above indeed defines a basis B of the linear program $(\mathrm{L}_S^{i_0})$ for the following reason. Let $Q_B \in \mathbb{R}^{(S \times \mathbb{A}) \times (S \times \mathbb{A})}$ be the associated submatrix of Q^S. For each slack variable with index (i, a) with $i \in S \setminus S'$ that was set to basic, one can perform column operations to remove all other non-zero entries of B in row (i, a) without affecting the remaining matrix. Moreover, after suitably exchanging rows and columns, respectively, we obtain a matrix Q'_B with the following structure:

$$Q'_B = \begin{bmatrix} A & 0 \\ 0 & I \end{bmatrix},$$

where the submatrix A is strictly row diagonally dominant and I is the identity matrix. Therefore, the matrices Q'_B and also Q_B are non-singular.

We mention that the constructed basis B for the linear program $(\mathrm{L}_S^{i_0})$ is usually primal infeasible and may also be dual infeasible in general. In our context, however, the basis B is almost always dual feasible, and if not, a dual feasible basic solution is usually obtained after very few iterations

of the dual simplex method. This is due to the following reason. Having solved ($\mathrm{L}_{S'}^{i_0}$), we obtained a corresponding optimal basis N' for the associated dual linear program ($\mathrm{DL}_{S'}^{i_0}$) (defined by the non-basic primal variables and constraints). Recall that by Theorem 3.5.6 each optimal basic solution for ($\mathrm{DL}_{S'}^{i_0}$) corresponds to at least one policy for the induced MDP $M(S')$, i. e., for each state $i \in S'$, there exists exactly one action $a \in \mathbb{A}(i)$ such that the dual variable u_{ia} is basic. Even though the basis N' for ($\mathrm{DL}_{S'}^{i_0}$) may not correspond to a policy, it is usually quite close to a policy in the sense that this property is only violated for very few states. In the way we extend B', the associated dual basis to B is then very close to a policy for $M(S)$, too. Since a policy basic solution is feasible for the dual linear program by Theorem 3.5.3, the basic solution for B is never far away from being dual feasible in practice. Consequently, the dual simplex method should be applied when using the described construction for an initial basis. See Section 4.2.2 for associated computational experiments.

3.5.3 Pricing

In this section we address details of the pricing problem encountered in the column generation process for approximating the optimal value vector at the considered state i_0. The first part covers structural results related to the reduced profits including the construction of upper bounds on $v_{i_0}^{\alpha}$ within the column generation algorithm, while the second part introduces possible pricing strategies.

Structural Results

In the following, we derive a combinatorial formula for computing the reduced profit of a state not contained in the set $S \subset \mathbb{S}$ during the column generation, given an optimal basic solution of the current linear program ($\mathrm{DL}_{S}^{i_0}$). This formula, similar to that for the policy basic solution of a given policy for $M(S)$, is given in terms of extended induced paths.

Theorem 3.5.10 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let $\underline{u}$ be an optimal basic solution of the dual linear program ($\mathrm{DL}_{S}^{i_0}$) and let π be a policy for $M(S)$ such that for the policy basic solution u^{π}, we have $u^{\pi} = \underline{u}$. Then the reduced profit $\bar{p}_j$ of a state $j \in \mathbb{S} \setminus S$ w. r. t. solution $\underline{u}$ equals:*

$$\bar{p}_j = \sum_{\substack{\text{extended } \pi\text{-induced} \\ (i_0, j)\text{-path } P}} w(P). \tag{3.5.11}$$

Particularly, a state $j \in \mathbb{S} \setminus S$ has a positive reduced profit if and only if there exists an extended π-induced (i_0, j)-path, i. e., using policy π the state j can be reached from i_0.

Proof. By Equation (3.3.1) the reduced profit of a state $j \in \mathbb{S} \setminus S$ equals:

$$\begin{aligned}\bar{p}_j &= \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia} \\ &= \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u^{\pi}_{ia}\end{aligned}$$

By using the formula for computing the policy basic solution u^{π} given in the Equations (3.5.7) and (3.5.8) in Theorem 3.5.8, we obtain:

$$\begin{aligned}\bar{p}_j &= \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u^{\pi}_{ia} \\ &= \alpha \sum_{i \in S} p_{ij}(\pi(i)) \bar{u}^{\pi}_{i} \\ &= \alpha \sum_{i \in S} p_{ij}(\pi(i)) \sum_{\substack{\pi\text{-induced} \\ (i_0, i)\text{-path } P}} w(P) \\ &= \sum_{i \in S} \sum_{\substack{\pi\text{-induced} \\ (i_0, i)\text{-path } P}} \alpha p_{ij}(\pi(i)) w(P) \\ &= \sum_{\substack{\text{extended } \pi\text{-induced} \\ (i_0, j)\text{-path } P}} w(P).\end{aligned}$$

□

Obviously, it is inappropriate in general to use Equation (3.5.11) for computing the reduced profits since an infinite series has to evaluated. In Section 3.5.4 we will propose approximation heuristics that aim to determine a good subset of states before the actual column generation algorithm based on linear programming is applied. One of the approaches is based on the idea to approximate the infinite series in Equation (3.5.11) by a finite sum. Moreover, the formula is used in the construction of alternative upper bounds for a component of the optimal value vector.

The following theorem establishes the basis for a combinatorial pricing method that is described in the second part of this section.

Theorem 3.5.11 *Given an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let $\underline{u}$ be any (possibly non-basic) optimal solution of the dual linear program $(\mathrm{DL}^{i_0}_S)$. Then, a state $j \in \mathbb{S} \setminus S$ has a positive reduced profit $\bar{p}_j > 0$ if and only if there exists a $\underline{u}$-induced path from i_0 to j.*

Proof. Obviously, the existence of a $\underline{u}$-induced path $(i_0, a_0, \ldots, a_{n-1}, i_n = j)$ from i_0 to j for some $n \in \mathbb{N}$ implies a positive reduced profit $\bar{p}_j$ since by Equation (3.3.1) and $p_{i_{n-1}j}(a_{n-1}), \underline{u}_{i_{n-1}j} > 0$ we have:

$$\bar{p}_j = \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia} \geq \alpha p_{i_{n-1}j}(a_{n-1}) \underline{u}_{i_{n-1}j} > 0.$$

Now let $\bar{p}_j > 0$ and assume that there does not exist a $\underline{u}$-induced path from i_0 to j. Define $S_2 \subset S$ to be the set of all states $i \in S$ such that there exists a $\underline{u}$-induced path from i to j, i. e., $i_0 \notin S_2$. Let $S_1 := S \setminus S_2$. Particularly, we have $\underline{u}_{ia} = 0$ for each $i \in S_1$ and each $a \in \mathbb{A}(i)$ such that $p_{ij}(a) > 0$, which implies:

$$\bar{p}_j = \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia} = \alpha \sum_{i \in S_2} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia}. \tag{3.5.12}$$

Moreover, for each $j' \in S_2$, we have $\underline{u}_{ia} = 0$ also for each $i \in S_1$ and each $a \in \mathbb{A}(i)$ such that $p_{ij'}(a) > 0$. Thus, Lemma 3.5.5 implies $\underline{u}_{ia} = 0$ for each $i \in S_2$ and each $a \in \mathbb{A}(i)$. Therefore, by Equation (3.5.12) we have for the reduced profit $\bar{p}_j = 0$, which is a contradiction. □

Next we describe a way to obtain an upper bound on the component $v^\alpha_{i_0}$ of the optimal value vector within the column generation process that is different from solving the linear program $(\mathrm{U}^{i_0}_S)$ for the current subset S. Using this bound is quite cheap since it can be determined very easily while solving the pricing problem. The bound is related to the upper bound $\overline{v}^\pi_{i_0}$ for an optimal policy π for the S-induced MDP $M(S)$ that is computed by solving the system of linear equations (3.1.4) on page 47. First we prove that $\overline{v}^\pi_{i_0}$ can also be computed in terms of the reduced profits of all states contained in $\mathbb{S} \setminus S$, providing an alternative way to obtain this bound.

Theorem 3.5.12 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, 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 $M(S)$, let u^π be the optimal policy basic solution of π for $(\mathrm{DL}^{i_0}_S)$. Moreover, let $\bar{p}_j$ be the reduced profit of state j for each $j \in \mathbb{S} \setminus S$ w. r. t. solution u^π. Then, the solution $\overline{v}^\pi$ of the system (3.1.4) at state i_0 equals the optimal value $\underline{v}_{i_0}$ of $(\mathrm{L}^{i_0}_S)$ plus a weighted sum of the reduced profits:*

$$\overline{v}^\pi_{i_0} = \underline{v}_{i_0} + v^\alpha_{\max} \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j, \tag{3.5.13}$$

where again $v^\alpha_{\max} := \max_{i \in \mathbb{S}, a \in \mathbb{A}(i)} c_i(a)/(1-\alpha)$.

Proof. Recall that $\overline{v}^{\pi}$ is computed as the unique solution of the system of linear equations (3.1.4):

$$v_i - \alpha \sum_{j \in S} p_{ij}(\pi(i)) v_j = c_i(\pi(i)) + \alpha v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S} p_{ij}(\pi(i)), \quad i \in S.$$

Consider the decomposition of this system into:

$$v'_i - \alpha \sum_{j \in S} p_{ij}(\pi(i)) v'_j = c_i(\pi(i)), \quad i \in S. \tag{3.5.14}$$

and

$$v''_i - \alpha \sum_{j \in S} p_{ij}(\pi(i)) v''_j = \alpha v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S} p_{ij}(\pi(i)), \quad i \in S. \tag{3.5.15}$$

Note that by Theorem 2.2.1 the system of linear equations (3.5.14) computes the value vector of π for the MDP $M(S)$ (except for state i_{end} that will also be ignored in the following). Thus, we have $v' = v^{\alpha}_{M(S)}(\pi)$. Since π is optimal for $M(S)$, its value vector is even optimal, which implies $v' = v^{\alpha}_{M(S)}$. By Theorem 3.1.8 the optimal value vector $v^{\alpha}_{M(S)}$ equals the unique optimal solution $\underline{v}$ of the linear program (L^{Σ}_S), which is by Corollary 3.1.5 optimal for $(\mathrm{L}^{i_0}_S)$, too. Therefore, the solution to (3.5.14) satisfies $v'_{i_0} = \underline{v}_{i_0}$.

Obviously, there exists a unique solution to (3.5.15), too. Next we state the following claim:

Claim 3.1 *The solution to* (3.5.15) *gives* $v''_{i_0} = v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j$.

Since $v' + v''$ defines the solution to the system (3.1.4), the claim implies:

$$\overline{v}^{\pi}_{i_0} = \underline{v}_{i_0} + v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j,$$

which proves the theorem.

It remains to prove Claim 3.1. Let us define a further decomposition of system (3.5.15) indexed by the set $\mathbb{S} \setminus S$. For each $j' \in \mathbb{S} \setminus S$, we consider the following system of linear equations:

$$v^{(j')}_i - \alpha \sum_{j \in S} p_{ij}(\pi(i)) v^{(j')}_j = \alpha v^{\alpha}_{\max} p_{ij'}(\pi(i)), \quad i \in S. \tag{3.5.16}$$

Note that in the system (3.5.16) we only assume the upper bound of $v^{\alpha}_{\max}$ for state j', while all other states from $\mathbb{S} \setminus S$ are ignored. It is clear that adding the unique solutions of these systems gives the solution for (3.5.15), i. e., we have $v'' = \sum_{j' \in \mathbb{S} \setminus S} v^{(j')}$.

Next we look at the structure of the solution to the system (3.5.16).

Claim 3.2 *For each state $j' \in \mathbb{S} \setminus S$, the solution to* (3.5.16) *is given by:*

$$v_i^{(j')} = v_{\max}^{\alpha} \sum_{\substack{\text{extended } \pi\text{-induced} \\ (i,\,j')\text{-path } P}} w(P), \quad i \in S. \tag{3.5.17}$$

Particularly, the claim implies together with Theorem 3.5.10 that the solution value $v_{i_0}^{(j')}$ for the state i_0 equals $v_{i_0}^{(j')} = v_{\max}^{\alpha} \bar{p}_{j'}$. Since $v'' = \sum_{j' \in \mathbb{S} \setminus S} v^{(j')}$ defines the solution to the system (3.5.15), we obtain $v''_{i_0} = v_{\max}^{\alpha} \sum_{j' \in \mathbb{S} \setminus S} \bar{p}_{j'}$ once the solution formula of Equation (3.5.17) has been established. This will prove Claim 3.1.

So it remains to prove Claim 3.2. For a path P, denote again by $|P|$ its length, i. e., the number of state transitions in P. It can easily be verified that (3.5.17) defines the solution to (3.5.16) by checking the corresponding equations for each state $i \in S$:

$$\begin{aligned}
v_i^{(j')} &= v_{\max}^{\alpha} \sum_{\substack{\text{extended } \pi\text{-induced} \\ (i,\,j')\text{-path } P}} w(P) \\
&= \alpha v_{\max}^{\alpha} p_{ij'}(\pi(i)) + v_{\max}^{\alpha} \sum_{\substack{\text{extended } \pi\text{-induced} \\ (i,\,j')\text{-path } P\,:\,|P| \geq 2}} w(P) \\
&= \alpha v_{\max}^{\alpha} p_{ij'}(\pi(i)) + v_{\max}^{\alpha} \sum_{j \in S} \left(\alpha p_{ij}(\pi(i)) \sum_{\substack{\text{extended } \pi\text{-induced} \\ (j,\,j')\text{-path } P}} w(P) \right) \\
&= \alpha v_{\max}^{\alpha} p_{ij'}(\pi(i)) + \alpha \sum_{j \in S} \left(p_{ij}(\pi(i)) \underbrace{v_{\max}^{\alpha} \sum_{\substack{\text{extended } \pi\text{-induced} \\ (j,\,j')\text{-path } P}} w(P)}_{v_j^{(j')}} \right) \\
&= \alpha v_{\max}^{\alpha} p_{ij'}(\pi(i)) + \alpha \sum_{j \in S} p_{ij}(\pi(i)) v_j^{(j')}.
\end{aligned}$$

Note that the two cases $p_{ij'}(\pi(i)) = 0$ and $p_{ij'}(\pi(i)) > 0$ are both included in the considerations above. This proves Claim 3.2, which completes the proof. □

Recall that the column generation algorithm constructs a sequence of subsets of states $S_1 \subset S_2 \subset \cdots \subset S_n \subseteq \mathbb{S}$ for some $n \in \mathbb{N}$. However, since we do not solve the linear programs of type (L_S^{Σ}) but ($\mathrm{L}_S^{i_0}$), we do not necessarily obtain associated policies $\pi_1, \pi_2, \ldots, \pi_n$ that are optimal for the corresponding induced MDPs and no corresponding optimal policy basic solutions for ($\mathrm{DL}_S^{i_0}$), either (note that solving the linear program ($\mathrm{L}_S^{i_0}$) for

some subset $S \subseteq \mathbb{S}$ only determines optimal actions in the MDP $M(S)$ for those states that are reachable from i_0 via these actions). Therefore, the construction of Theorem 3.5.12 cannot be used directly to obtain an upper bound on $v^\alpha_{i_0}$.

Nevertheless, the value $\underline{v}_{i_0} + v^\alpha_{\max} \sum_{j \in \mathbb{S}\setminus S} \bar{p}_j$ can obviously also be computed if $\bar{p}_j$ is the reduced profit w. r. t. an arbitrary optimal solution of $(\mathrm{DL}^{i_0}_S)$. For this case we obtain the following corollary of Theorem 3.5.12.

Corollary 3.5.13 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let $\underline{u}$ be any (possibly non-basic) optimal solution for $(\mathrm{DL}^{i_0}_S)$ and let $\bar{p}_j$ be the reduced profit of state j for each $j \in \mathbb{S}\setminus S$ w. r. t. solution $\underline{u}$. Then, we have:*

$$\min_{\substack{\text{optimal policy} \\ \pi \text{ for } M(S)}} \overline{v}^\pi_{i_0} \leq \underline{v}_{i_0} + v^\alpha_{\max} \sum_{j \in \mathbb{S}\setminus S} \bar{p}_j, \tag{3.5.18}$$

where $\underline{v}_{i_0}$ is the optimal value of $(\mathrm{L}^{i_0}_S)$ and $\overline{v}^\pi$ is defined as the solution of the system (3.1.4)*.*

Proof. By Theorem 3.5.6 the optimal solution $\underline{u}$ of $(\mathrm{DL}^{i_0}_S)$ is a finite convex combination of optimal policy basic solutions $u^{\pi_1}, \ldots, u^{\pi_n}$ of optimal policies $\pi_1, \ldots, \pi_n$ for $M(S)$ for some $n \in \mathbb{N}$:

$$\underline{u} = \sum_{k=1}^{n} \lambda_k u^{\pi_k},$$

where $\lambda_1, \ldots, \lambda_n \geq 0$ and $\sum_{k=1}^n \lambda_k = 1$. Therefore, we obtain for the sum of the reduced profits w. r. t. solution $\underline{u}$:

$$\begin{aligned}
\sum_{j \in \mathbb{S}\setminus S} \bar{p}_j &= \sum_{j \in \mathbb{S}\setminus S} \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \underline{u}_{ia} \\
&= \sum_{j \in \mathbb{S}\setminus S} \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) \sum_{k=1}^{n} \lambda_k u^{\pi_k}_{ia} \\
&= \sum_{k=1}^{n} \lambda_k \sum_{j \in \mathbb{S}\setminus S} \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u^{\pi_k}_{ia} \\
&= \sum_{k=1}^{n} \lambda_k \sum_{j \in \mathbb{S}\setminus S} \bar{p}^k_j,
\end{aligned}$$

where $\bar{p}^k_j$ denotes the reduced profit of state $j \in \mathbb{S}\setminus S$ w. r. t. the policy basic solution u^{π_k} for $k \in \{1, \ldots, n\}$. Let k^* be any index such that the sum of

the reduced profits associated with the policy basic solution $u^{\pi_{k^*}}$ is smallest, i. e.,

$$k^* \in \operatorname*{argmin}_{k=1}^{n} \left\{ \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j^k \right\}.$$

Now we obtain:

$$\sum_{j \in \mathbb{S} \setminus S} \bar{p}_j \geq \sum_{k=1}^{n} \lambda_k \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j^{k^*} = \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j^{k^*}.$$

By Theorem 3.5.12 this implies:

$$\min_{\substack{\text{optimal policy} \\ \pi \text{ for } M(S)}} \overline{v}_{i_0}^{\pi} \leq \overline{v}_{i_0}^{\pi_{k^*}} = \underline{v}_{i_0} + v_{\max}^{\alpha} \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j^{k^*} \leq \underline{v}_{i_0} + v_{\max}^{\alpha} \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j,$$

which completes the proof. □

Since $\overline{v}_{i_0}^{\pi}$ is by Theorem 3.1.13 an upper bound on $v_{i_0}^{\alpha}$ for each optimal policy π for $M(S)$, Corollary 3.5.13 implies the following result.

Corollary 3.5.14 *Given an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$, a state $i_0 \in \mathbb{S}$, and a subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, let $\underline{u}$ be any optimal solution for $(\mathrm{DL}_S^{i_0})$ and let $\bar{p}_j$ be the reduced profit of state j for each $j \in \mathbb{S} \setminus S$ w. r. t. solution $\underline{u}$. Then, we have:*

$$\underline{v}_{i_0} \leq v_{i_0}^{\alpha} \leq \underline{v}_{i_0} + v_{\max}^{\alpha} \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j, \tag{3.5.19}$$

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

Thus, by computing all reduced profits in one step of the column generation process it is possible to determine an upper bound on the value vector at state i_0 without solving the linear program $(\mathrm{U}_S^{i_0})$. However, as mentioned before, these bounds are typically weaker than the ones provided by the linear programs.

Pricing Strategies

In this paragraph we address details of the pricing operation in the column generation algorithm. Again, let $S \subseteq \mathbb{S}$ be any subset of states with $i_0 \in S$, and assume that optimal solutions $\underline{v}$ and $\underline{u}$ of the linear programs $(\mathrm{L}_S^{i_0})$ and $(\mathrm{DL}_S^{i_0})$ are at hand. The task of the pricing operation is to determine one or several states in $\mathbb{S} \setminus S$ with positive reduced profit to be added to S, yielding

new variables and constraints in the considered linear programs. In doing so, the questions arise which states should be added to S and how they can be determined algorithmically. We will refer to the answers of both questions as *pricing strategies*.

To simplify notation, the following descriptions of possible pricing strategies assume that only one state is to be added to S in each pricing step. Adding several states per step is conceptually equivalent to adding several times one state without recomputing the reduced profits in between.

Recall that the reduced profit of a state $j \in \mathbb{S} \setminus S$ w. r. t. the optimal dual solution $\underline{u}$ is given by Equation (3.3.1):

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

Moreover, recall that the set of candidate states $S_{\text{cand}} \subseteq \mathbb{S} \setminus S$ which may have a positive reduced profit is given by:

$$S_{\text{cand}} = \{j \in \mathbb{S} \setminus S \mid \exists i \in S \, \exists a \in \mathbb{A}(i) \colon p_{ij}(a) > 0\}.$$

In the following, we propose four different pricing strategies for adding a state $j^* \in S_{\text{cand}}$ with positive reduced profit (details are described below):

- direct: For all states in S_{cand} the reduced profit is computed sequentially. The reduced profit $\bar{p}_{j^*}$ of the returned state j^* is maximum.

- combinatorial: Starting from i_0 and recursively following actions such that the corresponding dual variable values are positive, one reaches all states in S_{cand} with positive reduced profit (Theorem 3.5.11). See Algorithm 5 for details. Again, the reduced profit $\bar{p}_{j^*}$ of the returned state j^* is maximum.

- min-depth: For each state $j \in S_{\text{cand}}$, define its *depth* $d_j \in \mathbb{N}_0$ w. r. t. the reduced state space S and state i_0 as the minimum length of a path from i_0 to j, where all states in the path except for j are contained in S. The state j^* is an arbitrary state in S_{cand} with positive reduced profit such that d_{j^*} is minimum, i. e., $d_{j^*} = \min\{d_j \mid j \in S_{\text{cand}} : \bar{p}_j > 0\}$. The method to obtain j^* is similar to that for combinatorial.

- improving: For all states $j \in S_{\text{cand}}$, compute the reduced profit $\bar{p}_j$ and a weight w_j defined by:

$$w_j = \min_{a \in \mathbb{A}(j)} \left\{ \frac{c_j(a) + \alpha \sum_{i \in S} p_{ji}(a) \underline{v}_i}{1 - \alpha p_{jj}(a)} \right\}. \tag{3.5.20}$$

Algorithm 5 Pricing strategy combinatorial

Input: an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ (given implicitly), a state $i_0 \in \mathbb{S}$, a subset $S \subseteq \mathbb{S}$ with $i_0 \in S$, an optimal solution $\underline{u}$ of $(\mathrm{DL}_S^{i_0})$
Output: a state $j^* \in S_{\text{cand}}$ with $\bar{p}_{j^*} > 0$ and $\bar{p}_{j^*} \geq \bar{p}_j$ for each $j \in S_{\text{cand}}$, or the message that no such state exists
$S_{\text{vis}} \leftarrow \emptyset$
$(j^*, \bar{p}_{j^*}) \leftarrow \textsc{Explore}(i_0, S_{\text{vis}})$
if $\bar{p}_{j^*} > 0$ **then**
    return j^*
else
    return "no state with positive reduced profit"
end if
procedure $\textsc{Explore}(i, S_{\text{vis}})$
    $S_{\text{vis}} \leftarrow S_{\text{vis}} \cup \{i\}$
    if $i \in S$ **then** ▷ explore recursively for $a \in \mathbb{A}(i)$ with $\underline{u}_{ia} > 0$
        $j^* \leftarrow \text{nil}, \bar{p}_{j^*} \leftarrow 0$
        for each $a \in \mathbb{A}(i)$ with $\underline{u}_{ia} > 0$ **do**
            for each $j \in \mathbb{S} \setminus S_{\text{vis}}$ with $p_{ij}(a) > 0$ **do**
                $(j', \bar{p}_{j'}) \leftarrow \textsc{Explore}(j, S_{\text{vis}})$
                if $\bar{p}_{j'} > \bar{p}_{j^*}$ **then**
                    $j^* \leftarrow j', \bar{p}_{j^*} \leftarrow \bar{p}_{j'}$ ▷ update best candidate
                end if
            end for
        end for
        return $(j^*, \bar{p}_{j^*})$
    else ▷ candidate state found
        compute $\bar{p}_i = \alpha \sum_{j \in S} \sum_{a \in \mathbb{A}(j)} p_{ji}(a) \underline{u}_{ja}$
        return $(i, \bar{p}_i)$
    end if
end procedure

As we will see below, the weight w_j is a feasible value for the variable v_j in the linear program $(\mathrm{L}_{S \cup \{j\}}^{i_0})$. If there exists a state $j \in S_{\text{cand}}$ with $\bar{p}_j w_j > 0$, the returned state j^* has a maximum value $\bar{p}_{j^*} w_{j^*}$. Otherwise, the state j^* has a maximum reduced profit $\bar{p}_{j^*}$.

Note that the pricing strategies direct and combinatorial are conceptually equivalent as both strategies aim to find a state with a maximum reduced profit. They only differ in the way to determine such a state algorithmically. The pricing strategy combinatorial makes use of the fact that each state with

positive reduced profit is reachable from i_0 via a $\underline{u}$-induced path, as has be shown in Theorem 3.5.11. Moreover, if $\underline{u}$ is a basic solution, we have by Theorem 3.5.6 that there exists for each state $i \in S$ at most one action $a \in \mathbb{A}(i)$ such that the associated dual variable $\underline{u}_{ia}$ is positive. That is, in this case the loop in Step 14 of Algorithm 5 consists of one iteration only.

The strategy min-depth is very simple. Among all states with positive reduced profits any one with minimum depth w. r. t. the current state space S and i_0 is added. Thus, this pricing strategy completely ignores the actual amounts of the positive reduced profits.

There are two issues to be considered in finding states that help to improve the lower bound for $v_{i_0}^{\alpha}$. On the one hand, it is easy to see that there are situations such that for each $j \in S_{\text{cand}}$, adding only the single state j to S does not increase the optimal value of the linear program $(\mathrm{L}_S^{i_0})$ (neither adding all states with positive reduced profits may do so), see Example 3.5.15. On the other hand, the three pricing strategies direct, combinatorial, and min-depth do not take into account the feasible range of the new variable. That is, even though a state $j \in S_{\text{cand}}$ has a large reduced profit, the constraints of the linear program $(\mathrm{L}_{S\cup\{j\}}^{i_0})$ may imply a very small feasible upper bound on v_j that may even be zero, which results in a very little increase in the objective function.

The idea of the strategy improving is to overcome this second problem. It aims to find a state with significant reduced profit that also guarantees that the new variable can be set to a certain positive value. Recall that the constraints w. r. t. a state $j \in S_{\text{cand}}$ in the linear program $(\mathrm{L}_{S\cup\{j\}}^{i_0})$ read:

$$(1 - \alpha p_{jj})v_j - \alpha \sum_{i \in S} p_{ji}(a)v_i \leq c_j(a) \quad \forall a \in \mathbb{A}(j),$$

which implies:

$$v_j \leq \frac{c_j(a) + \alpha \sum_{i \in S} p_{ji}(a)v_i}{1 - \alpha p_{jj}} \quad \forall a \in \mathbb{A}(j),$$

Thus, by adding state $j \in S_{\text{cand}}$ the associated variable v_j can be set to any value within the interval $[0, w_j]$. The theory of linear programming implies that the optimal value may potentially increase by $\bar{p}_j w_j$. The strategy improving computes a state that maximizes this product. This way, the strategy seems to offer a better chance to increase the optimal value of $(\mathrm{L}_S^{i_0})$ for each state when being added to S. If there do not exist other restricting constraints, the increase $\bar{p}_j w_j$ is guaranteed. In general, however, the bottleneck in the linear program may change, forcing other inequalities to equality. This obviously restricts the possible increase of the optimal value. These issues are illustrated by the following example.

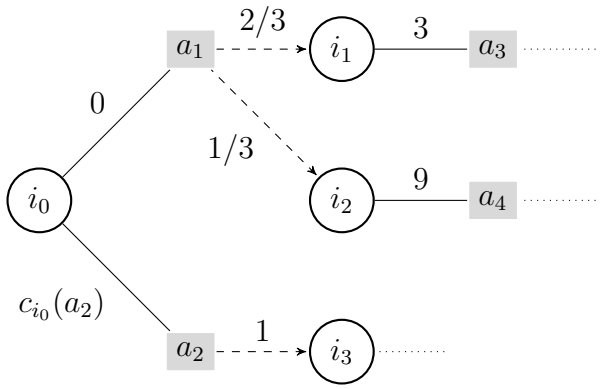


Figure 3.7: Part of a Markov decision process to illustrate the pricing strategy improving.

Example 3.5.15 Consider an MDP with the substructure shown in Figure 3.7, where the expected stage cost $c_{i_0}(a_2)$ will be used as a parameter. Moreover, assume that we have:

$$p_{i_1 i_0}(a_3) = p_{i_1 i_1}(a_3) = 0 \quad \text{and} \quad p_{i_2 i_0}(a_4) = p_{i_2 i_2}(a_4) = 0.$$

Let $S := \{i_0\}$, then the primal linear program $(\mathrm{L}_S^{i_0})$ simply reads:

$$\begin{aligned} \max \quad & v_{i_0} \\ \text{subject to} \quad & v_{i_0} \leq 0 && (i_0, a_1) \\ & v_{i_0} \leq c_{i_0}(a_2) && (i_0, a_2) \\ & v_{i_0} \in \mathbb{R}, \end{aligned}$$

and the optimal value equals 0. The associated dual linear program $(\mathrm{DL}_S^{i_0})$ looks as follows:

$$\begin{aligned} \min \quad & 0u_{i_0 a_1} + c_{i_0}(a_2) u_{i_0 a_2} \\ \text{subject to} \quad & u_{i_0 a_1} + u_{i_0 a_2} = 1 \\ & u_{i_0 a_1}, u_{i_0 a_2} \geq 0. \end{aligned}$$

Independently of the actual value of $c_{i_0}(a_2)$, an optimal solution for $(\mathrm{DL}_S^{i_0})$ is obviously given by $\underline{u}_{i_0 a_1} = 1$ and $\underline{u}_{i_0 a_2} = 0$. Consequently, we have by Equation (3.3.1) for the reduced profits:

$$\begin{aligned} \bar{p}_{i_1} &= \tfrac{2\alpha}{3}, \\ \bar{p}_{i_2} &= \tfrac{\alpha}{3}, \\ \bar{p}_{i_3} &= 0. \end{aligned}$$

Moreover, the weights of the states i_1 and i_2 as defined by Equation (3.5.20) equal $w_{i_1} = 3$ and $w_{i_2} = 9$, which gives $\bar{p}_{i_1} w_{i_1} = 2\alpha$ and $\bar{p}_{i_2} w_{i_2} = 3\alpha$. Therefore, the strategy improving adds state i_2 to the state space S. We obtain the following linear program $(\mathrm{L}^{i_0}_{S\cup\{i_2\}})$:

$$\begin{aligned} \max \quad & v_{i_0} \\ \text{subject to} \quad & v_{i_0} - \tfrac{\alpha}{3} v_{i_2} \leq 0 && (i_0, a_1) \\ & v_{i_0} \leq c_{i_0}(a_2) && (i_0, a_2) \\ & v_{i_2} \leq 9 && (i_2, a_4) \\ & v_{i_0}, v_{i_2} \in \mathbb{R}, \end{aligned}$$

which has an optimal value of $\underline{v}_{i_0} = \min\{c_{i_0}(a_2), 3\alpha\} = \min\{c_{i_0}(a_2), \bar{p}_{i_2} w_{i_2}\}$.

Thus in the case $c_{i_0}(a_2) = 0$, adding state i_2 did not improve the optimal value of the linear program providing a lower bound on $v^\alpha_{i_0}$. Note that in this situation it is impossible to increase the optimal value even by adding all states with positive reduced profits to S.

Now consider the case that the stage cost satisfies $c_{i_0}(a_2) \geq 3\alpha$. Then the full potential $\bar{p}_{i_2} w_{i_2} = 3\alpha$ for increasing the optimal value will be exploited. Notice that the pricing strategies direct and combinatorial select state i_1 instead of i_2, which only increases the optimal value by $\bar{p}_{i_1} w_{i_1} = 2\alpha$. Hence, the strategy improving increases the lower bound for the component $v^\alpha_{i_0}$ of the optimal value vector to a greater extend. △

We study the computational practicability of the proposed pricing strategies in Section 4.2.3 by presenting numerical results for different MDPs.

By Corollary 3.2.4 we know that for a given guarantee $\varepsilon > 0$, the construction of the r–neighborhood $S(i_0, r)$ of i_0 for a suitable radius $r \in \mathbb{N}$ provides an ε-approximation on $v^\alpha_{i_0}$ by the optimal values $\underline{v}_{i_0}$ and $\overline{v}_{i_0}$ of the linear programs $(\mathrm{L}^{i_0}_{S(i_0,r)})$ and $(\mathrm{U}^{i_0}_{S(i_0,r)})$, respectively, i. e., $\overline{v}_{i_0} - \underline{v}_{i_0} \leq \varepsilon$. In the following we show that for every pricing strategies except min-depth, the associated column generation algorithm may require more states to achieve guarantee ε than the neighborhood construction of Corollary 3.2.4.

Example 3.5.16 Consider an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ with the partial structure shown in Figure 3.8 that satisfies $c_{\max} = \max_{i\in\mathbb{S}, a\in\mathbb{A}(i)} c_i(a) = 1$, i. e., $v^\alpha_{\max} = 1/(1-\alpha)$. Moreover, for the transition probabilities p and q, we assume:

$$\begin{aligned} p &= \tfrac{1}{1+\alpha} + \beta \\ q &= \tfrac{\alpha}{1+\alpha} - \beta, \end{aligned}$$

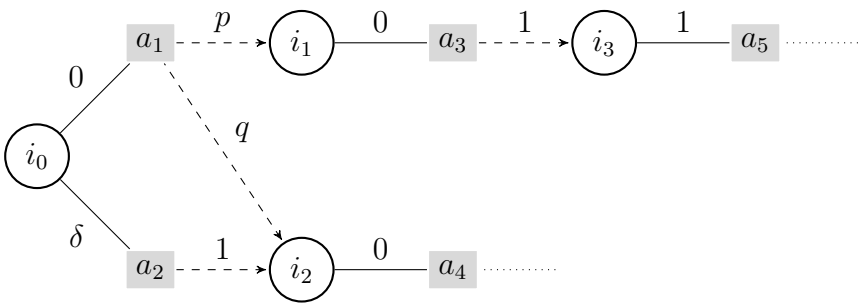


Figure 3.8: Part of a Markov decision process with the following property: in order to achieve a given approximation guarantee, the column generation algorithm requires more states than needed by the theoretical construction due to Corollary 3.2.4.

for any value $0 < \beta < \alpha/(1+\alpha)$. Notice that the choice of β implies $0 < p, q < 1$ and $p + q = 1$. Moreover, let $0 < \delta < \alpha q$. We consider an approximation guarantee of $\varepsilon = \alpha^2/(1-\alpha)$.

On the one hand, by Corollary 3.2.4 we obtain:

$$r = \left\lceil \log\left(\frac{\varepsilon}{v^{\alpha}_{\max}}\right) / \log \alpha \right\rceil - 1 = \left\lceil \log\left(\alpha^2\right) / \log \alpha \right\rceil - 1 = 1$$

for the radius of the required neighborhood to obtain the ε-approximation for the value $v^{\alpha}_{i_0}$. The associated 1–neighborhood equals $S' := S(i_0, 1) = \{i_0, i_1, i_2\}$ and yields the bounds $\underline{v}_{i_0} = 0$ and $\overline{v}_{i_0} = \alpha^2/(1-\alpha)$ for the optimal values of the linear programs $(\mathrm{L}^{i_0}_{S'})$ and $(\mathrm{U}^{i_0}_{S'})$. Thus, we indeed have for the approximation $\overline{v}_{i_0} - \underline{v}_{i_0} = \varepsilon$.

On the other hand, let us consider the column generation algorithm where the initial subset of states equals $S = \{i_0\}$, only one state is added to S per pricing step, and the pricing strategy is different from min-depth. In the first iteration the only optimal solution of the dual linear program $(\mathrm{DL}^{i_0}_{S})$ equals $\underline{u}_{i_0 a_1} = 1$ and $\underline{u}_{i_0 a_2} = 0$ which gives $\bar{p}_{i_1} = \alpha p$ and $\bar{p}_{i_2} = \alpha q$. Since we have $p > q$, all considered pricing strategies add state i_1 to S. In the second iteration we obtain for the dual prices $\underline{u}_{i_0 a_1} = 1$, $\underline{u}_{i_0 a_2} = 0$, and $\underline{u}_{i_1 a_3} = \alpha p$, which implies $\bar{p}_{i_2} = \alpha q$ and $\bar{p}_{i_3} = \alpha^2 p$. By $\alpha p > q$ each of the pricing strategies now add state i_3 to S.

Let us evaluate the approximation guarantee achieved by the resulting subset of states $S = \{i_0, i_1, i_3\}$. Note that $\delta < \alpha^2 p$ since $\delta < \alpha q$ and $q < \alpha p$. Thus, for the optimal value $\underline{v}_{i_0}$ of $(\mathrm{L}^{i_0}_{S})$ we obtain $\underline{v}_{i_0} = \min\{\delta, \alpha^2 p\} = \delta$. Furthermore, the optimal value $\overline{v}_{i_0}$ of $(\mathrm{U}^{i_0}_{S})$ equals:

$$\overline{v}_{i_0} = \min\left\{\delta + \tfrac{\alpha}{1-\alpha}, \alpha^2 p + \tfrac{\alpha^3}{1-\alpha} p + \tfrac{\alpha}{1-\alpha} q\right\}.$$

Since we have:

$$\begin{aligned}\alpha^2 p + \tfrac{\alpha^3}{1-\alpha} p + \tfrac{\alpha}{1-\alpha} q &= \tfrac{\alpha}{1-\alpha}\left(\alpha p - \alpha^2 p + \alpha^2 p + q\right)\\ &= \tfrac{\alpha}{1-\alpha}\left(\alpha p + q\right)\\ &< \tfrac{\alpha}{1-\alpha}\left(p + q\right)\\ &= \tfrac{\alpha}{1-\alpha}\\ &< \delta + \tfrac{\alpha}{1-\alpha},\end{aligned}$$

the minimum is attained for the second term and we obtain:

$$\overline{v}_{i_0} = \alpha^2 p + \tfrac{\alpha^3}{1-\alpha} p + \tfrac{\alpha}{1-\alpha} q.$$

Therefore, the approximation guarantee obtained by the subset S satisfies:

$$\begin{aligned}\overline{v}_{i_0} - \underline{v}_{i_0} &= \alpha^2 p + \tfrac{\alpha^3}{1-\alpha} p + \tfrac{\alpha}{1-\alpha} q - \delta\\ &> \alpha^2 p + \tfrac{\alpha^3}{1-\alpha} p + \tfrac{\alpha}{1-\alpha} q - \alpha q\\ &= \left(\tfrac{\alpha^2 - \alpha^3 + \alpha^3}{1-\alpha}\right) p + q \left(\tfrac{\alpha - \alpha + \alpha^2}{1-\alpha}\right)\\ &= \tfrac{\alpha^2}{1-\alpha} p + \tfrac{\alpha^2}{1-\alpha} q\\ &= \tfrac{\alpha^2}{1-\alpha}\\ &= \varepsilon.\end{aligned}$$

That is, the subset S consisting of three states does not suffice to obtain an ε-approximation on $v^\alpha_{i_0}$, whereas the 1–neighborhood $S(i_0, 1)$ does so. Thus, in the described situation the column generation algorithm has to generate more states than the neighborhood construction in order to achieve the desired approximation guarantee. △

We mention that the Example 3.5.16 can easily be extended for the case $\varepsilon = \alpha^n/(1-\alpha)$ with $n \in \mathbb{N}$ and $n > 2$, still giving the same outcome. Although the approximation via the neighborhood construction may be advantageous compared to that using the column generation algorithm in certain situations, usually the latter outperforms the former significantly. See Section 4.2.1 for computational results.

3.5.4 Approximation Heuristics

Similar to other applications of column generation, it is also convenient in our context to apply heuristics. Here the aim of an approximation heuristic

is to accelerate the overall approximation process by (quickly) constructing a good estimate of the required state space before the main column generation algorithm is employed.

The approach for using approximation heuristics in our context is the following. Once an initial subset of states $S_1 \subset \mathbb{S}$ with $i_0 \in S_1$ has been constructed by some heuristic, one can obtain lower and upper bounds $\underline{v}_{i_0}$ and $\overline{v}_{i_0}$ on the value $v_{i_0}^{\alpha}$ by solving the associated linear programs $(\mathrm{L}_{S_1}^{i_0})$ and $(\mathrm{U}_{S_1}^{i_0})$. That is, the approximation guarantee achieved by the heuristic can be evaluated. Depending on this information it may be beneficial to continue applying the heuristic. In doing so, the heuristic should be parameterized in terms of the known bounds $\underline{v}_{i_0}$ and $\overline{v}_{i_0}$. This procedure may be iterated until the heuristic has constructed a state space that yields some predefined approximation guarantee, which may even be the desired guarantee for the overall process. Thus, an approximation heuristic may be used in the spirit of a start heuristic as well as an improving heuristic. Obviously, it will depend on the considered MDP and the approximation heuristic itself to what extend employing the heuristic is advantageous. In designing an approximation heuristic it is crucial to find a suitable criterion to switch from the heuristic to the column generation algorithm. We will speak of a *heuristic stopping criterion*.

The simplest idea for an approximation heuristic is to initially construct a subset of states $S_1 \subseteq \mathbb{S}$ consisting of all states that can be reached from the state $i_0 \in \mathbb{S}$ with at most r transitions for some $r \in \mathbb{N}$. The resulting state space equals the r–neighborhood $S(i_0, r)$ as specified in Definition 3.2.1. Obviously, one can continue this heuristic when the approximation on $v_{i_0}^{\alpha}$ obtained by $S(i_0, r)$ does not satisfy the heuristic stopping criterion by generating $S(i_0, r')$ for some radius $r' > r$. We will call this generic approximation heuristic breadth-first-exploration.

Obviously, the main drawback of the heuristic breadth-first-exploration is that all states in the same depth w. r. t. state i_0 (cf. Definition 2.1.3 on page 11) are considered equally relevant for approximating $v_{i_0}^{\alpha}$, which is usually not the case. One possibility to partially overcome this disadvantage is to restrict breadth-first-exploration according to a particular policy, which leads to a so-called *policy-based approximation heuristic*.

Policy-Based Approximation Heuristics

Let $\varepsilon > 0$ be the desired approximation guarantee, and let π be a particular policy for the considered MDP. The goal of a policy-based approximation heuristic is to determine an initial subset of states $S_1 \subseteq \mathbb{S}$ with $i_0 \in S_1$ such

Algorithm 6 Approximation using a policy-based heuristic

1: **Input:** an MDP $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ (given implicitly), a state $i_0 \in \mathbb{S}$, a policy π for M, a generation heuristic heur, $\varepsilon > 0$
2: **Output:** lower and upper bounds $\underline{v}_{i_0}, \overline{v}_{i_0}$ on $v_{i_0}^\alpha$ with $(\overline{v}_{i_0} - \underline{v}_{i_0})/\underline{v}_{i_0} \leq \varepsilon$
3: let $M(\pi)$ be the restriction of M for π according to Theorem 3.4.1
4: use heur in the MDP $M(\pi)$ to obtain a subset of states $S_1 \subseteq \mathbb{S}$ such that the optimal values $\underline{v}_{i_0}(\pi)$ and $\overline{v}_{i_0}$ of the linear programs $(\mathrm{L}_{S,\pi}^{i_0})$ and $(\mathrm{U}_S^{i_0})$, respectively, satisfy:

$$\frac{\overline{v}_{i_0} - \underline{v}_{i_0}(\pi)}{\underline{v}_{i_0}(\pi)} \leq \varepsilon$$

5: apply Algorithm 4 with initial subset of states S_1

that we have:

$$\frac{\overline{v}_{i_0} - \underline{v}_{i_0}(\pi)}{\underline{v}_{i_0}(\pi)} \leq \varepsilon, \tag{3.5.21}$$

for the optimal value $\underline{v}_{i_0}(\pi)$ of the linear program $(\mathrm{L}_{S_1,\pi}^{i_0})$ described in Section 3.4.1 and the optimal value $\overline{v}_{i_0}$ of $(\mathrm{U}_{S_1}^{i_0})$. Recall that $\underline{v}_{i_0}(\pi)$ is a lower bound for the component $v_{i_0}^\alpha(\pi)$ of the value vector of policy π, while $\overline{v}_{i_0}$ is an upper bound on $v_{i_0}^\alpha$. We will argue later why it is advantageous to consider an upper bound on $v_{i_0}^\alpha$ instead of one for $v_{i_0}^\alpha(\pi)$.

Any method to construct a subset $S_1 \subseteq \mathbb{S}$ with $i_0 \in S_1$ satisfying Inequality (3.5.21) may be used. We refer to this method as a *generation heuristic*, each of which leads to a different policy-based heuristic. Clearly, it will typically take several iterations of the employed generation heuristic until the constructed state space S_1 fulfills Inequality (3.5.21). Note that each iteration requires two linear programs to be solved. Once the final state space S_1 has been obtained, the main approximation is carried out by the column generation algorithm. The concept for incorporating a policy-based heuristic in the main approximation process is summarized in Algorithm 6.

In using a policy-based heuristic, we hope that

1. the generation heuristic quickly computes a subset $S_1 \subseteq \mathbb{S}$ with $i_0 \in S_1$ such that Inequality (3.5.21) is satisfied,

2. the subset S_1 only contains few states not needed by the pure column generation algorithm to approximate $v_{i_0}^\alpha$,

3. and that the state space S_1 is already close to a subset of states which is required for the approximation of $v_{i_0}^\alpha$.

It is quite clear that these three properties are desirable as they will lead to a fast overall approximation process. In this context the following issues are to be taken into account. Solving linear programs of the type $(\mathrm{L}^{i_0}_{S,\pi})$ for some $S \subseteq \mathbb{S}$ with $i_0 \in S$ is easy since due to Corollary 3.4.2 an optimal solution can also be obtained by solving a system of linear equations. This obviously helps to achieve the first property. Nevertheless, solving the two linear programs is still quite time-consuming, which makes it beneficial to do so rarely by the policy-based heuristic.

Note further that for each subset $S \subseteq \mathbb{S}$ with $i_0 \in S$, the linear program $(\mathrm{U}^{i_0}_{S,\pi})$ is a relaxation of $(\mathrm{U}^{i_0}_{S})$. Therefore, we have for the associated optimal values $\overline{v}_{i_0}(\pi) \geq \overline{v}_{i_0}$. The difference between $\overline{v}_{i_0}(\pi)$ and $\overline{v}_{i_0}$ may be significant, even if the policy π is optimal. This is the reason for using upper bounds on $v^{\alpha}_{i_0}$ instead of $v^{\alpha}_{i_0}(\pi)$ in Inequality (3.5.21) since a policy-based heuristic may generate too many states otherwise. Obviously, such a behavior would be contrary to the second property above.

Usually, states that can be reached from i_0 by using good policies tend to be more important for the approximation of $v^{\alpha}_{i_0}$ than those reached by bad policies. Therefore, the second and the third property depend on the quality of the used policy π. Fortunately, for many MDPs, especially those arising in our context, often a policy is known that performs quite well.

Now assume that the considered policy-based heuristic terminates having constructed a state space $S_1 \subseteq \mathbb{S}$ with $i_0 \in S_1$. Let $\underline{v}_{i_0}(\pi)$ and $\overline{v}_{i_0}$ be the associated final bounds satisfying Inequality (3.5.21). Obviously, the value $\underline{v}_{i_0}(\pi)$ does not need to be a lower bound on $v^{\alpha}_{i_0}$, unless policy π is optimal. Switching to the main approximation process means to initially compute the valid lower bound on $v^{\alpha}_{i_0}$ w. r. t. subset S_1 as the optimal value $\underline{v}_{i_0}$ of the linear program $(\mathrm{L}^{i_0}_{S})$. Since $(\mathrm{L}^{i_0}_{S,\pi})$ is a relaxation of $(\mathrm{L}^{i_0}_{S})$, we have $\underline{v}_{i_0} \leq \underline{v}_{i_0}(\pi)$, which implies:

$$\frac{\overline{v}_{i_0} - \underline{v}_{i_0}}{\underline{v}_{i_0}} \geq \frac{\overline{v}_{i_0} - \underline{v}_{i_0}(\pi)}{\underline{v}_{i_0}(\pi)}.$$

Thus, the main approximation goal, i. e., $(\overline{v}_{i_0} - \underline{v}_{i_0})/\underline{v}_{i_0} \leq \varepsilon$, is usually not reached yet. Even if the policy π is optimal, we mostly still have $\underline{v}_{i_0} < \underline{v}_{i_0}(\pi)$ although $\underline{v}_{i_0}(\pi)$ is a valid lower bound on $v^{\alpha}_{i_0}$. We propose the following policy-based heuristics for a given policy π.

breadth-first-policy-exploration A simple policy-based heuristic can be derived from the heuristic breadth-first-exploration by restricting the construction w. r. t. a given policy π. That is, breadth-first-exploration is applied to the restricted MDP according to π as described in Section 3.4.1 to obtain lower bounds on $v^{\alpha}_{i_0}(\pi)$. We call this heuristic breadth-first-policy-exploration.

Note that for given $r \in \mathbb{N}$ breadth-first-policy-exploration constructs a subset $S(i_0, r, \pi)$ of the r–neighborhood $S(i_0, r)$, where $S(i_0, 0, \pi) := \{i_0\}$ and:

$$S(i_0, r, \pi) := S(i_0, r-1, \pi) \cup \{j \in \mathbb{S} \mid \exists i \in S(i_0, r-1, \pi) \colon p_{ij}(\pi(i)) > 0\},$$

for each $r \in \mathbb{N}$. A precise implementation of breadth-first-policy-exploration is given in Section 4.2.4 along with computational results.

However, the mentioned disadvantage of breadth-first-exploration is only partially resolved by restricting to policy π as the heuristic does not take into account transition probabilities, but solely the number of transitions for reaching a state by π. This observation motivates the following heuristics.

pricing-policy-exploration As described in Section 3.4.1, the column generation algorithm can be applied to approximate from below the component $v_{i_0}^{\alpha}(\pi)$ of the value vector of policy π. We will refer to the policy-based heuristic based on this generation heuristic as pricing-policy-exploration. The motivation for using pricing-policy-exploration is to benefit from the easier approximation of $v_{i_0}^{\alpha}(\pi)$ due to the possibility to faster solve the linear programs providing the lower bounds. However, this heuristic may still be a bit slow since new states are generated quite costly and usually many linear programs are solved.

weighting-policy-exploration The next policy-based heuristic can be seen as an approximation of pricing-policy-exploration. The idea is to generate preferably the same states, but without having to compute the reduced profits of the candidate states that could be added next to the reduced state space. This way, one can refrain from solving many linear programs since their solutions are only needed to check the heuristic stopping criterion given by Inequality (3.5.21).

Let $S \subseteq \mathbb{S}$ be a subset of states with $i_0 \in S$. Consider the policy basic solution of the given policy π for the dual linear program to $(\mathrm{L}_{S,\pi}^{i_0})$. By Theorem 3.5.10 the reduced profit $\bar{p}_j$ of a state $j \in \mathbb{S} \setminus S$ w. r. t. this basic solution equals:

$$\bar{p}_j = \sum_{\substack{\text{extended } \pi\text{-induced} \\ (i_0, j)\text{-path } P}} w(P).$$

We call a π-induced path $P = (i_1, \pi(i_1), i_2, \pi(i_2), \ldots, i_n = j)$ for some $n \in \mathbb{N}$ *acyclic* if $i_h \neq i_k$ for each $h, k \in \{1, \ldots, n\}$ with $h \neq k$. Instead of using the exact values for the reduced profits, the heuristic weighting-policy-exploration approximates the formula above by taking into account solely the weight of acyclic paths. Moreover, weighting-policy-exploration only considers a proper

subset of all acyclic paths, as we will see soon. This way, the heuristic computes lower bounds on the reduced profits.

The heuristic weighting-policy-exploration maintains a weight w_j for each candidate state $j \in \mathbb{S} \setminus S_1$ that should approximate the reduced profit $\bar{p}_j$. Initially, we start with the subset $S_1 = \{i_0\}$. Moreover, for each $j \in \mathbb{S} \setminus S_1$ that is a successor of i_0, we set its weight to $w_j = \alpha p_{i_0 j}(\pi)$, which obviously equals the weight of the path $(i_0, \pi(i_0), j)$. The weight of each other state $j \in \mathbb{S} \setminus S_1$ is initialized by $w_j = 0$. Then, in each iteration of the heuristic a state i with maximum weight w_i is added to the subset S_1 and the weights of its successor states are updated as follows. The weight w_j of each successor $j \in \mathbb{S} \setminus S_1$ of state i is increased by $\alpha p_{ij}(\pi) w_i$, which reflects the total weight of all (i_0, i)-paths considered by w_i extended by the transition leading to state j. Note that this update operation ignores all acyclic (i_0, j)-paths that are given by concatenating an (i_0, i)-path considered by w_i and a path $(i, \pi(i), i_1, \pi(i_1), \ldots, i_n, \pi(i_n), j)$ for $n \in \mathbb{N}$ such that $i_1, \ldots, i_n \in S_1$.

A crucial aspect in making the heuristic weighting-policy-exploration run fast is the question at which points the linear programs $(\mathrm{L}^{i_0}_{S_1,\pi})$ and $(\mathrm{U}^{i_0}_{S_1})$ should be solved for the current subset S_1 in order to check for Inequality (3.5.21). On the one hand, evaluating Inequality (3.5.21) too often will usually result in bad running times since many linear programs are to be solved. On the other hand, doing this check rarely may imply that the constructed states space becomes too large.

Note that by the Theorems 3.5.12 and 3.1.13 we have for each subset $S \subseteq \mathbb{S}$ with $i_0 \in S$:

$$\overline{v}_{i_0}(\pi) \leq \underline{v}_{i_0}(\pi) + v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S} \bar{p}_j,$$

where $\underline{v}_{i_0}(\pi)$ and $\overline{v}_{i_0}(\pi)$ are the optimal values of $(\mathrm{L}^{i_0}_{S,\pi})$ and $(\mathrm{U}^{i_0}_{S,\pi})$, respectively. That is, the absolute approximation guarantee $d_{\mathrm{abs}} := \overline{v}_{i_0}(\pi) - \underline{v}_{i_0}(\pi)$ w. r. t. policy π obtained by subset S_1 is bounded by $d_{\mathrm{abs}} \leq v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S_1} \bar{p}_j$. The approach of the heuristic weighting-policy-exploration is to derive an estimate for the relative difference $d_{\mathrm{rel}} := (\overline{v}_{i_0} - \underline{v}_{i_0}(\pi))/\underline{v}_{i_0}(\pi)$. However, the value d_{rel} is obviously hardly related to d_{abs} for the following reasons:

1. The relative difference d_{rel} considers the upper bound $\overline{v}_{i_0}$ for $v^{\alpha}_{i_0}$, while the value d_{abs} uses a bound on $v^{\alpha}_{i_0}(\pi)$.

2. Without taking into account a guess for $\underline{v}_{i_0}(\pi)$ the absolute difference d_{abs} has nothing to do with the relative difference d_{rel} in general.

We use the term $\delta v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S_1} w_j$ as estimate for the relative approximation guarantee d_{rel} for some value $\delta > 0$. Here the factor δ is used to outweigh

the estimation error due to the reasons above plus the fact that $\bar{p}_j \geq w_j$ for each state $j \in \mathbb{S} \setminus S_1$.

Note that the total weight $\sum_{j \in \mathbb{S} \setminus S_1} w_j$ decreases whenever a state $i \in \mathbb{S} \setminus S_1$ is added to the subset S_1: the sum decreases by w_i and increases at most by αw_i. Once we obtain a subset S_1 such that $\delta v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S_1} w_j \leq \varepsilon$, the linear programs $(\mathrm{L}^{i_0}_{S,\pi})$ and $(\mathrm{U}^{i_0}_{S})$ are solved and the achieved guarantee d_{rel} is evaluated. If Inequality (3.5.21) is not yet satisfied, the factor δ is updated such that the new estimate fulfills:

$$\delta v^{\alpha}_{\max} \sum_{j \in \mathbb{S} \setminus S_1} w_j = \frac{\overline{v}_{i_0} - \underline{v}_{i_0}(\pi)}{\underline{v}_{i_0}(\pi)}.$$

This way the heuristic weighting-policy-exploration is continued until Inequality (3.5.21) is satisfied or there do not exist further states with positive weights. Since it is possible to modify the error guess δ this way, its initial value is not that important. However, it has to be ensured that the initial value for δ is not too large as weighting-policy-exploration may construct too many states otherwise. Algorithm 7 describes the heuristic weighting-policy-exploration in detail. Note that after updating the error estimation factor δ in Step 14, the inequality in Step 9 will not be satisfied until at least one additional state has been added to S_1.

3.5.5 Column Generation Implementation

In this section, we mention some details concerning the implementation of our column generation algorithm for approximating the component $v^{\alpha}_{i_0}$ of the optimal value vector of a given MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ for some state $i_0 \in \mathbb{S}$. Moreover, we present the complete version of our approximation algorithm including all possible ingredients described earlier.

So far, we have not been dealing with the issue of solving the linear programs themselves arising in the column generation process. Recall that solving the linear programs $(\mathrm{U}^{i_0}_{S})$ for $S \subseteq \mathbb{S}$ to obtain upper bounds on the value $v^{\alpha}_{i_0}$ is only required to assess the approximation guarantee achieved so far. Apart from that, their solutions do not affect the approximation process at all. Therefore, it might be reasonable to solve these linear programs only once in a while. At each iteration of the column generation algorithm where no upper bound on $v^{\alpha}_{i_0}$ is determined via a linear program, we can still make use of the weaker upper bound computed by the reduced profits as described in Corollary 3.5.14. Note that these bounds can be determined almost for free by all proposed pricing strategies except for min-depth.

All linear programs encountered in the column generation algorithm are

Algorithm 7 Heuristic weighting-policy-exploration

Input: an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ (given implicitly), a state $i_0 \in \mathbb{S}$, a policy π for M, an initial error estimation $\delta > 0$, $\varepsilon > 0$
Output: a subset $S_1 \subseteq \mathbb{S}$ with $i_0 \in S_1$ such that Inequality (3.5.21) is satisfied for the associated bounds $\underline{v}_{i_0}(\pi)$ and $\overline{v}_{i_0}$
$S_1 \leftarrow \{i_0\}$
$w_j \leftarrow 0$ for each $j \in \mathbb{S} \setminus S_1$
for each $j \in \mathbb{S} \setminus S_1$ with $p_{i_0 j}(\pi) > 0$ **do**
    $w_j \leftarrow \alpha p_{i_0 j}(\pi)$ ▷ initialize weights
end for
while there exists $j \in \mathbb{S} \setminus S_1$ with $w_j > 0$ **do**
    if $\delta v_{\max}^{\alpha} \sum_{j \in \mathbb{S} \setminus S_1} w_j < \varepsilon$ **then**
        determine the optimal values $\underline{v}_{i_0}(\pi)$ and $\overline{v}_{i_0}$ of the linear programs $(\mathrm{L}_{S,\pi}^{i_0})$ and $(\mathrm{U}_{S}^{i_0})$, respectively
        if $\underline{v}_{i_0}(\pi)$ and $\overline{v}_{i_0}$ satisfy Inequality (3.5.21) **then**
            return S_1
        else
            update the error estimation δ by:

$$\delta \leftarrow \left(v_{\max}^{\alpha} \sum_{j \in \mathbb{S} \setminus S_1} w_j \right)^{-1} \frac{\overline{v}_{i_0} - \underline{v}_{i_0}(\pi)}{\underline{v}_{i_0}(\pi)}$$

        end if
    end if
    $i \leftarrow \operatorname{argmax}_{j \in \mathbb{S} \setminus S_1} \{w_j\}$ ▷ get state to be added
    $S_1 \leftarrow S_1 \cup \{i\}$
    for each $j \in \mathbb{S} \setminus S_1$ with $p_{ij}(\pi) > 0$ **do**
        $w_j \leftarrow w_j + \alpha p_{ij}(\pi) w_i$ ▷ update weights
    end for
end while
return S_1 ▷ $v_{i_0}^{\alpha}(\pi)$ computed exactly

solved using the linear and mixed integer programming solver Cplex [ILO], version 12.1. In Section 4.2.2 we will compare different possible solvers, i. e., the primal and dual simplex method and the barrier method.

We summarize the different ingredients of our approximation algorithm:

- a solver for the arising linear programs that may additionally feature the construction of initial bases (see the end of Section 3.5.2),

Algorithm 8 Column generation based approximation algorithm

1: **Input:** an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ (given implicitly), a state $i_0 \in \mathbb{S}$, $\varepsilon > 0$, a linear programming solver solver, an approximation heuristic heur, a pricing strategy pricer, $n \in \mathbb{N}$, $f \in \mathbb{N}$
2: **Output:** lower and upper bounds $\underline{v}_{i_0}, \overline{v}_{i_0}$ on $v_{i_0}^\alpha$ with $(\overline{v}_{i_0} - \underline{v}_{i_0})/\underline{v}_{i_0} \leq \varepsilon$
3: apply heur to construct an initial subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$
4: $i \leftarrow 0$ ▷ iteration counter
5: solve $(\mathrm{L}_S^{i_0})$ by solver, and let $\underline{v}_{i_0}$ be the optimal value
6: **if** $i \bmod f = 0$ **then**
7: solve $(\mathrm{U}_S^{i_0})$ by solver, and let $\overline{v}_{i_0}$ be the optimal value
8: **end if**
9: **if** $(\overline{v}_{i_0} - \underline{v}_{i_0})/\underline{v}_{i_0} \leq \varepsilon$ **then** ▷ check approximation goal
10: **return** $\underline{v}_{i_0}, \overline{v}_{i_0}$
11: **else**
12: $i \leftarrow i + 1$
13: apply pricer to determine a subset of states $S_{\text{new}} \subseteq \mathbb{S} \setminus S$ with $\bar{p}_j > 0$ for each $j \in S_{\text{new}}$, $|S_{\text{new}}| \leq n$, and $|S_{\text{new}}| = n$ if possible, as well as an upper bound $\overline{v}_{i_0}$ on $v_{i_0}^\alpha$ ▷ according to Corollary 3.5.14
14: **if** $S_{\text{new}} = \emptyset$ **then**
15: **return** $\underline{v}_{i_0}, \overline{v}_{i_0}$ ▷ $\underline{v}_{i_0} = v_{i_0}^\alpha$ due to Theorem 3.3.2
16: **else**
17: $S \leftarrow S \cup S_{\text{new}}$
18: go to step 5
19: **end if**
20: **end if**

- a frequency for computing the current upper bound on $v_{i_0}^\alpha$ via the associated linear program,
- a pricing strategy (see Section 3.5.3),
- a number of states to be added in one pricing iteration, and
- an optional approximation heuristic (see Section 3.5.4).

Algorithm 8 gives an overview of our approximation algorithm including the mentioned additional features. Note that Algorithm 8 is an implementation of the generic Algorithm 4.

We should mention that in practice, $S_{\text{new}} = \emptyset$ in Step 14 does not necessarily imply that the equation $\underline{v}_{i_0} = v_{i_0}^\alpha$ is fulfilled exactly due to numerical problems. Since we have $|\mathbb{S}| < \infty$, it is clear that Algorithm 8 is finite.

Finally, we compare our approximation algorithm 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 8 compared to this 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 as we will see in Section 4.2.3. 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 equivalent as described in Section 2.2.4. Our algorithm may be a bit slower per iteration when a second linear program is solved.

3.5.6 Approximation Without Linear Programming

In this section, we show that our approximation method given in Algorithm 8 can be implemented equivalently without using any technique from linear programming. Instead, we make use of the policy iteration method. Since we will handle different MDPs below, the optimal value vector of an MDP M will again be denoted by v^{α}_{M} to prevent ambiguity. Let $M = (\mathbb{S}, \mathbb{A}, p, c, \alpha)$ be an MDP and let $i_0 \in \mathbb{S}$ be a particular state. Given any initial subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, the alternative method to approximate the value v^{α}_{M,i_0} is as follows.

In a first step, we have to determine the lower and upper bounds $\underline{v}_{i_0}$ and $\overline{v}_{i_0}$ w. r. t. the subset S for the value v^{α}_{M,i_0} that are otherwise obtained as the optimal values of the linear programs $(\mathrm{L}^{i_0}_S)$ and $(\mathrm{U}^{i_0}_S)$, respectively. Using the policy iteration method as described in Algorithm 2 on page 20 we compute an optimal policy π for $M(S)$ and the optimal value vector $v^{\alpha}_{M(S)} = v^{\alpha}_{M(S)}(\pi)$. By Theorem 3.1.8 the vector $v^{\alpha}_{M(S)}$ equals the optimal solution of the linear program (L^{Σ}_S) which is by Corollary 3.1.5 also optimal for $(\mathrm{L}^{i_0}_S)$. Thus, the lower bound $\underline{v}_{i_0}$ is given by $\underline{v}_{i_0} = v^{\alpha}_{M(S),i_0}$. As described

Algorithm 9 Approximation algorithm without linear programming

1: **Input:** an MDP $(\mathbb{S}, \mathbb{A}, p, c, \alpha)$ (given implicitly), a state $i_0 \in \mathbb{S}$, $\varepsilon > 0$
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: determine any initial subset of states $S \subseteq \mathbb{S}$ with $i_0 \in S$, e. g., by using one of the proposed approximation heuristics
4: use the policy iteration method to compute optimal policies π and μ for the induced MDPs $M(S)$ and $M'(S)$ as well as the value vectors $v^\alpha_{M(S)}(\pi)$ and $v^\alpha_{M'(S)}(\mu)$
5: $\underline{v}_{i_0} \leftarrow v^\alpha_{M(S),i_0}(\pi)$ and $\overline{v}_{i_0} \leftarrow v^\alpha_{M'(S),i_0}(\mu)$
6: **if** $(\overline{v}_{i_0} - \underline{v}_{i_0})/\underline{v}_{i_0} \leq \varepsilon$ **then**
7: **return** $\underline{v}_{i_0}, \overline{v}_{i_0}$
8: **else**
9: determine the policy basic solution u^π of π by solving the associated system of linear equations
10: determine some subset of states $S_{\text{new}} \subseteq \mathbb{S} \setminus S$ with the property $\bar{p}_j := \alpha \sum_{i \in S} \sum_{a \in \mathbb{A}(i)} p_{ij}(a) u^\pi_{ia} > 0$ for each $j \in S_{\text{new}}$ by using one the proposed pricing methods
11: $S \leftarrow S \cup S_{\text{new}}$
12: go to step 4
13: **end if**

in Section 3.1.2 the upper bound $\overline{v}_{i_0}$ equals the component $v^\alpha_{M'(S),i_0}$ of the optimal value vector for the upper-bound S-induced MDP $M'(S)$, see Remark 3.1.11. Therefore, the value $\overline{v}_{i_0}$ can be obtained by the policy iteration method in a similar way as the lower bound.

After the bounds on $v^\alpha_{i_0}$ have been obtained, we check whether the desired approximation is achieved. If this is not the case, further states are to be generated. To this end, we compute the policy basic solution u^π of $(\mathrm{DL}^{i_0}_S)$ for the policy π that is optimal for the induced MDP $M(S)$ by solving the associated system of linear equations. Recall that by Theorem 3.5.3 the basic solution u^π is optimal for $(\mathrm{DL}^{i_0}_S)$ since π is optimal for $M(S)$. Having an optimal dual solution at hand, the pricing problem can be handled as described in Section 3.5.3 to added new states to S.

The described method is summarized in Algorithm 9. Typically, it will be advantageous to appoint the obtained optimal policies for $M(S)$ and $M'(S)$ as initial policies for the policy iteration methods in the next iteration. We will not compare the two equivalent versions of the approximation algorithm numerically. However, we are convinced that due to today's practical efficiency of linear programming solvers Algorithm 9 will not be competitive compared to the original column generation method given in Algorithm 8.

Chapter 4

Computational Results

In this chapter we apply our approximation algorithm to analyze various policies for exemplary discounted MDPs. The Markov decision processes we consider here emerge from academic and real-world online optimization problems. Our results will demonstrate that the algorithm is capable to provide realistic performance indicators for concrete policies. Additionally, we study computational aspects of the approximation algorithm.

The chapter is arranged as follows. Section 4.1 describes how to model a given online optimization problem as Markov decision process in general and introduces the instances considered in the sequel. In Section 4.2, we numerically compare our approximation algorithm with the neighborhood construction according to Theorem 3.2.2. Moreover, we analyze the performance of different solvers for the encountered linear programs and compare the pricing strategies and approximation heuristics introduced in Section 3.5.3 and Section 3.5.4, respectively. Finally, we apply the column generation algorithm to evaluate different policies for the considered MDPs in Section 4.3.

4.1 Modeling Markov Decision Processes for Online Optimization Problems

We already mentioned that we see the main motivation of our approximation algorithm in the need to analyze online optimization problems and especially online algorithms. The framework based on our approach is as follows. In order to analyze a given online algorithm for a given online optimization problem, the first step is to define a Markov decision process modeling the online problem. Usually, this construction is not completely straightforward since there are often different alternatives for formulating an associated Markov decision process and often restrictions have to be made as we will describe below. Furthermore, it is required to translate the considered online algorithm into a policy for the Markov decision process. This step is trivial as the chosen action of the policy for a given state is simply given by the decision made by the online algorithm. Then, we locally evaluate this policy using our approximation algorithm applied to a discounted MDP that is given by

the Markov decision process and some expedient discount factor. This evaluation is based on comparing the total expected discounted cost of the policy to that of an unknown optimal policy or other concrete policies. Similarly, the quality of a single action of the considered policy can be analyzed. This way, we obtain for the considered situation also an evaluation of the online algorithm or its current decision, respectively. It is important that the obtained evaluation has to be interpreted carefully by taking into account the possible restrictions and assumptions of the considered MDP model, e. g., the used discount factor and used transition probabilities. In the following, we will often use the notion of a policy as a synonym for online algorithm.

4.1.1 General Modeling Approach

In the following we describe in general how a given online optimization problem can be modeled as a Markov decision process. Most online optimization problems studied in the literature can be formulated as a so called *request-answer game* that was originally introduced by Ben-David et al. [BDBK+90]. For minimization problems, the definition is as follows.

Definition 4.1.1 (Request-answer game) A *request-answer game* is defined as a triple $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ consisting of a request set $\mathcal{R}$, an answer set $\mathcal{A}$, and a sequence $\mathcal{C}$ of cost functions $C_1, C_2, \ldots$, where $C_n \colon \mathcal{R}^n \times \mathcal{A}^n \to \mathbb{R}_+ \cup \{\infty\}$ for each $n \in \mathbb{N}$. Here $C_n(r_1, \ldots, r_n, a_1, \ldots, a_n)$ is the total incurred cost by giving answers $a_1, \ldots, a_n \in \mathcal{A}$ to requests $r_1, \ldots, r_n \in \mathcal{R}$. △

Definition 4.1.2 (Online algorithm) Let the triple $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ be an online optimization problem. An *online algorithm* for $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ is a sequence of functions $g_1, g_2, \ldots$, where $g_n \colon \mathcal{R}^n \to \mathcal{A}$ for each $n \in \mathbb{N}$. △

Note that this definition points out that an online algorithm must make decisions based only on the information obtained by previous requests. Obviously, depending on the sequence of requests $(r_1, \ldots, r_n)$ and the previously given answers $(a_1, \ldots, a_{n-1})$ some (or even all) answers $a_n \in \mathcal{A}$ may be infeasible. This is modeled by a cost of $C_n(r_1, \ldots, r_n, a_1, \ldots, a_n) = \infty$. In the following we assume that there always exists an answer $a_n \in \mathcal{A}$ such that the total incurred cost up to request r_n is finite.

One can distinguish two different classes of online optimization problems. In the so-called *sequence model* the requests must be served irrevocably in the order of their occurrence, i. e., only after r_j has been served, the next request r_{j+1} becomes known. In the *time stamp model* each request has an additional *release time* at which it becomes known and available for service.

Here the requests arrive in order of non-decreasing release times. An online algorithm must determine its behavior at a certain moment t in time depending on the requests released up to time t and the current time t. The difference to the sequence model is that the online algorithm is allowed to wait and to revoke tentative decisions, and that requests need not be served in the order of their occurrence. We mention that the request-answer game covers both models.

Our approach to construct a Markov decision process model $(\mathbb{S}, \mathbb{A}, p, c)$ of a request-answer game $(\mathcal{R}, \mathcal{A}, \mathcal{C})$ is as follows:

- The state space $\mathbb{S}$ is defined by:

$$\mathbb{S} = \{(r_1, \ldots, r_n, a_1, \ldots, a_{n-1}) \mid n \in \mathbb{N}, r_1, \ldots, r_n \in \mathcal{R}, a_1, \ldots, a_{n-1} \in \mathcal{A}\}.$$

- For each state $i = (r_1, \ldots, r_n, a_1, \ldots, a_{n-1}) \in \mathbb{S}$, the set of associated actions is given as:

$$\mathbb{A}(i) = \{a_n \in \mathcal{A} \mid C_n(r_1, \ldots, r_n, a_1, \ldots, a_n) < \infty\}.$$

- For each state $i = (r_1, \ldots, r_n, a_1, \ldots, a_{n-1}) \in \mathbb{S}$ and each possible action $a_n \in \mathbb{A}(i)$, the stage cost for the transition to a state $j \in \mathbb{S}$ is defined by:

$$c_i(a_n, j) = C_n(r_1, \ldots, r_n, a_1, \ldots, a_n) - C_{n-1}(r_1, \ldots, r_{n-1}, a_1, \ldots, a_{n-1}).$$

- The transition probabilities $p_{ij}(a_n)$ for $i, j \in \mathbb{S}$ and $a_n \in \mathbb{A}(i)$ such that $i = (r_1, \ldots, r_n, a_1, \ldots, a_{n-1})$ and $j = (r_1, \ldots, r_{n+1}, a_1, \ldots, a_n)$ may be chosen arbitrarily.

Obviously, this construction will result in an infinite state space since each state stores the complete history, which is required to obtain a stationary system. Thus, in the sense of Definition 2.1.1 on page 9, the tuple $(\mathbb{S}, \mathbb{A}, p, c)$ is not a Markov decision process. Recall that an infinite state space is not necessarily a problem for our approximation algorithm as the method only considers a smaller local part of the total set of states.

Note that it is usually not required to maintain the complete history: each state i only requires the information that is necessary

- to specify the possible actions $\mathbb{A}(i)$,
- to compute the expected stage cost $c_i(a, j)$ for each $a \in \mathbb{A}$ and each successor state j of i, and

- to determine the transition probability $p_{ij}(a)$ for each $a \in \mathbb{A}$ and each state j.

Most online optimization problems that are considered in the literature or arise from practical applications feature a structure that allows to reduced the information maintained by each state. This may often result in a finite state space.

Since competitive analysis captures the worst case behavior of online algorithms only, models for online optimization problems typically lack stochastic information. To formulate a Markov decision process, it is required to come up with some suitable distribution for the transition probabilities. Sometimes it is reasonable to use the uniform distribution. Examples follow in the Sections 4.1.3–4.1.5.

Recall that the performance of our approximation algorithm heavily depends on the number of successors of a state that are reached with significant probability. When this number is large, the local state spaces to obtain a given approximation guarantee will be large also, making the method perform poorly. In modeling a Markov decision process for an online optimization problem, this is the most serious issue. In order to construct a tractable Markov decision process, often simplifying restrictions especially on the set of possible requests R has to be made, as we will see in the models for the considered online optimization problems described next.

The three problems under consideration are interesting to be analyzed by our approach due to different reasons that are typical for online optimization and competitive analysis. In the first case we face a weird result from competitive analysis since a totally stupid online algorithm outperforms an apparently appropriate algorithm, see [KdPSR08]. In the second problem [HKM+05] the situation is that it seems to be difficult to evaluate a presumably improved variant of an online algorithm by means of competitive analysis (at least we were not able to do so). The last problem [HKR00] features the case that it is impossible to provide any performance distinctions for different online algorithms.

4.1.2 Issues for Analyzing Associated Discounted MDPs

Recall that our approximation algorithm analyzes discounted MDPs, i. e., the objective criterion is to minimize the total expected α-discounted cost for some discount factor $\alpha \in [0, 1)$. Unfortunately, the practical efficiency of the algorithm crucially depends on α. The greater the discount factor, the greater will usually be the size of the required state space to obtain a given approximation. In our context it is usually not required to consider

a discount factor close to one, as needed by typical finances applications. Instead, we should select a moderate discount factor that provides significant results concerning the comparison of different policies.

Moreover, the effect of discounting incurred costs has to be taken into account seriously in the analysis for a given Markov decision process. In the models described in the following sections, the stage costs represent either the increase of a maximum cost value or the increase of a total cost value. Due to discounting, we only consider the discounted increase. This makes early increases more costly than those occuring later. Thus, for a long (i, j)-path in any of these MDPs the total expected discounted cost is quantitatively hardly related to the originally modeled maximum or total cost value. Nonetheless, the total expected discounted cost of a given policy may still reflect its quality appropriately in comparison to other policies.

Using the trivial bounds $0 \leq v_j^\alpha \leq v_{\max}^\alpha$ for the components $v_j^\alpha, j \in \mathbb{S}$ of the optimal value vector in our approximation method typically gives weak approximation results when these bounds are weak also. We described in Remark 3.1.17 on page 50 that for each $j \in \mathbb{S}$ improved lower and upper bounds $0 \leq v_{\min}^\alpha(j) \leq v_j^\alpha \leq v_{\max}^\alpha(j) \leq v_{\max}^\alpha$ can be incorporated in the considered linear programs instead. Exploiting good state-specific bounds is very advantageous or even crucial to apply our algorithm effectively. For each considered MDP, we will also describe the employed lower and upper bounds.

Note that these improved bounds are valid for the components of the optimal value vector. The upper bound $v_j^\alpha \leq v_{\max}^\alpha(j)$ for a state $j \in \mathbb{S}$, however, may be infeasible for the value vector component of a given policy π in general, i. e., $v_j^\alpha(\pi) > v_{\max}^\alpha(j)$. On the other hand, each lower bound $v_{\min}^\alpha(j) \leq v_j^\alpha$ for state j is also valid for $v_j^\alpha(\pi)$ since we have $v_j^\alpha \leq v_j^\alpha(\pi)$.

4.1.3 Bin Coloring

A simple online optimization problem that was motivated by a real-world problem arising in an order picking problem is *online bin coloring* [KdPSR08]. We are given a set of colors $C \subseteq \mathbb{N}$ and a request sequence $r_1, r_2, \ldots, r_n$ for some $n \in \mathbb{N}$ consisting of unit size items, where each item r_k for $k \in \{1, \ldots, n\}$ has a color $c_k \in C$. The items are to be packed into bins, all of the same size $b \in \mathbb{N}$, as soon as they arrive. Repacking an item later on is not allowed. At each moment there are $m \in \mathbb{N}$ empty or partially filled bins. Whenever a bin is full, it is closed and replaced by a new empty bin. The objective is to pack the items in such a way that the maximum number of different colors in a bin is as small as possible.

Known Results

A natural greedy-type algorithm would put an item with a color already present in one of the bins into the same bin. If the color is currently not present in one of the bins, one puts it into a bin which currently has the least number of distinct colors. This algorithm is called GreedyFit. It has been shown by Krumke et al. [KdPSR08] that the competitive ratio of GreedyFit is at least $2m$, i. e., twice the number of bins that are available simultaneously. Furthermore, the totally stupid algorithm OneBin, which uses only one bin until it is filled completely and puts all items into that bin, achieves a competitive ratio of at most $2m - 1$, making it superior to GreedyFit in terms of the competitive ratio. Not surprisingly, the opposite behavior is observed in simulation experiments, where GreedyFit outperforms OneBin significantly. This weird result from competitive analysis motivates to analyze the online bin coloring problem and associated online algorithms w. r. t. a stochastic model, e. g., a Markov decision process, which may capture the observed behavior better. Using a new method based on the notion of stochastic dominance, Hiller and Vredeveld [HV08] proved that the performance of GreedyFit is stochastically better than that of OneBin for any number of items processed.

Markov Decision Process Model

Our Markov decision process model $(\mathbb{S}, \mathbb{A}, p, c)$ for online bin coloring is as follows. A state $i \in \mathbb{S}$ is of the form $i = (c, \chi, f_1, C_1, \ldots, f_m, C_m)$. It specifies the color $c \in C$ of the current request that is to be packed into a bin next. Moreover, the state i needs to keep track of the maximum *colorfulness* $\chi \in \mathbb{N}$, i. e., the maximum number of different colors attained by a bin so far. Note that the maximum colorfulness may have been attained by a bin that was closed already. Finally, the state i features the current configuration of each bin $k \in \{1, \ldots, m\}$ by the number of items $f_k \in \mathbb{N}$ and the set of different colors $C_k \subseteq C$ contained in that bin. It is quite clear that all the necessary information in a Markov decision process are available from this state structure. Due to the characteristics of the online bin coloring problem, the state space $\mathbb{S}$ is as follows:

$$\begin{aligned}\mathbb{S} = \{(c, \chi, f_1, C_1, \ldots, f_m, C_m) \mid\ & c \in C, \\ & \max_{1 \le k \le m} \{|C_k|\} \le \chi \le \min\{b, |C|\}, \\ & |C_k| \le f_k < b \ \forall k \in \{1, \ldots, m\}\}.\end{aligned}$$

We mention that it is not required to distinguish the different bins. Thus in order to reduce the state space, one can consider an (unordered) multiset

of bin configurations instead of the sequence given above. We did so in our implementation.

Since a new item can be packed into any one of the m bins, the set of possible actions is $\mathbb{A}(i) = \{1, \ldots, m\}$ for each state $i \in \mathbb{S}$. Assume that at a given state $i = (c, \chi, f_1, C_1, \ldots, f_m, C_m)$ action $a \in \mathbb{A}(i)$ is used. Then, each possible successor state $j \in \mathbb{S}$ is of the form $j = (c', \chi', f'_1, C'_1, \ldots, f'_m, C'_m)$, where the data are as follows. For each $k \in \{1, \ldots, m\} \setminus \{a\}$, we have $f'_k = f_k$ and $C'_k = C_k$, and for the action a, we obtain:

$$f'_a = \begin{cases} f_a + 1, & \text{if } f_a < b - 1, \\ 0, & \text{if } f_a = b - 1, \end{cases}$$

and

$$C'_a = \begin{cases} C_a \cup \{c\}, & \text{if } f_a < b - 1, \\ \emptyset, & \text{if } f_a = b - 1. \end{cases}$$

The new maximum colorfulness χ' is given by:

$$\chi' = \max\{\chi, |C_a \cup \{c\}|\}.$$

Finally, the color c' of the next item may be any color from C.

Note that the only random parameter in the Markov decision process is the color of future items. That is, for each two states $i, j \in \mathbb{S}$ (with parameters as denoted above) and each action $a \in \mathbb{A}(i)$, the transition probability $p_{ij}(a)$ only depends on the color $c'(j)$ of the new item. We assume that this random color is independent of the previous items and colors. One may choose any probability distribution, e. g., the uniform distribution. The associated stage cost $c_i(a, j)$ equals the increase of the maximum colorfulness due to the action a:

$$c_i(a, j) = \begin{cases} 1, & \text{if } |C_a| = \chi \text{ and } c \notin C_a, \\ 0, & \text{otherwise.} \end{cases}$$

Notice that the stage cost $c_i(a, j)$ is independent of the successor state $j \in \mathbb{S}$. This way the total sum of stage costs for the transitions of an (i, j)-path in the Markov decision process equals the total increase of the maximum colorfulness from state i to state j.

Remark 4.1.3 When we are only interested in uniform transition probabilities and in such online algorithms that always put an item with a color already present in a bin into such a bin, one can formulate a simpler Markov

decision process model. This model can neglect the explicit colors of the items. Concerning the colors, a state only specifies for each bin, the number of different colors contained, and whether the color of the current item is already present in a bin or not. The resulting states space is significantly smaller than the state space for the general model described above. △

We mention that the proposed model described above is very similar to the Markov chain model used in [HV08]. The described Markov decision process for bin coloring and policies associated with the mentioned online algorithms are analyzed in Section 4.3.2.

Improved Bounds

For the bin coloring MDP we do not employ state-specific lower bounds, but simply use the trivial bound $v^{\alpha}_{\min}(j) = 0$ for each state $j \in \mathbb{S}$. On the other hand, the following improved upper bound $v^{\alpha}_{\max}(j)$ for a state $j \in \mathbb{S}$ is incorporated. Clearly, the total expected α-discounted cost $v^{\alpha}_{j}(\pi)$ of any policy π gives a feasible upper bound on v^{α}_{j}. Let us call a bin $k \in \{1, \ldots, m\}$ *critical* w. r. t. a color $c \in C$ if the number of different colors $|C_k|$ present in that bin equals the maximum colorfulness χ and $c \notin C_k$. That is, the maximum colorfulness increases if color c is packed into a critical bin w. r. t. color c. We consider the online algorithm ACB (AvoidCriticalBins) which is as follows.

ACB If the color of the next item is already present in a bin, pack the item into that bin. Otherwise, if there exists a bin that is not critical choose such a bin, if all bins are critical pack the item into any bin.

Note that ACB generalizes GreedyFit.

Assume that there exist enough colors such that the maximum colorfulness $\chi = b$ is eventually attained by any policy for each initial state (one can show that $m(b-1)+1$ colors suffice). Consider a state j where the current color is $c \in C$. Obviously, a worst sequence of requests for the online algorithm ACB is such that each color is not present in one of the m open bins. Thus, we consider a sequence where each new color after the color c is not contained in any bin. Let $(f_1, C_1, \ldots, f_m, C_m)$ be the bin configuration after the item with color c has been packed by ACB, and let χ be the maximum colorfulness before that item was packed. In order to compute the total α-discounted cost of ACB for this sequence, one has to determine the stages $k_1, \ldots, k_{b-\chi}$, where the maximum colorfulness is increased. This

instance	m	b	n	$p_1, \ldots, p_n$
`bc-2-3-6-uni`	2	3	6	uniformly distributed
`bc-2-3-6-spe`	2	3	6	0.30 0.30 0.20 0.10 0.07 0.03
`bc-3-3-7-uni`	3	3	7	uniformly distributed
`bc-3-3-7-spe`	3	3	7	0.30 0.27 0.15 0.10 0.09 0.06 0.03
`bc-3-4-12-uni`	3	4	12	uniformly distributed
`bc-3-4-12-spe`	3	4	12	0.30 0.15 0.10 0.09 0.07 0.07
				0.06 0.05 0.04 0.03 0.02 0.02

Table 4.1: Considered instances of the bin coloring Markov decision process.

results in a total α-discounted cost of:

$$v^{\alpha}_{\max}(j) := \sum_{l=1}^{b-\chi} \alpha^{k_l},$$

which is the upper bound on v^{α}_{j} we will use for state j. It is straightforward to develop an algorithm for computing the numbers $k_1, \ldots, k_{b-\chi}$ and to show that the these numbers are independent of different possible implementations of ACB. We skip the details.

Studied Instances

In this section we introduce the instances of the described Markov decision process modeling the online bin coloring problem to be studied in the sequel. An instance is uniquely specified by the following parameters:

- a number of simultaneously available bins $m \in \mathbb{N}$,
- a bin capacity $b \in \mathbb{N}$,
- a number $n \in \mathbb{N}$ defining a set of different colors $C = \{1, \ldots, n\}$, and
- *color probabilities* $0 \leq p_1, \ldots, p_n \leq 1$ with $\sum_{k=1}^{n} p_k = 1$, where p_k is the probability that the next item has color k for each $k \in \{1, \ldots, n\}$.

We denote each concrete Markov decision process by the corresponding parameters m, b, n and the flag `uni` or `spe`, specifying whether the color probabilities are uniformly distributed or not. Table 4.1 shows the instances we will consider.

4.1.4 Target Date Assignment

The second online optimization problem we consider is a small *online target date assignment* problem. This kind of problems arises, e. g., in the context of dispatching service technicians. In [HKM+05] we introduced this type of online optimization problems and analyzed associated online algorithms by competitive analysis.

An instance of the online target date assignment problem is given by a sequence of requests $\sigma = r_1, r_2, \ldots, r_n$ for some $n \in \mathbb{N}$ and a *downstream problem* Π, which is an offline optimization problem for which arbitrary subsets of requests are feasible inputs. Each request r_k for $k \in \{1, \ldots, n\}$ has an integral release date t_k and must be assigned immediately and irrevocably to a target date in the time period $t_k + 1, \ldots, t_k + \delta_k$, where δ_k is the allowed time for deferring the service of request r_k, which is also revealed upon arrival of the request. In the following we consider only the case of uniform deferral times, that is, $\delta_k = \delta$ for all requests r_k and some $\delta \in \mathbb{N}$.

A solution of an online target date assignment problem w. r. t. a downstream problem Π is feasible if

- each request is assigned to a feasible target date, and
- for each single target date, the corresponding instance of Π is feasible, too.

Let σ_d be the subset of requests assigned to date d by an online algorithm. The optimal cost of Π for σ_d is called *downstream cost at date d*, and we denote it by $\text{downcost}(\sigma_d)$.

We consider here the online target date assignment problem with the objective to minimize the total downstream cost summed up over all target dates. The downstream problem we look at is bin packing. In bin packing n items with sizes $0 < s_1, \ldots, s_n \leq 1$ are to be packed into unit sized bins. The objective is to find a packing such that the total size of the items packed in one bin does not exceed the bin's capacity and the total number of bins needed to pack the items is minimized. In the online target date assignment problem w. r. t. bin packing, a request $r = (t(r)), s(r))$ is given by its release date $t(r)$ and its size $0 < s(r) \leq 1$. Therefore, the downstream cost at a date is the minimum number of bins required to pack all items assigned to that date. That is, in a sense repacking is allowed here. We assume that the number of available bins per date is not bounded because this would prevent any deterministic online algorithm to guarantee a feasible solution. The objective is to find an assignment of requests to feasible target dates that minimizes the total sum of the bins required over all target dates.

Known Results

We proposed the following online algorithms for this target date assignment problem in [HKM+05]. Let us say that a target date is *used*, if a request has been assigned to it.

Algorithm PackTogetherOrDelay (PTD) Assign a request r to the earliest date in the feasible range $t(r)+1,\ldots,t(r)+\delta$ which is already used. If no used target date is feasible for request r, then assign it to the latest feasible target date, that is, to $t(r)+\delta$.

Algorithm PackFirstOrDelay (PFD) If there exists a used target date to which the current request r can be assigned without increasing the number of necessary bins, then the earliest of these dates is chosen. Otherwise, assign the latest feasible date, $t(r)+\delta$.

Note that at any moment in time at most one feasible target date is used by the online algorithm PTD. We have proved that PTD is 2-competitive, see [HKM+05]. Moreover, we conjectured that the online algorithm PFD has a better performance guarantee than PTD although the analysis for the general problem seems more difficult. In general, PFD is not dominated by PTD, or vice versa: one can construct request sequences for which PFD performs better and worse than PTD, respectively.

Markov Decision Process Model

We assume that at each date at least one request is released. A Markov decision process model $(\mathbb{S},\mathbb{A},p,c)$ for the considered target date assignment problem can be formulated as follows. Since requests assigned to dates that are not feasible anymore are irrelevant for the future evolution of the system, it suffices to store in each state the requests assigned to the upcoming next δ dates, denoted by $1,\ldots,\delta$. Each state $i\in\mathbb{S}$ encodes the size $0<s\leq 1$ of the current item, the number of items n having been released at the current date, and multisets of item sizes assigned to each future date $k\in\{1,\ldots,\delta\}$ given by functions $S_k\colon(0,1]\to\mathbb{N}_0$. That is, the state i is given by the vector $i=(s,n,S_1,\ldots,S_\delta)$. We obtain the following state space:

$$\mathbb{S}=\{(s,n,S_1,\ldots,S_\delta)\mid 0<s\leq 1, n\in\mathbb{N}, \\ S_k\colon(0,1]\to\mathbb{N}_0\ \forall k\in\{1,\ldots,\delta\}\}.$$

Note that in order to obtain a finite state space in the Markov decision process, it is required to restrict the possible sizes an item can have.

Since a new request can be assigned to any one of the next δ dates, the set of possible actions is $\mathbb{A}(i) = \{1, \ldots, \delta\}$ for each state $i \in \mathbb{S}$. Assume that at a given state $i = (s, n, S_1, \ldots, S_\delta)$ action $a \in \mathbb{A}(i)$ is used. Depending on whether the next request arrives at the same date or the next one, there are two types of possible successor states. If the next request is released at the same date, a possible successor $j_1 \in \mathbb{S}$ is of the form $j_1 = (s', n+1, S'_1, \ldots, S'_\delta)$, where the new item has an arbitrary size $0 < s' \leq 1$ and the assignments are updated according to the selected action a:

$$S'_a(s) = S_a(s) + 1, \tag{4.1.1}$$

$$S'_a(x) = S_a(x) \quad \text{for each } x \in (0, 1] \setminus \{s\}, \tag{4.1.2}$$

$$S'_k = S_k \quad \text{for each } k \in \{1, \ldots, \delta\} \setminus \{a\}.$$

If the next request is released at the consecutive date (recall that we assume that at least one request is released each date), a successor state $j_2 \in \mathbb{S}$ has the data $j_2 = (s', 1, S''_1, \ldots, S''_\delta)$ for some $0 < s' \leq 1$. For the bin configuration, we now obtain:

$$S''_k = S_{k+1} \quad \text{for each } k \in \{1, \ldots, \delta - 1\} \setminus \{a - 1\},$$
$$S''_\delta(x) = 0 \quad \text{for each } x \in (0, 1],$$

and $S''_{a-1} = S'_a$ if $a > 1$, where S'_a is defined by the Equations (4.1.1) and (4.1.2).

In this Markov decision process we face two different random effects: the size of the new request and its release date given by a possible change of the current date. The stage cost $c_i(a, j)$ for states $i, j \in \mathbb{S}$ and an action $a \in \mathbb{A}(i)$ equals the increase of the downstream cost, i. e., we have:

$$\begin{aligned} c_i(a, j) &= \text{downcost}(S'_a) - \text{downcost}(S_a) \\ &= \begin{cases} 1, & \text{if downcost}(S'_a) \neq \text{downcost}(S_a), \\ 0, & \text{otherwise,} \end{cases} \end{aligned}$$

where S'_a is again given by the Equations (4.1.1) and (4.1.2). That is, the stage cost equals 1 if and only if packing all items given by S'_a optimally requires one bin more than an optimal solution for the items in S_a. Therefore, the total sum of stage costs for the transitions of a path in the Markov decision process equals the total downstream cost incurred along the path, i. e., the total number of new bins required. Computational results for the described model are given in Section 4.3.3.

Improved Bounds

As for bin coloring, we will always use the trivial lower bound $v^{\alpha}_{\min}(j) = 0$ for each state $j \in \mathbb{S}$ in an MDP modeling the target date assignment problem w. r. t. downstream bin packing. The idea for constructing a slightly improved upper bound for a state $j \in \mathbb{S}$ is simply to check whether there exists an action $a \in \mathbb{A}(i)$ such that the expected stage cost equals $c_i(a) = 0$. In this case, the upper bound is obtained by subtracting 1 from the trivial bound $v^{\alpha}_{\max} = c_{\max}/(1-\alpha) = 1/(1-\alpha)$. Otherwise, the upper bound $v^{\alpha}_{\max}$ is used. Therefore, we obtain:

$$v^{\alpha}_{\max}(j) = \begin{cases} \frac{\alpha}{1-\alpha}, & \text{if there exists } a \in \mathbb{A}(i) \text{ with } c_i(a) = 0, \\ \frac{1}{1-\alpha}, & \text{otherwise.} \end{cases}$$

Studied Instances

An instance of the Markov decision process for the online target date assignment problem with downstream bin packing is given by the following data:

- a deferral time $\delta \in \mathbb{N}$,
- a number of possible item types $n \in \mathbb{N}$ having different sizes each,
- possible item sizes $0 < s_1, \ldots, s_n \leq 1$,
- a probability distribution for the different items sizes: $p^s_1, \ldots, p^s_n \in (0, 1]$ with $\sum_{k=0}^{n} p^s_k = 1$, and
- probabilities $p^d_1, p^d_2, \ldots$, where $0 \leq p^d_k \leq 1$ is the probability for a date change (i. e., the next item will be given at the next date) when k requests have already been issued at the current date.

The instances to be considered in this chapter are denoted by their deferral time δ and the number of possible item sizes n. They are shown in Table 4.2.

Note that we restrict the possible requests to items of size 1/5 or 2/5. Thus, using an action at some the number of possible successor states is at most four. Moreover, our model assumes that the probability of changing the date is non-decreasing in the number of requests yet released at the current date.

instance	δ	n	$s_1, \ldots, s_n$	$p_1^s, \ldots, p_n^s$	$p_1^d, p_2^d, \ldots$
`tda-3-2`	3	2	1/5 2/5	0.5 0.5	0.2 0.3 0.5 0.7 0.9 1.0
`tda-4-2`	4	2	1/5 2/5	0.5 0.5	0.2 0.3 0.5 0.7 0.9 1.0

Table 4.2: Considered instances of the target date assignment Markov decision process.

4.1.5 Elevator Control

Controlling a group of elevators is one of the evident online optimization problems arising in practice. One can distinguish between passenger and cargo elevator systems. In both cases we face an online problem since future transport requests concerning human passengers or tangible goods, respectively, are unknown.

In controlling a group of elevators for serving human passengers the predominant goal is a good service (in recent years energy-saving issues came up as well). That is, one aims to achieve small average and maximum *waiting* and *flow times* for the passengers. The waiting time and the flow time is the time span between the release of the transportation call and the arrival of the serving elevator at the start floor and destination floor, respectively. A few approaches related to MDPs have already been successfully applied to elevator control. For instance, Crites and Barto [CB98] considered a detailed model with a continuous state space for changing traffic patterns. Using techniques from reinforcement learning the authors developed an algorithm that outperforms common control algorithms in simulation.

Some years ago, the paradigm of destination call elevator control has emerged [Clo70, Sch90]. In destination call systems, a passenger enters the destination floor instead of the direction only, which provides more information earlier. For such systems we developed control algorithms based on heuristics and exact reoptimization and evaluated their performance by simulation [HT08, HKT09, HKT10].

Cargo elevator systems frequently arise in logistics applications, for instance in high rack warehouses. As an example, consider an automated warehouse used to store and retrieve goods, see [GHKR99] for details. Pallets with goods have to be stored and are to be transported from the storage to vehicles for further distribution. Such a transport system may include conveyor belts, elevators, and other transportation devices. We concentrate on the control of the elevators, which constitutes an important subsystem. The important characteristics of such a system are the following:

- All pallets with the same start floor have to be served in order of their

arrival.

- The destination floors of all released pallets are known to the system.
- Each elevator has unit capacity, i. e., at any time it can serve at most one pallet.

The overall goal of a good control is to ensure a constant flow of pallets such that the vehicles do not have to wait too long, to avoid congestion, and to coordinate the traffic. Obviously, this goal is quite similar to that for passenger elevator systems.

In this section we develop a Markov decision process model for the problem of controlling cargo elevators.

Mathematical Formulation: The Online Dial-a-Ride Problem

A mathematical formulation for the problem of controlling a single cargo elevator is the *online dial-a-ride problem* on graphs, which is as follows (a generalized version of the online dial-a-ride problem [AKR00, Hau99] considers a metric space instead of a graph).

An instance of the online dial-a-ride problem on graphs consists of an undirected connected graph $G = (V, E)$ with edge weights $d(e)$ for each edge $e \in E$ and a special node $o \in V$, called *origin*. Moreover, we are given online a sequence of transportation requests $r_1, r_2, \ldots, r_n$ for some $n \in \mathbb{N}$. For each $k \in \{1, \ldots, n\}$, the request r_k is a triple $r_k = (t_k, a_k, b_k)$ that specifies its release time t_k and the source and target nodes $a_k, b_k \in V$ between which an object is to be transported. It is assumed that the release times are non-decreasing in the sequence of requests. Another aspect of the problem is that an online algorithm does not have information when the last request arrives nor about the total number of requests.

The requests are to be handled by a single *server* that can move at constant unit speed along the edges in G. The server is allowed to move continuously from one end point of an edge $uv \in E$ to the other one and possibly change its direction while at some position on the edge uv. The server has unit-capacity, i. e., it can transport at most one object at a time. At time 0 the server is located in the origin o and has to return to o after the end of its service. An online algorithm has to move the server along the edges in the graph G so as to fulfill all released transportation requests. Preemption is not allowed: once the server has picked up an object, it is not allowed to drop it at any other place than its destination.

In order to plan the work of the server, an online algorithm may maintain a preliminary *transportation schedule* for all known requests, according to

which it moves the server. A posteriori, the moves of the server induce a complete transportation schedule that may be compared to an offline schedule that is optimal with respect to some objective function (competitive analysis).

The goal is to find a transportation schedule whose cost is as small as possible, where the notion of cost depends on the objective function used. The commonly used objective functions are:

- the total completion time also called *makespan*, which is the time needed to serve all requests and to return to the origin o, and
- the average and maximum flow or waiting time.

We mentioned above that cargo elevator systems require to serve all pallets waiting at the same floor in order of their arrival. The corresponding version of the online dial-a-ride problem, where requests with the same source node are to be served in order of non-decreasing release times, is called *online FIFO dial-a-ride problem*.

Obviously, the online dial-a-ride problem models general transportation problems. The problem of controlling a single elevator can be represented by the special case when the graph G is a path.

Known Results

Various theoretical results for the general online dial-a-ride problem have been established. For the makespan objective there is a 2-competitive algorithm for the general online dial-a-ride problem which was proposed by Ascheuer et al. [AKR00]. The authors show also that no algorithm can be better than 2-competitive, proving that the presented algorithm is best possible w. r. t. competitive analysis. Moreover, they analyze two general online algorithms, namely Replan and Ignore. Both algorithms are based on computing and using an optimal schedule w. r. t. a certain objective function at some points in time, but differ in their frequency this schedule is updated. The Replan strategy recomputes an optimal schedule each time a new request becomes known, which is often used in practice. The algorithm Ignore works in phases. At the beginning of each phase, an optimal schedule for the currently known requests is computed and will be realized. Only when the schedule has been finished, a new schedule for the requests known at that time is computed and a new phase starts. During execution of a schedule all new requests are ignored, which gave the algorithm its name. Typically, the optimal schedules for Replan and Ignore are determined w. r. t. the makespan as objective function. In this case, both Replan and Ignore are 5/2-competitive. If the graph G is restricted to be a path, which represents the situation of an

elevator system, the lower bound of 2 reduces to $5/3 \approx 1.667$, see [AKR98]. Both bounds are true for the online FIFO dial-a-ride problem as well. It has been shown by Hauptmeier [Hau99] that for this problem Ignore achieves the same performance as before, while Replan is only 3-competitive.

However, considering an objective function based on waiting or flow times often seems to be more appropriate than using the makespan: for continuously operating systems with continuously arriving requests the total completion time is meaningless. That is, the positive results above cannot be applied. It is easy to see that for the task of minimizing the maximum or average waiting time or maximum flow time, no online algorithm can be competitive. Moreover, there does not exist an online algorithm with constant competitive ratio when the objective is to minimize the average flow time. Thus, it is impossible to distinguish any two online algorithms by classical competitive analysis for these objective functions. This necessitates alternative evaluation methods. For instance, Hauptmeier et al. [HKR00] showed that assuming a reasonably restricted class of request sequences, the algorithm Ignore achieves bounded maximum and average flow times, which is not true for Replan.

The cargo elevator control problem has also been studied in simulations, e. g., see Grötschel et al. [GHKR99]. In the case of a single elevator, the authors observed that algorithms producing good average flow times give high maximum flow times and vice versa. Moreover, Replan achieved very good average flow times, but also very high maximum waiting times. Ignore gave worse but still acceptable average flow times, but good maximum flow times as well. Furthermore, the online algorithm NN (NearestNeighbor) that is described below achieves even better average flow times than Replan. More recently, Friese and Rambau [FR06] studied a more involved algorithm based on integer programming for multi-elevator systems. Among others, they compared their new algorithm Reopt to the algorithms FIFO (FirstInFirstOut) and NN, each representing a certain class of typical algorithms. The simple rule-based algorithm FIFO assigns a new request in round-robin fashion to the elevators, serving each request of an elevator in the order of arrival. NN assigns a new request to the elevator giving the lowest waiting time for the new request if it is inserted such that no other request is postponed. The requests of each elevator are served such that the distance from the last to the next request is minimized. Simple greedy heuristics like NN are frequently used in practice. Finally, Reopt determines the schedule in an integrated way (the assignment and scheduling decisions are taken into account simultaneously) such that the average waiting time of all the requests is minimized. The simulation results reveal that for higher load intensity the algorithm NN outperforms FIFO and that Reopt is superior to both other algorithms.

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 $E = \{1, \ldots, n_E\}$ in a building with a set of floors $F = \{1, \ldots, n_F\}$. 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 $\sigma_f = r_1, \ldots, r_{n_f}$ of waiting requests, where $n_f \in \{0, \ldots, q\}$ is their number. Moreover, each request r_k for $k \in \{1, \ldots, n_f\}$ 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.

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.

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.

Remark 4.1.4 Similar to the bin coloring problem, it is not required to distinguish different elevators here. Instead of maintaining a sequence of elevators in a state, an unordered multiset is more appropriate to come up with a small state space.

We will deal with two different objective functions, namely minimizing the average or the maximum waiting time. As we will see below, the former does not require to store any waiting times. Thus, the state space can be reduced further and becomes finite for this objective. △

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 $a = (a(e_1), \ldots, a(e_{n_E}))$. 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.

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 $a(e) =$ load, the update for the waiting queue at floor f_e is $\sigma_{f_e}(j) = r_2, \ldots, r_{n_{f_e}}$, where $\sigma_{f_e}(i) = r_1, \ldots, r_{n_{f_e}}$. Otherwise, we have $\sigma_{f_e}(j) = \sigma_{f_e}(i)$.

- The current floor and loading of elevator e are updated by:

$$(f_e(j), d_e(j)) = \begin{cases} (f_e(i), d_e(i)), & \text{if } a(e) = \mathsf{wait}, \\ (f_e(i)+1, d_e(i)), & \text{if } a(e) = \mathsf{move\_up}, \\ (f_e(i)-1, d_e(i)), & \text{if } a(e) = \mathsf{move\_down}, \\ (f_e(i), f_1), & \text{if } a(e) = \mathsf{load}, \\ (f_e(i), 0), & \text{if } a(e) = \mathsf{drop}, \end{cases}$$

  where $r_1 = (f_e, f_1, w_1)$ denotes the first request in the sequence $\sigma_{f_e}(i)$ in the loading case.

In the case a new request $r = (a, b, 0)$ is released at a start floor $a \in F$ and should be transported to the 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:

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

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 mention that the Markov decision process model we consider here has not much in common with that used by Crites and Barto [CB98].

Improved Bounds

Recall that we consider two different elevator control MDPs, one for analyzing the total or average waiting time and another for dealing with the maximum waiting time.

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

The first lower bound $v_{\min}^{\alpha,1}(j)$ 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 $0 \leq p_f \leq 1$ be the probability that a request with start floor $f \in F$ is released at a time slot. Consider a permutation $f_1, \ldots, f_{|F|} \in F$ of the floors such that the probabilities are non-decreasing w. r. t. the permutation: $p_{f_1} \leq \cdots \leq p_{f_{|F|}}$. 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 $p_{f_1}, \ldots, p_{f_{|F|-|E|}}$, i. e., we have:

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

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:

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

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

The second part $v_{\min}^{\alpha,2}(j)$ of the lower bound on v_j^α for a state $j \in \mathbb{S}$ captures the total α-discounted cost resulting from the requests waiting in state j. 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 j will be denoted by 0 and the consecutive time slots by $1, 2, \ldots$. Algorithm 10 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 4.1.5 *Under the assumptions made, Algorithm 10 determines a schedule that serves all waiting requests in state j at a minimum total α-discounted cost for each $0 \leq \alpha < 1$. This cost equals:*

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

Proof. We refrain from giving a rigorous proof here. The idea is to compare an optimal schedule with the one obtained by Algorithm 10 and to use some exchange argument. □

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

$$v_{\min}^{\alpha,2}(j) := \sum_{k=0}^{t} \alpha^k n_k^{\text{wait}} \leq v_j^\alpha.$$

Algorithm 10 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.

Input: a state $j = ((\sigma_f)_{f \in F}, (f_e, d_e)_{e \in E})$ in a Markov decision process with elevator set E and floor set F
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^{wait} is the number of requests still waiting at time slot k for each $k \in \{0, \dots, t\}$
let $n \leftarrow |\bigcup_{f \in F} \sigma_f(j)|$ and let $\Delta_1 \leq \dots \leq \Delta_n$ be the non-decreasing sequence of distances $|a-b|$ of all waiting requests $(w, a, b) \in \bigcup_{f \in F} \sigma_f(j)$
for each $e \in E$ **do** ▷ get minimum loading time slot for elevator e
    if $d_e = 0$ **then** ▷ elevator empty
        $f \leftarrow f_e$ and $t_e \leftarrow 0$
    else ▷ elevator loaded
        $f \leftarrow d_e$ and $t_e \leftarrow |f_e - d_e| + 1$ ▷ driving and dropping time
    end if
    $t_e \leftarrow t_e + \min_{(a,b,w) \in \bigcup_{f \in F} \sigma_f(j)} |f - a|$ ▷ add time to reach request
end for
$t \leftarrow 0$
$n_t^{\text{wait}} \leftarrow n$ ▷ no request served yet
for $k = 1$ to n **do**
    let $e' \in \operatorname{argmin}_{e \in E} t_e$
    $n_{t+1}^{\text{wait}}, \dots, n_{t_{e'}-1}^{\text{wait}} \leftarrow n_t^{\text{wait}}$ ▷ no more requests loaded before time $t_{e'}$
    $n_{t_{e'}}^{\text{wait}} \leftarrow n_t^{\text{wait}} - 1$ ▷ one request loaded at time $t_{e'}$
    $t \leftarrow t_{e'}$
    $t_{e'} \leftarrow t_{e'} + \Delta_i + 2$ ▷ add driving and loading/dropping time
end for
return $n_0^{\text{wait}}, \dots, n_t^{\text{wait}}$

Notice that $v_{\min}^{\alpha,2}(j)$ takes into account the costs incurred from currently waiting requests only, while $v_{\min}^{\alpha,1}(j)$ solely considers costs due to future requests. Therefore, their sum $v_{\min}^{\alpha}(j) := v_{\min}^{\alpha,1}(j) + v_{\min}^{\alpha,2}(j)$ is a valid lower bound for the component v_j^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 $v_{\max}^\alpha(j) \geq v_j^\alpha$ for each state $j \in \mathbb{S}$ 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 num-

ber 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 $c_f := q - |\sigma_f(j)|$ be the remaining capacity at each floor $f \in F$ in state j. 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:

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

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:

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

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_j^α of the optimal value vector for some $j \in \mathbb{S}$, it is possible to take into account the processing of the requests waiting in state j 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 j. Consequently, we obtain an improved upper bound $v_{\max}^\alpha(j)$. 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.

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

The idea for constructing the lower bound on v_j^α is to determine for each floor $f \in F_{\text{wait}}(j)$ 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 $f \in F_{\text{wait}}(j)$ cannot be loaded before time t_f. Consequently, the smallest possible final waiting time of request r_f equals $w_f^{\text{final}} := w_f(j) + t_f$, where $w_f(j)$ denotes the present waiting time of request r_f in state j. Note that the current maximum waiting time $w_{\max}(j)$ will increase if we have $\max_{f \in F_{\text{wait}}(j)} w_f^{\text{final}} > w_{\max}(j)$. By considering all floors $f \in F_{\text{wait}}(j)$ in order of decreasing current waiting times $w_f(j)$, 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 11 describes in detail how to compute these time slots. It is easy to see that Algorithm 11 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_j^α:

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

Clearly, an upper bound $v_{\max}^\alpha(j)$ for a state $j \in \mathbb{S}$ can be derived by arbitrarily serving the requests waiting in state j and assuming that another

Algorithm 11 Algorithm for computing for a given state, a set of time slots, where the maximum waiting time will increase.

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

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 11). For constructing the second part of the bound let $w_{\max}$ be the maximum final waiting time of a request in state j 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 $w_{\max} + 1, w_{\max} + 2, \ldots$. Putting the two parts of the construction together, we obtain the following upper bound on v_j^α:

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

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 $v_j^\alpha(\pi)$ with $j \in \mathbb{S}$ obtained by computing the maximum current waiting time $w(j)$ of a request in state j. It is clear that the stage cost will equal 0 for the time slots $0, \ldots, w_{\max}(j) - w(j) - 1$. This implies

the upper bound:

$$v^{\alpha}_{\max}(j) := \frac{\alpha^{w_{\max}(j)-w(j)}}{1-\alpha} \geq v^{\alpha}_{j}(\pi).$$

We mention that using improved bounds for the elevator control MDPs is crucial in order to obtain acceptable approximation results. The results obtained by the trivial bounds are very weak. We sketch this observation briefly in Section 4.3.4.

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 two models differ only in the stage costs and in some data of the state space. 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 $F := \{1, \ldots, n_F\}$,
- a number of elevators $n_E \in \mathbb{N}$ defining the elevator set $E := \{1, \ldots, n_E\}$,
- 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 instances we will consider are given in Table 4.3. All instances denoted by `ela-*` are Markov decision processes for the case of minimizing the average waiting time, while those for minimizing the maximum waiting time are called `elm-*`. We only look at problems featuring a single elevator or two elevators. For more elevators the action space becomes quite large, which results in bad approximations of the column generation algorithm. 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, the floor 1 is either its start floor or its destination

instance	n_F	n_E	q	c_p	p^r	p^{sd}
`ela-1-2-100-02-ud`	8	1	2	100	0.2	`ub`
`ela-1-2-10-02-ud`	8	1	2	10	0.2	`ub`
`ela-1-4-10-02-ud`	8	1	4	10	0.2	`ub`
`ela-2-4-10-03-ud`	8	2	4	10	0.3	`ud`
`ela-1-4-10-02-sp`	8	1	4	10	0.2	`sp`
`elm-1-02-ud`	8	1	∞	-	0.2	`ud`

Table 4.3: 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 4.4 and Table 4.5, respectively.

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

Table 4.4: 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 $f_1, f_2 \in \{2, \ldots, 8\}$.

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	-	-

Table 4.5: The start-to-destination floor probability distribution `sp` representing a special situation.

instance	$\lvert\mathbb{S}\rvert$	$c_{\max}$	b	d
`bc-2-3-6-*`	5 424	1	2	6
`bc-3-3-7-*`	122 871	1	3	7
`bc-3-4-12-*`	172 635 060	1	3	12
`tda-3-2`	230 076	1	3	4
`tda-4-2`	16 421 870	1	4	4
`ela-1-2-100-02-ud`	6 357 609	36	4	15
`ela-1-2-10-02-ud`	6 357 609	18	4	15
`ela-1-4-10-02-ud`	11 160 234 375	34	4	15
`ela-2-4-10-03-ud`	$>$ 11 160 234 375	35	16	15
`ela-1-4-10-02-sp`	2 086 898 858	18	4	15
`elm-1-02-ud`	∞	1	4	15

Table 4.6: Overview on the structure of the considered Markov decision processes. The notation is taken from Theorem 3.2.2: $c_{\max} := \max_{i\in\mathbb{S},a\in\mathbb{A}(i)} c_i(a)$, $b := \max_{i\in\mathbb{S}}|\mathbb{A}(i)|$, and $d := \max_{i\in\mathbb{S},a\in\mathbb{A}(i)}|\{j \in \mathbb{S} \mid p_{ij}(a) > 0\}|$.

floor (see Table 4.4). 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 4.5 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.

4.2 Analysis of the Approximation Algorithm

In the Sections 4.1.3–4.1.5 Markov decision processes modeling the online bin coloring problem, an online target date assignment problem, and two online elevator control problems have been introduced. Table 4.6 gives an overview on the structure of all instances to be considered in the sequel. In the following we will refer to associated discounted MDPs as bin coloring, target date assignment, average waiting time elevator control, and maximum waiting time elevator control MDPs.

Before we turn to the analysis of policies for the considered MDPs, we will study in this section the performance of the approximation algorithm. Moreover, we will compare different linear programming solver settings as well the

MDP	Trivial state
bin coloring	$c = 1, \chi = 0, f_k = 0, C_k = \emptyset \; \forall k \in \{1, \ldots, m\}$
target date assignment	$s = 1/5, n = 1, S_k \equiv 0 \; \forall k \in \{1, \ldots, \delta\}$
elevator control	$w_{\max} = 0, \sigma_f = \emptyset \; \forall f \in F, (f_e, d_e) = (1, 0) \; \forall e \in E$

Table 4.7: Trivial states for the considered MDPs.

pricing strategies and approximation heuristics introduced in Section 3.5.3 and Section 3.5.4.

For all computations presented in this section we use initial states that can be seen as being trivial, reflecting a default situation. These states are as follows for the considered MDPs. For the bin coloring MDP this state is $i_{\text{bc}} = (c, \chi, f_1, C_1, \ldots, f_m, C_m)$ with $c = 1$, $\chi = f_k = 0$, and $C_k = \emptyset$ for each $k \in \{1, \ldots, m\}$. That is, all bins are empty and the current request with color 1 is the first one to be considered. Very similar is the definition of the trivial state i_{tda} for the target date assignment MDP. The state is of the form $i_{\text{tda}} = (s, n, S_1, \ldots, S_\delta)$ with $s = 1/5$, $n = 1$, and $S_k \equiv 0$ for each $k \in \{1, \ldots, \delta\}$, i. e., no request has been assigned to one of the feasible target dates yet and the first request at the current date is an item of size $1/5$. In the elevator control MDPs we assume for the trivial state that the maximum waiting time equals 0, no request is waiting, and all elevators are empty and situated at floor 1: we have $i_{\text{elv}} = (w_{\max}, (\sigma_f)_{f \in F}, (f_e, d_e)_{e \in E})$ with $w_{\max} = 0$, $\sigma_f = \emptyset$ for each $f \in F$, $(f_e, d_e) = (1, 0)$ for each $e \in E$. The trivial states are summarized in Table 4.7.

4.2.1 Neighborhood Construction versus Column Generation

First we analyze the performance of our approximation algorithm compared to the method that directly applies the construction of Theorem 3.2.2 on page 51 for approximating the component $v^{\alpha}_{i_0}$ of the optimal value vector for a particular state i_0. As an example we look at the target date assignment MDP (`tda-4-2`, 0.7), i. e., the underlying Markov decision process is given by the instance `tda-4-2` and the discount factor equals $\alpha = 0.7$. The studied initial state is the trivial state for the target date assignment MDP $i_0 = i_{\text{tda}}$ with $\delta = 4$, see Table 4.7.

Table 4.8 shows for different values for the radius $r \in \mathbb{N}_0$, the absolute approximation guarantee $\varepsilon(r)$ provided by the r–neighborhood $S(i_0, r)$ of i_0 in the worst-case, cf. Theorem 3.2.2. Moreover, the table specifies the size of the r–neighborhood and the absolute guarantee $\varepsilon(S(i_0, r))$ achieved practically

r	$\varepsilon_1 := \varepsilon(r)$	$\lvert S(i_0, r)\rvert$	$\varepsilon_2 := \varepsilon(S(i_0, r))$	$\lvert S_{\varepsilon_1}\rvert$	$\lvert S_{\varepsilon_2}\rvert$	$\frac{\lvert S_{\varepsilon_2}\rvert}{\lvert S(i_0,r)\rvert}$
0	2.33	1	2.33	1	1	100.0 %
1	1.63	16	1.63	8	9	56.3 %
2	1.14	154	1.09	45	49	31.8 %
3	0.80	824	0.68	99	142	17.2 %
4	0.56	3 224	0.39	238	437	13.6 %
5	0.39	10 286	0.25	447	1 091	10.6 %
6	0.27	25 086	0.15	955	2 125	8.5 %
7	0.19	53 490	0.09	1 528	3 880	7.3 %
8	0.13	103 678	0.06	2 422	5 946	5.7 %
9	0.09	187 264	0.03	3 736	9 349	5.0 %
10	0.07	319 694	0.02	5 291	14 230	4.5 %

Table 4.8: Required number of states: r–neighborhood vs. column generation algorithm. The start state is $i_0 = i_{\text{tda}}$ defined in Table 4.7. The value $v^{\alpha}_{i_0}$ is about 1.42 which yields an impression for the relative approximation quality.

by solving the associated linear programs $(\mathrm{L}^{i_0}_{S(i_0,r)})$ and $(\mathrm{U}^{i_0}_{S(i_0,r)})$. We compare the size of r–neighborhood of i_0 with the number of states required by our column generation algorithm to obtain an approximation guarantee of $\varepsilon(r)$ and $\varepsilon(S(i_0, r))$, respectively. It should be mentioned that all computations in this sections are based on the trivial lower and upper bounds: $v^{\alpha}_{\min} = 0$ and $v^{\alpha}_{\max} = 1/(1-\alpha)$.

The first observation is that the values for $\varepsilon_2 := \varepsilon(S(i_0, r))$ are significantly smaller than those for $\varepsilon_1 := \varepsilon(r)$ for larger values of r. That is, the actual bounds achieved by the neighborhood $S(i_0, r)$ may be in fact much better than guaranteed by Theorem 3.2.2. We know from Remark 3.2.6 that this is due to cycles within the r–neighborhood. Secondly, we compare the values of $\lvert S(i_0, r)\rvert$ and $\lvert S_{\varepsilon_1}\rvert$ in Table 4.8. As expected, the column generation algorithm requires substantially fewer states as in the theoretical worst-case for non-trivial approximation guarantees. Even the number of states $\lvert S_{\varepsilon_2}\rvert$ explored by our algorithm to obtain the guarantee ε_2, that is actually achieved by the neighborhood $S(i_0, r)$, is only a little fraction of the size of $S(i_0, r)$.

The observations above raise the question concerning the structure of the reduced state spaces $S \subset \mathbb{S}$ determined by the approximation algorithm. Considering the same MDP and state i_0 as above, Table 4.9 shows for each depth w. r. t. state i_0, how many states exist and how many of them were generated by the column generation algorithm. The considered approximation guarantee is 6 %. We see that the required subset of states can be far

r	$\lvert S_r\rvert$	$\lvert S \cap S_r\rvert$	$\lvert S_r \cap S\rvert / \lvert S_r\rvert$
0	1	1	100.0 %
1	15	15	100.0 %
2	138	52	37.7 %
3	670	92	13.7 %
4	2 400	343	14.3 %
5	7 062	627	8.9 %
6	14 800	799	5.4 %
7	28 404	898	3.2 %
8	50 188	796	1.6 %
9	83 586	442	0.5 %
10	132 430	84	0.1 %

Table 4.9: Distribution of the state space $S \subset \mathbb{S}$ generated by the approximation algorithm per depth. The subset of states in depth r w. r. t. state i_0 is denoted by $S_r := S(i_0, r) \setminus S(i_0, r-1)$, where $S(i_0, r)$ is again the r–neighborhood.

away from being an r–neighborhood for some $r \in \mathbb{N}$. This is typically the case for practical MDPs.

In [HKP+06] we assessed the performance of the approximation algorithm depending on different parameters, in particular the discount factor α. To shorty summarize the results, the number of states required by the algorithm to achieve a given approximation guarantee increases drastically for increasing values of α. As mentioned earlier, if α is very close to 1, an optimal policy w. r. t. the total expected α-discounted cost tends to be close to an optimal policy w. r. t. the average expected cost per stage. This, however, implies that it is insufficient to take into account only a local part of the state space in order to provide reasonable approximations. That is, for large values of the discount factor α our approach does not work unless the total state space is very small itself.

4.2.2 Linear Programming Solving

In this section we analyze the performance of the different linear programming solvers available in CPLEX [ILO], version 12.1, in the context of our column generation algorithm. Let again $S \subseteq \mathbb{S}$ with $i_0 \in S$ denote the currently considered subset of states. The last part of Section 3.5.2 described a method to construct a start basis for the linear program $(\mathrm{L}_S^{i_0})$ and $(\mathrm{U}_S^{i_0})$, respectively, from the optimal basis of the associated previously solved linear

MDP instance	no. vars. per it.	no start basis primal	no start basis dual	use start basis primal	use start basis dual
(bc-3-3-7-spe, 0.97)	10	4644	1599	15156	941
	100	2919	406	14173	225
	1000	4163	421	3121	248
(tda-4-2, 0.7)	10	1255	1030	1257	725
	100	846	592	782	304
	1000	832	533	498	241
(ela-1-2-10-02-ud, 0.8)	10	2346	1839	1192	1060
	100	563	361	200	147
	1000	476	248	60	48

Table 4.10: Required running times in seconds for solving the encountered linear programs using different settings for the simplex algorithm, depending on the maximum number of added variables in each pricing iteration. In each run 20 000 variables are generated by the approximation algorithm.

program. This basis is usually dual feasible. Otherwise, the dual simplex only requires very few pivot steps to obtain a dual feasible basis. Without passing an initial basis, Cplex constructs an initial basis from the optimal basis of the last iteration. This, however, usually causes some effort for the used simplex method only to obtain a feasible basis in phase one. In the following, we look at the performance of the primal and the dual simplex method, both with and without using our construction of an initial basis.

We consider the three MDPs (bc-3-3-7-spe, 0.97), (tda-4-2, 0.7), and (ela-1-2-10-02-ud, 0.8) and approximate the component $v_{i_0}^{\alpha}$ of the optimal value vector for each MDP by generating 20 000 variables, where i_0 is in each case the associate trivial state given in Table 4.7. We analyze the performance of the four solver settings depending on the number of generated variables per pricing iteration. Table 4.10 shows the associated total required runtimes for solving the linear programs encountered in the column generation algorithm for the three MDP instances. The results show that the dual simplex is faster than the primal simplex when both methods use the same way of setting the initial basis. Moreover, the dual simplex method combined with the proposed way to construct a start basis achieves the fastest running times in each approximation run. In the case that up to 1000 variables are generated in each pricing iteration, which seems to be most suitable for larger MDPs, the performance of this setting is significantly better than that of all other solver settings. Solely for the MDP (ela-1-2-10-02-ud, 0.8), the primal simplex method using the involved basis construction is only slightly

MDP instance	number of added variables per iteration					
	10		100		1000	
	dir.	com.	dir.	com.	dir.	com.
(bc-3-3-7-spe, 0.97)	833	391	77	44	8	5
(tda-4-2, 0.7)	2127	2377	202	198	21	21
(ela-1-2-10-02-ud, 0.8)	1706	4371	166	310	16	32
(elm-1-02-ud, 0.8)	8457	11628	954	1150	93	128

Table 4.11: Required running times for pricing new variables in seconds for the strategies direct (dir.) and combinatorial (com.) to generate 20 000 in different MDPs, depending on the maximum number of added variables in each pricing iteration.

slower.

We mention that the barrier method of CPLEX is not competitive at all: even a parallelized computation using four processors is much slower than the simplex algorithms. The reason is that the barrier method lacks the possibility of a warm start which could exploit information like suitable start bases. In all the following computations we apply the dual simplex method incorporating the proposed construction of an initial basis.

4.2.3 Pricing Strategies

In this section, the four pricing strategies introduced in Section 3.5.3 namely direct, combinatorial, min-depth, and improving are compared by means of numerical results. On the one hand, we aim to analyze the quality of these strategies, i. e., their capability to provide a good approximation by taking into account only a small subset of states. On the other hand, we assess the running time of our approximation algorithm depending on the employed pricing strategy.

Recall that direct and combinatorial are conceptually equivalent as both pricing strategies determine a state with a maximum reduced profit and only differ in their way to find such a state algorithmically. Since we intend to consider only one of these two strategies when dealing with the mentioned issues, we initially compare the run-times of direct and combinatorial.

Table 4.11 shows the required computing times for pricing new variables in the approximation of the component $v^{\alpha}_{i_0}$ of the optimal value vector for four MDPs. In each case, the state i_0 is the associated trivial state defined in Table 4.7 and 20 000 variables were generated in total. Apparently, the pricing strategy combinatorial is superior to direct for the bin coloring MDP

(bc-3-3-7-spe, 0.97), while the opposite is the case for the elevator control MDPs (ela-1-2-10-02-ud, 0.8) and (elm-1-02-ud, 0.8). Both strategies achieve similar run-times for the instance (tda-4-2, 0.7).

In the following we will argue how the structure of an MDP should affect the computing time for the two pricing strategies. Consider some point in the column generation process to approximate $v_{i_0}^{\alpha}$ with $i_0 \in \mathbb{S}$, and let $S \subset \mathbb{S}$ be the current subset of states and $S_{\text{cand}} \subseteq \mathbb{S} \setminus S$ be the associated set of candidate states as defined by Equation (3.3.2) on page 63. Note that the complexity of the pricing strategy direct scales with the total number of candidate states $|S_{\text{cand}}|$ since the reduced profit of each state in S_{cand} has to be computed. In contrast, the complexity of the strategy combinatorial scales with the number of visited states $|S_{\text{vis}}|$ in the recursive search for candidate states with positive reduced profits, see Algorithm 5 on page 91. Recall that S_{vis} includes both states in S and states in S_{cand}. Thus if the ratio $|S_{\text{cand}}|/|S_{\text{vis}}|$ is small, the pricing strategy direct should be more appropriate, and vice versa.

Unfortunately, this observation is only partially true for our implementation of the approximation algorithm especially since computing the reduced profit of a candidate state by the strategy combinatorial is more expensive than for direct. We maintain a data structure to store all candidate states together with the associated predecessors in S to determine the reduced profits. While the strategy direct employs this data structure directly, combinatorial requires an additional search operation whenever a candidate state is reached. Therefore, the run-times in Table 4.11 do not relate to the above argumentation: after having generated 10 000 states, the ratio $|S_{\text{cand}}|/|S_{\text{vis}}|$ is smallest for the instance (tda-4-2, 0.7) and greatest for (ela-1-2-10-02-ud, 0.8).

Due to memory requirements it may be reasonable to refrain from maintaining a data structure that stores the candidate states with their associated predecessors or to store only the candidates alone. Note that the pricing strategy direct cannot be used anymore in the first case, whereas combinatorial may still be applied. In the latter case, combinatorial will be advantageous compared direct since fewer predecessor states have to be determined.

Since the pricing strategy direct seems to be advantageous for most of the MDPs considered here, in particular for the more complex elevator control MDPs, we will refrain from analyzing the pricing strategy combinatorial further, but solely look at direct.

Quality of Pricing Strategies

In order to analyze the quality of the pricing strategies, we use each strategy to generate 4000 states for the studied MDPs. In doing so, exactly one state is

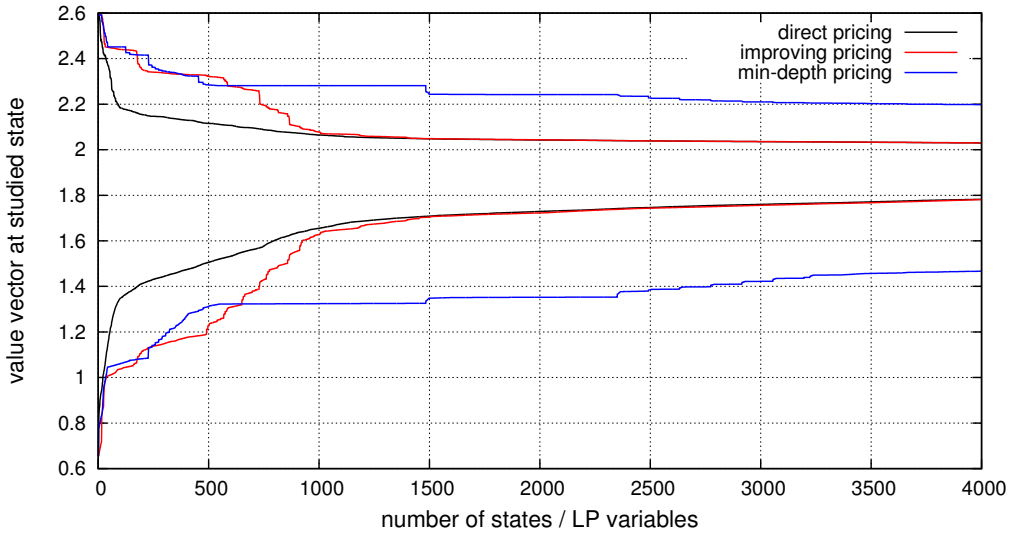


Figure 4.1: Comparison of the quality of different pricing strategies for the elevator control MDP (`ela-1-2-10-02-ud`, 0.8). Only one variable is added in each pricing iteration.

added in each pricing iteration. This ensures that only the state favored most by the considered strategy is generated. Adding several states would distort the actual difference between the strategies: the more states are generated each step, the less differs the quality of the pricing strategies.

Let us look at the elevator control MDPs (`ela-1-2-10-02-ud`, 0.8) and (`elm-1-02-ud`, 0.8). Figures 4.1 and 4.2 depict the progress in approximating $v^{\alpha}_{i_{\text{elv}}}$ achieved by the pricing strategies direct, improving, and min-depth when the associated involved state-specific bounds described in Section 4.1.5 are incorporated in the linear programs. Each picture shows the lower and upper bounds on $v^{\alpha}_{i_{\text{elv}}}$ obtained by the approximation algorithm depending on the number of explored states / variables in the linear programs.

Figure 4.1 shows that for the MDP (`ela-1-2-10-02-ud`, 0.8) the strategies direct and improving achieve a very similar approximation guarantee once about 1000 variables have been generated. For fewer states, direct provides much better bounds than improving. Moreover, the pricing strategy min-depth performs very bad. Its final relative approximation guarantee equals 44.6 %, whereas direct and improving yield a guarantee of 12.2 % and 12.4 %, respectively.

The results for the MDP (`elm-1-02-ud`, 0.8) given in Figure 4.2 are much different. Obviously, the bounds obtained by improving are disastrous. They even do not seem to improve in the approximation process at all (which, however, is only true for the upper bound, the lower bound improves very

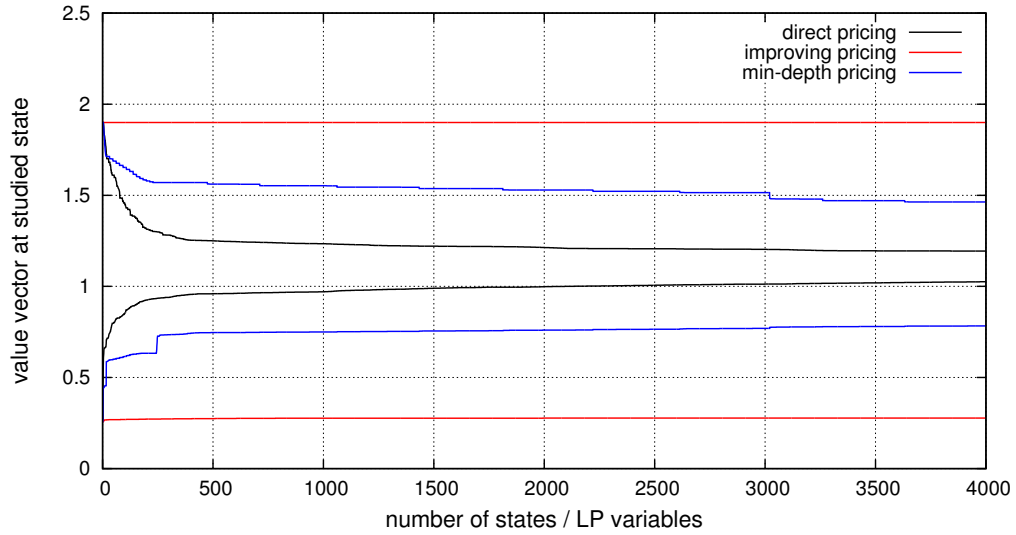


Figure 4.2: Comparison of the quality of different pricing strategies for the elevator control MDP (`elm-1-02-ud`, 0.8). Only one variable is added in each pricing iteration.

little). As before, the pricing strategy direct provides much better bounds than min-depth. The achieved approximation guarantee equals 14.6 % for direct, 84.2 % for min-depth, and 584.4 % for improving. The reason for the bad performance of improving is as follows. Recall that this pricing strategy always choose a state $j \in S_{\text{cand}}$ with positive weight w_j as defined in Equation (3.5.20) on page 90 if possible. For the considered approximation run improving can find such a state in each iteration. In particular, improving always adds such states $j \in \mathbb{S}$ that satisfy $\min_{a \in \mathbb{A}(j)} c_{ja} > 0$, which implies a positive weight w_j. However, these states tend to have really small reduced profits, while states with great reduced profits are never generated. Adding solely the state with maximum reduced profit in the first pricing iteration to the initial subset $\{i_0\}$ results in a better lower bound than that achieved by improving after having generated 4000 states.

We presented the results for the two elevator control MDPs as they reflect extreme outcomes. We mention that additional results for the bin coloring and target date assignment MDPs do not differ to that extend for the studied pricing strategies. Altogether, we conclude that the pricing strategy direct, which adds a state of maximum reduced profit, typically provides the best bounds by far.

The results showed that incorporating the amounts of the reduced profits pays off substantially: the simple min-depth strategy cannot cope with direct. Moreover, it is a bit disappointing that the bounds of improving are often

MDP instance	direct	improving	min-depth
(bc-3-3-7-spe, 0.97)	39	9	117
(tda-4-2, 0.7)	32	12	42
(ela-1-2-10-02-ud, 0.8)	9	3	11
(elm-1-02-ud, 0.8)	8	3	11

Table 4.12: Number of variables required to increase the initial lower bound obtained by $(\mathrm{L}^{i_0}_{\{i_0\}})$ for four MDPs.

worse than that of direct since improving was developed to improve the pure reduced cost criterion of direct by taking into account additional information namely the feasible range of new variables. Motivated by the bad results for improving, we also analyzed how many states are required to be added to the initial state space $\{i_0\}$ to reach the first increase of the computed lower bound on $v^{\alpha}_{i_0}$. Here we refrain from incorporating the involved lower bounds for the candidate states, but use the trivial lower bound $v^{\alpha}_{\min} = 0$ in the linear programs. For the considered MDPs and each of the different pricing strategies, Table 4.12 shows the size of the minimal subset of states $S \subseteq \mathbb{S}$ that is required to be generated such that the lower bound provided by the linear program $(\mathrm{L}^{i_0}_{S})$ is better than that due to $(\mathrm{L}^{i_0}_{\{i_0\}})$.

The results clearly show that improving outperforms the other pricing strategy when the goal is to increase the initial lower bound for $v^{\alpha}_{i_0}$ with a minimum number of new states. In the long run, however, using the pricing strategy improving seems to be disadvantageous for similar reasons as figured out for the concrete situation described above. An approach to prevent the observed drawbacks of the strategy improving would be to mainly focus on the reduced profits and to take into account the used weights with lower priority.

Performance of Pricing Strategies

Finally, we study the practical performance of the pricing strategies. Adding only one variable to the explored state space in each iteration is inadequate to achieve a good run-time of the approximation algorithm. Instead, generating up to about 1000 variables if possible per pricing iteration seems to be reasonable for the considered applications. Figure 4.3 shows the associated bounds achieved by the considered pricing strategies for the bin coloring MDP (bc-3-3-7-spe, 0.97) for approximating the value $v^{\alpha}_{i_{\mathrm{bc}}}$. The computation terminates once a relative guarantee of 10 % has been obtained.

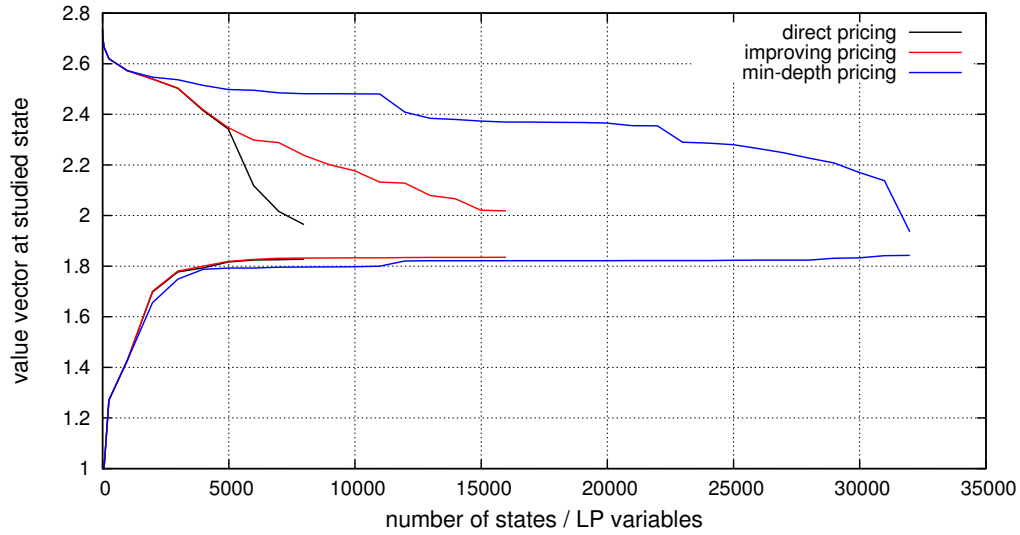


Figure 4.3: Comparison of the approximation progress of different pricing strategies for the bin coloring MDP (`bc-3-3-7-spe`, 0.97) to achieve a guarantee of 10 %. Up to 1000 variables are added in each pricing iteration if possible.

MDP instance	direct		improving		min-depth	
	time	$\|S\|$	time	$\|S\|$	time	$\|S\|$
(`bc-3-3-7-spe`, 0.97)	13	7981	79	15981	345	31981
(`tda-4-2`, 0.7)	9	3568	10	3568	8	3568
(`ela-1-2-10-02-ud`, 0.8)	84	7332	99	8332	41419	481332
(`elm-1-02-ud`, 0.8)	211	23675	approx. failed		approx. failed	

Table 4.13: Required running times in seconds and number of explored states to achieve a guarantee of 10 % for the considered pricing strategies. Up to 1000 variables are added in each pricing iteration if possible.

The strategies direct, improving, and min-depth require 7981, 15 981, and 31 981 states, respectively. For this example direct again outperforms the other strategies significantly w. r. t. the number of required states. The crucial aspect, however, is the actual running time of the approximation algorithm due to the used pricing strategy. Table 4.13 shows the run-times and the total number of explored states to achieve an approximation guarantee of 10 % for the four studied MDPs.

Obviously, using the pricing strategy direct results in the fastest version of the approximation algorithm. Only for the target date assignment MDP (`tda-4-2`, 0.7), the strategy min-depth yields a better running time

than direct. In this case, all pricing strategies generate the same restricted state space since in each pricing iteration all candidate states having a positive reduced profit added, i. e., their number is at most 1000. Therefore, it is clear that min-depth should be faster then the other more involved strategies. For the elevator control MDP (`elm-1-02-ud`, 0.8), only the pricing strategy direct is able to complete the approximation process. Using the other strategies, the algorithm runs out of memory since our implementation maintains all candidate states, which become extremely many for this MDP.

The results in this section revealed that the pricing strategy direct typically generates the least number of states and yields the best running times to reach a given approximation guarantee. In the following, all computations are based on the strategy direct.

4.2.4 Approximation Heuristics

In this section we computationally analyze the policy-based approximation heuristics proposed in Section 3.5.4, namely breadth-first-policy-exploration, pricing-policy-exploration, and weighting-policy-exploration. Their precise setting for a given policy π is as follows.

breadth-first-policy-exploration Let $S(i_0, r, \pi) \subseteq S(i_0, r)$ for $r \in \mathbb{N}$ denote the subset of states in the r–neighborhood $S(i_0, r)$ that can be reached from i_0 by policy π using at most r transitions. Initially, the state space $S_1 := S(i_0, r, \pi)$ with $r = 3$ is constructed. As long as S_1 does not satisfy Inequality (3.5.21) on page 98, we increase the radius r by one and extend S_1 such that $S_1 = S(i_0, r, \pi)$.

pricing-policy-exploration Recall that the heuristic pricing-policy-exploration applies the standard approximation algorithm restricted to policy π. We generate up to 1000 states in each pricing operation.

weighting-policy-exploration Recall that weighting-policy-exploration can be seen as an approximate variant of pricing-policy-exploration which tries to reduce the number of linear programs to be solved. We use the error estimate $\delta = 1$.

The policy-based heuristics employ the following policies: GreedyFit and PFD are used for the MDPs (`bc-3-3-7-spe`, 0.97) and (`tda-4-2`, 0.7), respectively, while NN is applied for the elevator control MDPs (`ela-1-2-10-02-ud`, 0.8) and (`elm-1-02-ud`, 0.8).

We approximate the value $v_{i_0}^{\alpha}$ for the considered MDPs up to a guarantee of 8 % using the mentioned heuristics, where i_0 is the associated trivial

MDP instance	bfpe		ppe		wpe		no heuristic	
	time	$\|S\|$	time	$\|S\|$	time	$\|S\|$	time	$\|S\|$
(bc-3-3-7-spe, 0.97)	188	23009	31	12072	16	11275	15	7981
(tda-4-2, 0.7)	19	5800	18	5652	11	4371	8	3568
(ela-1-2-10-02-ud, 0.8)	37685	532461	498	9204	548	8792	502	9332
(elm-1-02-ud, 0.8)	approx. failed		1099	49225	1203	52442	1120	48675

Table 4.14: Required running times in seconds and number of explored states to achieve a guarantee of 10 % for the considered pricing strategies. Up to 1000 variables are added in each pricing iteration if possible.

state given in Table 4.7. In each pricing iteration up to 1000 variables are generated. Table 4.14 shows the total run-times and the total number of generated states for the column generation algorithm using any or none of the approximation heuristics under consideration. As expected, the heuristic breadth-first-policy-exploration gives the worst running-time and requires the most states for all considered MDPs. Except for the target date assignment MDP (tda-4-2, 0.7), the results of breadth-first-policy-exploration are very bad. In the case of the MDP (elm-1-02-ud, 0.8) the heuristic breadth-first-policy-exploration even runs out of memory. The results of the three remaining settings are quite similar to each other. However, only for the elevator control MDPs the heuristic pricing-policy-exploration achieves a slighty better run-time than the column generation algorithm without incorporating an approximation heuristic. Thus, we conclude that incorporating the considered approximation heuristics in the column generation algorithm does seem to be advantageous.

Figure 4.4 illustrates in detail the performance of the approximation heuristics for the bin coloring MDP (bc-3-3-7-spe, 0.97). Each bend in the approximation curves shows the point where the heuristic terminates having computed a subset of states $S_1 \subseteq \mathbb{S}$ that fulfills Inequality (3.5.21). Up to this point the figure depicts the obtained lower bounds for $v_{i_0}^{\alpha}(\pi)$, afterwards those for the component $v_{i_0}^{\alpha}$ of the optimal value vector.

It is especially disappointing that using the heuristic weighting-policy-exploration does never pay off. We hoped that weighting-policy-exploration would be really fast since only few linear programs are solved in the heuristic. For two reasons we do not observe this behavior. On the one hand, the heuristic weighting-policy-exploration is hard to be parameterized such that only very few linear programs are solved. On the other hand, we observed in Section 4.2.2 that the linear programs arising in the approximation process can be solved really fast due to the construction of a suitable initial basis.

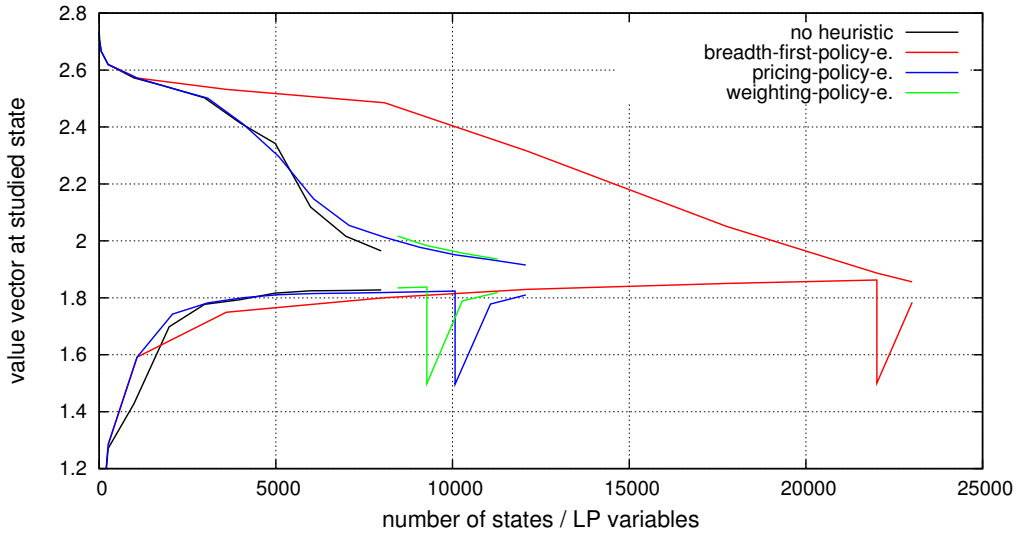


Figure 4.4: Approximation results depending on the used approximation heuristic for the bin coloring MDP (`bc-3-3-7-spe`, 0.97) and the trivial initial state i_{bc} defined in Table 4.7.

We believe that this second aspect significantly degrades the potential of employing an approximation heuristic in general. Furthermore, the results above substantiate that the standard approximation algorithm based on a good setting for the linear programming solver and the pricing strategy runs fast and is effective for generating such subsets of states that provide good approximations.

We have to mention that it is crucial for the practicability of our approach to employ the approximation algorithm based on the best arrangement for its components and possible extensions and to incorporate strong state-specific bounds in the linear programs. Otherwise, results like those provided in the next section cannot be obtained, or only at an extremely high computational effort.

4.3 Analysis for Exemplary MDPs

In this section our approximation algorithm is used to evaluate policies for discounted MDPs emerging from the introduced Markov decision processes modeling online optimization problems. In particular, we will consider the online algorithms described in the Sections 4.1.3–4.1.5. For all considered MDPs, no explicit optimal policy is known.

4.3.1 Subjects of Evaluation

As mentioned before, we see the main use of our approximation method in evaluating online algorithms / policies that possibly behave well in simulation experiments or even in practice, although the classical analysis tools (e. g., based on the competitive ratio) do not indicate this. We thus aim at estimating and reporting for each state $i_0 \in \mathbb{S}$, that is reached while running a simulation or real-world system, *online* the quantity:

$$\varepsilon_{i_0}^{\alpha}(\pi) := \frac{v_{i_0}^{\alpha}(\pi) - v_{i_0}^{\alpha}}{v_{i_0}^{\alpha}} \quad \text{or} \quad \varepsilon_{i_0}^{\alpha}(\pi(i_0)) := \frac{v_{i_0}^{\alpha}(\pi(i_0)) - v_{i_0}^{\alpha}}{v_{i_0}^{\alpha}}, \tag{4.3.1}$$

where π is a particular policy for the considered MDP. The values $\varepsilon_{i_0}^{\alpha}(\pi)$ and $\varepsilon_{i_0}^{\alpha}(\pi(i_0))$ give 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 (4.3.1) 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.

In the following, we will refrain to consider all states visited in a complete simulation run since the evaluation results obtained that way are typically quite boring for most situations. Instead, we look at a few specific states that highlight interesting outcomes.

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

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

In the following we will refer to the values $v_{i_0}^{\alpha}$, $v_{i_0}^{\alpha}(\pi)$, and $v_{i_0}^{\alpha}(\pi(i_0))$ simply as the *optimal cost*, the *cost of policy* π, and the *cost of action* $\pi(i_0)$ *at state* i_0, 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 values $\varepsilon_{i_0}^{\alpha}(\pi)$ and $\varepsilon_{i_0}^{\alpha}(\pi(i_0))$ for each analyzed policy π or action $\pi(i_0)$.

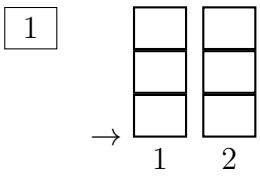


Figure 4.5: State i_1 in a bin coloring MDP with $m = 2$ bins of size $b = 3$. The state is of the form $i_1 = (c, \chi, f_1, C_1, f_2, C_2)$, where $c = 1$, $\chi = f_1 = f_2 = 0$, and $C_1 = C_2 = \emptyset$.

4.3.2 Bin Coloring MDPs

We start by considering different bin coloring MDPs and aim at investigating the following questions:

1. Is GreedyFit better than OneBin w. r. t. the total expected discounted cost (recall that OneBin outperforms GreedyFit in terms of competitive analysis)?

2. How close is GreedyFit to an optimal policy? And if GreedyFit is not almost optimal, can we design a policy that outperforms GreedyFit?

In order to answer these questions, we will consider the instances of the Markov decision process for bin coloring shown in Table 4.1 on page 115.

Instances `bc-2-3-6-*`

First we deal with the bin coloring MDPs $M_2^{\text{uni}} := (\texttt{bc-2-3-6-uni}, 0.97)$ and $M_2^{\text{spe}} := (\texttt{bc-2-3-6-spe}, 0.97)$. Note that both MDPs have the same state space $\mathbb{S}$. Using such a large discount factor of $\alpha = 0.97$ is possible since the state space $\mathbb{S}$ is quite small here: the MDPs only consist of $|\mathbb{S}| = 5424$ states. Thus, it is even possible for these MDPs to generate their complete state space $\mathbb{S}$ and to solve the associated linear programs $(\mathrm{L}_{\mathbb{S}}^{i_0}) = (\mathrm{P}^{i_0})$ for some state $i_0 \in \mathbb{S}$. The concrete value for the discount factor was chosen to obtain significant approximation results. We again mention that for a discount factor α' very close to 1, the total expected α'-discounted cost almost equals the total expected undiscounted cost. This implies that using the discount factor α' makes all policies perform similarly since each policy will eventually reach the maximum colorfulness of $b = 3$ with probability 1.

Initially, we consider the trivial state $i_1 := i_{\text{bc}} = (c, \chi, f_1, C_1, f_2, C_2) \in \mathbb{S}$ shown in Figure 4.5. This illustration is to be interpreted as follows. The new item with color c is depicted at the left, while the current configuration (f_1, C_1, f_2, C_2) of bin 1 and bin 2 is shown at the right. The arrow indicates the maximum colorfulness χ.

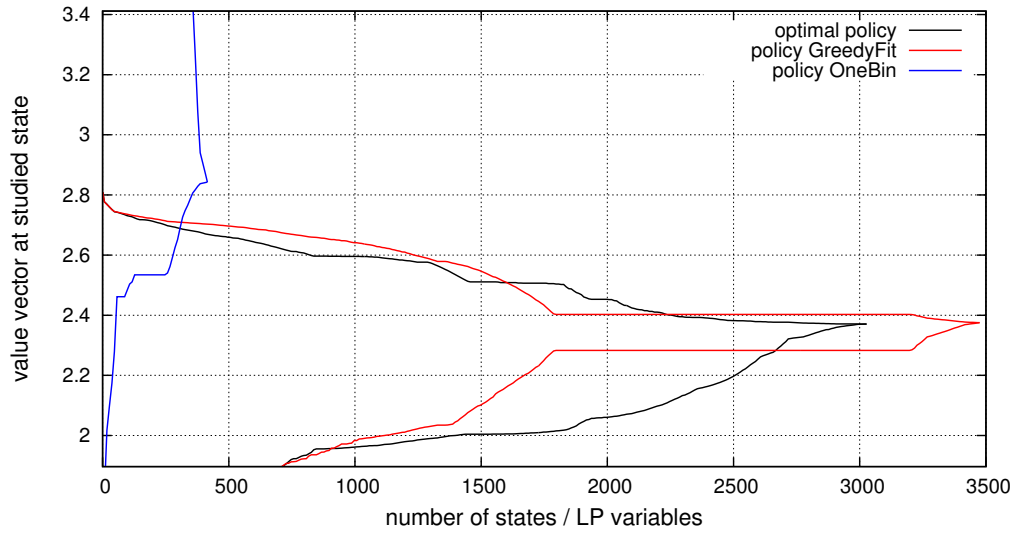


Figure 4.6: Approximation results for the bin coloring MDP M_2^{uni} and the trivial initial state i_1 defined in Figure 4.5. We have $\varepsilon_{i_1}^{\alpha}(\mathsf{GreedyFit}) = 0.2\,\%$ and $\varepsilon_{i_1}^{\alpha}(\mathsf{OneBin}) = 19.9\,\%$.

To investigate the performance of the policies OneBin and GreedyFit, we compare the approximation of $v_{i_0}^{\alpha}(\mathsf{GreedyFit})$ and $v_{i_0}^{\alpha}(\mathsf{OneBin})$ with that of the optimal total expected α-discounted cost $v_{i_0}^{\alpha}$ for the start state $i_0 = i_1$. The Figures 4.6 and 4.7 show the approximation results of these three values for the case of uniform (MDP M_2^{uni}) and special (MDP M_2^{spe}) transition probabilities. Obviously, the performance of OneBin is very poor in both cases: its value is greater than 2.8, while that of GreedyFit is below 2.4. In particular, the performance of OneBin is far away from that of an optimal: we have a relative cost increase of about 20 % for the MDP M_2^{uni} and more than 32 % for M_2^{spe}, while the associated values for GreedyFit are very small. This proves that OneBin reaches the maximum colorfulness of 3 significantly faster in expectation than GreedyFit. A second observation is that GreedyFit comes very close to the optimal value $v_{i_0}^{\alpha}$ for uniform transition probabilities, but is slighty inferior compared to an optimal policy for the special distribution for the transition probabilities where we have $\varepsilon_{i_1}^{\alpha}(\mathsf{GreedyFit}) = 2.8\,\%$. Note further that GreedyFit and an optimal policy perform better in the case of non-uniform transition probabilities. This behavior is what one would expect.

The disadvantage of GreedyFit compared to an optimal policy for the MDP M_2^{spe} shows that there must exist states, where the actions of GreedyFit are not optimal. By analyzing GreedyFit using our approximation tool, we were able to detect such states. One of them is the state $i_2 \in \mathbb{S}$, which is

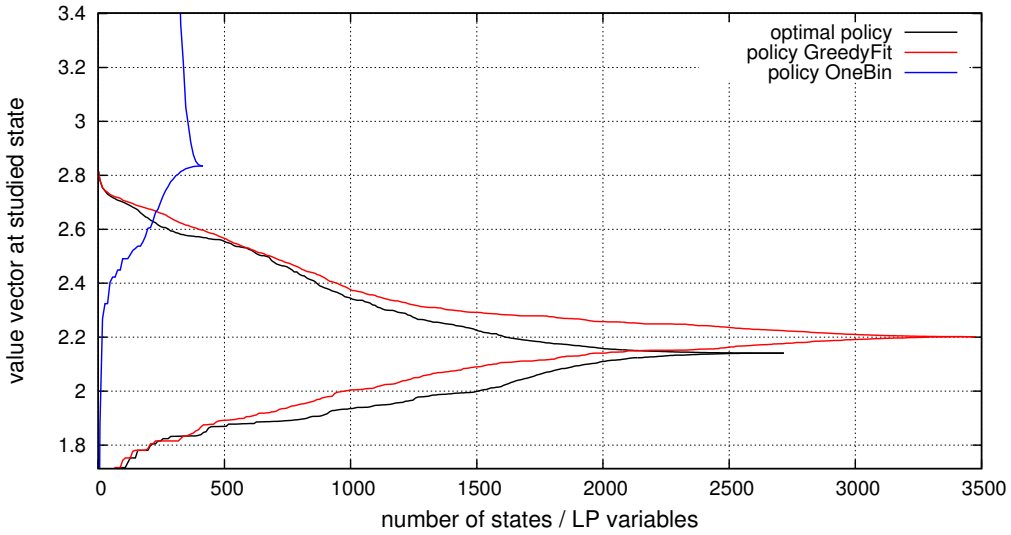


Figure 4.7: Approximation results for the bin coloring MDP M_2^{spe} and the trivial initial state i_1 defined in Figure 4.5. We have $\varepsilon_{i_1}^{\alpha}(\mathsf{GreedyFit}) = 2.8\,\%$ and $\varepsilon_{i_1}^{\alpha}(\mathsf{OneBin}) = 32.4\,\%$.

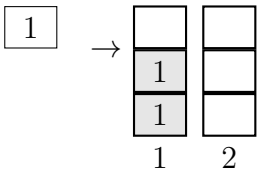


Figure 4.8: State i_2 in a bin coloring MDP with $m = 2$ bins of size $b = 3$. The state is of the form $i_2 = (c, \chi, f_1, C_1, f_2, C_2)$, where $c = 1$, $\chi = 2$, $f_1 = 2$, $C_1 = \{1\}$, $f_2 = 0$, and $C_2 = \emptyset$.

depicted in Figure 4.8.

We will refer to a bin k as being *safe* if the number of different colors present in that bin plus its remaining capacity does not exceed the maximum colorfulness, i. e., $|C_k| + b - f_k \leq \chi$. Since the number of different colors in a safe bin cannot exceed χ independently of the colors of further items packed into the bin, the maximum colorfulness will never be increased due to that bin. Obviously, bin 1 in state i_2 is safe. It seems to be a good idea to use safe bins only in disadvantageous situations. For instance, consider state i_2. If the probability of items with color 1 is sufficiently high, the bin configuration (f_1, C_1, f_2, C_2) with $f_1 = 2$, $f_2 = 1$, and $C_1 = C_2 = \{1\}$ (obtained by using bin 2) seems to be better than $f_1 = f_2 = 0$ and $C_1 = C_2 = \emptyset$ (reached by GreedyFit) since the first configuration increases the chance to make bin 2 safe. Note further that both situations guarantee that four items can be

packed without increasing the maximum colorfulness.

Based on the idea to avoid to use safe bins as long as possible while trying to make other bins safe, we developed a new bin coloring policy called SafeBin. Given a state $(c, \chi, f_1, C_1, \ldots, f_m, C_m)$, we refer to a bin $k \in \{1, \ldots, m\}$ as being *critical* w. r. t. color c if $|C_k| = \chi$ and $c \notin C_k$. Bins that are not critical are also called *non-critical*. That is, a bin is critical if and only if putting the next item into that bin will increase the maximum colorfulness. Moreover, we call a bin k *color-suited* w. r. t. color c if $c \in C_k$, i. e., color c is present in that bin. Note that a color-suited bin is non-critical. Finally, a bin is *emptiest (fulltest)* within a certain class of bins if it contains the smallest (greatest) number of items. The policy SafeBin works as follows:

1. If there exists a non-critical bin:
   (a) If there exists a color-suited bin that is not safe, put the item into such a bin.
   (b) Otherwise (all color-suited bins are safe), if there exists any non-critical bin that is not safe, choose such a bin containing the smallest number of different colors.
   (c) Otherwise (all non-critical bins are safe), pack the item into a fullest non-critical bin.
2. If all bins are critical, put the item in an emptiest bin.

The idea for choosing a bin with the smallest number of colors in Case 1b is to distribute colors uniformly among the bins that are not safe at the moment. This way, all of those bins have a similar chance to be color-suited for future colors by which safe bins may be created. We prefer to use a fullest non-critical bin if all non-critical bins are safe (Case 1c) since it seems to be desirable if one of the safe bins is soon relaced by a new empty bin, keeping the room in the other safe bins. Then, it is more likely that appropriate future items can be combined to make the new bin safe also, while one can get rid of unfavorable colors using the remaining safe bins. We mention that if the bin capacity is $b = 3$ as considered here, a bin $k \in \{1, \ldots, m\}$ can only be safe if we have $\chi = 2$, $f_k = 2$, and $|C_k| = 1$. Thus, all safe bins have the same number of items in Case 1c. In Case 2, when all bins are critical, SafeBin puts the item in the emptiest bin since this may imply that other bins containing more items will become safe.

Next we analyze the policy SafeBin and compare it to GreedyFit and an optimal policy. Figure 4.9 and 4.10 show the associated approximations for the MDPs M_2^{uni} and M_2^{spe} when the initial state is again the trivial one i_1. Except for SafeBin, these results have already been shown above.

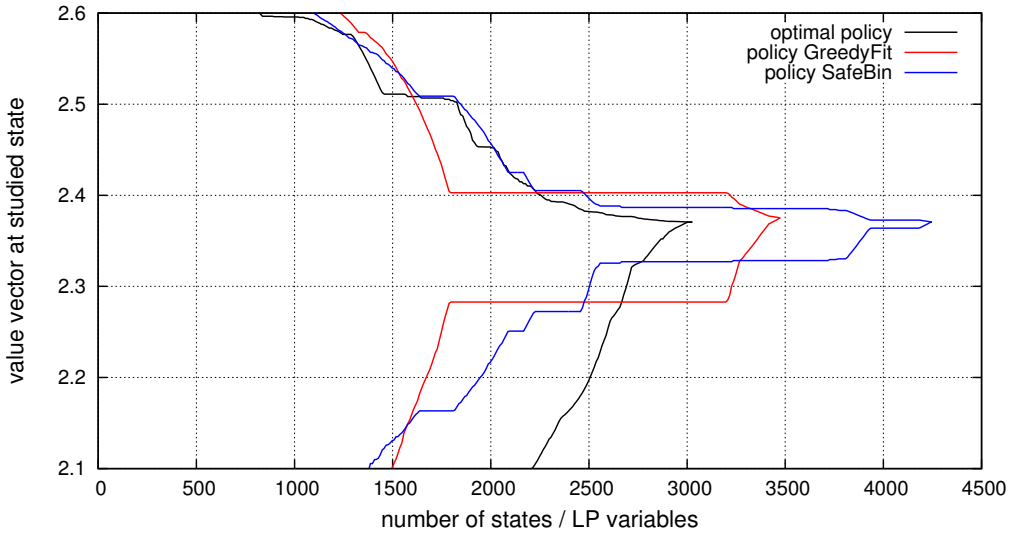


Figure 4.9: Approximation results for the bin coloring MDP M_2^{uni} and the trivial initial state i_1 defined in Figure 4.5. We have $\varepsilon_{i_1}^{\alpha}(\mathsf{GreedyFit}) = 0.2\,\%$ and $\varepsilon_{i_1}^{\alpha}(\mathsf{SafeBin}) = 0\,\%$.

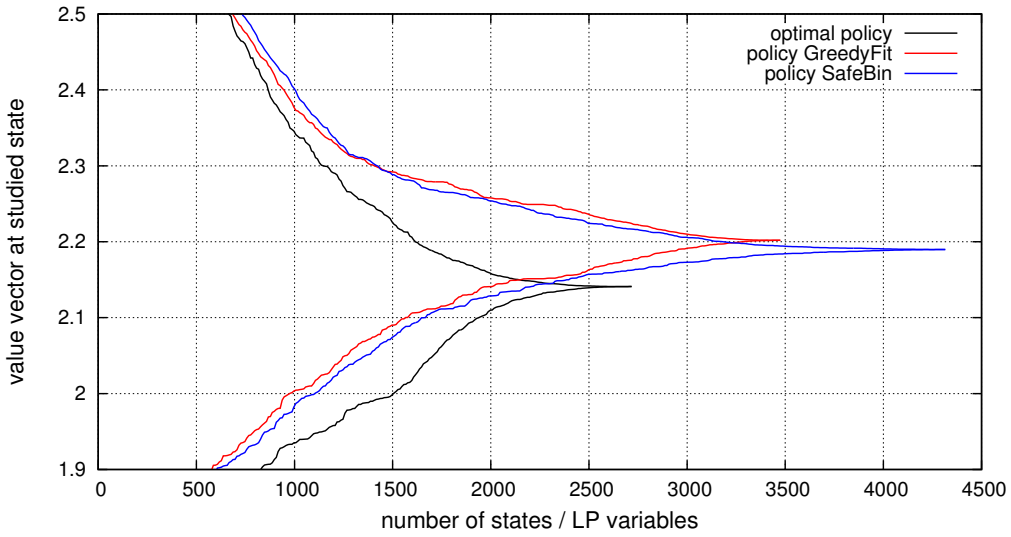


Figure 4.10: Approximation results for the bin coloring MDP M_2^{spe} and the trivial initial state i_1 defined in Figure 4.5. We have $\varepsilon_{i_1}^{\alpha}(\mathsf{GreedyFit}) = 2.8\,\%$ and $\varepsilon_{i_1}^{\alpha}(\mathsf{SafeBin}) = 2.3\,\%$.

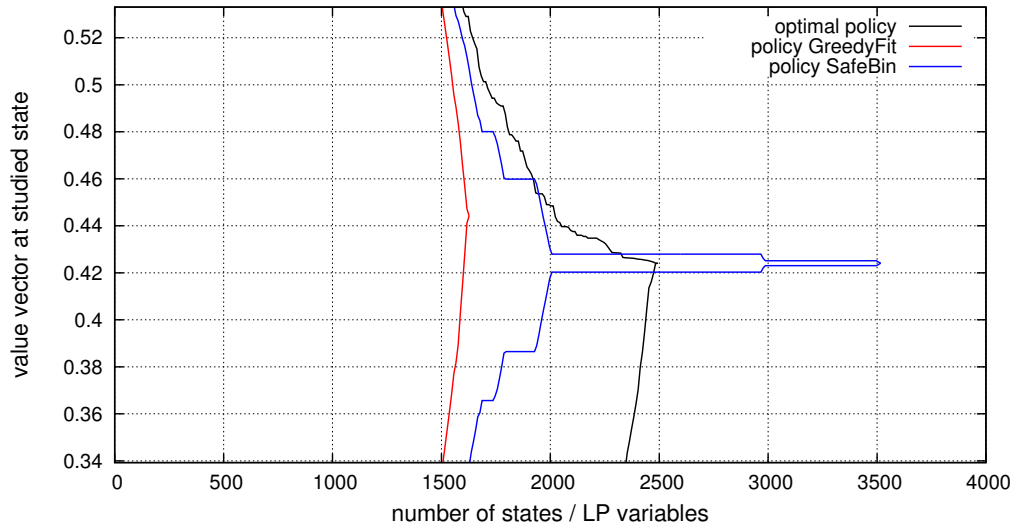


Figure 4.11: Approximation results for the bin coloring MDP M_2^{uni} and the initial state i_2 defined in Figure 4.8. We have $\varepsilon_{i_2}^{\alpha}(\mathsf{GreedyFit}) = 4.8\,\%$ and $\varepsilon_{i_2}^{\alpha}(\mathsf{SafeBin}) = 0\,\%$.

In the case of uniform transition probabilities, the total expected α-discounted costs of the two considered policies are very close to the optimal one. GreedyFit performs a little worse, while SafeBin is optimal for the considered start state. The approximation results for the non-uniform transition probabilities (Figure 4.10) look more interesting: here the two policies GreedyFit and SafeBin are slightly inferior compared to an optimal policy. The difference between the two studied policy is small, but SafeBin performs better than GreedyFit. It is quite clear that starting from the trivial state i_1 makes it unlikely to soon reach a somewhat difficult situation, where a reasonable policy takes a bad decision. For this reason the cost values of the considered policies only differ a little in the results above, apart from OneBin.

The approximations using the particular state i_2 as starting point are depicted in Figure 4.11 for uniform transition probabilities and in Figure 4.12 for the considered non-uniform distribution.

It is clear that the cost values of the considered policies are quite small for the initial state i_2 since the maximum colorfulness is already $\chi = 2$ and with high probability it will take a certain number of transitions until the final increase will occur. For uniform transition probabilities GreedyFit performs worse than SafeBin due to the reasons mentioned above that motivated developing SafeBin. The policy SafeBin itself is optimal for the initial state i_2. This changes in the case of the special distribution for the transition probabilities: the performance of the policies GreedyFit and SafeBin is substantially inferior

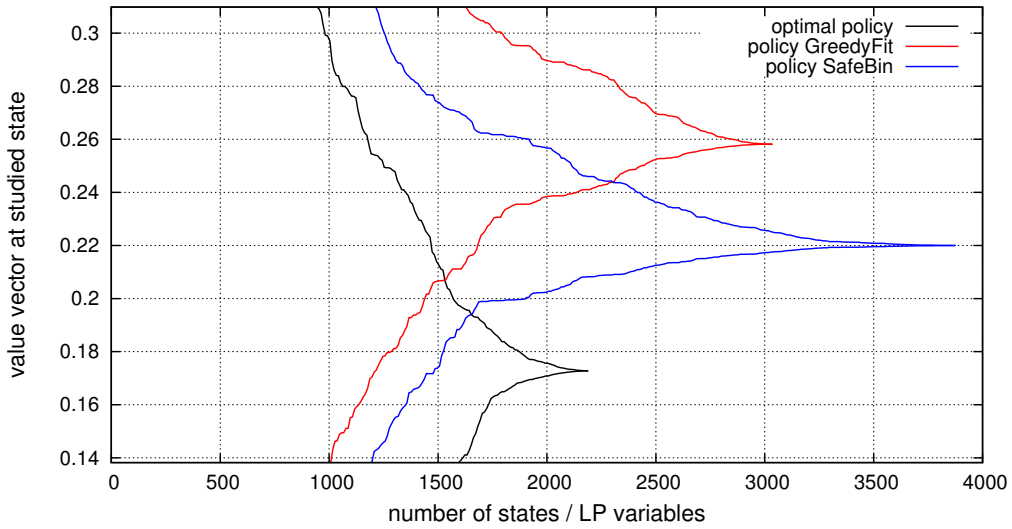


Figure 4.12: Approximation results for the bin coloring MDP M_2^{spe} and the initial state i_2 defined in Figure 4.8. We have $\varepsilon_{i_2}^{\alpha}(\mathsf{GreedyFit}) = 49.4\,\%$ and $\varepsilon_{i_2}^{\alpha}(\mathsf{SafeBin}) = 27.3\,\%$.

to that of an optimal policy. As before SafeBin outperforms GreedyFit, but now even to a larger extent. We believe that the high probability of 0.3 for an item with color 1 is the reason that the two policies differ more significantly as for the MDP M_2^{uni}: putting the item in bin 2 as done by SafeBin makes it more likely that the bin may become safe soon since the probability of the color is high.

Instances `bc-3-3-7-*`

Next we look at approximation results for the larger bin coloring MDPs $M_3^{\text{uni}} := (\texttt{bc-3-3-7-uni}, 0.97)$ and $M_3^{\text{spe}} := (\texttt{bc-3-3-7-spe}, 0.97)$, where $m = 3$ bins are available simulatenously. Notice that the number of different colors equals 7, which is still enough that each policy will eventually reach a maximum colorfulness of $b = 3$ with probability 1. The total number of states in both MDPs equals 122 871. Considering a trivial state as starting point for the MDPs M_3^{uni} and M_3^{spe} does not give further insight in the problem: still OneBin performs badly, while the differences between GreedyFit, SafeBin, and an optimal policy are even smaller than before. Instead, we will look at the two initial states i_3 and i_4 shown in Figure 4.13. Again, our approximation algorithm is able to compute the reported cost values exactly with reasonable effort, even for the large discount factor.

First we look for both initial states i_3 and i_4 at the approximation results

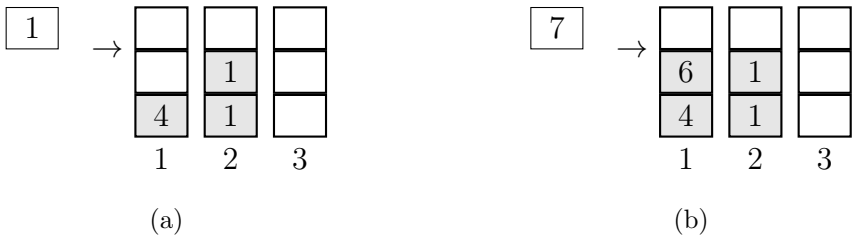


Figure 4.13: (a) State i_3 in a bin coloring MDP with $m = 3$ bins of size $b = 3$. The state is of the form $i_3 = (c, \chi, f_1, C_1, f_2, C_2, f_3, C_3)$, where $c = 1$, $\chi = 2$, $f_1 = 1$, $C_1 = \{4\}$, $f_2 = 2$, $C_2 = \{1\}$, $f_3 = 0$, and $C_3 = \emptyset$. (b) State i_4 in the same MDP. The state is of the form $i_4 = (c, \chi, f_1, C_1, f_2, C_2, f_3, C_3)$, where $c = 7$, $\chi = 2$, $f_1 = 2$, $C_1 = \{4, 6\}$, $f_2 = 2$, $C_2 = \{1\}$, $f_3 = 0$, and $C_3 = \emptyset$.

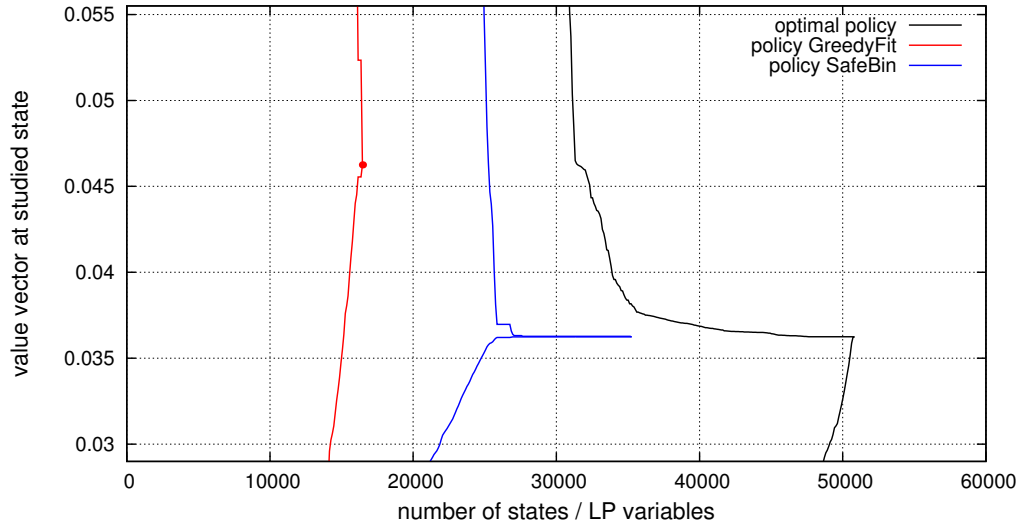


Figure 4.14: Approximation results for the bin coloring MDP M_3^{uni} and the initial state i_3 defined in Figure 4.13(a). We have $\varepsilon_{i_3}^{\alpha}(\mathsf{GreedyFit}) = 27.6\,\%$ and $\varepsilon_{i_3}^{\alpha}(\mathsf{SafeBin}) = 0\,\%$.

in the case of uniform transition probabilities, i. e., MDP M_3^{uni}, shown in the Figures 4.14 and 4.15. For both initial states the observations are similar: SafeBin is at least close to being optimal, while GreedyFit has a worse cost value.

However, there are different reasons for the poor performance of GreedyFit. In the case of the initial state i_3, the policy chooses bin 2 since color 1 is already present in that bin. Note that bin 2 is safe. Thus, the action of GreedyFit is bad for the same reason as mentioned before. For state i_4, we have $\mathsf{GreedyFit}(i_4) = 3$ since color 7 is not present in any bin and bin 3

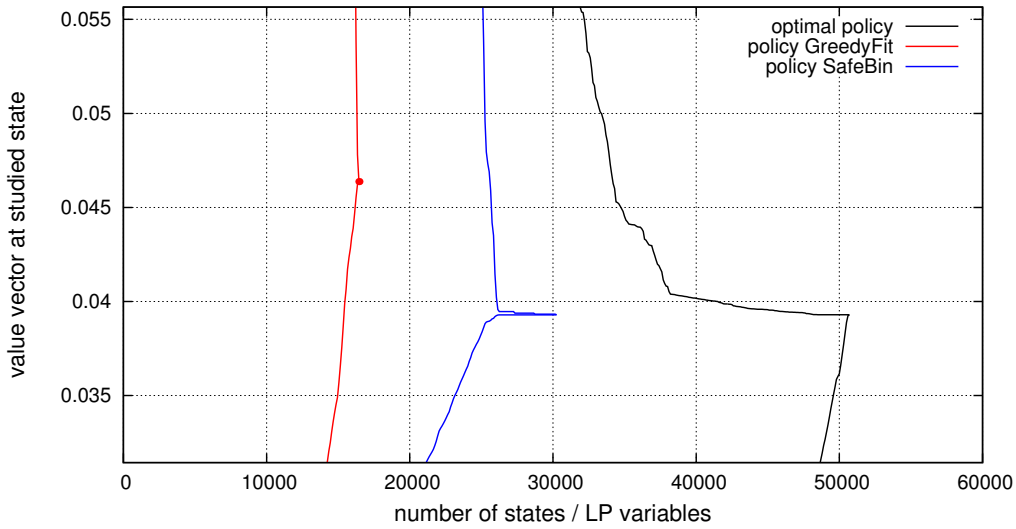


Figure 4.15: Approximation results for the bin coloring MDP M_3^{uni} and the initial state i_4 defined in Figure 4.13(b). We have $\varepsilon_{i_4}^{\alpha}(\mathsf{GreedyFit}) = 18.0\,\%$ and $\varepsilon_{i_4}^{\alpha}(\mathsf{SafeBin}) = 0\,\%$.

contains the least number of distinct colors. This seems to be a good action, which may help to make the bin safe. Indeed using the described approach based on Corollary 3.4.5 on page 68 we proved this action to be optimal. The reason for the bad performance of GreedyFit is due to decisions in later stages.

The policy SafeBin chooses bin 3 for both states i_3 and i_4. Note that SafeBin does not perform worse than an optimal policy in the case of uniform transition probabilities for all considered MDPs and initial states. In fact, we exhaustively tried to computationally find an initial state such that the resulting cost value of SafeBin is actually larger than the optimal one w. r. t. the considered MDPs M_2^{uni} and M_3^{uni}. Since no such state could be found, we conclude that SafeBin is almost optimal in the case of uniform transition probabilities independently of the considered start state.

Finally, let us consider the obtained approximations for the MDP M_3^{spe} with the non-uniform distribution for the transition probabilities. The results for the initial states i_3 and i_4 are depicted in Figures 4.16 and 4.17.

In both situations the cost values of the policies GreedyFit and SafeBin are far from being optimal, but still SafeBin outperforms GreedyFit. Note that the total expected α-discounted cost is very small in all cases. We could prove that the only optimal action at state i_3 is to choose bin $3 = \mathsf{SafeBin}(i_3)$, while GreedyFit puts the item into bin $2 = \mathsf{GreedyFit}(i_3)$. However, the bad performance of SafeBin shows that there must exist states where the action of

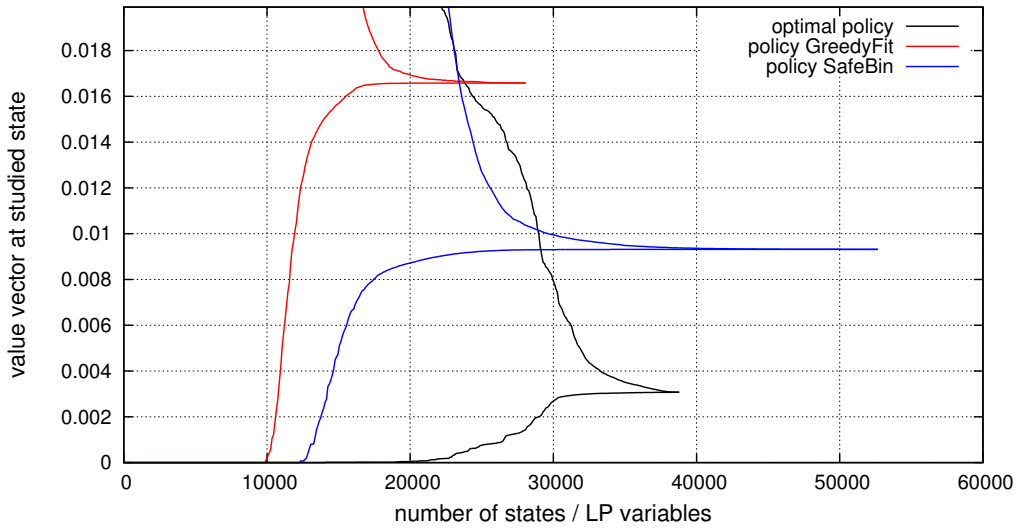


Figure 4.16: Approximation results for the bin coloring MDP M_3^{spe} and the initial state i_3 defined in Figure 4.13(a). We have $\varepsilon_{i_3}^{\alpha}(\mathsf{GreedyFit}) = 439.7\,\%$ and $\varepsilon_{i_3}^{\alpha}(\mathsf{SafeBin}) = 203.2\,\%$.

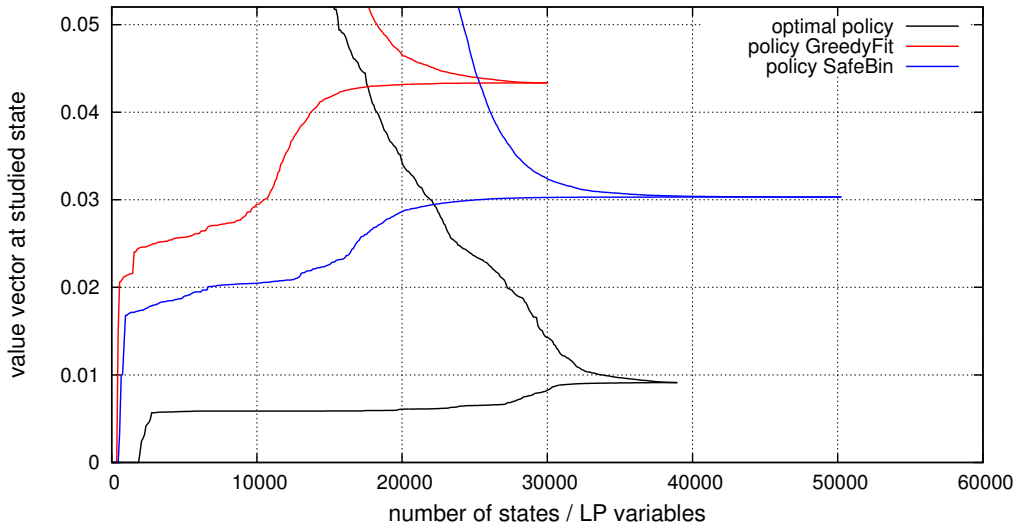


Figure 4.17: Approximation results for the bin coloring MDP M_3^{spe} and the initial state i_4 defined in Figure 4.13(b). We have $\varepsilon_{i_4}^{\alpha}(\mathsf{GreedyFit}) = 374.9\,\%$ and $\varepsilon_{i_4}^{\alpha}(\mathsf{SafeBin}) = 232.1\,\%$.

SafeBin is not optimal. Such a state is i_4. For this initial state, both policies choose bin 3 for the item with color 7. This decision is disadvantageous as the probability for another item with the same color equals 0.03, which is very small. Therefore, a bin containing only one item with color 7 is very hard to be made safe in the future. In the considered situation it is better to pack the item into the safe bin 2 to get rid of the annoying color 7.

Summary

In this paragraph, we summarize the important results of this section and make some further remarks. Concerning the comparision of GreedyFit and the stupid policy OneBin, the computational results obtained by our approximation algorithm give a more realistic picture than competitive analysis and reflect the behavior observed in simulations.

We developed a new online algorithm called SafeBin for the bin coloring problem that mostly outperforms GreedyFit and seems to be almost optimal in the case of uniform transition probabilities independently of the used start state. We mention that GreedyFit may be superior to SafeBin in the case of other distributions for the transition probabilities. For instance, consider the modified state i_4 (see Figure 4.13(b)), where the color of the items in bin 2 is color 7 instead of 1. For the same argument as above, it is not a good idea to put the new item having color 7 into bin 3, which SafeBin does. Similar as for state i_4, using bin 2 is an optimal action here. Due to their choises for the action to be applied, the cost of GreedyFit is smaller than that of SafeBin for the considered initial state. However, those situations seem quite unlikely to occur. This example shows that stochastic information have to be taken into account in order to come up with a policy that improves over SafeBin. Our first attempts in this direction were not successful.

In order to figure out whether our analyses for GreedyFit and SafeBin provide realistic performance indicators, we assessed the behavior of these two online algorithms by simulation also. In doing so, we simulated the exact setting given by our Markov decision process model. In the considered instances each online algorithm will eventually reach a maximum colorfulness that is equal to the bin capacity b. We evaluated for each number $k \in \{2, \ldots, b\}$, how many requests are required such that the maximum colorfulness becomes k. The simulation results reporting average values for 100 runs are shown in Table 4.15. Since both online algorithms work identically as long as the maximum colorfulness χ does not exceed 1, GreedyFit and SafeBin reach $\chi = 2$ at the same time for each request sequence. Thus, the associated average values are identical as well. For $\chi \in \{3, 4\}$, however, SafeBin

Markov decision process	maximum colorfulnes 2 GF	2 SB	3 GF	3 SB	4 GF	4 SB
bc-2-3-6-spe	4.10	4.10	69.47	91.52	-	-
bc-2-3-6-uni	3.98	3.98	35.54	39.95	-	-
bc-3-3-7-spe	6.64	6.64	1635.31	2259.52	-	-
bc-3-3-7-uni	5.91	5.91	693.34	771.55	-	-
bc-3-4-12-spe	5.36	5.36	24.30	25.93	5257.12	23527.80
bc-3-4-12-uni	4.79	4.79	12.65	12.81	1349.44	2430.52

Table 4.15: Average number of requests required by the online algorithms GreedyFit (GF) and SafeBin (SB) to reach a maximum colorfulness of 2, 3, and 4 according to 100 simulation runs.

significantly outperforms GreedyFit. We conclude that our approximation results for these online algorithms are in line with their behavior observed in simulation.

We make another remark concerning GreedyFit. In the case the color of an item to be packed is not present in one of the m bins, GreedyFit puts it into a bin having the least number of distinct colors. Often there may exist several bins with this property. Additional approximation results revealed that the used tie-breaking rule can slightly affect the policy's performance: choosing an emptiest bin in the described situation seems to be better than the opposite. Our approximation method also showed that the use of different tie-breaking rules for GreedyFit, depending on whether non-critical bins exist or not, barely changes the performance.

4.3.3 Target Date Assignment MDPs

An extensive analysis comparing the policies PTD and PFD for the target date assignment MDPs (tda-3-2, α) with discount factor $\alpha \in \{0.5, 0.7\}$ has been carried out in the diploma thesis of Heinz [Hei05, chapter 5.4] that is based on joint work. The results show that PFD selects an optimal action for most of the considered initial states and seems close to an optimal policy, while PTD uses worse actions than PFD in many situations. We mention that the MDPs have 230 076 states in total.

In this section we will analyze our Markov decision process formulation of the online target date assignment problem w. r. t. a modified objective cri-

terion. Instead of comparing the two policies in terms of the total expected α-discounted cost for some $\alpha \in (0, 1]$, we will discount stage costs depending on the date, not on the stage. Given a Markov decision process for the considered target date assignment problem with state space $\mathbb{S}$ and a state $i \in \mathbb{S}$, we denote the current date of state i by $t = 0$ and the consecutive dates by $t = 1, 2, \ldots$. Let $c_t \in \mathbb{N}_0$ be the sum of all stage costs incurred at date t, then the associated *total date-wise α-discounted cost* is defined by:

$$\sum_{t=0}^{\infty} \alpha^t c_t.$$

That is, each stage cost incurred at date t is being accounted by a factor of α^t. Obviously, using such an objective criterion is more natural for the target date assignment problem due to the specific time model of the problem.

In order to formally introduce the *total expected date-wise α-discounted cost* $v_i'^{\alpha}(\pi)$ of a policy π for initial state i, assume that the Markov decision process additionally encodes for each state $j \in \mathbb{S}$, the date $t(j) \in \mathbb{N}_0$ at which the current request is released. For the considered initial state i_0, we have $t(i_0) = 0$. Using the same notation as in Definition 2.1.9 on page 14, the cost value $v_i'^{\alpha}(\pi)$ is defined as:

$$\begin{aligned} v_i'^{\alpha}(\pi) &:= \sum_{k=0}^{\infty} \mathbb{E}_{i\pi}[\alpha^{t(X_k)} \cdot c_{X_k}(Y_k)] \\ &= \sum_{k=0}^{\infty} \sum_{j \in \mathbb{S}} \sum_{a \in \mathbb{A}(j)} \mathbb{P}_{i\pi}[X_k = j, Y_k = a] \cdot \alpha^{t(j)} c_j(a). \end{aligned}$$

The following result shows that the objective criterion of minimizing the total expected date-wise α-discounted cost is equivalent to minimize the total expected α-discounted cost for a modified Markov decision process.

Theorem 4.3.1 *Let $M = (\mathbb{S}, \mathbb{A}, p, c)$ be a Markov decision process for the considered target date assignment problem, where at least $n_{\min} \geq 1$ and at most $n_{\max} \geq n_{\min}$ requests are released each date, and let $\alpha \in [0, 1)$. Then, the total expected date-wise α-discounted cost of a policy π for M for an initial state $i_0 \in \mathbb{S}$ equals the total expected α-discounted cost of an associated policy π' for the Markov decision process $M' = (\mathbb{S}', \mathbb{A}', p', c')$ with the following components w. r. t. initial state $(i_0, 0, 1) \in \mathbb{S}'$.*

- *The state space $\mathbb{S}'$ is given by:*

  $$\mathbb{S}' = \{(s, n, S_1, \ldots, S_\delta, t, n_\Sigma) \in \mathbb{S} \times \mathbb{N}_0^2 \mid n + n_{\min} t \leq n_\Sigma \leq n + n_{\max} t\},$$

  where t is a date index and n_Σ is the total number of requests since date $t = 0$.

- *The action space $\mathbb{A}'$ is induced by $\mathbb{A}$: for a state $i' = (i, t, n_\Sigma) \in \mathbb{S}'$ with $i \in \mathbb{S}$, the set of possible actions is given by $\mathbb{A}'(i') = \mathbb{A}(i)$.*

- *Consider states $i' = (i, t(i'), n_\Sigma(i')) \in \mathbb{S}'$ and $j' = (j, t(j'), n_\Sigma(j')) \in \mathbb{S}'$, and for all $i, j \in \mathbb{S}$, let $n(i)$ and $n(j)$ denote the number of requests released at the current date. Then, the transition probability $p'_{i'j'}(a)$ for some action $a \in \mathbb{A}'(i')$ equals:*

$$p'_{i'j'}(a) = \begin{cases} p_{ij}(a), & \text{if } n_\Sigma(j') = n_\Sigma(i') + 1 \text{ and} \\ & [\ (t(j') = t(i') \text{ and } n(j) = n(i) + 1) \text{ or} \\ & \quad (t(j') = t(i') + 1 \text{ and } n(j) = 1)\ ], \\ 0, & \text{otherwise.} \end{cases}$$

- *For each state $i' = (i, t(i'), n_\Sigma(i')) \in \mathbb{S}'$ and an action $a \in \mathbb{A}'(i')$, the stage cost $c'_{i'}(a, j')$ for each possible successor $j' = (j, t(j'), n_\Sigma(j')) \in \mathbb{S}'$ is defined as:*

$$c'_{i'}(a, j') = \frac{c_i(a, j)}{\alpha^{n_\Sigma(i') - t(i') - 1}}.$$

Proof. Let π be any policy in the original Markov decision process M, and let $i_0 \in \mathbb{S}$ be any state in M. Note that each state $i' = (i, t, n_\Sigma)$ in M', which is reachable from $i'_0 = (i_0, 0, 1)$ by the policy π', is reached via exactly $n_\Sigma - 1$ transitions. That is, the state parameter n_Σ may be seen as a stage index, where the initial index equals 1. Consequently, each stage cost $c'_{i'}(a, j')$ for some $j' \in \mathbb{S}'$ and $a \in \mathbb{A}'(i')$ at state i' is being discounted by the factor $\alpha^{n_\Sigma - 1}$. Thus, we have an accounted stage cost of:

$$\alpha^{n_\Sigma - 1} c'_{i'}(a, j') = \alpha^t c_i(a, j),$$

which equals the corresponding date-wise α-discounted cost for date t. □

By Theorem 4.3.1 we can apply our column generation algorithm to approximate also the total expected date-wise α-discounted cost of a concrete or unknown optimal policy in the considered Markov decision process given an initial state (note that $\sum_{k=0}^{\infty} n_{\max} \alpha^k = n_{\max}/(1-\alpha)$ is a general upper bound on $v_i'^{\alpha}(\pi)$ for any policy π and state $i \in \mathbb{S}$). Moreover, we mention that the approximation algorithm can easily be modified to handle this objective criterion based on the original Markov decision process, i. e., the blow up of the state space according to the theorem is not required.

In the following, we consider the Markov decision process `tda-3-2` and a discount factor of $\alpha = 0.7$. We analyze the policies PTD and PFD w. r. t. the total expected date-wise α-discounted cost. The initial states we consider

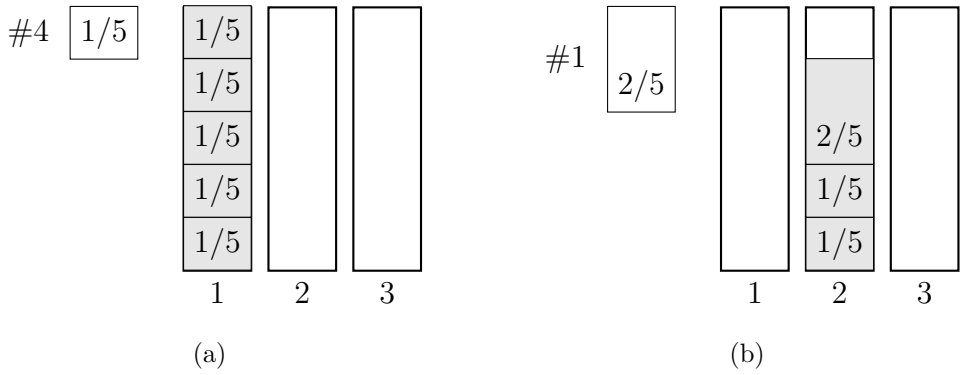


Figure 4.18: (a) State i_1 in a Markov decision process for target date assignment with deferral time $\delta = 3$. The state is of the form $i_1 = (s, n, S_1, S_2, S_3)$, where $s = 1/5$, $n = 4$, $S_1(1/5) = 5$, $S_1(x) = 0$ for each $x \in (0, 1] \setminus \{1/5\}$, $S_2 \equiv 0$, and $S_3 \equiv 0$. (b) State i_2 in the same. The state is of the form $i_2 = (s, n, S_1, S_2, S_3)$, where $s = 2/5$, $n = 1$, $S_2(1/5) = 2$, $S_2(2/5) = 1$, $S_2(x) = 0$ for each $x \in (0, 1] \setminus \{1/5, 2/5\}$, $S_1 \equiv 0$, and $S_3 \equiv 0$.

here have been chosen to unfold the disadvantages of the two considered policies, see Figure 4.18. It is easy to see that each of the policies PTD and PFD can reach these two states. We required a relative approximation guarantee of $\varepsilon = 0.5\,\%$ for these computations.

Figure 4.19 shows the approximation results for the initial state i_1. Note that already four request have been released at the current date. This implies that the probability for a date change is quite high in this situation: we have $p_4^d = 0.7$. Thus, assigning the request to date $1 = \mathsf{PTD}(i_1)$ is not an optimal action since it is unlikely that the new bin required at date 1 could be filled reasonably before the date changes. That is, using this action will badly exploit the capacity of the newly required bin with high probability. Notice further that the action cost $v_{i_1}'^{\alpha}(\mathsf{PTD}(i_1))$ is smaller than the policy cost $v_{i_1}'^{\alpha}(\mathsf{PTD})$, which implies that PTD selects non-optimal actions for other states, too.

On the other hand, choosing date $3 = \mathsf{PFD}(i_1)$ is optimal. This way, one maximizes the chance for combining the current item of size $1/5$ with further items, which allows for exploiting the bin capacity as much as possible. Note that the results from Figure 4.19 cannot prove a difference between the cost of the policy PTD and the optimal cost.

Nevertheless, PFD is neither optimal as our results show for the second initial state i_2, see Figure 4.20. Here the cost of the policy PFD and its action $\mathsf{PFD}(i_2) = 3$ are slightly worse than the optimal cost. Interestingly, the action of PTD, which assigns date 2, is optimal now. The reason why it is better to choose date 2 may be as follows. With high probability some

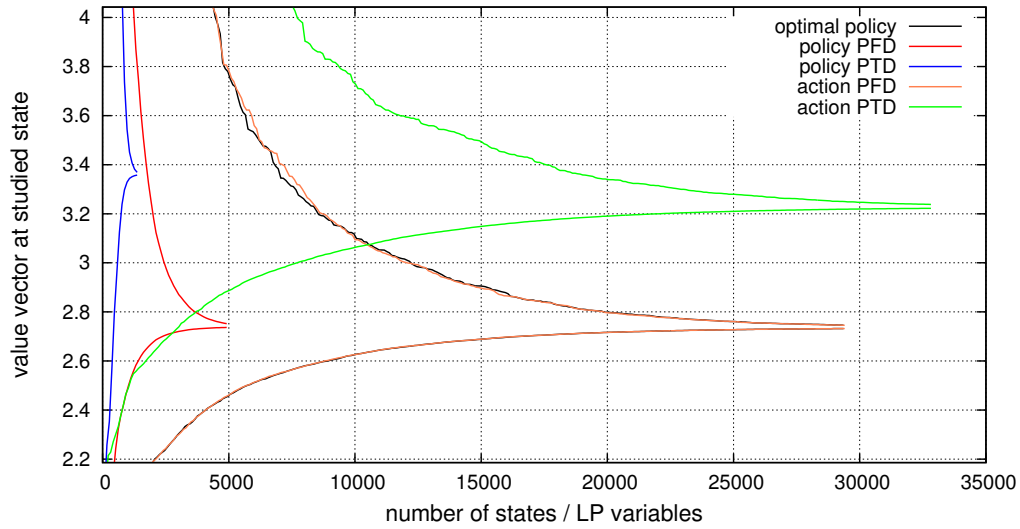


Figure 4.19: Approximation results w. r. t. the total expected date-wise α-discounted cost with $\alpha = 0.7$ for the Markov decision process `tda-3-2` and the initial state i_1 defined in Figure 4.18(a). We have $\varepsilon^{\alpha}_{i_1}(\mathsf{PTD}) \geq 22.3\,\%$, $\varepsilon^{\alpha}_{i_1}(\mathsf{PTD}(i_1)) \geq 17.4\,\%$, $\varepsilon^{\alpha}_{i_1}(\mathsf{PFD}) \geq 0\,\%$, $\varepsilon^{\alpha}_{i_1}(\mathsf{PFD}(i_1)) \geq 0\,\%$.

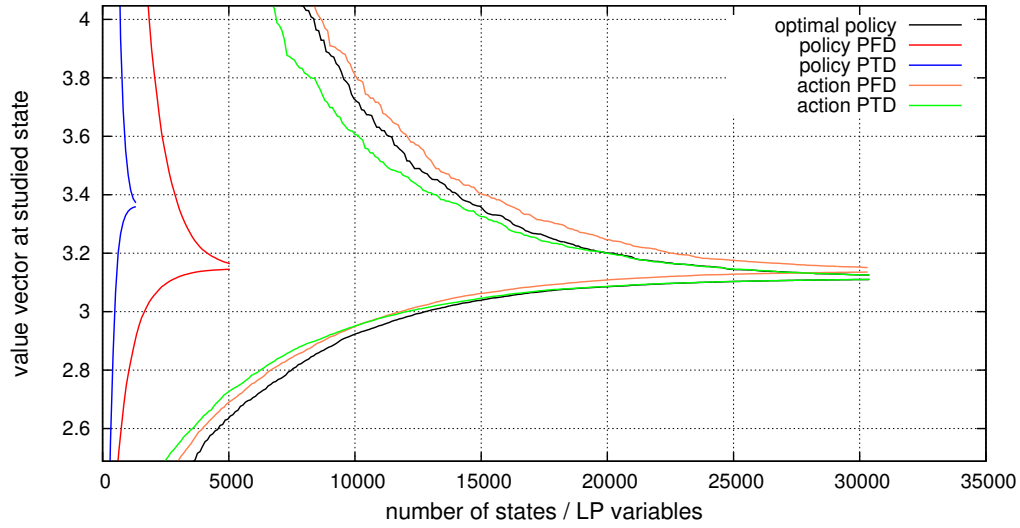


Figure 4.20: Approximation results w. r. t. the total expected date-wise α-discounted cost with $\alpha = 0.7$ for the Markov decision process `tda-3-2` and the initial state i_2 defined in Figure 4.18(b). We have $\varepsilon^{\alpha}_{i_2}(\mathsf{PTD}) \geq 7.5\,\%$, $\varepsilon^{\alpha}_{i_2}(\mathsf{PTD}(i_2)) \geq 0\,\%$, $\varepsilon^{\alpha}_{i_2}(\mathsf{PFD}) \geq 0.6\,\%$, $\varepsilon^{\alpha}_{i_2}(\mathsf{PFD}(i_2)) \geq 0.3\,\%$.

Markov decision process	Number of required bins	
	PFD	PTD
tda-3-2	3060.07	3504.91
tda-4-2	3019.87	3377.23

Table 4.16: Average number of bins required by the online algorithms PFD and PTD according to 100 simulation runs each of which consists of 10 000 requests.

more requests will be released before the current date will change twice: the probability for at least three and four additional requests in that time span is greater or equal than 0.86 and 0.69, respectively. That is, assigning date 2 offers a good chance that the remaining capacity of 4/5 given by the used bins at that date can be exploited completely. Moreover, using two bins at one date offers more flexibility for packing. The advantage of using date 2 is that one could add two items of size 2/5 without having to open another bin. This is not the case if date 3 was chosen. We mention that for the usually considered total expected α-discounted cost, the difference between the action cost $v_{i_2}^{\alpha}(\mathsf{PFD}(i_2))$ and the optimal cost $v_{i_2}^{\alpha}$ is even larger.

Though there exist states like i_2 where PFD selects an action that is not optimal, the policy seems to be quite close to an optimal policy w. r. t. the total expected date-wise discounted cost, too. Finally, we look at simulation results for the policies PFD and PTD assuming the exact setting of the Markov decision process model again. We investigate the total number of bins required by the policies for 10 000 randomly generated requests. The average values for 100 simulation runs are given in Table 4.16. As expected and supposed by our approximation results, PFD outperforms PTD. We mention that the advantage of PFD degrades for increasing deferral time δ.

4.3.4 Elevator Control MDPs

In this section, we analyze policies for the elevator control MDP w. r. t. the average and maximum waiting time. In doing so, it is crucial to incorporate the involved lower and upper bounds described in Section 4.1.5 in the linear programs. Otherwise, the approximations for the optimal cost and the cost of a given policy or single action at a given state i_0 will be very weak, making a comparison of different policies almost impossible. We will sketch this observation very briefly below. Moreover, we should mention that computing approximations for the considered elevator control MDPs by our algorithm is quite expensive. The run-times for generating 100 000 variables are up to

3 hours in the case of a single elevator. For the studied instance featuring two elevators it takes even up to 12 hours for approximating a component of the optimal value vector.

We start by giving an overview of the online algorithms / policies considered for scheduling a single elevator (some of them have already been described in Section 4.1.5). Except for implementing FIFO and Ignore, a state in the MDP is not required to encode a tentative schedule for serving the waiting requests. Thus for each state, the associated action according to a given policy has to be computed separately, although it follows directly from the schedule determined for the predecessor state unless a new request was released. Therefore, we can describe each policy by specifying either its next action, the request to be served next unless another request appears, or the complete schedule. Recall that no decision has to be made, whenever the elevator is loaded since the elevator simply serves the loaded request. The same is true if no request is waiting. In this case, all policies let the elevator wait at the current floor. For the description of the online algorithms, we thus assume that the elevator is empty and there exists at least one waiting request. 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.

ReplanS The same as Replan, but a schedule minimizing the sum of the final waiting times of all requests is used.

ReplanSQ The same as Replan, but a schedule minimizing the sum of the squared final waiting times of all requests is used.

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

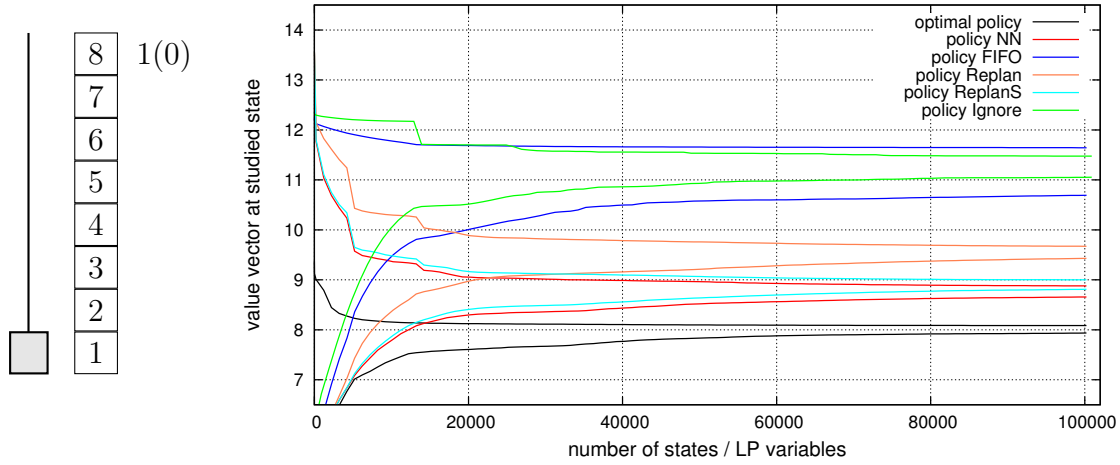
store the schedule. The policy Ignore requires a modified MDP where each state encodes a schedule containing a (possibly empty) subset of the waiting requests. Moreover, if for some state this schedule is empty and a request is waiting, each associated action has a second component that sets the schedule for all waiting requests.

In the case of two elevators, we only consider FIFO and NN extended by some strategy for assigning requests to elevators. The used assignment strategies are described later. To prevent the approximation pictures to be presented in the sequel from being overloaded, we will only display results for selected policies. However, we provide the bounds ε^α for the relative cost increase for all policies.

Average Waiting Time

We start by analyzing the behavior of the mentioned policies concerning their average waiting time. Initially, the MDP $M_{1,2}^{\mathrm{ud}} := (\texttt{ela-1-2-100-02-ud}, 0.8)$ will be considered that models combined up and down traffic and assumes a large penalty cost of $c_p = 100$ and a very small waiting queue length of $q = 2$. The first initial state we look at is the state i_1, where the elevator is located at floor 1 and one request with start floor 8 and destination floor 1 is currently waiting, see Figure 4.21(a). Note that this illustration renders all parameters of the state. Specifying the waiting times of the requests in the model for the average waiting time is only required for the policies FIFO and ReplanSQ. In the following, we will use such diagrams to present states in elevator control MDPs without specifying all of their parameters explicitly. Moreover, we will denote each request as a triple of its start and destination floors and its current waiting time, i. e., the request in state i_1 is denoted by $(8, 1, 0)$.

The approximation results for the policies NN, FIFO, Replan, ReplanS, and Ignore w. r. t. the initial state i_1 are given in Figure 4.21(b). The picture shows that the performance of most of the considered policies differs substantially: The policy NN seems to be slightly better than ReplanS which itself outperforms Replan. For ReplanSQ, we have $v_{i_1}^\alpha(\mathsf{NN}) < v_{i_1}^\alpha(\mathsf{ReplanSQ}) < v_{i_1}^\alpha(\mathsf{Replan})$. The policies FIFO and Ignore give the worst results and are proven to be more than 30 % away from an optimal policy. Note that the latter two policies will serve the present request $r_1 := (8, 1, 0)$ first, independently of possible future requests. Therefore, one would also expect FIFO and Ignore to give bad results for the initial state i_1 because serving the request r_1 keeps the elevator busy for a long time and prevents the service of possible other requests that could be transported faster. Recall that all waiting requests in a given state are of the same importance as we analyze the (discounted) average waiting

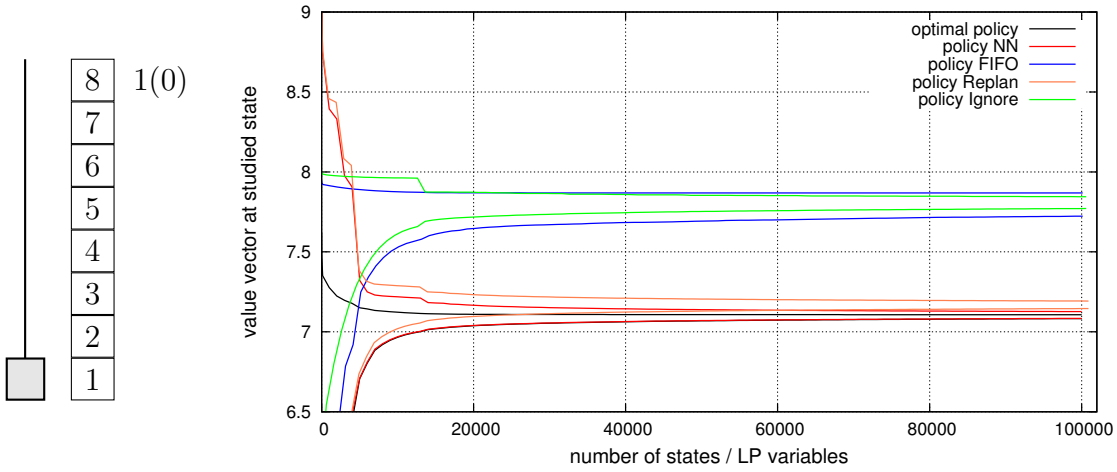


(a) (b)

Figure 4.21: (a) State i_1 in an elevator control MDP w. r. t. the average waiting time for a single elevator e and a set F of eight floors. The state is of the form $i_1 = ((\sigma_f)_{f \in F}, f_e, d_e)$, where $\sigma_8 = (8, 1, 0)$ and $\sigma_f = \emptyset$ for each $f \in F \setminus \{8\}$, $f_e = 1$, and $d_e = 0$. (b) Approximation results for the elevator control MDP $M^{\mathrm{ud}}_{1,2}$ and the initial state i_1. We have $\varepsilon^{\alpha}_{i_1}(\mathsf{NN}) \geq 7.1\,\%$, $\varepsilon^{\alpha}_{i_1}(\mathsf{FIFO}) \geq 32.2\,\%$, $\varepsilon^{\alpha}_{i_1}(\mathsf{Replan}) \geq 16.6\,\%$, $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanS}) \geq 9.0\,\%$, and $\varepsilon^{\alpha}_{i_1}(\mathsf{Ignore}) \geq 36.7\,\%$. Moreover, $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanSQ}) \geq 9.9\,\%$.

time. Thus upon arrival of another request r_2, the initial request r_1 is just as important to be served as r_2, although it has been waiting for a longer time.

However, one has to mention that the analyzed policies are at a disadvantage for our Markov decision process formulation since they disregard the very restricted length of the waiting queues, possibly incurred penalty costs, and the probability distribution for the start and destination floors for future requests. This is especially awkward for the considered instance `ela-1-2-100-02-ud`: in expectation each second request has start floor 1, which makes this floor crucial concerning penalty costs. In particular, one can show that an optimal policy will not serve the request r_1 until further requests arrive (this is also the case for a penalty cost of $p_c = 10$). This observation can be interpreted twofold. On the one hand, it can indeed be reasonable for certain traffic situations in some elevator system to let an elevator wait although a request could be served. Such aspects will be taken into account in a stochastic model for the future, which is impossible if it is assumed that no information about the future is available, as done by competitive analysis. On the other hand, one can look upon the MDP $M^{\mathrm{ud}}_{1,2}$ as being somehow unrealistic, in particular it is not fair to compare the studied policies with an optimal one for the considered setting.



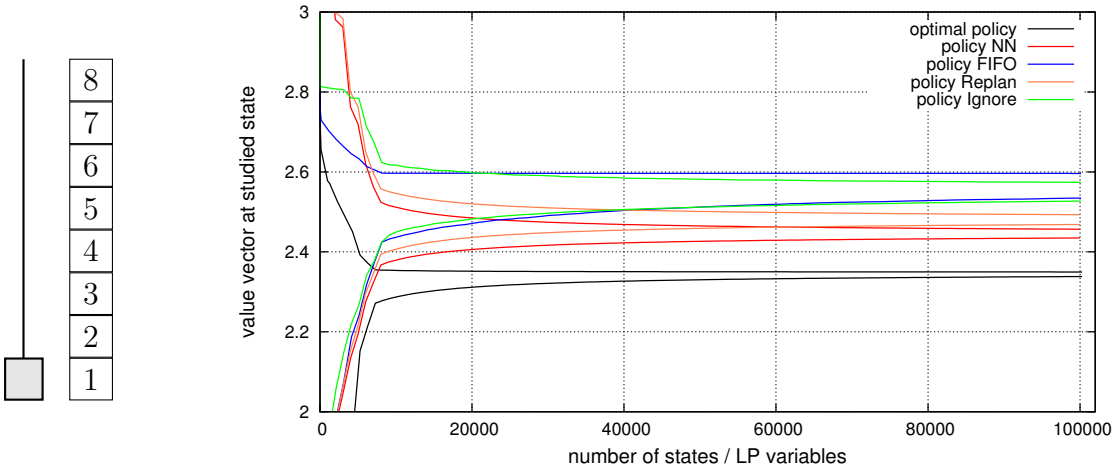
(a) (b)

Figure 4.22: (a) State i_1 in an elevator control MDP w. r. t. the average waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP $M^{\mathrm{sp}}_{1,4}$ and the initial state i_1. We have $\varepsilon^{\alpha}_{i_1}(\mathsf{NN}) \geq 0\,\%$, $\varepsilon^{\alpha}_{i_1}(\mathsf{FIFO}) \geq 8.7\,\%$, $\varepsilon^{\alpha}_{i_1}(\mathsf{Replan}) \geq 0.5\,\%$, and $\varepsilon^{\alpha}_{i_1}(\mathsf{Ignore}) \geq 9.3\,\%$. Moreover, $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanS}) \geq 0\,\%$ and $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanSQ}) \geq 0.4\,\%$.

Nonetheless, the results give a reasonable comparison for the concrete policies under consideration since all of them serve the request r_1 immediately unless further requests are released before the elevator loads r_1. We mention that the large penalty cost and the high probability for the occurrence of penalties are the reasons that the policies differ significantly here. Thus, the results in Figure 4.21(b) particularly show that NN and ReplanS are most suitable among the analyzed policies to prevent penalty costs.

Remark 4.3.2 As mentioned before, it is important to estimate the optimal or policy cost at candidate states using the involved lower and upper bounds developed in Section 4.1.5. For instance, the approximation process for the state i_1 given in Figure 4.21(b) yields $v^{\alpha}_{i_1} \in [7.94, 8.08]$ and $v^{\alpha}_{i_1}(\mathsf{NN}) \in [8.66, 8.88]$, which gives a relative guarantee of 1.9 % and 2.6 %, respectively. On the other hand, estimating the cost at candidate states by the trivial bounds gives $v^{\alpha}_{i_1} \in [7.84, 8.49]$ and $v^{\alpha}_{i_1}(\mathsf{NN}) \in [8.56, 9.39]$ after having generated 100 000 states, which corresponds to gaps of 8.4 % and 9.7 %. Obviously, the former approximations using the improved state-specific bounds are much better. △

Due to the mentioned drawbacks of the MDP $M^{\mathrm{ud}}_{1,2}$, we study next the initial state i_1 in the setting of the MDP $M^{\mathrm{sp}}_{1,4} := (\texttt{ela-1-4-10-02-sp}, 0.8)$ for



(a) (b)

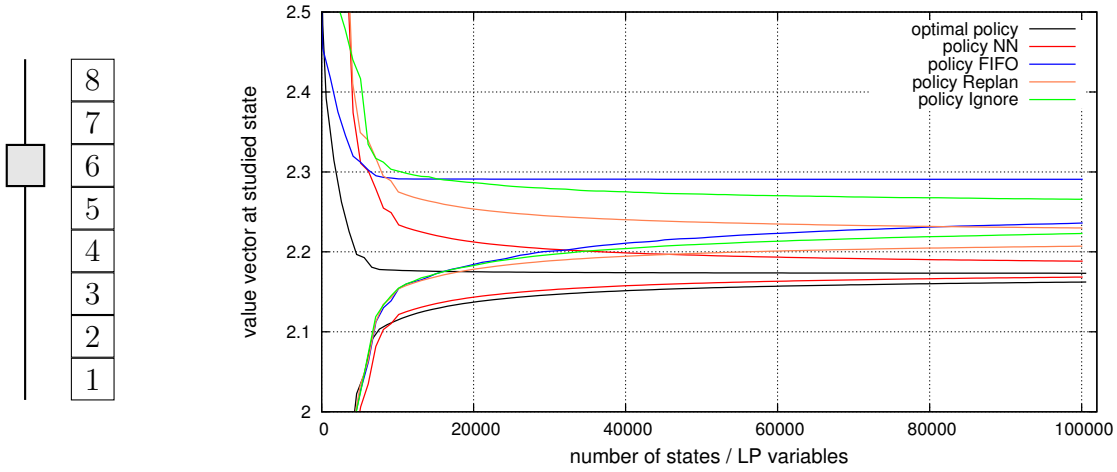
Figure 4.23: (a) Trivial state i_2 in an elevator control MDP w. r. t. the average waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP $M^{\mathrm{sp}}_{1,4}$ and the initial state i_2. We have $\varepsilon^{\alpha}_{i_2}(\mathsf{NN}) \geq 3.6\,\%$, $\varepsilon^{\alpha}_{i_2}(\mathsf{FIFO}) \geq 7.9\,\%$, $\varepsilon^{\alpha}_{i_2}(\mathsf{Replan}) \geq 5.1\,\%$, and $\varepsilon^{\alpha}_{i_2}(\mathsf{Ignore}) \geq 7.5\,\%$. Moreover, $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanS}) \geq 4.1\,\%$ and $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanSQ}) \geq 4.9\,\%$.

the special traffic situation in the case of a waiting queue length of $q = 4$ and a moderate penalty cost of $p_c = 10$. In this case, the waiting times produced by a policy instead of the incurred penalty cost dominates the total expected discounted cost. See Figure 4.22(b) for the associated approximation results. Obviously, the policies under consideration differ less than before: NN and ReplanS seem to be close to an optimal policy for the state i_1, while Replan and ReplanSQ are only slightly worse than an optimal policy but also outperformed a little by NN. Still, FIFO and Ignore are both clearly inferior compared to the other policies. This is what one would expect.

Next we consider the trivial initial state $i_2 = i_{\mathrm{elv}}$ shown in Figure 4.23(a), where no transport request is waiting and the elevator is situated at floor 1. The associated approximation results for the MDP $M^{\mathrm{sp}}_{1,4}$ are depicted in Figure 4.23(b).

Obviously, none of the considered policy is really close to an optimal policy for the initial state i_2. Concerning the studied policies the results show the same ranking as before. However, the most interesting observation we made relating to these results is that 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

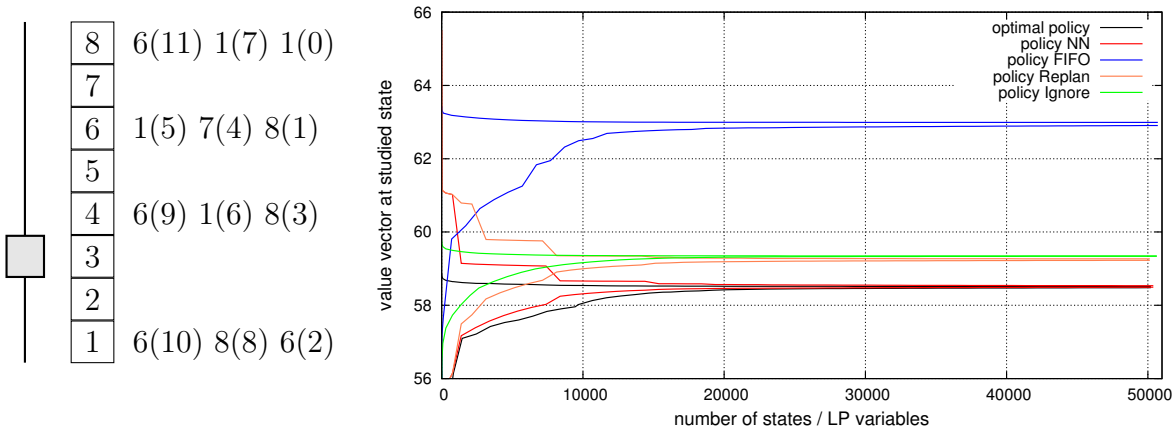


(a) (b)

Figure 4.24: (a) Modified trivial state i_3 in an elevator control MDP w. r. t. the average waiting time, where the elevator is located at floor 6. (b) Approximation results for the elevator control MDP $M^{\mathrm{sp}}_{1,4}$ and the initial state i_3. We have $\varepsilon^{\alpha}_{i_3}(\mathsf{NN}) \geq 0\,\%$, $\varepsilon^{\alpha}_{i_3}(\mathsf{FIFO}) \geq 2.9\,\%$, $\varepsilon^{\alpha}_{i_3}(\mathsf{Replan}) \geq 1.6\,\%$, and $\varepsilon^{\alpha}_{i_3}(\mathsf{Ignore}) \geq 2.3\,\%$. Moreover, $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanS}) \geq 0.1\,\%$ and $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanSQ}) \geq 1.1\,\%$.

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 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 3.4.5 on page 68. 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 MDP $M^{\mathrm{sp}}_{1,4}$.

Since all considered policies do trivial parking, using the modified trivial state i_3, where the elevator is situated at floor 6 (see Figure 4.24(a)) seems to be a fair initial state. The approximations for state i_3 are shown in Figure 4.24(b). In this situation the differences between the studied policies are



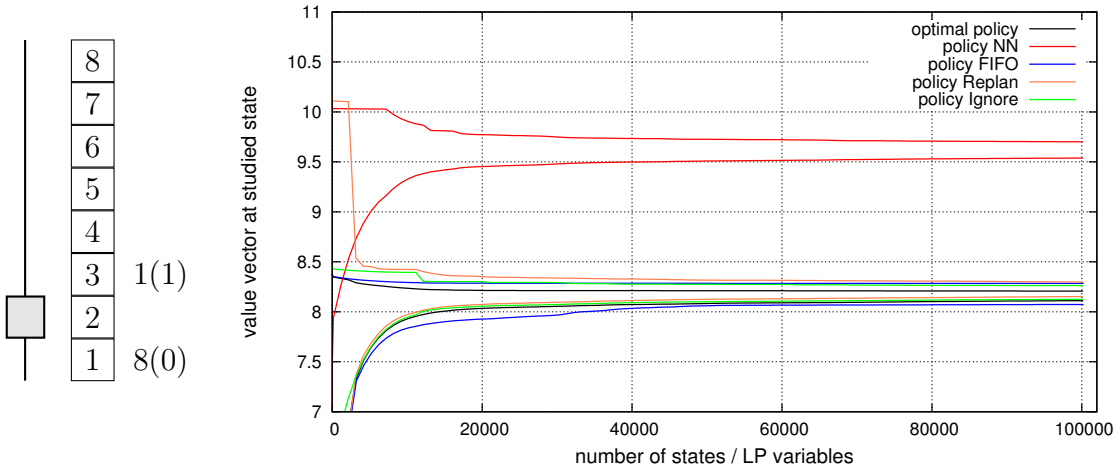
(a) (b)

Figure 4.25: (a) State i_4 in an elevator control MDP w. r. t. the average waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP $M^{\mathrm{sp}}_{1,4}$ and the initial state i_4. We have $\varepsilon^{\alpha}_{i_4}(\mathsf{NN}) \geq 0\,\%$, $\varepsilon^{\alpha}_{i_4}(\mathsf{FIFO}) \geq 7.5\,\%$, $\varepsilon^{\alpha}_{i_4}(\mathsf{Replan}) \geq 1.2\,\%$, and $\varepsilon^{\alpha}_{i_4}(\mathsf{Ignore}) \geq 1.4\,\%$. Moreover, $\varepsilon^{\alpha}_{i_1}(\mathsf{ReplanS}), \varepsilon^{\alpha}_{i_1}(\mathsf{ReplanSQ}) \geq 0\,\%$.

little. NN seems to be close to optimal and provably outperforms the other concrete policies except for ReplanS. Moreover, the results let us suppose that Replan is better than Ignore, which itself might be superior to FIFO. However, we do not have a proof for that. We mention that the computed bounds for ReplanSQ are similar but slightly better than those of Replan.

Next we analyze the policies w. r. t. the initial state i_4 shown in Figure 4.25(a) that captures a situation of a high system load, i. e., many requests are waiting. The approximation progess for the initial state i_4 in the MDP $M^{\mathrm{sp}}_{1,4}$ is depicted in Figure 4.25(b). Note that FIFO serves the requests in a quite inappropriate order, which results in a bad performance of this policy: it is about 7.5 % away from an optimal policy. The costs of the policies NN, ReplanS, and ReplanSQ are almost optimal, while Replan and Ignore give slightly worse results. Moreover, Replan and Ignore perform similarly. This shows that possible future requests affect the cost of a policy only very little here. We mention that in this situation one can directly determine the costs of the considered policies quite accurately since possible future requests barely affect them.

Except for a missing parking approach, the policy NN performs almost optimally for the initial states considered above. In the following we study the state i_5 defined in Figure 4.26(a), which is designed to give a bad per-



(a) (b)

Figure 4.26: (a) State i_5 in an elevator control MDP w. r. t. the average waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP $M_{1,4}^{\mathrm{ud}}$ and the initial state i_5. We have $\varepsilon_{i_5}^{\alpha}(\mathsf{NN}) \geq 16.2\,\%$ and $\varepsilon_{i_5}^{\alpha}(\mathsf{FIFO}), \varepsilon_{i_5}^{\alpha}(\mathsf{Replan}), \varepsilon_{i_5}^{\alpha}(\mathsf{Ignore}) \geq 0\,\%$. Moreover, $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanS}), \varepsilon_{i_1}^{\alpha}(\mathsf{ReplanSQ}) \geq 0\,\%$.

formance for NN, and consider the MDP $M_{1,4}^{\mathrm{ud}} := (\texttt{ela-1-4-10-02-ud}, 0.8)$ modeling combined up and down traffic in an elevator system with a waiting queue length of $q = 4$ and a penalty cost of $p_c = 10$. Recall that the tie-breaking rule of NN is to move down if there does not exist a unique request located nearest to the elevator. Figure 4.26(b) shows the approximation results of the policies under consideration for the initial state i_5. As expected, NN performs badly as we have $\varepsilon_{i_5}^{\alpha}(\mathsf{NN}) \geq 16.2\,\%$. Serving the request $r_1 := (1, 8, 1)$ before $r_2 := (3, 1, 2)$ makes r_1 and r_2 wait for 1 and 15 more time slots, respectively, while the opposite order is optimal and results in additional waiting times of 6 and 1 time units. All of the remaining policies choose the latter order for serving the present requests and consequently perform very similar to each other and much better than NN. These results show that the different behavior of the policies after serving the requests r_1 and r_2, which takes at least 14 time units, has only a little impact on the total expected 0.8-discounted costs of the policies. This observation points out that our approximation approach seems to be inappropriate to investigate the long-term behavior of the policies unless a greater discount factor is considered.

Clearly, the reason for the bad performance of the policy NN w. r. t. initial state i_5 is its tie-breaking rule. Of course one can construct symmetric situ-

ations using another elevator control MDP and another initial state, where the opposite tie-breaking rule is bad also. Apart from taking advantage of the tie-breaking rule, we were not able to detect "fair" initial states such that the cost of the policy NN is actually greater than that of an optimal policy. All further situations that feature a non-optimal performance of NN seem to be due to the trivial parking policy of NN and possible penalty costs that are ignored by all considered policies.

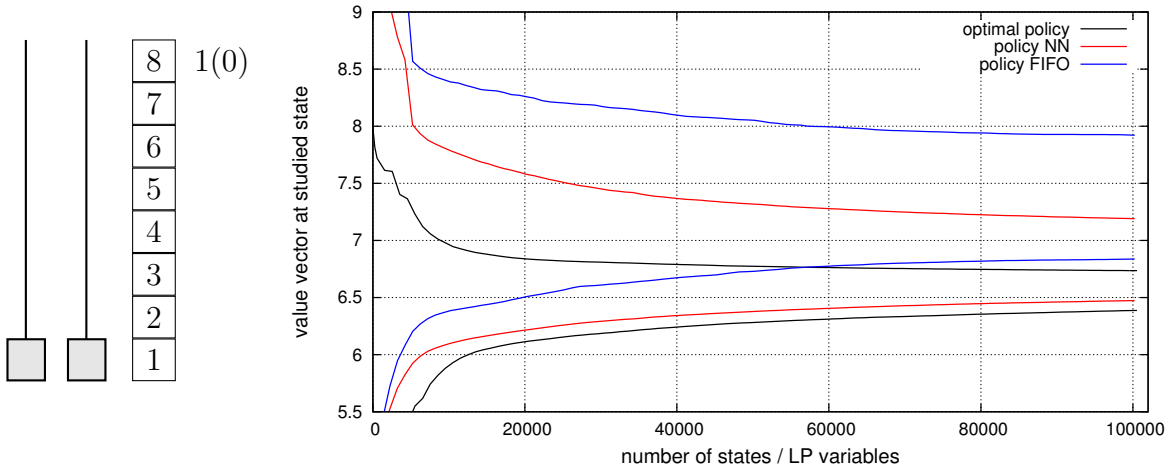
In the last part of this section we study a group of two elevators serving combined up and down traffic of identical intensities. Let us consider the MDP $M_{2,4}^{\mathrm{ud}} := (\texttt{ela-2-4-10-03-ud}, 0.8)$, where the probability that a new request is released at a time slot equals 0.3. As mentioned before, we only consider the policies FIFO and NN in the case of two elevators. Again, an empty elevator waits at its current floor if it does not have an assigned request, which is especially the case if there are no further requests waiting at all. Assuming that there exists at least one waiting request, the assignment strategies are as follows:

FIFO Consider the request with smallest waiting time and let f be its start floor. Then, both elevators compete for serving this request: the elevator that can reach floor f first taking into account possibly loaded requests, serves the request next. Then, the request with the next larger waiting time is considered and assigned similarly, and so on. This is done for all waiting requests until both elevators have been assigned a request or there are no waiting requests left (at this point the next actions for both elevators have been determined).

NN For both elevators e_1 and e_2, determine the smallest time t_k by which elevator e_k can reach the start floor of any waiting request after serving possibly loaded requests. Some elevator e^s with the smallest time is assigned the corresponding request. It remains to find the next request to be served by the other elevator $e^o \in \{e_1, e_2\} \setminus \{e^s\}$. For the remaining requests, this procedure is repeated until the elevator e^o reaches its nearest request earlier than e^s can pick up another request or no requests are left. In the first case, the elevator e^o is assigned the associated request, while e^o waits at its current floor otherwise.

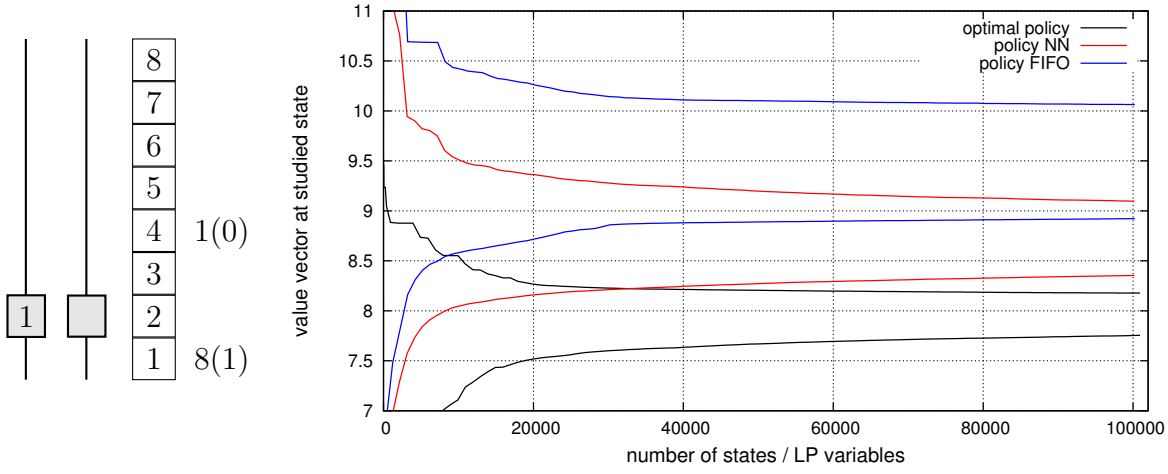
These assignment strategies are used as they anticipate the future evolution of the system due to the interaction of both elevators if no further requests are released. This would be achieved easier if the state space encoded tentative schedules.

Firstly, we analyze the performance of the policies NN and FIFO for the initial state i_6 defined in Figure 4.27(a). Note that state i_6 captures the same



(a) (b)

Figure 4.27: (a) State i_6 in an elevator control MDP w. r. t. the average waiting time for two elevators and 8 floors. (b) Approximation results for the elevator control MDP $M_{2,4}^{\mathrm{ud}}$ and the initial state i_6. We have $\varepsilon_{i_6}^{\alpha}(\mathsf{NN}) \geq 0\,\%$ and $\varepsilon_{i_6}^{\alpha}(\mathsf{FIFO}) \geq 1.5\,\%$.



(a) (b)

Figure 4.28: (a) State i_7 in an elevator control MDP w. r. t. the average waiting time for two elevators and 8 floors. (b) Approximation results for the elevator control MDP $M_{2,4}^{\mathrm{ud}}$ and the initial state i_7. We have $\varepsilon_{i_7}^{\alpha}(\mathsf{NN}) \geq 2.2\,\%$ and $\varepsilon_{i_7}^{\alpha}(\mathsf{FIFO}) \geq 9.1\,\%$.

situation as state i_1 (see Figure 4.21(a)) except for the number of available elevators. Obviously, at state i_6, the two policies NN and FIFO choose the action to move one elevator one floor up, while the second elevator waits at floor 1 (approximation results let us suppose that this action is optimal, although we did not obtain a proof for that). Thus, their cost values will differ only due to decisions made depending on possible future request. Using i_6 as initial state, we obtain the approximation results depicted in Figure 4.27(b). On the one hand, the policy NN seems to be close to an optimal policy. Nonetheless, the results suggest that NN does not perform optimally for the initial state i_6. On the other hand, we obtain a proof that the cost of FIFO exceeds the optimal cost.

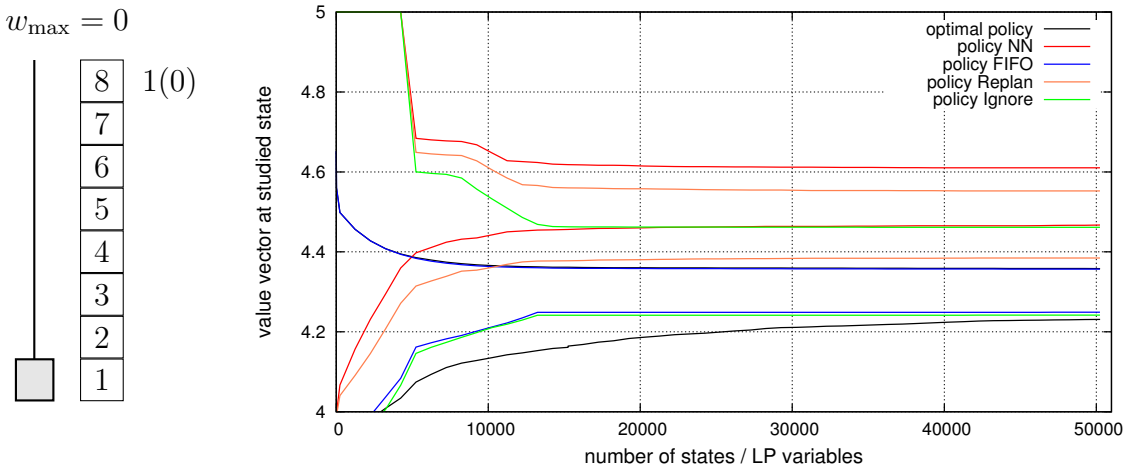
Next we study the initial state i_7 defined in Figure 4.28(a). Denote by e_1 and e_2 the loaded and the empty elevator, respectively, i. e., we have for the destination floors $d_{e_1} = 1$ and $d_{e_2} = 0$. The state i_7 represents a situation where the assignment decision of the policies NN and FIFO seems to be bad: both policies serve the request $r_1 := (1, 8, 1)$ by elevator e_2 and the request $r_2 := (4, 1, 0)$ by elevator e_1. Note that this schedule results in additional waiting times for the requests r_1 and r_2 of 1 and 5 time units, respectively, whereas the opposite assignment gives an additional waiting time of 2 time units for both requests. As expected, the two policies do not perform optimally for the initial state i_7, see Figure 4.28(b). Moreover, the approximation results let us suppose that the policy NN is still better than FIFO.

We mention that considering the special traffic situation as done in the case of a single elevator instead of combined up and down traffic, yields similar approximation results to those given in the Figures 4.27(b) and 4.28(b), but separates the policies more clearly from each other.

Maximum Waiting Time

This section analyzes the performance of the policies NN, FIFO, Replan, ReplanS, ReplanSQ, 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 start by considering the MDP $M_1^{\text{ud}} := (\texttt{elm-1-02-ud}, 0.8)$ and the state i_1 defined in Figure 4.29(a). Note that the illustrations of states in this section additionally specify the current maximum waiting time $w_{\max}$.

Figure 4.29(b) 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 Re-



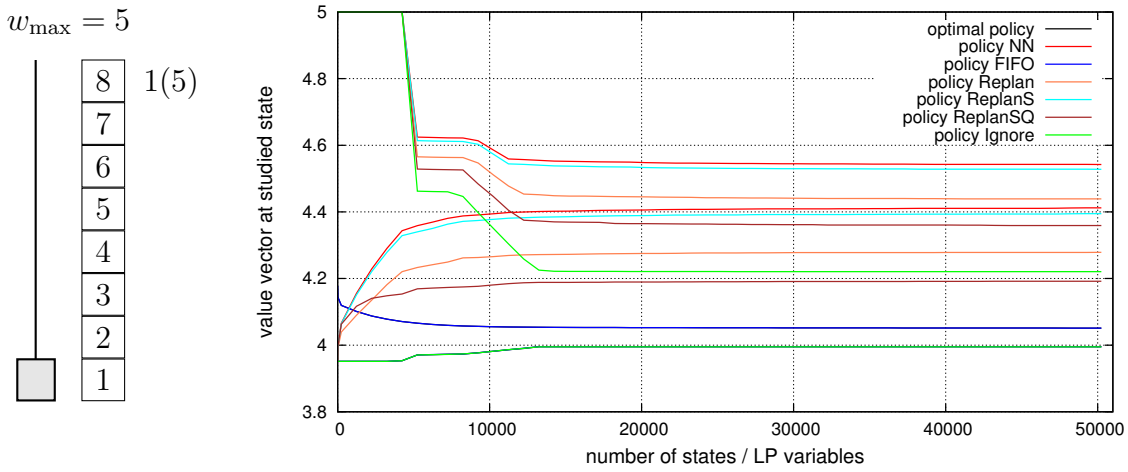
(a) (b)

Figure 4.29: (a) State i_1 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP M_1^{ud} and the initial state i_1. 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 $\varepsilon_{i_1}^{\alpha}(\mathsf{Ignore}) \geq 0\,\%$. Moreover, $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanS}) \geq 2.2\,\%$ and $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanSQ}) \geq 0.3\,\%$.

plan is shown to be greater than the cost of FIFO and the optimal cost. We mention that ReplanSQ achieves similar but slightly better bounds than Replan. The same is true for ReplanS and NN. It is clear that the behavior of a policy like NN may be inappropriate in order to achieve a small maximum waiting time: it lacks any stability in the service of the present requests as it may postpone the service of single requests again and again due to serving others. For instance, the request $(8, 1, 0)$ in state i_1 may not be served for a long time, while FIFO and Ignore guarantee this request to be served next. Apparently, serving the request immediately is strongly rewarded by considering the total expected discount cost as objective criterion.

Note that the state i_2 specified in Figure 4.30(a) is similar to i_1, but now the only present request has already been waiting for 5 time units. Obviously, this makes the request $r_1 := (8, 1, 5)$ even more important to be served soon. Note that the final waiting time of request r_1 will be 12, which allows that each new request can wait at least 12 time units before it may cause the maximum waiting time to increase. Thus we expect the same ranking of the policies as before, but the costs to differ at a greater extent. The approximation results in the case of the initial state i_2 are depicted in Figure 4.30(b) and show the expected behavior.

On the one hand, the policies NN and ReplanS are proved to perform worse



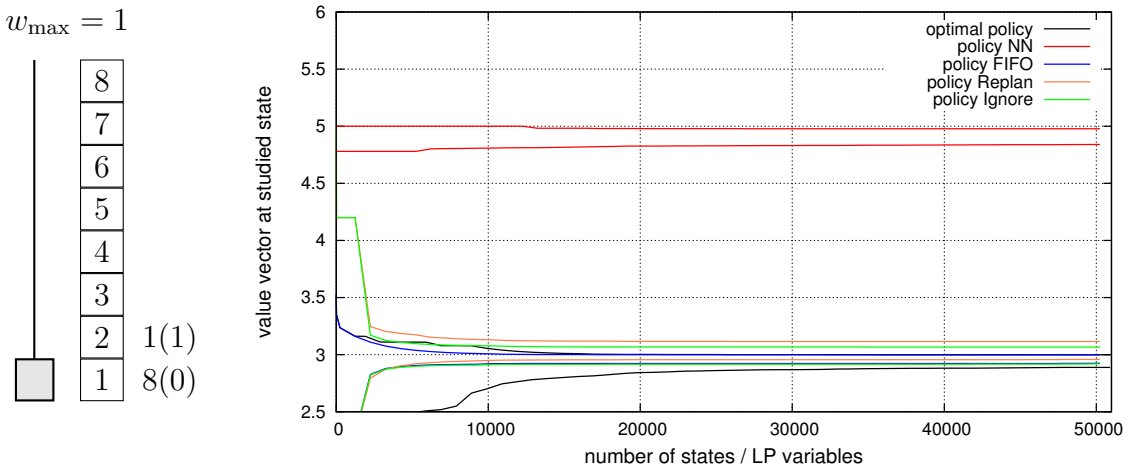
(a) (b)

Figure 4.30: (a) State i_2 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP M_1^{ud} and the initial state i_2. We have $\varepsilon_{i_2}^{\alpha}(\mathsf{NN}) \geq 8.9\,\%$, $\varepsilon_{i_2}^{\alpha}(\mathsf{FIFO}) \geq 0\,\%$, $\varepsilon_{i_2}^{\alpha}(\mathsf{Replan}) \geq 5.6\,\%$, $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanS}) \geq 8.5\,\%$, $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanSQ}) \geq 3.5\,\%$, and $\varepsilon_{i_2}^{\alpha}(\mathsf{Ignore}) \geq 0\,\%$.

than ReplanSQ which seems to have also a better cost value than Replan. On the other hand, FIFO is close to optimal and outperforms all policies except for Ignore which may be close to optimal, too. Note that the approximation of Ignore is typically worse than that of the other policies since modeling this policy requires a special Markov decision process with a larger state space, as described in the beginning of the section.

Next we consider three situations with the two requests $r_1 := (1, 8, w_1)$ and $r_2 := (2, 1, w_2)$ for different waiting times $w_1, w_2 \in \mathbb{N}_0$. Note that the state i_3 in Figure 4.31(a) represents a very disadvantageous situation for the policy NN since it will serve the request r_1 with the small current waiting time $w_1 = 0$ first. Moreover, serving r_1 keeps the elevator busy for a long time. For these reasons scheduling the requests in the opposite order should be much better. The results in Figure 4.31(b) show a very bad performance of the policy NN, while the costs of all other policies seem to be near the optimal optimal cost.

Let us now consider the case where the current waiting times of the requests r_1 and r_2 are exchanged, i. e., we have $w_1 = 1$ and $w_2 = 0$. The resulting state i_4 is given in Figure 4.32(a). Note that, similar to NN, the policy FIFO now also serves request r_1 first, whereas Replan, all its variants, and Ignore schedule the requests in the opposite order. See Figure 4.32(b) for

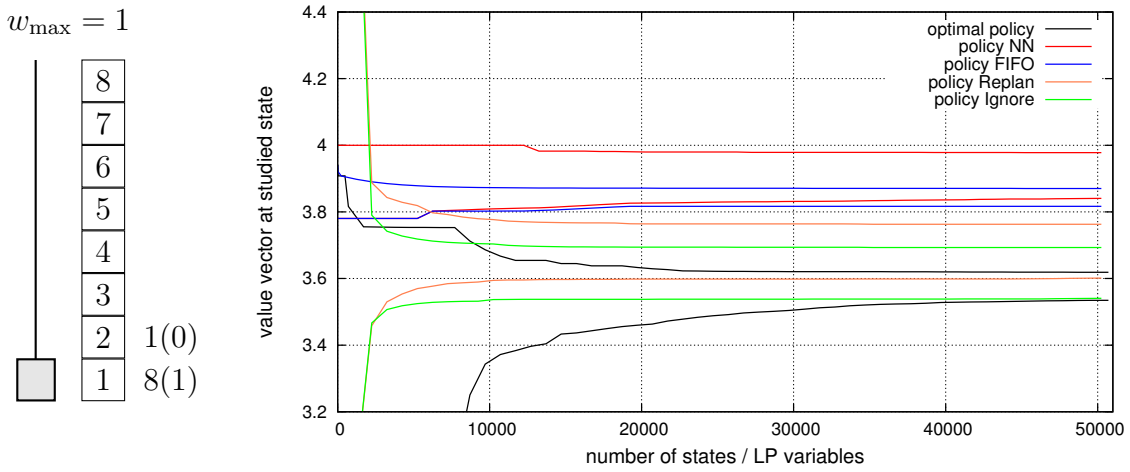


(a) (b)

Figure 4.31: (a) State i_3 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP M_1^{ud} and the initial state i_3. We have $\varepsilon_{i_3}^{\alpha}(\mathsf{NN}) \geq 61.4\,\%$ and $\varepsilon_{i_3}^{\alpha}(\mathsf{FIFO}), \varepsilon_{i_3}^{\alpha}(\mathsf{Replan}), \varepsilon_{i_3}^{\alpha}(\mathsf{Ignore}) \geq 0\,\%$. Moreover, $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanS}), \varepsilon_{i_1}^{\alpha}(\mathsf{ReplanSQ}) \geq 0\,\%$.

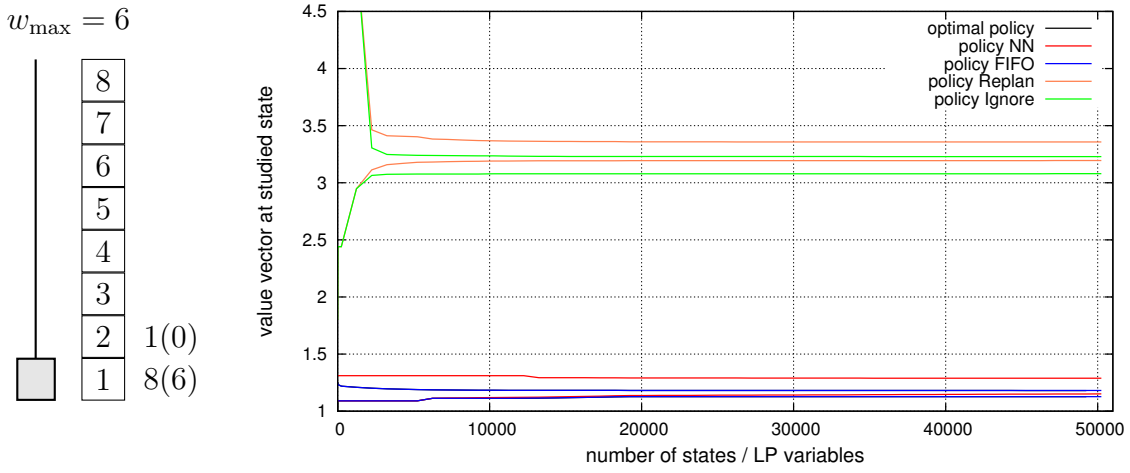
the approximation results w. r. t. start state i_4. We see that the scheduling order that serves r_1 before r_2 as used by NN and FIFO is still worse than the opposite one. The costs of NN and FIFO are proven to be greater than those of the other policies. Ignore is probably close to an optimal policy and seems to outperform all variants of Replan, which may serve other requests before r_1. Altogether, we see that the advantage of postponing the service of the request r_1 requiring a very long trip, which implies the better makespan here, outweighs the possible advantage of serving the requests in order of their arrival.

This may change if the waiting time w_1 of request r_1 is greater than 1. In state i_5, see Figure 4.33(a), we assume this waiting time to be 6. Moreover, we consider a maximum waiting time of $w_{\max} = 6$, too. For the initial state i_5, we obtained the approximation results shown in Figure 4.33(b). Now we observe the opposite behavior of the policies: request r_1 is better served before r_2. However, we mention that serving r_2 before r_1 will be much better in practice: the resulting final waiting times are 1 and 10, whereas the opposite order produces waiting times of 15 and 6. This order is used by Replan, ReplanS, ReplanSQ, and Ignore. Consequently, their performance is bad, while that of FIFO and NN is close to optimal. Considering the discounted cost is again responsible for these unrealistic results. Clearly, a



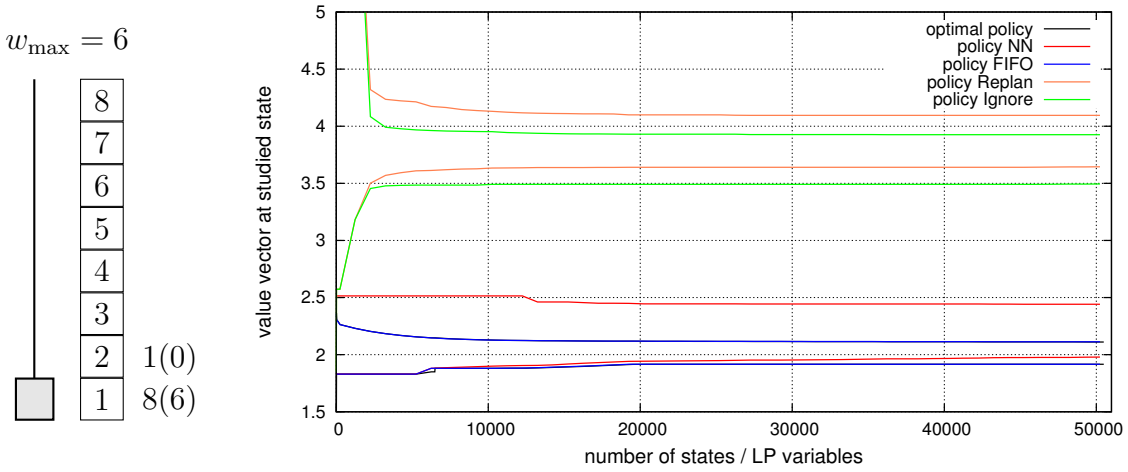
(a) (b)

Figure 4.32: (a) State i_4 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP M_1^{ud} and the initial state i_4. We have $\varepsilon_{i_4}^{\alpha}(\mathsf{NN}) \geq 6.1\,\%$, $\varepsilon_{i_4}^{\alpha}(\mathsf{FIFO}) \geq 5.5\,\%$, and $\varepsilon_{i_4}^{\alpha}(\mathsf{Replan}), \varepsilon_{i_4}^{\alpha}(\mathsf{Ignore}) \geq 0\,\%$. Moreover, $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanS}), \varepsilon_{i_1}^{\alpha}(\mathsf{ReplanSQ}) \geq 0\,\%$.



(a) (b)

Figure 4.33: (a) State i_5 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP M_1^{ud} and the initial state i_5. We have $\varepsilon_{i_5}^{\alpha}(\mathsf{NN}), \varepsilon_{i_5}^{\alpha}(\mathsf{FIFO}) \geq 0$, $\varepsilon_{i_5}^{\alpha}(\mathsf{Replan}) \geq 170.6\,\%$, and $\varepsilon_{i_5}^{\alpha}(\mathsf{Ignore}) \geq 160.8\,\%$. Moreover, $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanS}) \geq 174.3\,\%$ and $\varepsilon_{i_1}^{\alpha}(\mathsf{ReplanSQ}) \geq 169.2\,\%$.



(a) (b)

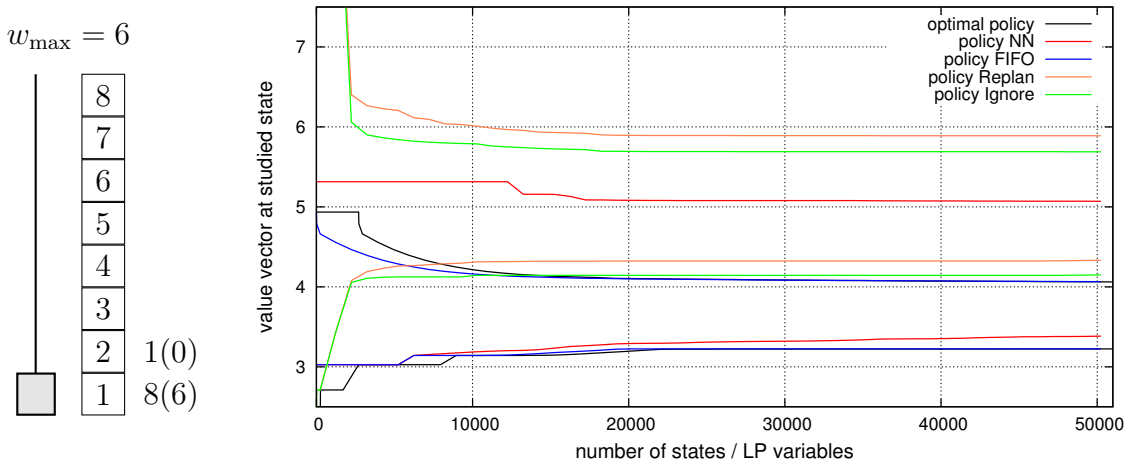
Figure 4.34: (a) State i_5 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP (`elm-1-02-ud`, 0.85) and the initial state i_5. We have $\varepsilon^{\alpha}_{i_5}(\mathsf{NN}), \varepsilon^{\alpha}_{i_5}(\mathsf{FIFO}) \geq 0$, $\varepsilon^{\alpha}_{i_5}(\mathsf{Replan}) \geq 72.6\,\%$, and $\varepsilon^{\alpha}_{i_5}(\mathsf{Ignore}) \geq 65.5\,\%$.

discount factor close to 1 would favor the opposite scheduling order.

To investigate whether our approximation algorithm is able to show this behavior, we also looked at the initial state i_5 in the Markov decision process `elm-1-02-ud` assuming a discount factor of 0.85, 0.9, and 0.95. For these situations we did not yet consider the policies ReplanS and ReplanSQ. The associated results are given in the Figures 4.34–4.36. For the discount factors 0.85 and 0.9, we obtain the same ranking of the considered policies as before. However, the extent by which the costs of the policies differ reduces with increasing disount factor. Unfortunately, nothing can be shown by generating 50 000 variables in the case of the discount factor 0.95. We conclude that our algorithm is unable to even indicate for the considered Markov decision process model that serving request r_2 before r_1 is better.

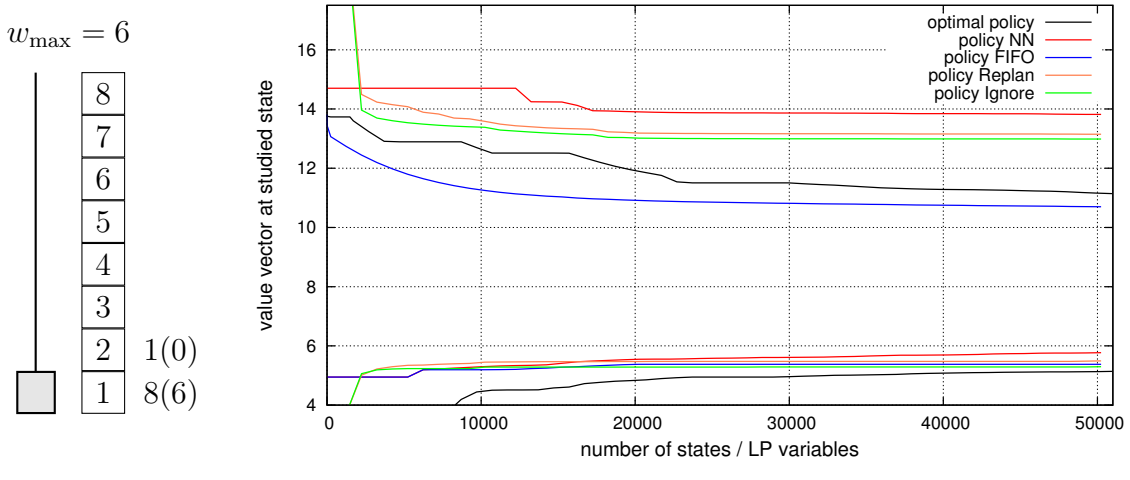
Summary

The results for the considered average and maximum waiting time elevator control MDPs show the following. On the one hand, by analyzing several initial states we saw that the approximation results for the studied policies sometimes reflect observations made in simuation, e. g., see [GHKR99, Hau99]. This includes the good performance of NN w. r. t. the average waiting time



(a) (b)

Figure 4.35: (a) State i_5 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP (`elm-1-02-ud`, 0.9) and the initial state i_5. We have $\varepsilon^{\alpha}_{i_5}(\mathsf{NN}), \varepsilon^{\alpha}_{i_5}(\mathsf{FIFO}) \geq 0$, $\varepsilon^{\alpha}_{i_5}(\mathsf{Replan}) \geq 6.6\,\%$, and $\varepsilon^{\alpha}_{i_5}(\mathsf{Ignore}) \geq 2.2\,\%$.



(a) (b)

Figure 4.36: (a) State i_5 in an elevator control MDP w. r. t. the maximum waiting time for a single elevator and 8 floors. (b) Approximation results for the elevator control MDP (`elm-1-02-ud`, 0.95) and the initial state i_5. We have $\varepsilon^{\alpha}_{i_5}(\mathsf{NN}), \varepsilon^{\alpha}_{i_5}(\mathsf{FIFO}) \geq 0$ and $\varepsilon^{\alpha}_{i_5}(\mathsf{Replan}), \varepsilon^{\alpha}_{i_5}(\mathsf{Ignore}) \geq 0\,\%$.

and the insight that policies which achieve small average waiting times may often produce high maximum waiting times, and vise versa.

On the other hand, we have seen that total expected discounted cost criterion strongly favors greedy solutions which only focus on the costs incurred for few stages, especially for small discount factors. Moreover, long-term simulation runs often show very large performance differences for some policies, e. g., bad policies like FIFO cannot cope with the traffic load and the number of waiting requests increases continuously. Note that it requires a lot of stage transitions until a large system load, i. e., many waiting requests, can emerge. Thus, it is impossible that two policies will reach substantially different situations concerning the system load within few stages. Since our analysis takes into account only a smaller local part of the total state space and differences occuring after many transitions account only very little to the total expected discounted cost, one cannot expect to prove large gaps as observed in simulations. Instead, the total expected discounted cost is mostly dominated by present requests.

We conclude that analyzing elevator control policies in our MDP model is inadequate to really capture the long-term behavior observed in practice and simulation unless a discount factor close to 1 is used. Unfortunately, for a great discount factor, our approximation algorithm is not suitable to provide good approximation guarantees.

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.

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.

Markov decision process	NN		NNPark-f		NNMaxPark-f	
	avg.	max.	avg.	max.	avg.	max.
`elm-1-01-sp`	13.66	116.15	13.34	113.38	13.69	98.76
`elm-1-01-ud`	12.26	139.33	11.93	128.59	12.26	97.94

Table 4.17: Average value for the average and maximum waiting times achieved by the online algorithms NN and its variants NNPark-f and NNMaxPark-f according to 100 simulation runs for 10 000 time steps. The parking floor is chosen to be $f = 6$ for the instance `elm-1-01-sp` and $f = 1$ for `elm-1-01-ud`.

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 modeling the maximum waiting time, i. e., the queue length is infinite, is simulated for 10 000 time steps. We compute average values for the observed average and maximum waiting times for 100 simulation runs. Table 4.17 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. 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. We mention that an arrival probability of $p^r = 0.2$ already yields a high traffic intensity the system does not seem able to cope with in the long run. Thus, we focused on a lower traffic intensity here.

Chapter 5

Conclusions

The contribution of this thesis is a new framework for analyzing the quality of online algorithms or policies in a discounted MDP, respectively. The method is based on a column generation algorithm we developed for approximating the total expected discounted cost of an unknown optimal policy, a concrete policy, or a single action at a given initial state. We proposed various approaches for designing and extending the standard approximation algorithm in order to improve its practical efficiency. Computational results showed that our method often works well: we obtained realistic performance indicators for the quality of different online algorithms and were able to improve existing algorithms due to the insight provided by our analysis.

In the following we will summarize the highlights of our work in detail. We established a theorem which shows that the size of the total state space of an MDP does not impose any barrier on the practicability of our approximation algorithm. We developed an involved example proving that the approximation algorithm may generate more states than required by the construction of the theorem. In practice, however, the opposite is true: the algorithm typically inspects substantially fewer states.

Further theoretical issues related to our column generation algorithm were investigated. We obtained insight in the structure of dual bases and the pricing problem. For the latter, we gave a combinatorial interpretation and proved a combinatorial formula for computing the reduced profit of a candidate state. We deduced an alternative way to determine upper bounds on the considered value vector component based on the reduced profits. However, these bounds are in practice clearly inferior to those provided by the linear programs.

To figure out how to make our approximation algorithm most effective, we proposed and analyzed different required and optional settings:

- It turned out that the proposed construction of initial (almost dual feasible) bases for the linear programs combined with the dual simplex method is most suitable for solving the encountered linear programs fast.

- The probably most important issue in making the algorithm practically efficient is to guide the dynamic extension of the considered subset of states by the reduced profit structure of the current reduced linear program. Ignoring the amount of reduced profits as done by the strategy min-depth seems to be substantially inferior to provide strong bounds.

  For the considered MDPs, employing the pricing strategy direct is most appropriate for our implementation regarding both the quality of the added states concerning good approximations and the running time. The strategy combinatorial, which is derived from theoretical insight in the pricing problem and is conceptually equivalent to direct, is only slower than direct for the elevator control MDPs. However, the pricing strategy combinatorial is more flexible in the sense that it may still be employed if the candidate states are not maintained explicitly by the column generation algorithm.

- The approximation heuristics we developed seem not to be advantageous if the mentioned linear programming solver setting is used. Unfortunately, the promising heuristic weighting-policy-exploration, which is based on the combinatorial formula for computing reduced profits, lacks a mechanism to be properly parameterized in order to solve very few linear programs. Otherwise, using this heuristic may be beneficial.

The fact that incorporating reasonable heuristics does not pay off substantiates that the standard approximation algorithm is very effective if a good setting for the linear programming solver and the pricing strategy is employed.

For different discounted MDPs emerging from online optimization problems, various online algorithms were analyzed by our method. In all cases good approximations could be determined by inspecting only relatively small subsets of states. Although there are methods based on MDPs for providing practical solutions, e. g., see [CB98], other approaches do not seem able to give similarly good approximations for a component of the optimal value vector. Our results, representing the first of their kind, provide quite realistic performance indicators for the considered policies in the case of the online bin coloring problem and the studied online target date assignment problem. For online elevator control, the obtained results reflect observations from simulation only partially since our approach offers insight solely for short-term considerations.

Moreover, our method revealed weaknesses of the considered policies. Based on this valuable information, improved policies like SafeBin and NN-

MaxPark-f including a provably optimal parking strategy could be developed. In particular, SafeBin seems to be the currently best known online algorithm for the online bin coloring problem, which shows the potential of our method concerning the design of state-of-the-art policies.

By having considered different examples we obtained a feeling whether the column generation algorithm is suitable to provide good approximations for a given MDP or not. First of all one should keep in mind that our method represents a general framework that can be applied to an arbitrary discounted MDP. Consequently, one cannot expect the method to work properly for each MDP directly. For instance, the trivial lower and upper bounds on the components of the optimal value vector may be too weak. The elevator control MDPs are examples where involved problem-specific bounds are required in order to make the approach work at all.

Nonetheless, there exist parameters of an MDP that affect the practical performance of the approximation algorithm. Generally, our method will provide good approximations if

- the discount factor is "considerably smaller" than 1,
- only few actions are possible at a given state,
- for each state and action, the associated transition probabilities are concentrated on a small number of successor states, i. e., there exist few successors that are reached with significant probability, and
- the difference between available lower and upper bounds on the components of the optimal value vector is small, which follows, e. g., from a small difference between the minimal and maximal expected stage costs.

Typically, our method is unable to yield good approximations for large discount factors. Sometimes, however, a large discount factor is necessary to obtain significant results, as we have observed partially for the studied elevator control MDPs. It depends on the structure of the considered Markov decision process whether the total expected discounted cost of a policy reflects its practical performance realistically already for smaller discount factors, i. e., a local approximation method like ours may work properly.

On the one hand, we believe that computations based on smaller discount factors may provide results of high practical relevance for Markov decision processes that exhibit a strong local character in the sense that already few transitions can make a significant difference. This is the case for the described Markov decision process models for the online bin coloring problem

and the studied online target date assignment problem. For some states in our model for online elevator control, however, many or even all policies proceed identically for several transitions.

On the other hand, local approximation methods seem to be less appropriate if there exist many requests competing simultaneously for a common resource, e. g., an elevator. Then smaller discount factors tend to favor more greedy online algorithms that achieve low costs for the next few stages. For instance, we observed such a behavior for the maximum waiting time elevator control MDP and the initial state shown in Figure 4.33(a). Moreover, a local approximation method cannot yield meaningful results if high stage costs are only incurred in the long run.

Outlook We conclude this thesis with an outlook concerning possible improvements and applications of the proposed approximation method.

We believe that the second curse of dimensionality, i. e., many possible actions at a state, can be handled by an extension of our algorithm. Similarly to generating improving states, actions or constraints, respectively, could be added dynamically to the current reduced linear program in the spirit of a separation algorithm. For instance, such a method will be advantageous in the elevator control MDPs for groups of multiple elevators: already in the case of two elevators there can exist 16 different actions at a state.

Moreover, possible approaches to enhance our approximation algorithm include a memory saving implementation that particular refrains from storing the candidate states. Another idea would be to adaptively aggregate and disaggregate candidate states depending on the current solution of the reduced linear program similar to an approach described by Bertsekas and Castañon [BC89]. This way, more complicated problems suffering from the third curse of dimensionality, i. e., many possible successor states, might by tackled.

On the practical side, we are still interested in results for the online control of elevator groups that address the long-term performance of online algorithms. Using our Markov decision process model for elevator control, this requires computations for a greater discount factor. In order to obtain good approximations in this case, better bounds on the optimal value vector for estimating the impact of candidate states have to be constructed.

More promising may be considering a continuous-time model for elevator control with very few possible interarrival times for the requests. Contrary to the Markov decision process we studied in this thesis, in this model each state contains a newly released request. The resulting discounted MDPs with infinite state space will probably be more difficult to handle computationally,

but offer the important advantage that discounting takes place w. r. t. requests instead of time slots. Consequently, this model will be more suitable to evaluate the behavior of policies for longer periods.

Generally, it will be very interesting to think about possibilities to analyze more involved online algorithms, e. g., reoptimization algorithms. The challenge we face here is that such algorithms are typically computationally expensive. Thus, our approach can most likely not be employed directly as the online algorithm is required to solve many instances in the column generation algorithm. Exploiting warm-start techniques for the computations may be beneficial in this context since consecutive states are usually very similar. That is, in the case of a reoptimization algorithm, many auxiliary offline optimization problems to be solved are almost identical.

Bibliography

[AFG05] Susanne Albers, Lene M. Favrholdt, and Oliver Giel. On paging with locality of reference. *J. Comput. System Sci.*, 70(2):145–175, 2005. [2]

[AKR98] Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. The online transportation problem: Competitive scheduling of elevators. Report 98–34, ZIB, 1998. `opus.kobv.de/zib/volltexte/1998/378/`. [123]

[AKR00] Norbert Ascheuer, Sven O. Krumke, and Jörg Rambau. Online dial-a-ride problems: Minimizing the completion time. In *Proceedings of the 17th International Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science*, pages 639–650. Springer, 2000. [121, 122]

[Alb03] Susanne Albers. Online algorithms: A survey. *Math. Programming*, 97:3–26, 2003. [3]

[Bau01] Heinz Bauer. *Measure and Integration Theory.* De Gruyter, 1st edition, 2001. [15]

[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. [29]

[BC89] Dimitri R. Bertsekas and David A. Castañon. Adaptive aggregation for infinite horizon dynamic programming. *IEEE Trans. Automat. Control*, 34(6):589–598, 1989. [29, 190]

[BDBK+90] Shai Ben-David, Allan Borodin, Richard M. Karp, Gabor Tardos, and Avi Wigderson. On the power of randomization in online algorithms. In *Algorithmica*, pages 379–386, 1990. [108]

[Ber01] Dimitri P. Bertsekas. *Dynamic Programming and Optimal Control*, volume 1 and 2. Athena Scientific, Belmont, 2nd edition, 2001. [3, 9, 10, 15, 17, 60]

[BEY98] Allan Borodin and Ran El-Yaniv. *Online Computation and Competitive Analysis.* Cambridge University Press, 1998. [1, 3]

[BF07] Joan Boyar and Lene M. Favrholdt. The relative worst order ratio for online algorithms. *ACM Trans. Algorithms*, 3(2):Article 22, 2007. [3]

[Bie87] K. J. Bierth. An expected average reward criterion. *Stochastic Process. Appl.*, 26, 1987. [31]

[BIRS95] Allan Borodin, Sandy Irani, Prabhakar Raghavan, and Baruch Schieber. Competitive paging with locality of reference. *J. Comput. System Sci.*, 50(2):244–258, 1995. [2]

[Bla62] David Blackwell. Discrete dynamic programming. *Annals of Mathematical Statistics*, 33(2):719–726, 1962. [32]

[BLMS+06] Luca Becchetti, Stefano Leonardi, Alberto Marchetti-Spaccamela, Guido Schäfer, and Tjark Vredeveld. Average case and smoothed competitive analysis for the multi-level feedback algorithm. *Math. Oper. Res.*, 31(1):85–108, 2006. [3]

[BLS92] Allan Borodin, Nathan Linial, and Michael E. Saks. An optimal online algorithm for metrical task systems. *J. ACM*, 39(4):745–763, 1992. [2]

[BT96] Dimitri P. Bertsekas and John N. Tsitsiklis. *Neuro-Dynamic Programming*, volume 1. Athena Scientific, Belmont, 1st edition, 1996. [27, 28, 29]

[CB98] Robert H. Crites and Andrew G. Barto. Elevator group control using multiple reinforcement learning agents. *Machine Learning*, 33(2–3):235–262, 1998. [120, 127, 188]

[CGS97] Riccardo Cambini, Giorgio Gallo, and Maria Grazia Scutellà. Flows on hypergraphs. *Mathematical Programming*, 78:195–217, 1997. [27]

[Clo70] Gordon David Closs. *The computer control of passenger traffic in large lift systems.* PhD thesis, Victoria University of Manchester, 1970. [120]

[CSHY80] E. G. Coffman, Jr., K. So, M. Hofri, and A. C. Yao. A stochastic model of bin-packing. *Information and Control*, 44:105–115, 1980. [3]

[DDS05] Guy Desaulniers, Jacques Desrosiers, and Marius M. Solomon, editors. *Column generation.* GERAD 25th anniversary series. Springer, 2005. [59]

[d'E63] F. d'Epenoux. A probabilistic production and inventory problem. *Management Science*, 10(1):98–108, 1963. [21]

[dFV03] Daniela P. de Farias and Benjamin Van Roy. The linear programming approach to approximate dynamic programming. *Operations Research*, 51(6):850–865, 2003. [21, 28, 29]

[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. [21, 29]

[dG60] Guy T. de Ghellinck. Les problèmes de décisions séquentielles. *Cahiers Centre d'Etudes Recherche Opérationnelle*, 2:161–179, 1960. [24]

[dGE67] Guy T. de Ghellinck and Gary D. Eppen. Linear programming solutions for separable markovian decision problems. *Management Science*, 13(5):371–394, January 1967. [23]

[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. [29, 58, 105]

[DLO05] Reza Dorrigiv and Alejandro López-Ortiz. A survey of performance measures for on-line algorithms. *SIGACT News*, 36(3):67–81, 2005. [3]

[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. [123]

[FS02] Eugene A. Feinberg and Adam Shwartz, editors. *Handbook of Markov Decision Processes: Methods and Applications*. Kluwer Academic Publishers, 2002. [3, 9, 17, 18, 19, 22, 23, 32]

[FW98] Amos Fiat and Gerhard J. Woeginger, editors. *Online Algorithms: The State of the Art*, volume 1442 of *Lecture Notes in Computer Science*. Springer, 1998. [1]

[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/`. [6, 120, 123, 182]

[GLNP93] Giorgio Gallo, Giustino Longo, Sang Nguyen, and Stefano Pallottino. Directed hypergraphs and applications. *Discrete Applied Mathematics*, 42:177–201, 1993. [25]

[Hau99] Dietrich Hauptmeier. Online algorithms for transport systems. Master's thesis, Technische Universität Berlin, 1999. [121, 123, 182]

[Hei05] Stefan Heinz. Policies for online target date assignment problems: Competitive analysis versus expected performance. Master's thesis, Technische Universität Berlin, 2005. [7, 161]

[HKM+05] Stefan Heinz, Sven O. Krumke, Nicole Megow, Jörg Rambau, Andreas Tuchscherer, and Tjark Vredeveld. The online target date assignment problem. In Thomas Erlebach and Giuseppe Persiano, editors, *Proceedings of the 3rd Workshop on Approximation and Online Algorithms*, volume 3879 of *Lecture Notes in Computer Science*, pages 230–243, 2005. [6, 110, 116, 117]

[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/`. [5, 51, 138]

[HKR00] Dietrich Hauptmeier, Sven O. Krumke, and Jörg Rambau. The online Dial-a-Ride problem under reasonable load. In *CIAC 2000*, volume 1767 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2000. [110, 123]

[HKT09] Benjamin Hiller, Torsten Klug, and Andreas Tuchscherer. Improving the performance of elevator systems using exact reoptimization algorithms. In *Proceedings of MAPSP*, 2009. [120]

[HKT10] Benjamin Hiller, Torsten Klug, and Andreas Tuchscherer. Improved destination call elevator control algorithms for up peak traffic. In *Operations Research Proceedings 2011*. Springer, 2010. to appear. [120]

[HT08] Benjamin Hiller and Andreas Tuchscherer. Real-time destination-call elevator group control on embedded microcontrollers. In *Operations Research Proceedings 2007*. Springer, 2008. [120]

[HV08] Benjamin Hiller and Tjark Vredeveld. Probabilistic analysis of online bin coloring algorithms via stochastic comparison. In *Proceedings of the 16th Annual European Symposium on Algorithms*, volume 5193 of *Lecture Notes in Computer Science*, pages 528–539, 2008. [112, 114]

[ILO] IBM ILOG. CPLEX. `http://www.ilog.com/products/cplex/`. [103, 138]

[JMRW92] Robert G. Jeroslow, Kipp Martin, Ronald L. Rardin, and Jinchang Wang. Gainfree leontief substitution flow problems. *Mathematical Programming*, 57:375–414, May 1992. [27]

[KdPSR08] Sven Oliver Krumke, Willem E. de Paepe, Leen Stougie, and Jörg Rambau. Bincoloring. *Theoret. Comput. Sci.*, 407(1-3):231–241, 2008. [6, 110, 111, 112]

[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. [29, 57]

[MHK+98] Nicolas Meuleau, Milos Hauskrecht, Kee-Eung Kim, Leonid Peshkin, Leslie Pack Kaelbling, Thomas Dean, and Craig Boutilier. Solving very large weakly coupled Markov decision processes. In *In Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pages 165–172, 1998. [29]

[NK06] Lars Relund Nielsen and Anders Ringgaard Kristensen. Finding the k best policies in a finite-horizon Markov decision process. *European Journal of Operational Research*, 175(2):1164–1179, December 2006. [25]

[Old01] Jeffrey D. Oldham. Combinatorial approximation algorithms for generalized flow problems. *J. Algorithms*, 38(1):135–169, 2001. [71]

[Pad99] Manfred W. Padberg. *Linear Optimization and Extensions: Algorithms and Combinatorics*. Springer, 2nd edition, 1999. [25]

[Pla06] Robert Plato. *Numerische Mathematik kompakt*. Vieweg, 3rd edition, 2006. [23, 40]

[Pow07] Warren B. Powell. *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. John Wiley and Sons, Inc., Hoboken, New Jersey, 1st edition, 2007. [4, 27, 29]

[Put05] Martin L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley and Sons, Inc., Hoboken, New Jersey, 2nd edition, 2005. [3, 4, 9, 15, 17, 20, 30, 32]

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, 1st edition, 1998. [27, 29]

[Sch90] Jordis Schröder. Advanced dispatching: Destination hall calls + instant car-to-call assignments: M10. *Elevator World*, pages 40–46, March 1990. [120]

[SS85] Paul J. Schweitzer and Abraham Seidmann. Generalized polynomial approximations in markov decision processes. *Journal of Mathematical Analysis and Applications*, 110:568–582, 1985. [21, 29]

[SS06] Alexander Souza and Angelika Steger. The expected competitive ratio for weighted completion time scheduling. *Theory of Computing Systems*, 39:121–136, 2006. [3]

[SSS06] Mark Scharbrodt, Thomas Schickinger, and Angelika Steger. A new average case analysis for completion time scheduling. *J. ACM*, pages 121–146, 2006. [3]

[ST85] Daniel Dominic Sleator and Robert Endre Tarjan. Amortized efficiency of list update and paging rules. *Communications of the ACM*, 28(2):202–208, 1985. [1]

[SY94] Satinder P. Singh and Richard C. Yee. An upper bound on the loss from approximate optimal-value functions. *Machine Learning*, 16(3):227–233, 1994. [29]

[Tor98] Eric Torng. A unified analysis of paging and caching. *Algorithmica*, 20(2):175–200, 1998. [2]

[Van06] Benjamin Van Roy. Performance loss bounds for approximate value iteration with state aggregation. *Mathematics of Operations Research*, 31(2):234–244, 2006. [29]

[Ye05] Yinyu Ye. A new complexity result on solving the markov decision problem. *Mathematics of Operations Research*, 30(3):733–749, 2005. [27]