Adaptive Resource Control

Machine Learning Approaches to Resource Allocation
in Uncertain and Changing Environments

Ph.D. Thesis

Balázs Csanád Csáji

Supervisor: László Monostori, D.Sc.

Faculty of Informatics (IK),
Eötvös Loránd University (ELTE)

Doctoral School of Computer Science,
Foundations and Methods in Informatics Ph.D. Program,
Chairman: Prof. János Demetrovics, Member of HAS

Computer and Automation Research Institute (SZTAKI),
Hungarian Academy of Sciences (HAS, MTA)

Budapest, Hungary, 2008

[...] καὶ τὴν ἀκρίβειαν μὴ ὁμοίως ἐν ἅπασιν ἐπιζητεῖν, ἀλλ᾽ ἐν ἑκάστοις κατὰ τὴν ὑποκειμένην ὕλην καὶ ἐπὶ τοσοῦτον ἐφ᾽ ὅσον οἰκεῖον τῇ μεθόδῳ. καὶ γὰρ τέκτων καὶ γεωμέτρης διαφερόντως ἐπιζητοῦσι τὴν ὀρθήν: ὃ μὲν γὰρ ἐφ᾽ ὅσον χρησίμη πρὸς τὸ ἔργον, ὃ δὲ τί ἐστιν ἢ ποῖόν τι: θεατὴς γὰρ τἀληθοῦς. τὸν αὐτὸν δὴ τρόπον καὶ ἐν τοῖς ἄλλοις ποιητέον, ὅπως μὴ τὰ πάρεργα τῶν ἔργων πλείω γίνηται. (Aristotle, Nicomachean Ethics, 1098a; based on: Ingram Bywater, editor, Oxford, Clarendon Press, 1894)

[...] *we must not look for equal exactness in all departments of study, but only such as belongs to the subject matter of each, and in such a degree as is appropriate to the particular line of enquiry. A carpenter and a geometrician both try to find a right angle, but in different ways; the former is content with that approximation to it which satisfies the purpose of his work; the latter, being a student of truth, seeks to find its essence or essential attributes. We should therefore proceed in the same manner in other subjects also, and not allow side issues to outbalance the main task in hand.* (Aristotle in 23 Volumes, Vol. 19, translated by Harris Rackham, Harvard University Press, 1934)

Declaration

Herewith I confirm that all of the research described in this dissertation is my own original work and expressed in my own words. Any use made within it of works of other authors in any form, e.g., ideas, figures, text, tables, are properly indicated through the application of citations and references. I also declare that no part of the dissertation has been submitted for any other degree — either from the Eötvös Loránd University or another institution.

Balázs Csanád Csáji
Budapest, April 2008

Acknowledgments

Though, the thesis is my own work, I received many support from my colleagues and family. Without them, the dissertation would not be the same. I would like to take the opportunity to express my gratitude here to all who helped and encouraged me during my studies.

First of all, I want to thank those people that had a direct influence on my thesis. These include first and foremost my supportive supervisor, László Monostori, but also the people whom I have collaborated with at the Engineering and Management Intelligence (EMI) Laboratory of the Computer and Automation Research Institute (SZTAKI).

Furthermore, I warmly thank Csaba Szepesvári for the many helpful discussions on Markov decision processes. I am very grateful to László Gerencsér, as well, from whom I learned a lot about stochastic models. I am also thankful for expanding my knowledge on machine learning to László Györfi. Finally, the first researcher who motivated my interest in artificial intelligence research during my graduate studies was András Lőrincz.

I am also grateful for the Ph. D. scholarship that I received from the Faculty of Informatics (IK) of the Eötvös Loránd University (ELTE) and, later, for the young researcher scholarship of the Hungarian Academy of Sciences (HAS, MTA). I greatly acknowledge the contribution of SZTAKI, as well, where I performed the research presented in the dissertation.

Last but not least, I am very thankful for the support and encouragement of my parents and family, especially, for the continuous help and care of my wife, Hildegard Anna Stift.

Abstract

The dissertation aims at studying *resource allocation problems* (RAPs) in *uncertain* and *changing* environments. In order to do this, first a brief introduction to the motivations and classical RAPs is given in Chapter 1, followed by a section on Markov decision processes (MDPs) which constitute the basis of the approach. The core of the thesis consists of two parts, the first deals with *uncertainties*, namely, with stochastic RAPs, while the second studies the effects of *changes* in the environmental dynamics on learning algorithms.

Chapter 2, the first core part, investigates stochastic RAPs with scarce, reusable resources and non-preemtive, interconnected tasks having temporal extensions. These RAPs are natural generalizations of several standard resource management problems, such as scheduling and transportation ones. First, reactive solutions are considered and defined as policies of suitably reformulated MDPs. It is highlighted that this reformulation has several favorable properties, such as it has finite state and action spaces, it is acyclic, hence all policies are proper and the space of policies can be safely restricted. Proactive solutions are also proposed and defined as policies of special partially observable MDPs. Next, reinforcement learning (RL) methods, such as fitted Q-learning, are suggested for computing a policy. In order to compactly maintain the value function, two representations are studied: hash tables and support vector regression (SVR), particularly, ν-SVRs. Several additional improvements, such as the application of rollout algorithms in the initial phases, action space decomposition, task clustering and distributed sampling are investigated, as well.

Chapter 3, the second core part, studies the possibility of applying value function based RL methods in cases when the environment may change over time. First, theorems are presented which show that the optimal value function and the value function of a fixed control policy Lipschitz continuously depend on the immediate-cost function and the transition-probability function, assuming a discounted MDP. Dependence on the discount factor is also analyzed and shown to be non-Lipschitz. Afterwards, the concept of (ε, δ)-MDPs is introduced, which is a generalization of MDPs and ε-MDPs. In this model the transition-probability function and the immediate-cost function may vary over time, but the changes must be asymptotically bounded. Then, learning in changing environments is investigated. A general relaxed convergence theorem for stochastic iterative algorithms is presented and illustrated through three classical examples: value iteration, Q-learning and TD-learning.

Finally, in Chapter 4, results of numerical experiments on both benchmark and industry-related problems are shown. The effectiveness of the proposed adaptive resource allocation approach as well as learning in presence of disturbances and changes are demonstrated.

Contents

Chapter 1

Introduction

Information technology has been making an explosion-like progress since the middle of the past century. However, as computer science broke out from laboratories and classrooms and started to deal with "real world" problems, it had to face major difficulties. Namely, in practise, we mostly have only *incomplete* and *uncertain* information on the system and the environment that we must work with, additionally, they may even *change* dynamically, the problem may be *non-stationary*. Moreover, we also have to face *complexity* issues, viz., even if we deal with static, highly simplified and abstract problems and it can be known that the solution exists and can be attained in finitely many steps, the problem could still be *intractable*, viz., we might not have enough computation power (or even enough storage space) to achieve it in practise, as this is the case, e.g., with many NP-hard problems.

One way to overcome these difficulties is to apply *machine learning* techniques. It means designing systems which can *adapt* their behavior to the current state of the environment, *extrapolate* their knowledge to the unknown cases and learn how to *optimize* the system. These approaches often use *statistical* methods and satisfy with *approximate*, *suboptimal* but *tractable* solutions concerning both computational demands and storage space.

The importance of *learning* was recognized even by the founders of computer science. It is well known, e.g., that John von Neumann (1948) was keen on *artificial life* and, besides many other things, designed *self-organizing* automata. Alan Turing (1950) can be another example, who in his famous paper, which can be treated as one of the starting articles of *artificial intelligence* research, wrote that instead of designing extremely complex and large systems, we should design programs that can learn how to work efficiently by themselves.

In the dissertation we consider an important problem with many practical applications, which has all the difficulties mentioned in the previous parts, namely: *resource allocation*. In this chapter, first, a brief introduction to resource allocation is given followed by a section on Markov decision processes (MDPs), since they constitute the basis of the presented approach. At the end of Chapter 1 the main contributions of the dissertation are summarized. Chapter 2 deals with *uncertainties* concerning resource allocation, namely, it defines a generalized framework for stochastic problems, then, an MDP based reformulation is given and efficient solution methods are suggested applying various machine learning techniques, such as reinforcement learning, support vector regression and clustering. Chapter 3 studies the effects of environmental *changes* on learning algorithms. First, different value function bounds for environmental changes are presented followed by an analysis of stochastic iterative algorithms in a special class of non-stationary environments. Finally, in Chapter 4 results of numerical experiments on benchmark and industry-related data are presented.

1.1 Resource Allocation

Resource allocation problems (RAPs) are of high practical importance, since they arise in many diverse fields, such as manufacturing production control (e.g., production scheduling), warehousing (e.g., storage allocation), fleet management (e.g., freight transportation), personnel management (e.g., in an office), scheduling of computer programs (e.g., in massively parallel GRID systems), managing a construction project or controlling a cellular mobile network. RAPs are also central to management science (Powell and Van Roy, 2004). In the thesis we consider optimization problems that include the assignment of a finite set of reusable resources to non-preemtive, interconnected tasks that have stochastic durations and effects. Our main objective in the thesis is to investigate efficient decision-making processes which can deal with the allocation of scarce resources over time with a goal of optimizing the objectives. For "real world" applications, it is important that the solution should be able to deal with large-scale problems and handle environmental changes, as well.

1.1.1 Industrial Motivations

One of our main motivations for investigating RAPs is to enhance manufacturing production control. Regarding contemporary manufacturing systems, difficulties arise from unexpected tasks and events, non-linearities, and a multitude of interactions while attempting to control various activities in dynamic shop floors. Complexity and uncertainty seriously limit the effectiveness of conventional production control approaches (e.g., deterministic scheduling). In the thesis we apply *mathematical programming* and *machine learning* (ML) techniques to achieve the *suboptimal control* of a generalized class of stochastic RAPs, which can be vital to an *intelligent manufacturing system* (IMS). The term of IMS can be attributed to a tentative forecast of Hatvany and Nemes (1978). In the early 80s IMSs were outlined as the next generation of manufacturing systems that utilize the results of *artificial intelligence* research and were expected to solve, within certain limits, unprecedented, unforeseen problems on the basis of even incomplete and imprecise information. Naturally, the applicability of the different proposed solutions to RAPs are not limited to industrial problems.

1.1.2 Curse(s) of Dimensionality

Different kinds of RAPs have a huge number of exact and approximate solution methods, e.g., (see Pinedo, 2002) in the case of scheduling problems. However, these methods primarily deal with the static (and often strictly deterministic) variants of the various problems and, mostly, they are not aware of uncertainties and changes. Special (deterministic) RAPs which appear in the field of *combinatorial optimization*, e.g., the traveling salesman problem (TSP) or the job-shop scheduling problem (JSP), are *strongly NP-hard* and, moreover, they do not have any good polynomial-time approximation, either (Lawler et al., 1993; Lovász and Gács, 1999). In the stochastic case, RAPs can be often formulated as *Markov decision processes* (MDPs) and by applying *dynamic programming* (DP) methods, in theory, they can be solved *optimally*. However, due to the phenomenon that was named *curse of dimensionality* by Bellman, these methods are highly intractable in practice. The "curse" refers to the *combinatorial explosion* of the required computation as the size of the problem increases. Some authors, e.g., Powell and Van Roy (2004), talk about even three types of curses concerning DP algorithms. This has motivated *approximate* approaches that require a more tractable computation, but often yield *suboptimal* solutions (Bertsekas, 2005).

1.1.3 Related Literature

It is beyond our scope to give a general overview on different solutions to RAPs, hence, we only concentrate on the part of the literature that is closely related to our approach. Our solution belongs to the class of *approximate dynamic programming* (ADP) algorithms which constitute a broad class of discrete-time control techniques. Note that ADP methods that take an actor-critic point of view are often called *reinforcement learning* (RL).

Zhang and Dietterich (1995) were the first to apply an RL technique for a special RAP. They used the $TD(\lambda)$ method with iterative repair to solve a static scheduling problem, namely, the NASA space shuttle payload processing problem. Since then, a number of papers have been published that suggested using RL for different RAPs. The first reactive (closed-loop) solution to scheduling problems using ADP algorithms was briefly described in (Schneider et al., 1998). Riedmiller and Riedmiller (1999) used a *multilayer perceptron* (MLP) based neural RL approach to learn local heuristics. Aydin and Öztemel (2000) applied a modified version of Q-learning to learn dispatching rules for production scheduling. In (Csáji et al., 2003, 2004; Csáji and Monostori, 2005b,a, 2006a,b; Csáji et al., 2006) multi-agent based versions of ADP techniques were used for solving dynamic scheduling problems.

Powell and Van Roy (2004) presented a formal framework for RAPs and they applied ADP to give a general solution to their problem. Later, a parallelized solution to the previously defined problem was given by Topaloglu and Powell (2005). Note that our RAP framework, presented in Chapter 2, differs from the one in (Powell and Van Roy, 2004), since in our system the goal is to accomplish a set of tasks that can have widely different stochastic durations and precedence constraints between them, while the approach of Powell and Van Roy (2004) concerns with satisfying many similar demands arriving stochastically over time with demands having unit durations but not precedence constraints.

Recently, *support vector machines* (SVMs) were applied by Gersmann and Hammer (2005) to improve iterative repair (local search) strategies for *resource constrained project scheduling problems* (RCPSPs). An agent-based resource allocation system with MDP-induced preferences was presented in stepDolgov2006. Beck and Wilson (2007) gave proactive solutions for job-shop scheduling problems based on the combination of *Monte Carlo simulation*, solutions of the associated deterministic problem, and either constraint programming or tabu-search. Finally, the effects of environmental changes on the convergence of reinforcement learning algorithms was theoretically analyzed by Szita et al. (2002).

1.1.4 Classical Problems

In this section we give a brief introduction to RAPs through three classical problems: job-shop scheduling, traveling salesman and container loading. All of these problems are known to be NP-hard. Throughout the thesis we will apply them to demonstrate our ideas.

Job-Shop Scheduling

First, we consider the classical *job-shop scheduling problem* (JSP) which is a standard deterministic RAP (Pinedo, 2002). We have a set of jobs, $\mathcal{J} = \{J_1, \ldots, J_n\}$, to be processed through a set of machines, $\mathcal{M} = \{M_1, \ldots, M_k\}$. Each $j \in \mathcal{J}$ consists of a sequence of n_j tasks, for each task $t_{ji} \in \mathcal{T}$, where $i \in \{1, \ldots, n_j\}$, there is a machine $m_{ji} \in \mathcal{M}$ which can process the task, and a processing time $p_{ji} \in \mathbb{N}$. The aim of the optimization is to find a *feasible schedule* which minimizes a given performance measure. A solution, i.e., a schedule, is a suitable "task to starting time" assignment, Figure 1.1 presents an example schedule. The concept of "feasibility" will be defined in Chapter 2. In the case of JSP a feasible schedule can be associated with an ordering of the tasks, i.e., the order in which they will be executed on the machines. There are many types of performance measures available for JSP, but probably the most commonly applied one is the maximum completion time of the tasks, also called "makespan". In case of applying makespan, JSP can be interpreted as the problem of finding a schedule which completes all tasks in every job as soon as possible.

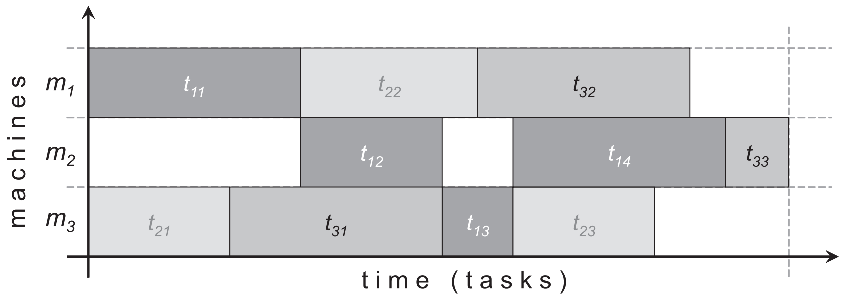


Figure 1.1: A possible solution to JSP, presented in a Gantt chart. Tasks having the same color belong to the same job and should be processed in the given order. The vertical gray dotted line indicates the maximum completion time of the tasks.

Later, we will study an extension of JSP, the *flexible job-shop scheduling problem* (FJSP). In FJSP the machines may be interchangeable, i.e., there may be tasks that can be executed on several machines. In this case the processing times are given by a *partial* function, $p : \mathcal{M} \times \mathcal{T} \hookrightarrow \mathbb{N}$. Recall that a partial function, denoted by "$\hookrightarrow$", is a binary relation that associates the elements of its domain set with *at most* one element of its range set.

An even more general version of JSP, which is often referred to as *resource constrained project scheduling problem* (RCPSP), arises when the tasks may require several resources, such as machines and workers, simultaneously, in order to be executed (Pinedo, 2002).

Traveling Salesman

One of the basic transportation problems is the famous *traveling salesman problem* (TSP) that can be stated as follows. Given a number of cities and the costs of travelings between them, which is the least-cost round-trip route that visits each city exactly once and then returns to the starting city (Papadimitriou, 1994). Several variants of TSP are known, here we present one of the standard versions. It can be formally characterized by a connected, undirected, edge-weighted graph $G = \langle V, E, w \rangle$, where the components are as follows. The vertex set, $V = \{1, \ldots, n\}$, is corresponding to the set of "cities", $E \subseteq V \times V$ is the set of edges which represents the "roads" between the cities, and function $w : E \to \mathbb{N}$ defines the weights of the edges: the durations of the trips. The aim of the optimization is to find a *Hamilton-circuit* with the smallest possible weight. Note that a Hamilton-circuit is a graph cycle that starts at a vertex, passes through every vertex exactly once and, finally, returns to the starting vertex. Take a look at Figure 1.2 for an example Hamilton-circuit.

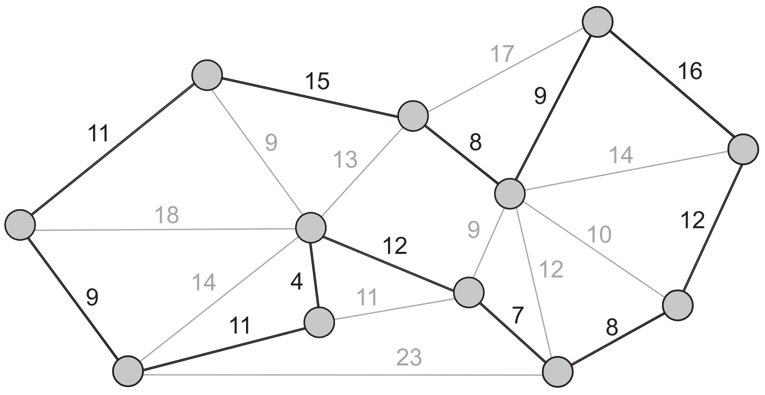


Figure 1.2: A possible solution to TSP, a path in the graph. The black edges constitute a Hamilton-circuit in the given connected, undirected, edge-weighted graph.

Container Loading

Our final classical example is the *container loading problem* (CLP) which is an inventory management problem (Davies and Bischoff, 1999). CLP is related to the bin packing problem with the objective of high volumetric utilization. It involves the placement of a set of items in a container. In practical applications there are a number of requirements concerning container loading, such as stacking conditions, cargo stability, visibility and accessibility considerations. Now, as a simplification, we concentrate only on the weight distribution of the loaded container, focusing on the location of the *center of gravity* (CoG). The exact requirements concerning CoG depend on the specific application, especially on the means of transport. In aircraft loading or loading containers lifted by cranes, for example, the CoG has to be located in the center of the container. In contrast, in road transport, it is often preferred to have the CoG above the axles of the vehicle (Kovács and Beck, 2007).

Now, we describe the problem of loading homogeneous, box-shaped items into a rectangular container. We assume that the rotation of the items is disallowed. The location of the CoG can be constrained to an arbitrary rectangular region of the container. For simplicity, we present a two-dimensional variant of the problem, although it is straightforward to extend the model to higher dimensions. We have a set of two-dimensional boxes, $B_1, \dots, B_n$, to be placed in a rectangular container of length L and width W. Each box B_i is characterized by its length a_i, width b_i and weight w_i. The location of B_i is represented by two parameters, x_i and y_i, which describe the coordinate of the "south-west" corner of the box. Since the boxes are homogeneous, the CoG of the cargo, $\langle x^*, y^* \rangle$, can be computed as follows

$$\langle\, x^*, y^* \,\rangle = \left\langle\, \frac{\sum_{i=1}^n w_i(x_i + a_i/2)}{\sum_{i=1}^n w_i},\; \frac{\sum_{i=1}^n w_i(y_i + b_i/2)}{\sum_{i=1}^n w_i} \,\right\rangle .$$

The objective of CLP is to find a placement of the boxes in the container such that the CoG of the cargo is located in a given rectangular area, more precisely, $x^*_{min} \le x^* \le x^*_{max}$ and $y^*_{min} \le y^* \le y^*_{max}$ should hold. Take a look at Figure 1.3 for an example solution to CLP.

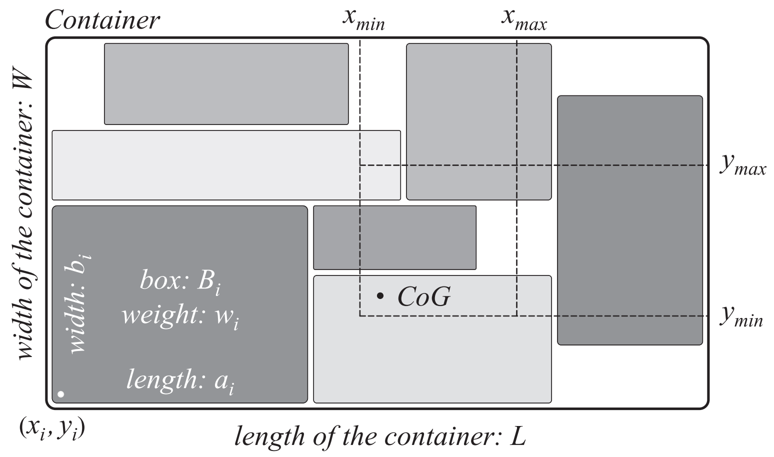


Figure 1.3: A possible solution to CLP. The objective is to place the given boxes in the container in a way that the constraints on the center of gravity are satisfied.

Assuming that all of the parameters are natural numbers, this problem could be transformed into a scheduling problem which allows tasks with multiple resource requirements (Kovács and Beck, 2007). In order to illustrate this reformulation, consider the following similarities between scheduling and container loading problems. The container, e.g., corresponds to the hull of the schedule, defined by the scheduling horizon (horizontal axis) and the resource capacity (vertical axis). Boxes can be associated with tasks. Box length corresponds to task duration and box width can be seen as the resource requirements of the task. Finally, the physical weight of the box can be associated with the weight of the task. Regarding our RAP framework, the weights can be included in the performance measure.

1.2 Markov Decision Processes

Stochastic control problems are often modeled by MDPs that constitute a fundamental tool for computational learning theory. The theory of MDPs has grown extensively since Bellman introduced the discrete stochastic variant of the optimal control problem in 1957. These kinds of stochastic optimization problems have great importance in diverse fields, such as engineering, manufacturing, medicine, finance or social sciences. Several solution methods are known, e.g., from the field of [neuro-]dynamic programming (NDP) or reinforcement learning (RL), which compute or approximate the optimal control policy of an MDP. These methods succeeded in solving many different problems, such as transportation and inventory control (Van Roy et al., 1996), channel allocation (Singh and Bertsekas, 1997), robotic control (Kalmár et al., 1998), logical games and problems from financial mathematics. Many applications of RL and NDP methods are also considered by the textbooks of Bertsekas and Tsitsiklis (1996), Sutton and Barto (1998) and Feinberg and Shwartz (2002).

This section contains the basic definitions, the applied notations and some preliminaries. MDPs are of special interest for us, since they constitute the fundamental theory of our approach. In Chapter 2, e.g., generalized stochastic RAPs are presented and, in order to apply machine learning techniques to solve them, they are reformulated as MDPs. Later, in Chapter 3, environmental changes are investigated within the concept of MDPs, as well.

Definition 1 *By a (finite, discrete-time, stationary, fully observable) Markov decision process (MDP) we mean a stochastic system characterized by a 6-tuple* $\langle \mathbb{X}, \mathbb{A}, \mathcal{A}, p, g, \alpha \rangle$*, where the components are as follows:* $\mathbb{X}$ *is a finite set of discrete states and* $\mathbb{A}$ *is a finite set of control actions. Mapping* $\mathcal{A} : \mathbb{X} \to \mathcal{P}(\mathbb{A})$ *is the availability function that renders each state a set of actions available in that state where* $\mathcal{P}$ *denotes the power set. The transition-probability function is given by* $p : \mathbb{X} \times \mathbb{A} \to \Delta(\mathbb{X})$*, where* $\Delta(\mathbb{X})$ *is the space of probability distributions over* $\mathbb{X}$*. Let* $p(y \,|\, x, a)$ *denote the probability of arrival at state* y *after executing action* $a \in \mathcal{A}(x)$ *in state* x*. The immediate-cost function is defined by* $g : \mathbb{X} \times \mathbb{A} \to \mathbb{R}$*, where* $g(x, a)$ *is the cost of taking action* a *in state* x*. Finally, constant* $\alpha \in [0, 1]$ *denotes the discount rate. If* $\alpha = 1$*, then the MDP is called undiscounted, otherwise it is called discounted.*

An interpretation of an MDP can be given, which viewpoint is often taken in RL, if we consider an *agent* that acts in an uncertain environment. The agent receives information about the state of the environment, x, at each state x the agent is allowed to choose an action $a \in \mathcal{A}(x)$. After the action is selected, the environment moves to the next state according to the probability distribution $p(x, a)$ and the decision-maker collects its one-step cost, $g(x, a)$. The aim of the agent is to find an *optimal behavior* (policy), such that applying this strategy minimizes the expected cumulative costs over a finite or infinite horizon.

A *stochastic shortest path* (SSP) problem is a special MDP in which the aim is to find a control policy such that reaches a pre-defined terminal state starting from a given initial state, additionally, minimizes the expected total costs of the path, as well. A policy is called *proper* if it reaches the terminal state with probability one. A usual assumption when dealing with SSP problems is that *all policies are proper*, which is abbreviated as APP.

It is possible to extend the theory to more general state and action spaces, but at the expense of increased mathematical complexity. Finite state and action sets are mostly sufficient for digitally implemented controls and, therefore, we restrict ourselves to this case.

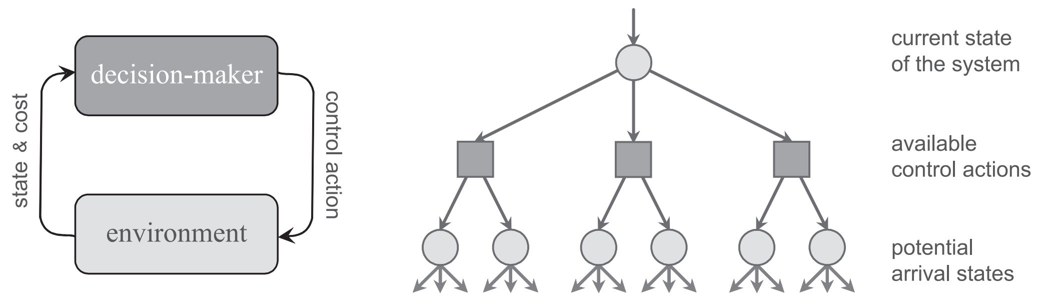


Figure 1.4: Markov decision processes - the interaction of the decision-maker and the uncertain environment (left); the temporal progress of the system (right).

1.2.1 Control Policies

The behavior of the learning agent at a given time is defined by a policy. Roughly speaking, a (stationary, Markovian) control policy determines the action to take in each state.

Definition 2 *A deterministic policy, $\pi : \mathbb{X} \to \mathbb{A}$, is simply a function from states to control actions. A randomized policy, $\pi : \mathbb{X} \to \Delta(\mathbb{A})$, is a function from states to probability distributions over actions. We denote the probability of executing action a in state x by $\pi(x)(a)$ or, for short, by $\pi(x, a)$. Unless indicated otherwise, we consider randomized policies.*

For any $\widetilde{x}_0 \in \Delta(\mathbb{X})$ initial probability distribution of the states, the transition probabilities p together with a control policy π completely determine the progress of the system in a stochastic sense, namely, they define a *homogeneous Markov chain* on $\mathbb{X}$,

$$\widetilde{x}_{t+1} = P(\pi)\widetilde{x}_t,$$

where $\widetilde{x}_t$ is the state probability distribution vector of the system at time t, and $P(\pi)$ denotes the probability transition matrix induced by control policy π, defined as follows

$$[P(\pi)]_{x,y} = \sum_{a \in \mathbb{A}} p(y \,|\, x, a)\, \pi(x, a).$$

The *Kolmogorov extension theorem* guarantees that any initial state x_0 and any policy π define a stochastic process (sequence) $x_0, a_0, x_1, a_1, \dots$ (Feinberg and Shwartz, 2002).

1.2.2 Value Functions

The performance of a control policy in the long run is specified by its the *value function.* The value of a state with respect to a given policy is, roughly, the total amount of cost an agent can expect to incur starting from that state and following the policy thereafter.

Definition 3 *The value or cost-to-go function of a policy π is a function from states to costs, $J^{\pi} : \mathbb{X} \to \mathbb{R}$. Function $J^{\pi}(x)$ gives the expected value of the cumulative (discounted) costs when the system is in state x and it follows policy π thereafter,*

$$J^{\pi}(x) = \mathbb{E}\left[\sum_{t=0}^{N} \alpha^t g(X_t, A_t^{\pi}) \,\middle|\, X_0 = x\right], \tag{1.1}$$

where X_t and A_t^{π} are random variables, A_t^{π} is selected according to control policy π and the distribution of X_{t+1} is $p(X_t, A_t^{\pi})$. The horizon of the problem is denoted by $N \in \mathbb{N} \cup \{\infty\}$. Unless indicated otherwise, we will always assume that the horizon is infinite, $N = \infty$.

Similarly to the definition of J^{π}, one can define *action-value* functions of control polices,

$$Q^{\pi}(x, a) = \mathbb{E}\left[\sum_{t=0}^{N} \alpha^t g(X_t, A_t^{\pi}) \,\middle|\, X_0 = x, A_0^{\pi} = a\right],$$

where the notations are the same as in equation (1.1). Action-value functions are especially important for model-free approaches, such as the classical Q-learning algorithm.

1.2.3 Bellman Equations

We saw that the agent aims at finding an *optimal* policy which minimizes the expected costs. In order to define optimal solutions, we also need a concept for *comparing* policies.

Definition 4 *We say that $\pi_1 \leq \pi_2$ if and only if $\forall x \in \mathbb{X} : J^{\pi_1}(x) \leq J^{\pi_2}(x)$. A control policy is (uniformly) optimal if it is less than or equal to all other control policies.*

There always exists at least one optimal policy (Sutton and Barto, 1998). Although there may be many optimal policies, they all share the same *unique* optimal cost-to-go function, denoted by J^*. This function must satisfy the Bellman optimality equation, $TJ^* = J^*$, where T is the *Bellman operator* (Bertsekas and Tsitsiklis, 1996), defined for all $x \in \mathbb{X}$, as

$$(TJ)(x) = \min_{a \in \mathcal{A}(x)} \Big[g(x, a) + \alpha \sum_{y \in \mathbb{X}} p(y \,|\, x, a) J(y)\Big]. \tag{1.2}$$

The *Bellman equation* for an arbitrary (stationary, Markovian, randomized) policy is

$$(T^{\pi} J)(x) = \sum_{a \in \mathcal{A}(x)} \pi(x, a) \Big[g(x, a) + \alpha \sum_{y \in \mathbb{X}} p(y \,|\, x, a) J(y)\Big],$$

where the notations are the same as in equation (1.2) and we also have $T^{\pi} J^{\pi} = J^{\pi}$.

Definition 5 *We say that function $f : \mathcal{X} \to \mathcal{Y}$, where $\mathcal{X}$, $\mathcal{Y}$ are normed spaces, is Lipschitz continuous if there exists a $\beta \geq 0$ such that $\forall\, x_1, x_2 \in \mathcal{X} : \|f(x_1) - f(x_2)\|_{\mathcal{Y}} \leq \beta \|x_1 - x_2\|_{\mathcal{X}}$, where $\|\cdot\|_{\mathcal{X}}$ and $\|\cdot\|_{\mathcal{Y}}$ denote the norm of $\mathcal{X}$ and $\mathcal{Y}$, respectively. The smallest such β is called the Lipschitz constant of f. Henceforth, assume that $\mathcal{X} = \mathcal{Y}$. If the Lipschitz constant $\beta < 1$, then the function is called a contraction. A mapping is called a pseudo-contraction if there exists an $x^* \in \mathcal{X}$ and a $\beta \geq 0$ such that $\forall\, x \in \mathcal{X}$, we have $\|f(x) - x^*\|_{\mathcal{X}} \leq \beta \|x - x^*\|_{\mathcal{X}}$.*

Naturally, every contraction mapping is also a pseudo-contraction, however, the opposite is not true. The pseudo-contraction condition implies that x^* is the fixed point of function f, namely, $f(x^*) = x^*$, moreover, x^* is unique, thus, f cannot have other fixed points.

It is known that the Bellman operator is a supremum norm contraction with Lipschitz constant α. In case we consider stochastic shortest path (SSP) problems, which arise if the MDP has an absorbing terminal (goal) state, then the Bellman operator becomes a pseudo-contraction in the weighted supremum norm (Bertsekas and Tsitsiklis, 1996).

1.2.4 Approximate Solutions

From a given value function J, it is straightforward to get a policy, e.g., by applying a *greedy* and deterministic policy (w.r.t. J) that always selects actions with minimal costs,

$$\pi(x) \in \operatorname*{arg\,min}_{a \in \mathcal{A}(x)} \Big[g(x,a) + \alpha \sum_{y \in \mathbb{X}} p(y \,|\, x, a) J(y) \Big].$$

MDPs have an extensively studied theory and there exist a lot of exact and approximate solution methods, e.g., value iteration, policy iteration, the Gauss-Seidel method, Q-learning, Q(λ), SARSA and TD(λ) - temporal difference learning (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998; Feinberg and Shwartz, 2002). Most of these reinforcement learning algorithms work by iteratively approximating the optimal value function.

If J is “close” to J^*, then the greedy policy with one-stage lookahead based on J will also be “close” to an optimal policy, as it was proven by Bertsekas and Tsitsiklis (1996):

Theorem 6 *Let M be a discounted MDP and J is an arbitrary value function. The value function of the greedy policy based on J is denoted by J^π. Then, we have*

$$\left\| J^\pi - J^* \right\|_\infty \leq \frac{2\,\alpha}{1-\alpha} \left\| J - J^* \right\|_\infty,$$

where $\|\cdot\|_\infty$ denotes the supremum norm, more precisely, $\|f\|_\infty = \sup\{|f(x)| : x \in dom(f)\}$. Moreover, there exists an $\varepsilon > 0$ such that if $\|J - J^*\|_\infty < \varepsilon$, then $J^* = J^\pi$.

Consequently, if we could obtain a good approximation of the optimal value function, then we immediately had a good control policy, as well, e.g., the greedy policy with respect to our approximate value function. Therefore, the main question for most RL approaches is that how a good approximation to the optimal value function could be achieved.

1.2.5 Partial Observability

In an MDP it is assumed that the agent is perfectly informed about the current state of the environment, which presupposition is often unrealistic. In *partially observable Markov decision processes* (POMDPs), which are well-known generalizations of MDPs, the decision-maker does not necessarily know the precise state of the environment: some observations are available to ground the decision upon, however, these information can be partial and noisy. Formally, a POMDP has all components of a (fully observable) MDP and, additionally, it has a finite *observation* set $\mathbb{O}$ and a function for the *observation probabilities* $q : \mathbb{X} \times \mathbb{A} \to \Delta(\mathbb{O})$.

The notation $p(z \mid x, a)$ shows the probability that the decision-maker receives observation z after executing action a in state x. Note that in POMDPs the availability function depends on the observations rather than the real states of the underlying MDP, $\mathcal{A} : \mathbb{O} \to \mathcal{P}(\mathbb{A})$.

Control policies of POMDPs are also defined on observations. Thus, a (non-Markovian) deterministic policy takes the form of $\pi : \mathbb{O}^* \to \mathbb{A}$, where $\mathbb{O}^*$ denotes the set of all finite sequences over $\mathbb{O}$. Respectively, randomized policies are defined as $\pi : \mathbb{O}^* \to \Delta(\mathbb{A})$.

An important idea in the theory of POMDPs is the concept of *belief states*, which are probability distributions over the states of the environment. They were suggested by Åström (1965) and they can be interpreted as the decision-maker's ideas about the current state. We denote the belief space by $\mathbb{B} = \Delta(\mathbb{X})$. The belief state is a *sufficient statistic* in the sense that the agent can perform as well based upon belief states as if it had access to the whole history of observations (Smallwood and Sondik, 1973). Therefore, a Markovian control policy based on belief states, $\pi_b : \mathbb{B} \to \Delta(\mathbb{A})$, as it was shown, can be as efficient as a non-Markovian control policy that applies all past observations, $\pi_o : \mathbb{O}^* \to \Delta(\mathbb{A})$.

Given a belief state b, a control action a and an observation z, the successor belief state $\tau(b, a, z) \in \mathbb{B}$ can be calculated by the *Bayes rule*, more precisely, as follows

$$\tau(b, a, z)(y) = \frac{\sum\limits_{x \in \mathbb{X}} p(z, y \mid x, a)\, b(x)}{p(z \mid b, a)}\,,$$

where $p(z, y \mid x, a) = p(z \mid y, a) \cdot p(y \mid x, a)$ and $p(z \mid b, a)$ can be computed by

$$p(z \mid b, a) = \sum_{x, y \in \mathbb{X}} p(z, y \mid x, a)\, b(x)\,.$$

It is known (Aberdeen, 2003) that with the concept of belief states, a POMDP can be transformed into a fully observable MDP. The resulting process is called the *belief state MDP*. The state space of the belief state MDP is $\mathbb{B}$, its action space is $\mathbb{A}$, and the transition-probabilities from any state b_1 to state b_2 after executing action a can be determined by

$$p(b_2 \mid b_1, a) = \begin{cases} p(z \mid b_1, a) & \text{if } b_2 = \tau(b_1, a, z) \text{ for some } z \\ 0 & \text{otherwise} \end{cases}$$

The immediate-cost function of the belief state MDP for all $b \in \mathbb{B}$, $a \in \mathbb{A}$ is given by

$$g(b, a) = \sum_{x \in \mathbb{X}} b(x)\, g(x, a),$$

consequently, the optimal cost-to-go function of the belief state MDP, denoted by $\tilde{J}^*$, is

$$\tilde{J}^*(b) = \min_{a \in \mathcal{A}(b)} \Big[g(b, a) + \alpha \sum_{z \in \mathbb{O}} p(z \mid b, a)\, \tilde{J}^*(\tau(b, a, z)) \Big].$$

Due to this reformulation, solving a POMDP, *in theory*, can be accomplished by solving the corresponding belief state MDP. However, usually it is hard to translate this approach into efficient solution methods. Some approximate solutions are considered by Aberdeen (2003).

1.3 Main Contributions

The main contributions and the new scientific results of the dissertation can be summarized in six points which can be organized in two thesis groups. The first group concerns with efficiently solving RAPs in presence of *uncertainties*, while the second contains results on managing *changes* in the environmental dynamics. It is expected to formulate the contributions in first-person singular form, in order to express that they are my own results.

1.3.1 Stochastic Resource Allocation

In Chapter 2 I study RAPs in presence of uncertainties. I also suggest machine learning based solution methods to handle them. My main contributions are as follows:

T 1.1 *I propose a formal framework for studying stochastic resource allocation problems with reusable resources and non-preemtive, interconnected tasks having temporal extensions. I provide a reformulation of it as a controlled Markov process and I show that this system is capable of handling both reactive and proactive solutions.*

I define a formal RAP which is a natural generalization of several standard resource management problems, such as scheduling, transportation and inventory management ones. I reformulate this general RAP as a stochastic shortest path (SSP) problem (a special MDP) having favorable properties, such as, it is acyclic, its state and action spaces are finite, all policies are proper and the space of control policies can be safely restricted. I define reactive solutions of stochastic RAPs as control policies of the reformulated problem. I also investigate proactive solutions and treat them as policies of the non-observable MDP corresponding to the reformulated MDP. I analyze the relation between the optimal cost-to-go of the reactive and the proactive solutions, as well. These results can be found in Section 2.1 of the dissertation.

T 1.2 *I suggest methods based on the combination of approximate dynamic programming, simulated annealing and either hash tables or kerner regression, in order to compute and represent reactive solutions. I confirm the effectiveness of this approach with results of numerical experiments on both benchmark and industry related problems.*

In order to compute a good approximation of an optimal policy, I suggest ADP methods, particularly, fitted Q-learning. Regarding value function representation, I study two approaches: hash tables and support vector regression (SVR), especially, ν-SVRs. In both cases, I define the inputs as numerical feature vectors. Since the problem to be faced is an SSP, I apply off-line learning after each episode. An episode consists of a state-action-cost trajectory, generated by simulation. Regarding controlling the ratio of exploration and exploitation during the simulation I apply the Boltzmann formula. These ideas are described in Sections 2.2.1 and 2.2.2 of the dissertation. I also present results of numerical experiments on both benchmark and industry-related data, in order to demonstrate the effectiveness of the approach. I measure the performance on hard benchmark flexible job-shop scheduling problems and I also demonstrate the scaling properties by experiments on a simulated factory producing mass-products. These experiments are presented in Sections 4.1.2 and 4.1.4.

T 1.3 *I provide further improvements based on rollout algorithms, action space decomposition, clustering and distributed sampling, in order to speed up the computation of a solution. I present results of numerical experiments to support their effectiveness.*

The suggested improvements are: application of limited lookahead rollout algorithms in the initial phases to guide the exploration and to provide the first samples to the approximator; decomposing the action space to decrease the number of available actions in the states; clustering the tasks to reduce the length of the trajectories and so the variance of the cumulative costs; as well as two methods to distribute the proposed algorithm among several processors having either a shared or a distributed memory architecture. These approaches are contained in Sections 2.2.3 and 2.2.4. I present results of numerical experiments concerning the improvements in Sections 4.1.3 and 4.1.5. These experiments illustrate the effects of clustering depending on the size of the clusters and the speedup relative to the number of processors.

1.3.2 Varying Environments

In Chapter 3 I analyze the effects of changes in the environment. I also investigate value function based RL methods in varying environments. My main contributions are as follows:

T 2.1 *I deduce bounds in discounted MDPs concerning the dependence of the optimal value function and value functions of (stationary, Markovian, randomized) control policies on the transition-probabilities, the immediate-costs and the discount factor.*

I prove that the value function of a (stationary, Markovian, randomized) control policy in a discounted MDP Lipschitz continuously depends on the immediate-cost function (Theorem 11). A similar result was already known for the case of transition-probability functions, however, I present an improved bound for that case, as well (Theorem 10). I also present value function bounds (Theorem 12) for the case of changes in the discount factor and demonstrate through an example that this dependence is not Lipschitz continuous. Then (with Lemma 14) I extend these results to optimal value functions, too. These theorems can be found in Section 3.1.

T 2.2 *I introduce a new MDP model, called (ε, δ)-MDP, in order to study varying environments. It allows asymptotically bounded changes in the transition-probabilities and the immediate-costs. I prove that changes in the discount factor can be incorporated into the immediate-costs, thus, discount changes do not have to be modeled.*

In order to study changing environments, I introduce (ε, δ)-MDPs (Definition 23) that are generalizations of classical MDPs and ε-MDPs. In this extended model the transition-probability function and the immediate-cost function may change over time, provided that the accumulated changes remain asymptotically bounded, viz. bounded in the limit. I show (Lemma 24) that potential changes in the discount factor can be incorporated into the immediate-cost function, thus, discount changes do not have to be considered. These contributions are presented in Section 3.2.2.

T 2.3 *I prove a general convergence theorem for time-dependent stochastic iterative algorithms. As a corollary, I deduce an approximation theorem for value function based reinforcement learning (RL) methods working in (ε, δ)-MDPs. I also illustrate these results through three classical RL algorithms as well as numerical experiments.*

I analyze stochastic iterative algorithms where the value function update operator may change over time. I prove a relaxed convergence theorem for this kind of algorithm (Theorem 26). As a corollary, I get an approximation theorem for value function based RL methods working in (ε, δ)-MDPs (Corollary 27). Furthermore, I illustrate my results through three classical RL algorithms. I deduce relaxed convergence properties in (ε, δ)-MDPs for asynchronous value iteration, Q-learning and TD(λ) – temporal difference learning. In order to demonstrate the results, I present two simple stochastic iterative algorithms, a "well-behaving" and a "pathological" one. These contributions are described in Sections 3.2.3 and 3.2.4. I also present results of numerical experiments which highlight some features of working in varying environments. I show two experiments concerning adaptation in Section 4.2.

Chapter 2

Stochastic Resource Allocation

As we saw in Chapter 1, *resource allocation problems* (RAPs) have may important practical applications and, usually, they are difficult to solve, even in deterministic cases. It is known, for example, that both JSP and TSP are *strongly NP-hard*, moreover, they do not have any good *polynomial time approximation* algorithm, either. Additionally, in "real world" problems we often have to face *uncertainties*, e.g., in many cases the processing times of the tasks or the durations of the trips are not known exactly in advance, only estimations are available to work with, e.g., these values are given by suitable random variables.

Unfortunately, it is not trivial to extend classical approaches, such as *branch and cut* or *constraint satisfaction* algorithms, to handle stochastic RAPs. Simply replacing the random variables with their expected values and, then, applying standard deterministic algorithms, usually, does not lead to efficient solutions. The issue of additional uncertainties in RAPs makes them even more challenging and call for advanced techniques. In the dissertation we suggest applying statistical *machine learning* (ML) methods to handle these problems.

In this chapter, first, we define a general resource allocation *framework* which is a natural extension of several standard resource management problems, such as JSP and TSP. Then, in order to apply ML methods, we reformulate it as an MDP. Both *proactive* (off-line) and *reactive* (on-line) resource allocation are considered and their relation is analyzed. Concerning efficient solution methods, we restrict ourselves to reactive solutions. We suggest regression based RL methods to solve RAPs and, later, we extend the solution with several additional improvements to speed up the computation of an efficient control policy.

2.1 Markovian Resource Control

This section aims at precisely defining RAPs and reformulating them in a way that they could be effectively solved by machine learning methods presented in Section 2.2. First, a general resource allocation framework is described. We start with deterministic variants and then extend the definition to the stochastic case. Afterwards, we reformulate the reactive problem as an MDP. Later, with the help of POMDPS, we study how this approach could be extended to proactive solutions. Finally, we show that the solution of the proactive problem can be lower and upper bounded with the help of the corresponding reactive solution.

2.1.1 Deterministic Framework

Now, we present a general formal framework to model diverse resource allocation problems. As we will see, this framework is an extension of several classical combinatorial optimization type RAPs, such as scheduling and transportation problems, e.g., JSP and TSP.

First, a deterministic resource allocation problem is considered: an instance of the problem can be characterized by an 8-tuple $\langle \mathcal{R}, \mathcal{S}, \mathcal{O}, \mathcal{T}, \mathcal{C}, d, e, i \rangle$. In details the problem consists of a set of reusable *resources* $\mathcal{R}$ together with $\mathcal{S}$ that corresponds to the set of possible *resource states*. A set of allowed *operations* $\mathcal{O}$ is also given with a subset $\mathcal{T} \subseteq \mathcal{O}$ which denotes the *target operations* or *tasks*. $\mathcal{R}$, $\mathcal{S}$ and $\mathcal{O}$ are supposed to be finite and they are pairwise disjoint. There can be *precedence constrains* between the tasks, which are represented by a partial ordering $\mathcal{C} \subseteq \mathcal{T} \times \mathcal{T}$. The *durations* of the operations depending on the state of the executing resource are defined by a *partial* function $d : \mathcal{S} \times \mathcal{O} \hookrightarrow \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, thus, we have a discrete-time model. Every operation can *affect* the state of the executing resource, as well, that is described by $e : \mathcal{S} \times \mathcal{O} \hookrightarrow \mathcal{S}$ which is also a partial function. It is assumed that $dom(d) = dom(e)$, where $dom(\cdot)$ denotes the domain set of a function. Finally, the *initial states* of the resources are given by $i : \mathcal{R} \to \mathcal{S}$.

The state of a resource can contain all relevant information about it, for example, its type and current setup (scheduling problems), its location and load (transportation problems) or condition (maintenance and repair problems). Similarly, an operation can affect the state in many ways, e.g., it can change the setup of the resource, its location or condition. The system must allocate each task (target operation) to a resource, however, there may be cases when first the state of a resource must be modified in order to be able to execute a certain task (e.g., a transporter may need, first, to travel to its loading/source point, a machine may require repair or setup). In these cases non-task operations may be applied. They can modify the states of the resources without directly serving a demand (executing a task). It is possible that during the resource allocation process a *non-task* operation is applied several times, but other *non-task* operations are completely avoided (for example, because of their high cost). Nevertheless, finally, all *tasks* must be completed.

Feasible Resource Allocation

A *solution* for a deterministic RAP is a partial function, the *resource allocator function*, $\varrho : \mathcal{R} \times \mathbb{N} \hookrightarrow \mathcal{O}$ that assigns the *starting times* of the operations on the resources. Note that the operations are supposed to be *non-preemptive* (they may not be interrupted).

A solution is called *feasible* if and only if the following four properties are satisfied:

1. All tasks are associated with exactly one (resource, time point) pair:
   $\forall v \in \mathcal{T} : \exists! \, \langle r, t \rangle \in dom(\varrho) : v = \varrho(r, t)$.

2. Each resource executes, at most, one operation at a time:
   $\neg \exists u, v \in \mathcal{O} : u = \varrho(r, t_1) \,\wedge\, v = \varrho(r, t_2) \,\wedge\, t_1 \leq t_2 < t_1 + d(s(r, t_1), u)$.

3. The precedence constraints of the tasks are kept:
   $\forall \, \langle u, v \rangle \in \mathcal{C} : [u = \varrho(r_1, t_1) \,\wedge\, v = \varrho(r_2, t_2)] \Rightarrow [t_1 + d(s(r_1, t_1), u) \leq t_2]\,.$

4. Every operation-to-resource assignment is valid:
   $\forall \langle r,t\rangle \in dom(\varrho) : \langle s(r,t), \varrho(r,t)\rangle \in dom(d)$,

where $s : \mathcal{R} \times \mathbb{N} \to \mathcal{S}$ describes the states of the resources at given times

$$s(r,t) = \begin{cases} i(r) & \text{if } t = 0 \\ s(r,t-1) & \text{if } \langle r,t\rangle \notin dom(\varrho) \\ e(s(r,t-1), \varrho(r,t)) & \text{otherwise} \end{cases}$$

A RAP is called *correctly specified* if there exists at least one feasible solution. In what follows it is assumed that the problems are correctly specified. Take a look at Figure 2.1.

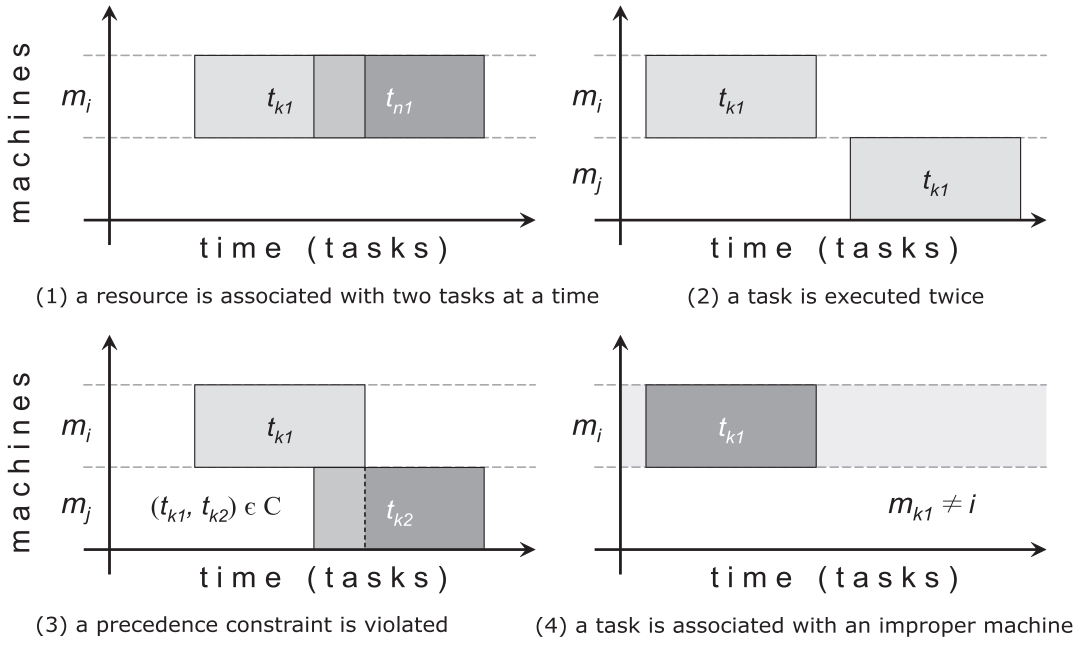


Figure 2.1: Feasibility - an illustration of the four forbidden properties, using JSP as an example. The presented four cases are excluded from the set of feasible schedules.

Performance Measures

The set of all feasible solutions is denoted by $\mathbb{S}$. There is a performance (or cost) associated with each solution, which is defined by a *performance measure* $\kappa : \mathbb{S} \to \mathbb{R}$ that often depends on the task completion times, only. Typical performance measures that appear in practice include: maximum completion time or mean flow time. The aim of the resource allocator system is to compute a feasible solution with maximal performance (or minimal cost).

Note that the performance measure can assign penalties for violating *release* and *due dates* (if they are available) or can even reflect the *priority* of the tasks. A possible generalization of the given problem is the case when the operations may require more resources simultaneously, which is important to model, e.g., resource constrained project scheduling problems.

However, it is straightforward to extend the framework to this case: the definition of d and e should be changed to $d : \mathcal{S}^{\langle k \rangle} \times \mathcal{O} \to \mathbb{N}$ and $e : \mathcal{S}^{\langle k \rangle} \times \mathcal{O} \to \mathcal{S}^{\langle k \rangle}$, where $\mathcal{S}^{\langle k \rangle} = \cup_{i=1}^{k} \mathcal{S}^{i}$ and $k \leq |\mathcal{R}|$. Naturally, we assume that for all $\langle \hat{s}, o \rangle \in dom(e) : dim(e(\hat{s}, o)) = dim(\hat{s})$. Although, managing tasks with multiple resource requirements may be important in some cases, to keep the analysis as simple as possible, we do not include them in the model. Nevertheless, the presented model could be easily generalized to this case and, moreover, the solution methods presented in Section 2.2 are applicable to handle such tasks, as well.

Demonstrative Examples

Now, as demonstrative examples, we reformulate (F)JSP and TSP in the given framework.

It is straightforward to formulate scheduling problems, such as JSP, in the presented resource allocation framework: the tasks of JSP can be directly associated with the tasks of the framework, machines can be associated with resources and processing times with durations. The precedence constraints are determined by the linear ordering of the tasks in each job. Note that there is only one possible resource state for every machine. Finally, feasible schedules can be associated with feasible solutions. If there were setup-times in the problem, as well, then there would be several states for each resource (according to its current setup) and the "set-up" procedures could be associated with the non-task operations.

Regarding the RAP formulation of TSP, $\mathcal{R} = \{r\}$, where r corresponds to the "salesman". $\mathcal{S} = \{s_1, \dots, s_n\}$, if the state (of r) is s_i, it indicates that the salesman is in city i. $\mathcal{O} = \mathcal{T} = \{t_1, \dots, t_n\}$, where the execution of task t_i symbolizes that the salesman goes to city i from his current location. The constraints $\mathcal{C} = \{\langle t_2, t_1 \rangle, \langle t_3, t_1 \rangle \dots, \langle t_n, t_1 \rangle\}$ are used for forcing the system to end the whole round-tour in city 1, which is also the starting city, thus, $i(r) = s_1$. For all $s_i \in \mathcal{S}$ and $t_j \in \mathcal{T}$: $\langle s_i, t_j \rangle \in dom(d)$ if and only if $\langle i, j \rangle \in E$. For all $\langle s_i, t_j \rangle \in dom(d) : d(s_i, t_j) = w_{ij}$ and $e(s_i, t_j) = s_j$. Note that $dom(e) = dom(d)$ and the first feasibility requirement guarantees that each city is visited exactly once. The performance measure κ is the latest arrival time, $\kappa(\varrho) = \max\{t + d(s(r,t), \varrho(r,t)) \mid \langle r, t \rangle \in dom(\varrho)\}$.

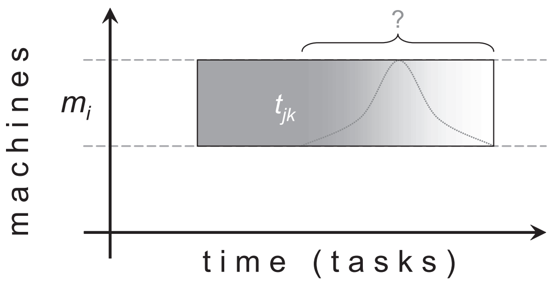
Computational Complexity

If we use a performance measure which has the property that a solution can be precisely defined by a bounded sequence of operations (which includes all tasks) with their assignment to the resources and, additionally, among the solutions generated this way an optimal one can be found, then the RAP becomes a *combinatorial optimization* problem. Each performance measure monotone in the completion times, these measures are called *regular*, has this property. Because the above defined RAP is a generalization of, e.g., JSP and TSP, it is *strongly NP-hard* and, furthermore, no good polynomial-time approximation of the optimal resource allocating algorithm exists, either (Papadimitriou, 1994).

2.1.2 Stochastic Framework

So far our model has been deterministic, now we turn to stochastic RAPs. The stochastic variant of the described general class of RAPs can be defined by randomizing functions d,

e and i. Consequently, the operation durations become random, $d : \mathcal{S} \times \mathcal{O} \to \Delta(\mathbb{N})$, where $\Delta(\mathbb{N})$ is the space of probability distributions over $\mathbb{N}$. Also the effects of the operations are uncertain, $e : \mathcal{S} \times \mathcal{O} \to \Delta(\mathcal{S})$ and the initial states of the resources can be stochastic, as well, $i : \mathcal{R} \to \Delta(\mathcal{S})$. Note that the ranges of functions d, e and i contain probability distributions, we denote the corresponding random variables by D, E and I, respectively. The notation $X \sim f$ indicate that random variable X has probability distribution f. Thus, $D(s,o) \sim d(s,o)$, $E(s,o) \sim e(s,o)$ and $I(r) \sim i(r)$ for all $s \in \mathcal{S}$, $o \in \mathcal{O}$ and $r \in \mathcal{R}$. Take a look at Figure 2.2 for an illustration of the stochastic variants of the JSP and TSP.



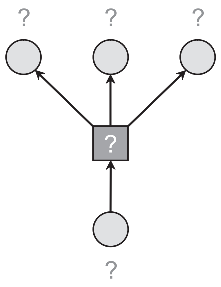


Figure 2.2: Randomization in case of JSP (left) and in case of TSP (right). In the latter, the initial state, the durations and the arrival vertex could be uncertain, as well.

Stochastic Dominance

In stochastic RAPs the performance of a solution is also a random variable. Therefore, in order to compare the performance of different solutions, we have to compare random variables. Many ways are known to make this comparison. We may say, for example, that a random variable has stochastic dominance over another random variable "almost surely", "in likelihood ratio sense", "stochastically", "in the increasing convex sense" or "in expectation". In different applications different types of comparisons may be suitable, however, probably the most natural one is based upon the expected values of the random variables. In the dissertation we apply this kind of comparison for solutions of stochastic RAPs.

Solution Classification

In this subsection we classify the basic types of resource allocation techniques. First, in order to give a proper classification we need the concepts of "open-loop" and "closed-loop" controllers. An *open-loop* controller, also called a non-feedback controller, computes its input into a system by using only the current state and its model of the system. Therefore, an open-loop controller does not use feedback to determine if its input has achieved the desired goal, it does not observe the output of the processes being controlled. In contrast, a *closed-loop* controller uses feedback to control the system (Sontag, 1998). Figure 2.3 demonstrates the two control concepts. Closed-loop control has a clear advantage over open-loop solutions in dealing with *uncertainties*. Hence, it also has improved reference tracking performance, it can stabilize unstable processes and reduced sensitivity to parameter variations.

In *deterministic* RAPs there is no significant difference between open- and closed-loop controls. In this case we can safely restrict ourselves to open-loop methods. If the solution is aimed at generating the resource allocation off-line in advance, then it is called *predictive*. Thus, predictive solutions perform open-loop control and assume a deterministic environment. In *stochastic* resource allocation there are some data (e.g., the actual durations) that will be available only during the execution of the plan. Based on the usage of this information, we identify two basic types of solution techniques. An open-loop solution that can deal with the uncertainties of the environment is called *proactive*. A proactive solution allocates the operations to resources and defines the orders of the operations, but, because the durations are uncertain, it does not determine precise starting times. This kind of technique can be applied only when the durations of the operations are stochastic, but, the states of the resources are known perfectly (e.g., stochastic JSP). Finally, in the stochastic case closed-loop solutions are called *reactive*. A reactive solution is allowed to make the decisions on-line, as the process actually evolves providing more information. Naturally, a reactive solution is not a simple sequence, but rather a resource allocation *policy* (to be defined later) which controls the process. The thesis mainly focuses on reactive solutions, only. We will formulate the reactive solution of a stochastic RAP as a control policy of a suitably defined Markov decision process (specially, a stochastic shortest path problem). Even though we focus on reactive solutions, we will briefly investigate how our approach could be extended to handle proactive resource allocation problems, as well.

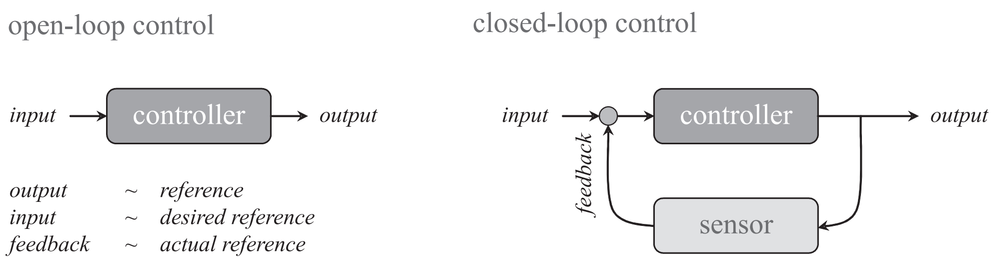


Figure 2.3: The concepts of open-loop (non-feedback) and closed-loop (feedback) controllers. The latter observes the output of the controlled process with sensors.

2.1.3 Reactive Resource Control

In this section we formulate reactive solutions of stochastic RAPs as control policies of suitably reformulated SSP problems. The current task durations and resource states will only be incrementally available during the resource allocation control process.

Problem Reformulation

In order to reformulate RAPs as SSPs (which are special MDPs), we have to define the state space (including the initial and terminal states), the action space, the action constraint function, the effects of actions (transition-probabilities) and the immediate-cost function.

A state $x \in \mathbb{X}$ is defined as a 4-tuple, more precisely, $x = \langle \tau, \mu, \varrho, \varphi \rangle$, where $\tau \in \mathbb{N}$ is the current time and the function $\mu : \mathcal{R} \to \mathcal{S}$ determines the current states of the resources. The *partial* functions ϱ and φ store the past of the process, namely, $\varrho : \mathcal{R} \times \mathbb{N}_{\tau-1} \hookrightarrow \mathcal{O}$ contains the resources and the times in which an operation was started and $\varphi : \mathcal{R} \times \mathbb{N}_{\tau-1} \hookrightarrow \mathbb{N}_\tau$ describes the stopping times of the already completed operations, where $\mathbb{N}_\tau = \{0, \dots, \tau\}$. Naturally, it is always true that $dom(\varphi) \subseteq dom(\varrho)$. By $\mathcal{T}_S(x) \subseteq \mathcal{T}$ we denote the set of tasks which have been started in state x (before the current time τ) and by $\mathcal{T}_F(x) \subseteq \mathcal{T}_S(x)$ the set of tasks that have been finished already in state x. It is easy to see that $\mathcal{T}_S(x) = rng(\varrho) \cap \mathcal{T}$ and $\mathcal{T}_F(x) = rng(\varrho|_{dom(\varphi)}) \cap \mathcal{T}$, where $rng(\cdot)$ denotes the range set (also called image set) of a function. The resource allocation process starts from an initial state $x_s = \langle 0, \mu, \emptyset, \emptyset \rangle$, which corresponds to the situation at time zero when none of the operations have been started. The initial probability distribution of the problem, $\widetilde{x}_0$, can be calculated as follows

$$\widetilde{x}_0(x_s) = \mathbb{P}\left(\mu(r_1) = I(r_1), \dots, \mu(r_n) = I(r_n)\right),$$

where $I(r) \sim i(r)$ denotes the random variable that determines the initial state of resource $r \in \mathcal{R}$ and n is the number of resources, thus, $n = |\mathcal{R}|$. Therefore, $\widetilde{x}_0$ renders initial states to resources according to probability distribution I. We introduce a set of terminal states, as well. A state x is considered as a terminal state ($x \in \mathbb{T}$) if and only if $\mathcal{T}_F(x) = \mathcal{T}$ and it can be reached from a state $\hat{x}$, where $\mathcal{T}_F(\hat{x}) \neq \mathcal{T}$. If the system reaches a terminal state, which means that all tasks are finished, then we treat the control process completed.

It is easy to see that, in theory, we can *aggregate* all terminal states to a global unique terminal state and introduce a new unique initial state, x_0, that has only one available action which takes us randomly (with $\widetilde{x}_0$ distribution) to the real initial states. Then, the problem becomes a *stochastic shortest path* problem and the aim can be described as finding a routing having minimal expected cost from the new initial state to the goal state.

At every time τ the system is informed on the finished operations, and it can decide on the operations to apply (and by which resources). The control action space contains operation-resource assignments $a_{vr} \in \mathbb{A}$, where $v \in \mathcal{O}$ and $r \in \mathcal{R}$, and a special a_{wait} control that corresponds to the action when the system does not start a new operation at the current time. In a non-terminal state $x = \langle \tau, \mu, \varrho, \varphi \rangle$ the available actions are

$$a_{wait} \in \mathcal{A}(x) \Leftrightarrow \mathcal{T}_S(x) \setminus \mathcal{T}_F(x) \neq \emptyset$$

$$\forall v \in \mathcal{O} : \forall r \in \mathcal{R} : a_{vr} \in \mathcal{A}(x) \Leftrightarrow (v \in \mathcal{O} \setminus \mathcal{T}_S(x) \ \wedge \ \forall \langle \hat{r}, t \rangle \in dom(\varrho) \setminus dom(\varphi) : \hat{r} \neq r \ \wedge$$
$$\wedge \ \langle \mu(r), v \rangle \in dom(d) \ \wedge \ v \in \mathcal{T} \Rightarrow (\forall u \in \mathcal{T} : \langle u, v \rangle \in \mathcal{C} \Rightarrow u \in \mathcal{T}_F(x)))$$

Thus, action a_{wait} is available in every state with an unfinished operation; action a_{vr} is available in states in which resource r is idle, it can process operation v, additionally, if v is a task, then it was not executed earlier and its precedence constraints are satisfied.

If an action $a_{vr} \in \mathcal{A}(x)$ is executed in a state $x = \langle \tau, \mu, \varrho, \varphi \rangle$, then the system moves with probability one to a new state $\hat{x} = \langle \tau, \mu, \hat{\varrho}, \varphi \rangle$, where $\hat{\varrho} = \varrho \cup \{\langle \langle r, t \rangle, v \rangle\}$. Note that we treat functions as sets of ordered pairs. The resulting $\hat{x}$ corresponds to the state where operation v has started on resource r if the previous state of the environment was x.

The effect of the a_{wait} action is that from $x = \langle \tau, \mu, \varrho, \varphi \rangle$ it takes to an $\hat{x} = \langle \tau + 1, \hat{\mu}, \varrho, \hat{\varphi} \rangle$, where an unfinished operation $\varrho(r, t)$ that was started at t on r finishes with probability

$$\mathbb{P}(\langle r, t \rangle \in dom(\hat{\varphi}) \mid x, \langle r, t \rangle \in dom(\varrho) \setminus dom(\varphi)) = \frac{\mathbb{P}(D(\mu(r), \varrho(r, t)) + t = \tau)}{\mathbb{P}(D(\mu(r), \varrho(r, t)) + t \geq \tau)},$$

where $D(s, v) \sim d(s, v)$ is a random variable that determines the duration of operation v when it is executed by a resource which has state s. This quantity is called *completion rate* in stochastic scheduling theory and *hazard rate* in reliability theory. We remark that for operations with continuous durations, this quantity is defined by $f(t)/(1 - F(t))$, where f denotes the density function and F the distribution of the random variable that determines the duration of the operation. If operation $v = \varrho(r, t)$ has finished ($\langle r, t \rangle \in dom(\hat{\varphi})$), then $\hat{\varphi}(r, t) = \tau$ and $\hat{\mu}(r) = E(r, v)$, where $E(r, v) \sim e(r, v)$ is a random variable that determines the new state of resource r after it has executed operation v. Except the extension of its domain set, the other values of function φ do not change, consequently, $\forall \langle r, t \rangle \in dom(\varphi)$: $\hat{\varphi}(r, t) = \varphi(r, t)$. In other words, $\hat{\varphi}$ is a conservative extension of φ, formally, $\varphi \subseteq \hat{\varphi}$.

The cost function g, for a given κ performance measure (which depends only on the operation-resource assignments and the completion times), is defined as follows. Let $x = \langle \tau, \mu, \varrho, \varphi \rangle$ and $\hat{x} = \langle \hat{\tau}, \hat{\mu}, \hat{\varrho}, \hat{\varphi} \rangle$. Then, if the system arrives at state $\hat{x}$ after executing action a in state x, it incurs the cost $\kappa(\varrho, \varphi) - \kappa(\hat{\varrho}, \hat{\varphi})$. Note that, though, in Section 2.1.1 performance measures were defined on complete solutions, for most measures applied in practice (e.g., makespan, weighted total lateness) it is straightforward to generalize the measure to partial solutions, as well. One may, for example, treat the partial solution of a problem as a complete solution of a smaller (sub)problem, viz., a problem with fewer tasks.

Favorable Features

Let us call the introduced SSPs, which describe stochastic RAPs, *RAP-MDPs.* In this section we overview some basic properties of RAP-MDPs. First, it is straightforward to see that these MDPs have finite action spaces, since $|\mathbb{A}| \leq |\mathcal{R}|\,|\mathcal{O}| + 1$ always holds.

We may also observe that RAP-MDPs are *acyclic*, namely, none of the states can appear multiple times, because during the resource allocation process τ and $dom(\varrho)$ are non-decreasing and, additionally, each time the state changes, the quantity $\tau + |dom(\varrho)|$ strictly increases. Therefore, the system cannot reach the same state twice. As an immediate consequence, we can notice that all control policies eventually terminate and, therefore, proper.

Though, the state space of a RAP-MDP is denumerable in general, if the allowed number of non-task operations is bounded and the random variables describing the operation durations are finite, the state space of the reformulated MDP becomes finite, as well.

For the effective computation of a good control policy, it is important to try to reduce the number of states. We can do so by recognizing that if the performance measure κ is non-decreasing in the completion times, then an optimal control policy of the reformulated RAP-MDP can be found among the policies which start new operations only at times when another operation has been finished or in an initial state. This statement can be supported by the fact that without increasing the cost (κ is non-decreasing) every operation can be shifted earlier on the resource which was assigned to it until it reaches another operation, or until it reaches a time when one of its preceding tasks is finished (if the operation was a task with precedence constrains), or, ultimately, until time zero. Note that most of the performance measures used in practice (e.g., makespan, weighted completion time, average tardiness) are non-decreasing. As a consequence, the states in which no operation has been finished can be omitted, except the initial states. Therefore, each a_{wait} action may lead to a state where an operation has been finished. We may consider it, as the system executes automatically an a_{wait} action in the omitted states. By this way, the state space can be decreased and, therefore, a good control policy can be calculated more effectively.

Composable Measures

For a large class of performance measures, the state representation can be simplified by leaving out the past of the process. In order to do so, we must require that the performance measure be composable with a suitable function. In general, a function $f : \mathcal{P}(X) \to \mathbb{R}$ is called γ-*composable* if for any $A, B \subseteq X$, $A \cap B = \emptyset$ it holds that $\gamma(f(A), f(B)) = f(A \cup B)$, where $\gamma : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is called the *composition function*, and X is an arbitrary set. This definition can be directly applied to performance measures. If a performance measure, e.g., is γ-composable, it indicates that the value of any complete solution can be computed from the values of its disjoint subsolutions (solutions to subproblems) with function γ. In practical situations the composition function is often the max, the min or the "+" function.

If the performance measure κ is γ-composable, then the past can be omitted from the state representation, because the performance can be calculated incrementally. Thus, a state can be described as $x = \langle \bar{\tau}, \bar{\kappa}, \bar{\mu}, \mathcal{T}_U \rangle$, where $\bar{\tau} \in \mathbb{N}$, as previously, is the current time, $\bar{\kappa} \in \mathbb{R}$ contains the performance of the current (partial) solution and $\mathcal{T}_U$ is the set of unfinished tasks. The function $\bar{\mu} : \mathcal{R} \to \mathcal{S} \times (\mathcal{O} \cup \{\iota\}) \times \mathbb{N}$ determines the current states of the resources together with the operations currently executed by them (or ι if a resource is idle) and the starting times of the operations (needed to compute their completion rates).

In order to keep the analysis as simple as possible, we restrict ourselves to composable functions, since almost all performance measures that appear in practice are γ-composable for a suitable γ (e.g., makespan or total production time is max-composable).

Reactive Solutions

Now, we are in a position to define the concept of reactive solutions for stochastic RAPs. A *reactive solution* is a (stationary, Markovian) control policy of the reformulated SSP problem. It performs a closed-loop (on-line) control, since at each time step the controller is informed about the current state of system and it can choose a control action based upon this information. Section 2.2 deals with the computation of effective control policies.

2.1.4 Proactive Resource Control

In order to apply a reactive solution, which performs closed-loop control, sufficient feedback should be available from the system, e.g., provided by sensors. In some situations, however, we do not have such feedback and we must apply an open-loop controller, still, we should also take uncertainties into account. In these cases one can design a proactive solution. Here, we suggest a way of extending our approach to be able to handle proactive problems.

The problem of proactive resource allocation can be formulated as a *non-observable Markov decision process* (NOMDP) that is a special POMDP, in which the observation set contains only one element, $|\mathbb{O}| = 1$. In this case the equations describing the progress of the belief state MDP can be simplified, e.g., the successor belief state can be computed by

$$\tau(b, a)(y) = \sum_{x \in \mathbb{X}} p(y \mid x, a)\, b(x),$$

where the observation parameter z is omitted, for obvious reasons. The Bellman equation, which describes the optimal cost-to-go function of the transformed MDP, is also simpler

$$\hat{J}^*(b) = \min_{a \in \mathbb{A}} \Big[g(b, a) + \alpha \hat{J}^*(\tau(b, a)) \Big].$$

We argue that the already calculated reactive solution, namely, the cost-to-go function of the fully observable MDP (FOMDP), J^*, can be used to accelerate the computation of finding a good policy for the proactive problem, since the cost-to-go function $\hat{J}^*$ of the NOMDP can be lower and upper bounded by J^*. Computing a reactive solution is a much easier task than computing a proactive one, because, e.g., there are much fewer states to count with, therefore, partially tracing back the problem of approximating the optimal cost-to-go of a NOMDP to the case of a FOMDP can mean a remarkable speedup. Moreover, the bounds are valid for arbitrary POMDPs, as well. Although, the first inequality of the presented statement follows from the results of Åström (1965), White (1976), or, more recently, Hauskrecht (2000), we also present another direct proof of it in the appendix.

Theorem 7 *Consider a POMDP and its fully observable MDP counterpart, which system has the same state space, action space, transition-probabilities and costs as the original POMDP, only the observability is different. The optimal cost-to-go functions of the POMDP and the MDP are denoted by $\tilde{J}^*$ and J^*, respectively. Then, for all belief state b, we have*

$$\sum_{x \in \mathbb{X}} b(x)\, J^*(x) \leq \tilde{J}^*(b) \leq \sum_{x \in \mathbb{X}} b(x)\, J^*(x) + \frac{c}{1 - \alpha},$$

where $c = (g_{max} - g_{min})$; $g_{max} = \max\{g(x, a) \mid x \in \mathbb{X}, a \in \mathbb{A}\}$ and similarly for g_{min}. If the immediate-cost function g is not constant ($c \neq 0$), then the second inequality is strict.

The bounds for the optimal cost-to-go function of the belief state MDP are sharp, as the following lemma shows (applying the usual notations for POMDPs and FOMDPs):

Lemma 8 *(1) There exists a POMDP and a corresponding fully observable MDP such that, for a belief state b: $\tilde{J}^*(b) = \langle J^*, b \rangle$; as well as (2) for all $\varepsilon > 0$ there exists a POMDP and a corresponding MDP such that for a belief state b: $|\tilde{J}^*(b) - \langle J^*, b \rangle - c/(1 - \alpha)| < \varepsilon$.*

In this lemma $\langle \cdot, \cdot \rangle$ denoted inner product, namely, $\langle f, g \rangle = \sum_x f(x)g(x)$. If take a closer look at the proof, located in the appendix, we can notice that the bounds remain sharp even if we restrict ourselves to NOMDPs. The lower bound can be a remarkable help, e.g., when such an ADP method is applied that strictly requires non-underestimating initial values for the approximated optimal cost-to-go function. An example of a learning method with such assumption could be the *real-time dynamic programming* (RTDP) algorithm.

Proactive Solutions

As a summary, we can conclude that a *proactive solution* of a stochastic RAP is a (stationary, Markovian) control policy of the corresponding NOMDP. It is known that POMDPs can be transformed into belief state MDPs, however, it is usually hard to translate this result into efficient practical solution algorithms. We have shown that the optimal solution of the reactive problem can be applied to accelerate the calculation of the proactive solution. Furthermore, there exists several approximation methods to ensure efficient solutions of POMDPs (Hauskrecht, 2000). In the thesis we present proactive solutions only as a possibility and leave their theoretical and practical investigations to further research. Henceforth, we will restrict ourselves to machine learning based *reactive* solutions.

2.2 Machine Learning Approaches

In this section we aim at giving an effective solution to large-scale RAPs in uncertain and dynamic environments with the help of different machine learning approaches. First, we overview some reinforcement learning (RL) methods to compute a "good" policy. Afterwards, we investigate two function approximation techniques to enhance the solution. Clustering, rollout algorithm and action decomposition as well as distributed sampling are also considered, as they can speedup the computation of a good control policy and, therefore, are important additions if we face large-scale problems. Later, in Chapter 3, we will see that this kind of approach can effectively work in changing environments, as well.

2.2.1 Reinforcement Learning

In the previous sections we have formulated RAPs as acyclic SSP problems. Now, we face the challenge of finding a good policy. In theory, the optimal value function of a finite MDP can be exactly computed by *dynamic programming* (DP) methods, such as value iteration or the Gauss-Seidel method. Alternatively, an exact optimal policy can be directly calculated by policy iteration. However, due to the "curse of dimensionality", computing an exact optimal solution by these methods is practically infeasible, e.g., typically both the required amount of computation and the needed storage space, viz., memory, grows combinatorially with the size of the problem. In order to handle the "curse", we should apply *approximate dynamic programming* (ADP) techniques to achieve a good approximation of an optimal policy. Here, we suggest using sampling-based *fitted Q-learning* (FQL). In each trial a Monte-Carlo estimate of the value function is computed and projected onto a suitable function space. The methods described in this section (FQL, MCMC and the Boltzmann formula) should be applied simultaneously, in order to achieve an efficient solution. First, for the sake of better understandability, we overview two basic ADP methods which are combined with function approximation: fitted value iteration and fitted policy iteration.

Fitted Value Iteration

The *value iteration* algorithm is one of the basic dynamic programming methods. It is defined by the iteration $J_{k+1} = TJ_k$, where J_k is a value function and T is the Bellman operator, defined by equation (1.2). It is known that J_k converges in the supremum norm to J^* for any initial J_0, even if the updates of different states are performed asynchronously and also for more general operators (Bertsekas and Tsitsiklis, 1996; Szepesvári and Littman, 1999). The method of *fitted value iteration* (FVI) arises when the cost-to-go function is represented by a (typically parametric) function from a suitable function space $\mathcal{F} \subseteq \mathbb{J}(\mathbb{X})$, where $\mathbb{J}(\mathbb{X})$ denotes the space of value functions over $\mathbb{X}$. In the case of finite MDPs, $\mathbb{J}(\mathbb{X})$ contains all $J : \mathbb{X} \to \mathbb{R}$ functions. Thus, after each iteration the resulted value function is projected back onto $\mathcal{F}$. Consequently, in general, the FVI algorithm proceeds as follows

$$J_{k+1} \in \operatorname*{arg\,min}_{f \in \mathcal{F}} \|f - TJ_k\| ,$$

where $\|\cdot\|$ is an appropriate norm. FVI is a special case of *approximate value iteration* (AVI), where the updates follow the form $J_{k+1} = TJ_k + \varepsilon_k$. Bertsekas and Tsitsiklis (1996) have presented L^∞ bounds for the convergence of AVI. Szepesvári and Munos (2005) have shown finite time bounds for the convergence of FVI in case of weighted L^p norms.

Fitted Policy Iteration

There is an alternative to value iteration, called *policy iteration*, which always terminates finitely, in case of finite MDPs. Moreover, value iteration can be seen as a special case of asynchronous policy iteration. In policy iteration, first, the policy is *evaluated* by applying the Bellman operator of policy π, then its is *improved.* It is known (Bertsekas and Tsitsiklis, 1996) that iteration $J_{k+1} = T^\pi J_k$ converges to J^π for any initial J_0. Naturally, we do not need to exactly compute J^π. In asynchronous policy iteration, e.g., a subset of states $\mathbb{X}_k \subseteq \mathbb{X}$ is considered in iteration k and either value function J_k is updated according to

$$J_{k+1}(x) = \begin{cases} T^{\pi_k} J_k(x) & \text{if } x \in \mathbb{X}_k \\ J_k(x) & \text{otherwise} \end{cases} \tag{2.1}$$

while leaving the policy unchanged, $\pi_{k+1} = \pi_k$, or else the policy is improved, while the cost-to-go estimation remains the same, $J_{k+1} = J_k$. The policy is updated as follows

$$\pi_{k+1}(x) = \begin{cases} \operatorname*{arg\,min}_{a \in \mathcal{A}(x)} \Big[g(x,a) + \alpha \sum_{y \in \mathbb{X}} p(y \mid x,a)\, J_k(y) \Big] & \text{if } x \in \mathbb{X}_k \\ \pi_k(x) & \text{otherwise} \end{cases} \tag{2.2}$$

where $\pi_{k+1} : \mathbb{X} \to \mathbb{A}$ is a deterministic policy, which is defined as greedy with respect to J_k on $\mathbb{X}_k$. It is known, e.g., (Bertsekas and Tsitsiklis, 1996), that if the value update (2.1) and the policy update (2.2) are executed infinitely often for all states and J_0, π_0 satisfy $T^{\pi_0} J_0 \leq J_0$ then J_k converges in finitely many steps to the optimal cost-to-go, J^*. The technique of asynchronous policy iteration can be interpreted as an actor-critic method. Similarly to FVI we can define *fitted policy iteration* by redefining the value update step as

$$J_{k+1} \in \left\{ f \in \mathcal{F} : \left\| f - \widetilde{T}^{\pi_k} J_k \right\| \leq \varepsilon \right\},$$

where $\widetilde{T}_{\pi_k}$ denotes the operator that only updates the estimates of $\mathbb{X}_k$ according to T^{π_k}, viz., as defined by (2.1). Naturally, we should choose an ε such that $\varepsilon > \inf_{f \in \mathcal{F}} \left\| f - \widetilde{T}^{\pi_k} J_k \right\|$ or assume that $\mathcal{F}$ is dense in $\mathbb{J}(\mathbb{X})$. Note that if we assume that each value update step is automatically followed by a policy update step (on the same states), then the actual control policy does not need to be stored explicitly, it can be computed on-line, as needed.

Fitted Q-learning

Watkins' Q-learning is a very popular off-policy, model-free reinforcement learning algorithm (Watkins, 1989; Watkins and Dayan, 1992; Even-Dar and Mansour, 2003). Q-learning is *model-free*, since it does not require a model of the environment, more precisely, it does not need the transition-probabilities and the immediate-costs, it satisfies with data coming from simulation. Moreover, Q-learning is *off-policy*, since it approximates the optimal value function independently of the applied policy, as long as it makes sufficient explorations.

Q-learning works with action-value functions and iteratively approximates the optimal action-value function, Q^*. The one-step (tabular) Q-learning rule is defined as follows

$$Q_{i+1}(x,a) = (1 - \gamma_i(x,a)) Q_i(x,a) + \gamma_i(x,a)(\widetilde{T}_i Q_i)(x,a),$$
$$(\widetilde{T}_i Q_i)(x,a) = g(x,a) + \alpha \min_{B \in \mathcal{A}(Y)} Q_i(Y,B),$$

where $\gamma_i(x,a)$ are the learning rates and Y is a random variable representing a state generated from the pair (x,a) by simulation, that is, according to the probability distribution $p(x,a)$. It is known (Bertsekas and Tsitsiklis, 1996) that if $\gamma_i(x,a) \in [0,1]$ and they satisfy

$$\sum_{i=0}^{\infty} \gamma_i(x,a) = \infty \qquad \text{and} \qquad \sum_{i=0}^{\infty} \gamma_i^2(x,a) < \infty,$$

then the Q-learning algorithm converges with probability one to the optimal action-value function, Q^*, in the case of lookup table representation when each state-action value is stored independently. We speak about the method of *fitted Q-learning* (FQL) when the value function is represented by a (typically parametric) function from a suitable function space, $\mathcal{F}$, and after each iteration, the updated value function is projected back onto $\mathcal{F}$.

A useful observation is that we need the "learning rate" parameters only to overcome the effects of random disturbances. However, if we deal with deterministic problems, this part of the method can be simplified. The resulting algorithm simply updates $Q(x,a)$ with the minimum of the previously stored estimation and the current outcome of the simulation, which is also the core idea of the LRTA* algorithm (Bulitko and Lee, 2006). When we dealt with deterministic resource allocation problems, we applied this simplification, as well.

Evaluation by Simulation

Naturally, in large-scale problems we cannot update all states at once. Therefore, we perform *Markov chain Monte Carlo* (MCMC) simulations (Hastings, 1970; Andrieu et al., 2003) to generate samples (with the model or with a simulator), which are used for computing the new approximation of the estimated value function. Therefore, the set of states to be updated in episode i, namely $\mathbb{X}_i$, is generated by simulation. Because RAP-MDPs are acyclic, we apply *prioritized sweeping*, which means that after each iteration the cost-to-go estimations are updated in the reverse order in which they appeared during the simulation. Assume, for example, that $\mathbb{X}_i = \{x_1^i, x_2^i, \dots, x_{t_i}^i\}$ is the set of states for the update of the value function after iteration i, where $j < k$ implies that x_j^i appeared earlier during the simulation than x_k^i. In this case the order in which the updates are performed is $x_{t_i}^i, \dots, x_1^i$. Moreover, we do not need a *uniformly* optimal value function, it is enough to have a good approximation of the optimal cost-to-go function for the relevant states. A state is called *relevant* if it can appear with positive probability during the application of an optimal policy. Therefore, it is sufficient to consider the case when $x_1^i = x_0$, where x_1^i is the first state in episode i and x_0 is the (aggregated) initial state of the SSP problem.

The Boltzmann Formula

In order to ensure the convergence of the FQL algorithm, one must guarantee that each cost-to-go estimation be continuously updated. A technique used often to balance between *exploration* and *exploitation* is the *Boltzmann formula* (also called *softmin* action selection):

$$\pi_i(x, a) = \frac{\exp(-Q_i(x, a)/\tau_i)}{\sum\limits_{b \in \mathcal{A}(x)} \exp(-Q_i(x, b)/\tau_i)},$$

where $\tau_i \geq 0$ is the Boltzmann (or Gibbs) temperature and i is the episode number. It is easy to see that high temperatures cause the actions to be (nearly) equiprobable, low ones cause a greater difference in selection probability for actions that differ in their value estimations. Thus, if τ_i was close to zero, then π_i was close to the greedy policy w.r.t. Q_i. Note that here we applied the Boltzmann formula for minimization, viz., small values result in high probability. It is advised to extend this approach by a variant of *simulated annealing* (Kirkpatrick et al., 1983) or *Metropolis algorithm* (Metropolis et al., 1953), which means that τ_i should be decreased over time, at a suitable rate (see Singh et al., 2000).

2.2.2 Cost-to-Go Representations

In Section 2.2.1 we suggested FQL for iteratively approximating the optimal value function. However, the question of a suitable function space, onto which the resulted value functions can be effectively projected, remained open. In order to deal with large-scale problems (or problems with continuous state spaces) this question is crucial. In this section, first, we suggest features for stochastic RAPs, then describe two methods that can be applied to compactly represent value functions. The first and simpler one applies hash tables while the second, more sophisticated one, builds upon the theory of support vector machines.

Feature Vectors

In order to efficiently apply a function approximator, first, the states and the actions of the reformulated MDP should be associated with numerical vectors representing, e.g., typical features of the system. In the case of stochastic RAPs, we suggest using features as follows:

- For each resource in $\mathcal{R}$, the *resource state id*, the *operation id* of the operation being currently processed by the resource (could be idle), as well as the *starting time* of the last (and currently unfinished) operation can be a feature. If the model is available to the system, the *expected ready time* of the resource should be stored instead.
- For each task in $\mathcal{T}$, the *task state id* could be treated as a feature that can assume one of the following values: "not available" (e.g., some precedence constraints are not satisfied), "ready for execution", "being processed" or "finished". It is also advised to apply "1-out-of-n" coding, viz., each value should be associated with a separate bit.
- In case we use action-value functions, for each action (resource-operation assignment) the *resource id* and the *operation id* could be stored. If the model is available, then the *expected finish time* of the operation should also be taken into account.

In the case of a model-free approach which applies action-value functions, for example, the feature vector would have $3 \cdot |\mathcal{R}| + |\mathcal{T}| + 2$ components. Note that for features representing temporal values, it is advised to use *relative* time values instead of *absolute* ones.

In theory, if the dimension of the feature vector was too large, it could be decreased by statistical dimension reduction methods, such as *principal component analysis* (PCA, also called Karhunen-Loève transform), if Gaussian distribution is assumed, or *independent component analysis* (ICA), in case of non-Gaussian distributions (Hyvärinen et al., 2001).

Hash Tables

Suppose that we have a vector $w = \langle w_1, w_2, \dots, w_k \rangle$, where each component w_i corresponds to a feature of a state or an action. Usually, the value estimations for all of these vectors cannot be stored in the memory. In this case one of the simplest methods to be applied is to represent the estimations in a *hash table*. A hash table is, basically, a dictionary in which keys are mapped to array positions by hash functions. If all components can assume finite values, e.g., in our finite-state, discrete-time case, then a key could be generated as follows. Let us suppose that for all w_i we have $0 \leq w_i < m_i$, then w can be seen as a number in a *mixed radix numeral system* and, therefore, a unique *key* can be calculated as

$$\varphi(w) = \sum_{i=1}^{k} w_i \prod_{j=1}^{i-1} m_j,$$

where $\varphi(w)$ denotes the key of w, and the value of an empty product is treated as one.

The *hash function*, ψ, maps feature vector keys to memory positions. More precisely, if we have memory for storing only d value estimations, then the hash function takes the form $\psi : rng(\varphi) \to \{0, \dots, d-1\}$, where $rng(\cdot)$ denotes the range set of a function.

It is advised to apply a d that is *prime.* In this case a usual hashing function choice is $\psi(x) = y$ if and only if $y \equiv x \pmod{d}$, namely, if y is congruent to x modulo d.

Having the keys of more than one item map to the same position is called a *collision.* There are many collision resolution schemes, they may be divided into open addressing, chaining, and keeping one special overflow area. Here, we do not concern with hash collisions in general, however, in the case of RAP-MDPs we suggest a method as follows. Suppose that during a value update the feature vector of a state (or a state-action pair) maps to a position that is already occupied by another estimation corresponding to another item (which can be detected, e.g., by storing the keys). Then we have a collision and the estimation of the new item should overwrite the old estimation if and only if the MDP state corresponding to the new item appears with higher probability during execution starting from the (aggregated) initial state than the one corresponding to the old item. In case of a model-free approach, the item having a state with smaller current time component can be kept.

Despite its simplicity, the hash table representation has several disadvantages, e.g., it still needs a lot of memory to work efficiently, it cannot easily handle continuous values and, it only stores individual data, moreover, it does not *generalize* to "similar" items. In the next section we present a statistical approach that can deal with these issues, as well.

Support Vector Regression

A promising choice for compactly representing the cost-to-go function is to use *support vector regression* (SVR) from statistical learning theory. In order to maintain the value function estimations, we suggest applying ν-SVRs which were proposed by Schölkopf et al. (2000). They have an advantage over classical ε-SVRs according to which, through the new parameter ν, the number of support vectors can be controlled. Additionally, parameter ε can be eliminated. First, we briefly overview the core ideas of ν-SVRs.

In general, SVR faces the problem as follows. We are given a sample, a set of data points $\{\langle x_1, y_1\rangle, \dots, \langle x_l, y_l\rangle\}$, such that $x_i \in \mathcal{X}$ is an input, where $\mathcal{X}$ is a measurable space, and $y_i \in \mathbb{R}$ is the target output. For simplicity, we shall assume that $\mathcal{X} \subseteq \mathbb{R}^k$, where $k \in \mathbb{N}$. The aim of the learning process is to find a function $f : \mathcal{X} \to \mathbb{R}$ with a small risk

$$R[f] = \int_{\mathcal{X}} l(f, x, y) dP(x, y), \tag{2.3}$$

where P is a probability measure, which is responsible for the generation of the observations and l is a loss function, such as $l(f, x, y) = (f(x) - y)^2$. The learning algorithms usually use other kinds of loss functions, since additional constrains are also maintained by this function. A common error function used in SVRs is the so-called ε-*insensitive* loss function, $|f(x) - y|_\varepsilon = \max\{0, |f(x) - y| - \varepsilon\}$. Unfortunately, we cannot minimize (2.3) directly, since we do not know P, we are given the sample, only (which are generated by simulation in our case). We try to obtain a small risk by minimizing the regularized risk functional in which we replace the average over $P(x, y)$ by an average over the training sample

$$\frac{1}{2} \|w\|^2 + C \cdot R^{\varepsilon}_{emp}[f], \tag{2.4}$$

where, $\|w\|^2$ is a term that characterizes the model complexity and $C > 0$ a constant responsible for a certain trade-off. They will be explained later. The function $R^{\varepsilon}_{emp}[f]$ is

$$R^{\varepsilon}_{emp}[f] = \frac{1}{l}\sum_{i=1}^{l} |f(x_i) - y_i|_{\varepsilon}.$$

It measures the ε-insensitive average training error. The problem which arises when we try to minimize (2.4) is called *empirical risk minimization* (ERM). In regression problems we usually have a Hilbert space $\mathcal{F}$, containing $\mathcal{X} \to \mathbb{R}$ type (typically non-linear) functions, and our aim is to find a function f that is "close" to y_i in each x_i and takes the form

$$f(x) = \sum\nolimits_j w_j \phi_j(x) + b = w^T \phi(x) + b,$$

where $\phi_j \in \mathcal{F}$, $w_j \in \mathbb{R}$ and $b \in \mathbb{R}$. Specially, the aim of the ν-SVR is to minimize

$$\Phi(w, \xi, \xi^*, \varepsilon) = \frac{1}{2}\|w\|_2^2 + C\Big(\nu\varepsilon + \frac{1}{l}\sum_{i=1}^{l}(\xi_i + \xi_i^*)\Big),$$

subject to the constraints on w, b, ξ, ξ^* and ε as follows

$$w^T\phi(x_i) + b - y_i \le \varepsilon + \xi_i \tag{2.5}$$

$$y_i - w^T\phi(x_i) - b \le \varepsilon + \xi_i^* \tag{2.6}$$

$$\xi_i, \xi_i^* \ge 0, i \in \{1, \dots, l\}, \varepsilon \ge 0$$

The parameters ξ_i and ξ_i^* are *slack variables* to cope with the otherwise often infeasible problem. They measure the deviation of a sample point from its ideal condition; $\varepsilon > 0$ defines the amount of deviation that we totally tolerate (viz., see the formulation of ε-insensitive loss functions). The constant $C > 0$ determines the trade-off between the flatness of the regression and the amount up to which deviations larger than ε are tolerated. The particular data points $\langle x_i, y_i\rangle$ for which inequality (2.5) or inequality (2.6) is satisfied with the equality sign, are called *support vectors*, hence the name support vector machines.

Using Lagrange multiplier techniques, we can rewrite the regression problem in its dual form (Schölkopf et al., 2000) and arrive at the final ν-SVR optimization problem. The aim of the dual problem is to maximize for $\nu \ge 0, C > 0$ the function $W(\alpha, \alpha^*)$ which is

$$W(\alpha, \alpha^*) = \sum_{i=1}^{l}(\alpha_i^* - \alpha_i)y_i - \frac{1}{2}\sum_{i=1}^{l}\sum_{j=1}^{l}(\alpha_i^* - \alpha_i)(\alpha_j^* - \alpha_j)K(x_i, x_j),$$

subject to the constraints on the Lagrange multipliers α_i and α_i^*

$$\sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \quad \text{and} \quad \sum_{i=1}^{l}(\alpha_i + \alpha_i^*) \le C \cdot \nu \quad \text{and} \quad \alpha_i, \alpha_i^* \in \left[0, \frac{C}{l}\right].$$

The resulting regression estimate then takes the form as follows

$$f(x) = \sum_{i=1}^{l} (\alpha_i^* - \alpha_i) K(x_i, x) + b,$$

where K is an *inner product kernel* defined by $K(x, y) = \langle \phi(x), \phi(y) \rangle$, where $\langle \cdot, \cdot \rangle$ denotes inner product. Note that $\alpha_i, \alpha_i^* \neq 0$ holds usually only for a small subset of training samples, furthermore, parameter b (and ε) can be computed by taking into account that inequations (2.5) and (2.6) become equalities with $\xi_i, \xi_i^* = 0$ for points with $0 < \alpha_i, \alpha_i^* < C/l$, respectively, due to the *Karush-Kuhn-Tucker* (KKT) conditions. Additionally, it can be proven that the hyperparameter $\nu \geq 0$ is an upper bound on the fraction of training errors and a lower bound on fraction of support vectors Schölkopf et al. (2000).

Mercer's theorem in functional analysis characterizes which functions correspond to an inner product in some space $\mathcal{F}$. Basic kernel types include linear, polynomial, Gaussian and sigmoid functions. In our experiments with RAP-MDPs we have used Gaussian kernels which are also called *radial basis function* (RBF) kernels. RBF kernels are defined by $K(x, y) = \exp(-\|x - y\|^2 / (2\sigma^2))$, where $\sigma > 0$ is an adjustable kernel parameter.

A variant of the fitted Q-learning algorithm combined with regression and softmin action selection is described in Table 2.1. Most of our RAP solutions are based on that algorithm.

The notations of the pseudocode shown in Table 2.1 are as follows. Variable i contains the episode number, t_i is the length of episode i and j is a parameter for time-steps inside an episode. The Boltzmann temperature is denoted by τ, π_i is the control policy applied in

```
Regression Based Q-learning
 1.  Initialize $Q_0$, $\mathcal{L}_0$, $\tau$ and let $i = 1$.
 2.  Repeat (for each episode)
 3.      Set $\pi_i$ to a soft and semi-greedy policy w.r.t. $Q_{i-1}$, e.g.,
         $\pi_i(x, a) = \exp(-Q_{i-1}(x, a)/\tau) / \left[\sum_{b \in \mathcal{A}(x)} \exp(-Q_{i-1}(x, b)/\tau)\right]$.
 4.      Simulate a state-action trajectory from $x_0$ using policy $\pi_i$.
 5.      For $j = t_i$ to 1 (for each state-action pair in the episode) do
 6.          Determine the features of the state-action pair, $y_j^i = h(x_j^i, a_j^i)$.
 7.          Compute the new action-value estimation for $x_j^i$ and $a_j^i$, e.g.,
             $z_j^i = (1 - \gamma_i) Q_{i-1}(x_j^i, a_j^i) + \gamma_i \left[g(x_j^i, a_j^i) + \alpha \min_{b \in \mathcal{A}(x_{j+1}^i)} Q_{i-1}(x_{j+1}^i, b)\right]$.
 8.      End loop (end of state-action processing)
 9.      Update sample set $\mathcal{L}_{i-1}$ with $\left\{\langle y_j^i, z_j^i \rangle : j = 1, \dots, t_i\right\}$, the result is $\mathcal{L}_i$.
10.      Calculate $Q_i$ by fitting a smooth regression function to the sample of $\mathcal{L}_i$.
11.      Increase the episode number, $i$, and decrease the temperature, $\tau$.
12.  Until some terminating conditions are met, e.g., $i$ reaches a limit
     or the estimated approximation error to $Q^*$ gets sufficiently small.
Output: the action-value function $Q_i$ (or $\pi(Q_i)$, e.g., the greedy policy w.r.t. $Q_i$).
```

Table 2.1: Pseudocode for regression-based Q-learning with softmin action selection.

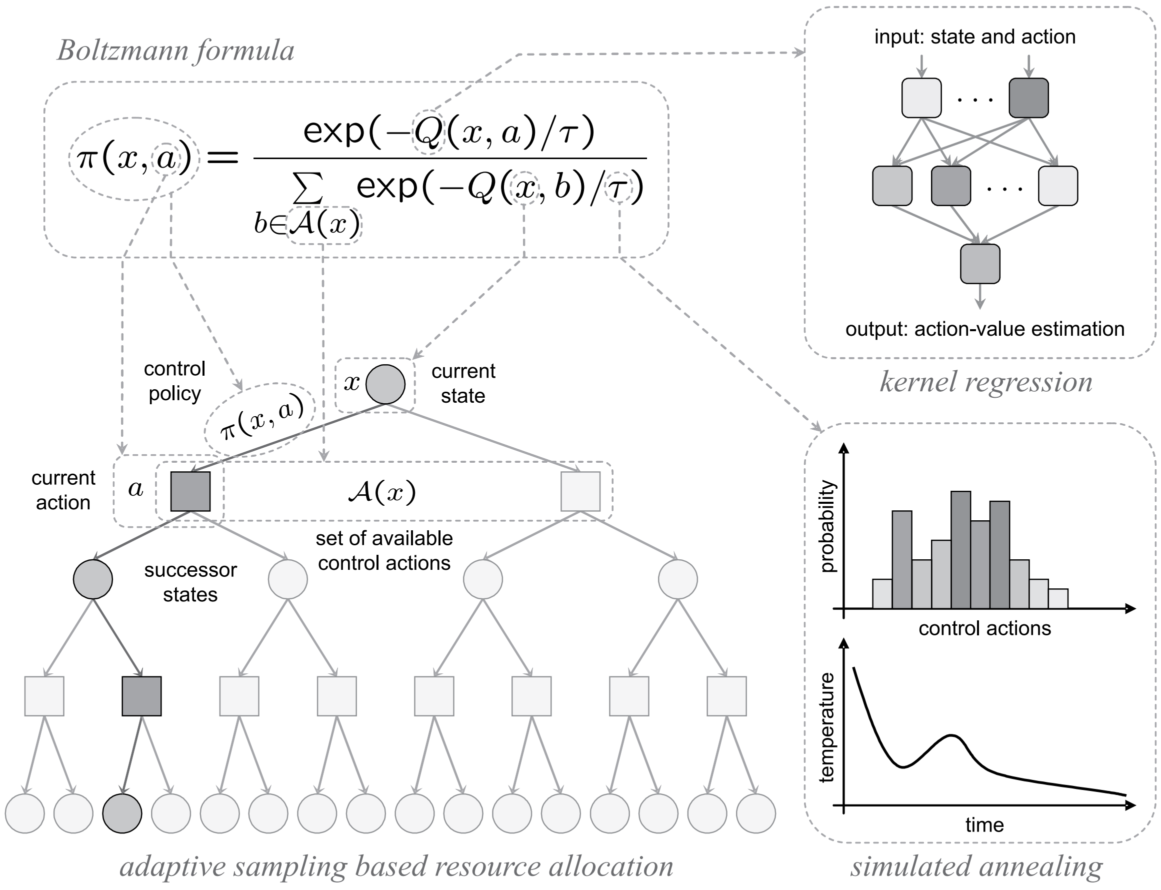


Figure 2.4: The combination of the applied machine learning techniques. The control policy is defined by the Boltzmann formula; the policy is evaluated by adaptive sampling; the value function is maintained by support vector (kernel) regression; and the exploration-exploitation ratio is controlled by simulated annealing.

episode i and x_0 is the (aggregated) initial state. State x_j^i and action a_j^i correspond to step j in episode i. Function h computes features for state-action pairs while γ_i denotes learning rates. Finally, $\mathcal{L}_i$ denotes the regression sample and Q_i is the fitted value function.

Take a look at Figure 2.4 for an illustration of combining the proposed machine learning algorithms, such as reinforcement learning, kernel regression and simulated annealing.

Though, support vector regression offers an elegant and efficient solution to the value function representation problem, we presented the hash table representation possibility not only because it is much easier to implement, but also because it requires less computation, thus, provides faster solutions. Moreover, the values of the hash table could be accessed independently; this was one of the reasons why we applied hash tables when we dealt with distributed solutions, e.g., on architectures with uniform memory access. Nevertheless, SVRs have other advantages, most importantly, they can "generalize" to "similar" data.

2.2.3 Additional Improvements

Computing a (close-to) optimal solution with RL methods, such as (fitted) Q-learning, could be very inefficient in large-scale systems, even if we apply prioritized sweeping and a capable representation. In this section we present some additional improvements in order to speed up to optimization process, even at the expense of achieving only suboptimal solutions.

Rollout Algorithms

During our experiments, presented in Section 4.1.5, it turned out that using a *suboptimal base policy*, such as a greedy policy with respect to the immediate costs, to guide the exploration, speeds up the optimization considerably. Therefore, at the initial stage we suggest applying a *rollout policy*, which is a limited lookahead policy, with the optimal cost-to-go approximated by the cost-to-go of the base policy (Bertsekas, 2001). In order to introduce the concept more precisely, let $\hat{\pi}$ be the greedy policy with respect to immediate-costs,

$$\hat{\pi}(x) \in \operatorname*{arg\,min}_{a \in \mathcal{A}(x)} g(x, a).$$

The value function of $\hat{\pi}$ is denoted by $J^{\hat{\pi}}$. The one-step lookahead rollout policy π based on policy $\hat{\pi}$, which is an improvement of $\hat{\pi}$ (cf. policy iteration), can be calculated by

$$\pi(x) \in \operatorname*{arg\,min}_{a \in \mathcal{A}(x)} \mathbb{E}\left[G(x, a) + J^{\hat{\pi}}(Y)\right],$$

where Y is a random variable representing a state generated from the pair (x, a) by simulation, that is, according to probability distribution $p(x, a)$. The expected value (viz., the expected costs and the cost-to-go of the base policy) is approximated by Monte Carlo simulation of several trajectories that start at the current state. If the problem is deterministic, then a single simulation trajectory suffices, and the calculations are greatly simplified.

Take a look at Figure 2.5 for an illustration. In scheduling theory, a similar (but simplified) concept can be found and a rollout policy would be called a *dispatching rule.*

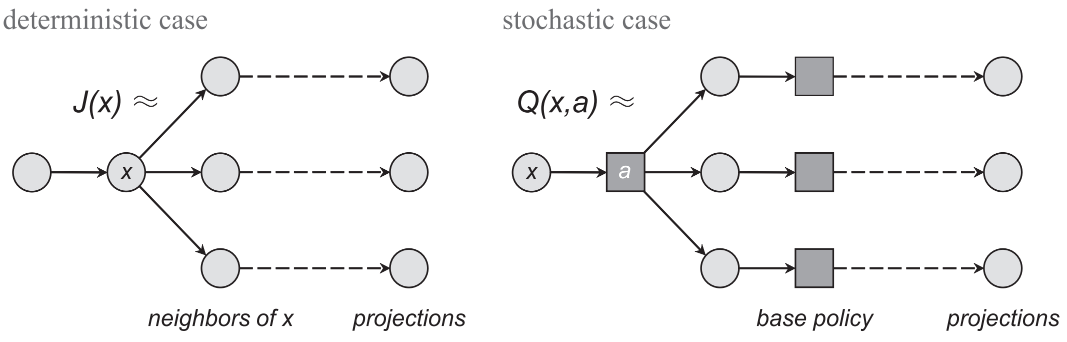


Figure 2.5: The evaluation of state x with rollout algorithms in the deterministic and the stochastic case. Circles denote states and rectangles represent actions.

The two main issues why we suggest the application of rollout algorithms are

1. We need several initial samples before the first application of approximation techniques and these first samples can be generated by simulations guided by a rollout policy.

2. General reinforcement learning methods perform quite poorly in practice without any initial guidance. However, the learning algorithm can start improving the rollout policy π, especially, in case we apply (fitted) Q-learning, it can learn directly from the trajectories generated by a rollout policy, since it is an *off-policy* learning method.

Action Space Decomposition

In large-scale problems the set of available actions in a state may be very large, which can slow down the system significantly. In the current formulation of the RAP the number of available actions in a state is $O(|\mathcal{T}|\,|\mathcal{R}|)$. Though, even in real world situations $|\mathcal{R}|$ is, usually, not very large, but $\mathcal{T}$ could contain thousands of tasks. Here, we suggest decomposing the action space as shown in Figure 2.6. First, the system selects a task, only, and it moves to a new state where this task is fixed and an executing resource should be selected. In this case the state description can be extended by a new variable $\tau \in \mathcal{T} \cup \{\emptyset\}$, where $\emptyset$ denotes the case when no task has been selected yet. In every other case the system should select an executing resource for the selected task. Consequently, the new action space is $\mathbb{A} = \mathbb{A}_1 \cup \mathbb{A}_2$, where $\mathbb{A}_1 = \{\, a_v \mid v \in \mathcal{T}\,\} \cup \{a_\omega\}$ and $\mathbb{A}_2 = \{\, a_r \mid r \in \mathcal{R}\,\}$. As a result, we radically decreased the number of available actions, however, the number of possible states was increased, as well. Nevertheless, our numerical experiments showed that it was a reasonable trade-off.

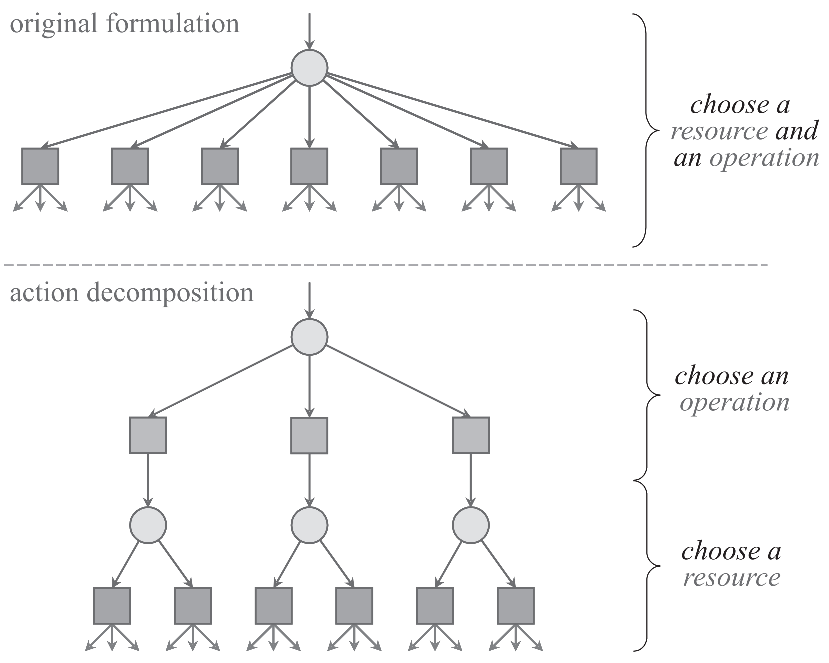


Figure 2.6: Action selection before (up) and after (down) action space decomposition.

Clustering the Tasks

The idea of divide-and-conquer is widely used in artificial intelligence and recently it has appeared in the theory of dealing with large-scale MDPs. Partitioning a problem into several smaller subproblems is also often applied to decrease computational complexity in combinatorial optimization problems, for example, in scheduling theory.

We propose a simple and still efficient partitioning method for a practically very important class of performance measures. In real world situations the tasks very often have release dates and due dates, and the performance measure, e.g., total lateness and number of tardy tasks, depends on meeting the deadlines. Note that these measures are regular. We denote the (possibly randomized) functions defining the *release* and *due dates* of the tasks by $A : \mathcal{T} \rightarrow \mathbb{N}$ and $B : \mathcal{T} \rightarrow \mathbb{N}$, respectively. In this section we restrict ourselves to performance measures that are regular and depend on due dates. In order to cluster the tasks, we need the definition of *weighted expected slack time* which is given as follows

$$S_w(v) = \sum_{s \in \Gamma(v)} w(s)\, \mathbb{E}\Big[B(v) - A(v) - D(s,v)\Big],$$

where $\Gamma(v) = \{\, s \in \mathcal{S} \mid \langle s, v\rangle \in dom(D)\,\}$ denotes the set of resource states in which task v can be processed, and $w(s)$ are weights corresponding, for example, to the likelihood that resource state s appears during execution, or they can be simply $w(s) = 1/\,|\Gamma(v)|$.

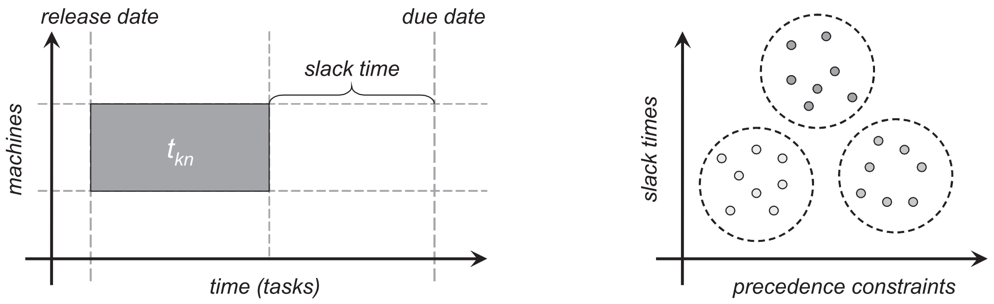


Figure 2.7: Clustering the tasks according to their slack times and precedence constraints.

In order to increase computational speed, we suggest clustering the tasks in $\mathcal{T}$ into successive disjoint subsets $\mathcal{T}_1, \dots, \mathcal{T}_k$ according to the precedence constraints and the expected slack times; take a look at Figure 2.7 for an illustration. The basic idea behind our approach is that we should handle the most constrained tasks first. Therefore, *ideally*, if $\mathcal{T}_i$ and $\mathcal{T}_j$ are two clusters and $i < j$, then tasks in $\mathcal{T}_i$ had expected slack times smaller than tasks in $\mathcal{T}_j$. However, in most of the cases clustering is not so simple, since the precedence constraints must also be taken into account and this clustering criterion has the priority. Thus, if $\langle u, v\rangle \in \mathcal{C}$, $u \in \mathcal{T}_i$ and $v \in \mathcal{T}_j$, $i \leq j$ must hold. During learning, first, tasks in $\mathcal{T}_1$ are allocated to resources, only. After some episodes, we fix the allocation policy concerning tasks in $\mathcal{T}_1$ and we start sampling to achieve a good policy for tasks in $\mathcal{T}_2$, and so on.

Naturally, clustering the tasks is a two-edged weapon, making too small clusters may seriously decrease the performance of the best achievable policy, making too large clusters may considerably slow down the system. This technique, however, has several advantages, e.g., (1) it effectively decreases the search space; (2) further reduces the number of available actions in the states; and, additionally (3) speeds up the learning, since the sample trajectories become smaller (only a small part of the tasks is allocated in a trial and, consequently, the variance of the total costs is decreased). The effects of clustering relative to the size of the clusters were analyzed experimentally and are presented in Section 4.1.5.

2.2.4 Distributed Systems

In this section we further improve the previously given solution by distributing the computation of a good control policy among several processor. Before we present our distributed approach, first, we give an overview on a few widespread distributed optimization techniques and compare some of their properties, such as the guarantees of finding an optimal (or a near optimal) solution and their robustness against disturbances, such as breakdowns.

In the past decades considerable amount of research was done to enhance decision-making, such as resource allocation, and several new paradigms appeared to distributed optimization. Distributed decision-making is often favorable (Perkins et al., 1994), not only because it can *speed up* the computation considerably, but also because it can result in more *robust* and *flexible* solutions. For example, a multi-agent based point of view combined with a heterarchical architecture can show up several advantages (Baker, 1998), such as self-configuration, scalability, fault tolerance, massive parallelism, reduced complexity, increased flexibility, reduced cost and, last but not least, emergent behavior (Ueda et al., 2001).

Agent Based Approaches

Multi-agent systems (MASs) constitute a popular approach to distributed computation in artificial intelligence. A MAS is a special distributed system with localized decision-making and, usually, localized storage. An *agent* is basically a self-directed entity with its own value system and a means to communicate with other such objects (Baker, 1998).

Multi-agent based or *holonic* manufacturing with adaptive agents received a great deal of recent attention (Baker, 1998; Monostori et al., 2006). They became a promising tool for managing complexity in manufacturing. A MAS approach to production control is useful for manufacturers who often need to change the configuration of their factories by adding or removing machines, workers, product lines, manufacturers who cannot predict the possible manufacturing scenarios according to which they will need to work in the future.

A *complex adaptive system* (CAS) can be considered as a MAS with highly adaptive agents (Holland, 1992, 1995). Agents may represent any entity with self-orientation, such as cells, species, individuals, firms or nations. Environmental conditions are changing, due to the agents' interactions as they compete and cooperate for the same resources or for achieving a given goal. This, in turn, changes the behavior of the agents themselves. The most remarkable phenomenon exhibited by a CAS is the *emergence* of highly structured collective behavior over time from the interactions of simple subsystems. The typical characteristics of a CAS include dynamics involving interrelated spatial and temporal effects, correlations over long length- and time-scales, strongly coupled degrees of freedom and non-interchangeable system elements, to name only the most important ones. Adaptive agents become important tools for managing complexity and optimizing diverse types of production control systems (Monostori, 2003; Monostori et al., 2004; Monostori and Csáji, 2006; Schuh et al., 2008).

PROSA (Van Brussel et al., 1998) is a standard MAS based architecture for manufacturing systems. The general idea underlying this approach is to consider both the resources (e.g., machines) and the jobs (interconnected tasks) as active entities. The basic architecture of the PROSA approach consists of three types of agents: *order agents* (internal logistics), *product agents* (process plans), and *resource agents* (resource handling). Resource agents correspond to physical parts (production resources in the system, such as factories, shops, machines, furnaces, conveyors, pipelines, material storages, personnel, etc.), and contain an information processing part that controls the resource. Product agents hold the process and product knowledge to assure the correct making of the product. They act like information servers to other agents. Order agents represent a task or a job (an ordered set of tasks) in the manufacturing system. They are responsible for performing the assigned work correctly, effectively and on time. Though, the PROSA architecture is a production control approach, it is only a framework, it does not offer any direct resource allocation solutions. It is only a possible starting point for designing multi-agent based manufacturing systems.

A lot of distributed optimization techniques were inspired by various biological systems (Kennedy and Eberhart, 1995), such as bird flocks, wolf packs, fish schools, termite hills or ant-colonies. These approaches can show up strongly robust and parallel behavior. The *ant-colony* optimization algorithm (Moyson and Manderick, 1988) is, in general, a distributed and randomized algorithm to solve *shortest path* problems in graphs. RAPs can be formulated as special stochastic shortest path problems, as well, as we saw in Section 2.1.3.

The PROSA architecture can also be extended by ant-colony type optimization methods (Hadeli et al., 2004). In this case a new type of agent is introduced, called *ant*. Agents of this type are mobile and they gather and distribute information in the manufacturing system. The main assumption is that the agents are much faster than the ironware that they control, and that makes the system capable to forecast. Agents are faster and, therefore, can emulate the behavior of the system several times before the actual decision is taken. The resource allocation in this system is made by local decisions. Each order agent sends ants (mobile-agents), which are moving downstream in a virtual manner. They gather information about the possible schedules from the resource agents and then they return to the order agent with the information. The order agent chooses a schedule and then it sends ants to book the needed resources. After that the order agent regularly sends booking ants to re-book the previously found best schedule, because if the booking is not refreshed then it *evaporates* after a while (like the *pheromone* in the analogy of food-foraging ants). From time to time the order agent sends ants to survey the possible new (and better) schedules. If they find a better solution, the order agent sends ants to book the resources that are needed for the new schedule and the old booking information will simply evaporate, eventually.

Swarm optimization methods are very robust, they can naturally adapt to disturbances and environmental changes, since the ants continuously explore the current situation and the obsolete data simply evaporates if not refreshed regularly. However, these techniques often have the disadvantage that finding an optimal or even a relatively good solution cannot be easily guaranteed, theoretically. The ant-colony based extension of PROSA, for example, faces almost exclusively the *routing* problem in resource allocation and it mostly ignores *sequencing* problems, namely, it does not concern with the efficient ordering of the tasks.

There are multi-agent systems which use some kinds of *negotiation* or market-based mechanism (Márkus et al., 1996). In this case, the tasks or the jobs are associated with order agents, while the resources are controlled by resource agents, like in the case of PROSA.

Market-based resource allocation is a recursive, iterative process with announce-bid-award cycles. During resource allocation the tasks are announced to the agents that control the resources, and they can bid for the available works. The jobs or tasks are, usually, announced one-by-one, which can lead to *myopic* behavior and, therefore, guaranteeing an optimal or even an approximately good solution is often very hard. Regarding adaptive behavior, market-based resource allocation is often less robust than swarm optimization methods, such as ant-colony optimization, however, their performance is usually better.

Parallel Optimization

It is known (Modi et al., 2001) that resource allocation problems (at least their deterministic variants) can be often formulated as *constraint satisfaction* (CS) problems. In this case, they aim at solving a multi-dimensional, constraint optimization problem, defined as

$$\begin{aligned} &\textit{optimize} && f(x_1, x_2, \dots, x_n), \\ &\textit{subject to} && g_j(x_1, x_2, \dots, x_n) \leq c_j, \end{aligned}$$

where $x_i \in \mathcal{X}_i$, $i \in \{1, \dots, n\}$ and $j \in \{1, \dots, m\}$. Functions f and g_j are real-valued and $c_j \in \mathbb{R}$, as well. Moreover, most RAPs, for example, resource constrained project scheduling, can be formulated as a *linear programming* (LP) problem, which can be written as follows

$$\begin{aligned} &\textit{optimize} && \langle c, x \rangle, \\ &\textit{subject to} && Ax \leq b, \end{aligned}$$

where $A \in \mathbb{R}^{n \times m}$, $c \in \mathbb{R}^n$, $b \in \mathbb{R}^m$ and $\langle \cdot, \cdot \rangle$ denotes inner product. Then, distributed variants of *constrained optimization* approaches can be applied to compute a solution. In this case, a close-to-optimal solution is often guaranteed, however, the computation time is usually large. The main problems with these approaches are that they cannot easily take uncertainties into account and, moreover, they are not robust against disturbances.

The idea of divide-and-conquer is often applied to decrease computational complexity when dealing with combinatorial optimization problems. The main idea behind this approach is to *decompose* the problem and solve the resulted sub-problems independently. In most cases calculating the sub-solutions can be done in a distributed way (Wu et al., 2005).

These approaches can be effectively applied in many cases, however, defining a decomposition which guarantees both efficient computational speedup together with the property that combining the optimal solutions of the sub-problems results in a global optimal solution is very demanding. Therefore, when we apply decomposition, we usually have to give-up optimality and satisfy with fast but sometimes far-from-optimal solutions. Moreover, it is hard to make these systems robust against disturbances. Tracking environmental changes can be often accomplished by the complete recalculation of the whole solution, only.

Distributed Sampling

Finally, we argue that the approach presented in Section 2.2 can be easily and efficiently distributed among several processors with *distributed sampling.* We will consider extensions of the algorithm using both *shared memory* and *distributed memory* architectures. Take a look at Figure 2.8 for an illustration of these architectures. We suppose that we have k processors, and let us denote the set of all processors by $\mathcal{P} = \{p_1, p_2, \dots, p_k\}$.

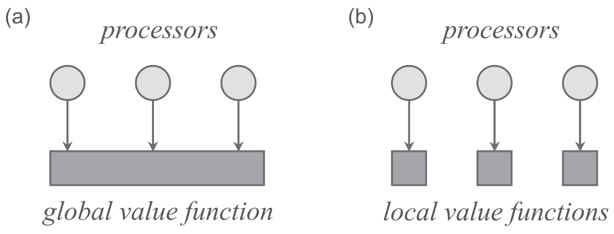


Figure 2.8: Shared (a) and distributed (b) memory access.

In case we have a parallel system with a shared memory architecture, e.g., UMA (uniform memory access), then it is straightforward to parallelize the computation of a control policy. Namely, each processor $p \in \mathcal{P}$ can sample the search space independently, while by using the same, shared value function. The (joint) control policy can be calculated using this common, global value function, e.g., the greedy policy w.r.t. this function can be applied.

Parallelizing the solution by using an architecture with distributed memory is more challenging. Probably the simplest way to parallelize our approach to several processors with distributed memory is to let the processors search independently by letting them working with their own, local value functions. After a given time or number of iterations, we may treat the best achieved solution as the joint policy. More precisely, if we denote the aggregated initial state by x_0, then the joint control policy π can be defined as follows

$$\pi \in \operatorname*{arg\,min}_{\pi_p\,(p \in \mathcal{P})} J^{\pi_p}(x_0) \qquad \text{or} \qquad \pi \in \operatorname*{arg\,min}_{\pi_p\,(p \in \mathcal{P})} \min_{a \in \mathcal{A}(x_0)} Q^{\pi_p}(x_0, a),$$

where J^{π_p} and Q^{π_p} are (approximate) state- and action-value functions calculated by processor $p \in \mathcal{P}$. Control policy π_p is the solution of processor p after a given number of iterations. During our numerical experiments we usually applied 10 000 iterations per processor.

Naturally, there could be many (more sophisticated) ways to parallelize the computation using several processors with distributed memory. From time to time, e.g., the processors could exchange some of their best episodes (trajectories with the lowest costs) and learn from the experiments of the others. In this way, they could help improving the value functions of each other. Our numerical experiments, presented in Section 4.1.3, showed that even in the simplest case, distributing the calculation speeds up the optimization considerably. Moreover, in case of applying a shared memory architecture, the speedup was almost linear.

As parallel computing represents a very promising way do deal with large-scale systems, their further theoretical and experimental investigation would be very important. For example, by harmonizing the exploration of the processors, the speedup could be improved.

Chapter 3

Varying Environments

In Chapter 2 we investigated stochastic RAPs in which some of the components, e.g., the executing times, were *uncertain*. We took the approach to associate the uncertain parameters with random variables. So far, only the *stationary* case was studied, however, many problems that appear in practise come from the sudden *changes* of the environment. In this chapter we analyze how MDP based approaches could work in these circumstances.

The dynamics of stochastic RAPs presented in Chapter 2 can be modeled as MDPs, but what happens when the model was *wrong*, e.g., if the real transition-probabilities were different, or the dynamics had *changed* meanwhile? Naturally, the changing of the dynamics can also be modeled as a (higher level) MDP, however, in this way the problem could very easily become practically intractable. In "real word" problems computing an optimal or even a relatively good policy is extremely time-consuming and the needed computation grows rapidly with the number of states. Including environmental changes in the model is very likely to lead to problems which do not have any practically efficient solution method.

In what follows, we argue that if the model was "close" to the environment, then the performance of a control *policy* in the model would also be "close" to the performance of the same policy applied in the real environment and, moreover, "slight" changes in the environment result only in "slight" changes in the *value functions*. More precisely, we show that the value function of a (stationary, Markovian, randomized) control policy in a discounted MDP *Lipschitz continuously* depends on the *immediate-cost* and the *transition-probability* functions. We also present value function bounds for the case of changes in the *discount factor* and demonstrate through an example that this dependence is not Lipschitz continuous. Even though *optimal* policies might also change if the environment has changed, we will show that these results can be extended to optimal value functions, as well.

As a consequence, it is not unconditionally important to include small dynamics changes in the model. If the dynamics have changed, then the old and obsolete cost-to-go values can be effectively used as starting points for adapting, since they will not be very far from the changed value function of the new environment, if the change was not radical. Therefore, for slowly varying systems, we can safely avoid modeling the changes of the dynamics if we use a method that iteratively approximates the optimal value function. This approach is also supported by the results of our numerical experiments, presented in Chapter 4.

Afterwards, in order to theoretically analyze learning in changing environments, the concept of (ε, δ)-MDPs is introduced in which the transition-probability and the immediate-cost functions are allowed to *vary* over time, as long as the cumulative changes remain *asymptotically bounded.* Then, we study learning algorithms in changing environments. In order to do this, first, a general framework for analyzing *stochastic iterative algorithms* is presented. A novelty of our approach is that we allow the value function update operator to be *time-dependent.* Later, we apply this framework to deduce an *approximate convergence theorem* for time-dependent stochastic iterative algorithms. With the help of this general convergence theorem we also show relaxed convergence properties (more precisely, κ-approximation) for value function based reinforcement learning methods working in (ε, δ)-MDPs.

3.1 Changes in the Dynamics

In many control problems it is typically not possible to "practise" in the real environment, only a dynamic *model* is available to the system and this model can be used for predicting how the environment will respond to the control signals (model predictive control). MDP based solutions usually work by *simulating* the environment with the model, through simulation they produce *simulated experience* and by *learning* from these experience they improve their value functions. Computing an approximately optimal value function is essential because, as we have seen (Theorem 6), close approximations to optimal value functions lead directly to good control policies. Though, there are alternative approaches which directly approximate optimal control policies (see Sutton et al., 2000). However, what happens if the model was inaccurate or the environment had changed slightly? In what follows we investigate the effects of inaccurate models and environmental changes. For continuous Markov processes questions like these were already analyzed (Gordienko and Salem, 2000; Favero and Runggaldier, 2002; Montes de Oca et al., 2003), hence, we focus on *finite* MDPs.

3.1.1 Transition Changes

First, we will see that the value function of a Markovian control policy in *discounted* MDP Lipschitz continuously depends on the applied transition-probability function.

Theorem 9 *Assume that two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, differ only in their transition-probability functions, and let these two functions be denoted by p_1 for $\mathcal{M}_1$ and p_2 for $\mathcal{M}_2$. Let the value functions of a (stationary, Markovian, randomized) control policy π be denoted by J_1^π related to $\mathcal{M}_1$ and J_2^π related to $\mathcal{M}_2$. Then, the following inequality holds*

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{\alpha\,|\mathbb{X}|\,\|g\|_\infty}{(1-\alpha)^2}\,\|p_1 - p_2\|_\infty\,.$$

The proof of Theorem 9 can be found in the appendix. The same estimation was derived by Kalmár et al. (1998), but only for the case of *optimal* value functions. The dependence of the optimal value function of an MDP on the transition-probabilities was analyzed by Müller (1996), as well. Later, we will see that our theorems on the changes of the value function of a fixed policy can also be extended to the case of optimal value functions.

A disadvantage of this theorem is that the estimation heavily depends on the size of the state space, $n = |\mathbb{X}|$. However, this bound can be improved if we consider an induced matrix norm for transition-probabilities instead of the supremum norm. The following theorem presents our improved estimation, its proof can also be found in the appendix.

Theorem 10 *With the assumptions and notations of Theorem 9, we have*

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{\alpha \, \|g\|_\infty}{(1-\alpha)^2} \, \|p_1 - p_2\|_1 \, ,$$

where $\|\cdot\|_1$ *is a norm on* $f : \mathbb{X} \times \mathbb{A} \times \mathbb{X} \to \mathbb{R}$ *type functions, e.g.,* $f(x, a, y) = p(y \,|\, x, a)$,

$$\|f\|_1 = \max_{x,\, a} \sum_{y \in \mathbb{X}} |\, f(x, a, y)\,| \, . \tag{3.1}$$

If we consider f as a matrix which has a column for each state-action pair $(x, a) \in \mathbb{X} \times \mathbb{A}$ and a row for each state $y \in \mathbb{X}$, then the above definition gives us the usual "maximum absolute column sum norm" definition for matrices, which is conventionally denoted by $\|\cdot\|_1$.

It is easy to see that for all f, we have $\|f\|_1 \leq n \, \|f\|_\infty$, where n is size of the state space. Therefore, the estimation of Theorem 10 is at least as good as the estimation of Theorem 9. In order to see that it is a real improvement consider, e.g., the case when we choose a particular state-action pair, $(\hat{x}, \hat{a})$, and take a p_1 and p_2 that only differ in $(\hat{x}, \hat{a})$. For example, $p_1(\hat{x}, \hat{a}) = \langle 1, 0, 0, \ldots, 0\rangle$ and $p_2(\hat{x}, \hat{a}) = \langle 0, 1, 0, \ldots, 0\rangle$, and they are equal for all other $(x, a) \neq (\hat{x}, \hat{a})$. Then, by definition, $\|p_1 - p_2\|_1 = 2$, but $n \, \|p_1 - p_2\|_\infty = n$. Consequently, in this case, we have improved the bound of Theorem 9 by a factor of $2/n$.

3.1.2 Cost Changes

In some situations changes in the cost function should also be taken into account. This problem could be important for resource allocation, since, e.g., a resource might go wrong and it should be replaced by another one (e.g., an external resource) which has much higher operational cost. Another example could be the changing of task priorities (e.g., a task becomes urgent, its due date becomes much tighter) which can also be modeled through the changing of the immediate-cost function. The following theorem shows that a similar Lipschitz continuity type dependence can be proven for the case of changes in the immediate-cost function as for the case of changes in the transition-probability function.

Theorem 11 *Assume that two discounted MDPs,* $\mathcal{M}_1$ *and* $\mathcal{M}_2$*, differ in their immediate-cost functions only, and let these two functions be denoted by* g_1 *for* $\mathcal{M}_1$ *and* g_2 *for* $\mathcal{M}_2$*, respectively. Let the value functions of a (stationary, Markovian, randomized) policy* π *be denoted by* J_1^π *related to* $\mathcal{M}_1$ *and* J_2^π *related to* $\mathcal{M}_2$*. Then, the following inequality holds*

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{1}{1-\alpha} \, \|g_1 - g_2\|_\infty \, .$$

Our proof of Theorem 11, together with all of the proofs, can be found in the appendix.

3.1.3 Discount Changes

The following theorem shows that the change of the value function can also be estimated in case there were changes in the discount rate (all proofs can be found in the appendix).

Theorem 12 *Assume that two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, differ only in the discount factors, denoted by $\alpha_1, \alpha_2 \in [0,1)$. Let the value functions of a (stationary, Markovian, randomized) policy π be denoted by J_1^π related to $\mathcal{M}_1$ and J_2^π related to $\mathcal{M}_2$. Then, we have*

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{|\alpha_1 - \alpha_2|}{(1-\alpha_1)(1-\alpha_2)} \|g\|_\infty .$$

The next example demonstrates, however, that this dependence is not Lipschitz continuous. Consider, e.g., an MDP that has only one state x and one action a. Taking action a loops back deterministically to state x with cost $g(x,a) = 1$. Therefore, the only available control policy takes the form of $\pi(x,a) = 1$. Suppose that the MDP has discount factor $\alpha_1 = 0$, thus, $J_1^\pi(x) = 1$. Now, if we change the discount rate to $\alpha_2 \in (0,1)$, then $|\alpha_1 - \alpha_2| < 1$ but $\|J_1^\pi - J_2^\pi\|_\infty$ could be arbitrarily large, since $J_2^\pi(x) \to \infty$ as $\alpha_2 \to 1$.

At the same time, we can notice that if we fix a constant $\alpha_0 < 1$ and only allow discount factors from the interval $[0, \alpha_0]$, then this dependence became Lipschitz continuous, as well.

3.1.4 Action-Value Changes

Many reinforcement learning algorithms, such as Q-learning and SARSA, work with action-value functions which are important, e.g., for model-free approaches. Now, we investigate how the previously presented theorems apply to this type of value functions. The action-value function of policy π, denoted by Q^π, can be rewritten with the help of J^π as follows

$$Q^\pi(x,a) = g(x,a) + \alpha \sum_{y \in \mathbb{X}} p(y \,|\, x,a) J^\pi(y),$$

where J^π is the (state-) value function of policy π. Note that in the case of action-value functions, first, we take a given action (which can have very high immediate-cost) and, only after the action was taken, follow we the actual policy. Thus, we can estimate $\|Q^\pi\|_\infty$ by

$$\|Q^\pi\|_\infty \leq \|g\|_\infty + \alpha \|J^\pi\|_\infty .$$

Nevertheless, the next lemma shows that the same estimations can be derived for environmental changes in the case of action-value functions as in the case of value functions.

Lemma 13 *Assume that we have two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, which differ only in the transition-probability functions or only in the immediate-cost functions or only in the discount factors. Let π denote an arbitrary (stationary, Markovian, randomized) control policy. The (state-) value and action-value functions of control policy π are denoted by J_1^π, Q_1^π and J_2^π, Q_2^π for $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. Then, the previously given value function bounds for $\|J_1^\pi - J_2^\pi\|_\infty$ of Theorems 9, 10, 11 and 12 are also bounds for $\|Q_1^\pi - Q_2^\pi\|_\infty$.*

3.1.5 Optimal Cost-to-Go Changes

Another interesting question is the effects of environmental changes on the *optimal* value function. Theorems 9, 10, 11 and 12 cannot be directly applied to deduce estimations for the case of optimal value functions, since they presuppose that the policy is *fixed*, however, optimal policies might also change if the environment had changed. Nevertheless, with the help of the following lemma, the case of optimal value functions becomes a consequence of the previous theorems. The proof of Lemma 14 can be found in the appendix, as well.

Lemma 14 *For all* $f_1, f_2 : \mathcal{X} \to \mathbb{R}$ *bounded functions such that* $\min_x f_1(x) \leq \min_x f_2(x)$ *and* $\hat{x} = \arg\min_x f_1(x)$*, we have the inequality* $|\min_x f_1(x) - \min_x f_2(x)| \leq |f_1(\hat{x}) - f_2(\hat{x})|$*.*

Assume that we have two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, which may differ in their transition-probability functions, immediate-cost functions and discount factors. In order to deduce bounds for the distance of their optimal value functions, first, let us introduce an auxiliary (stationary, Markovian) deterministic control policy, defined as follows

$$\hat{\pi}(x) = \begin{cases} \pi_1^*(x) & \text{if } J_1^*(x) \leq J_2^*(x), \\ \\ \pi_2^*(x) & \text{if } J_2^*(x) < J_1^*(x), \end{cases}$$

where π_i^* denotes an *optimal* control policy in $\mathcal{M}_i$ and J_i^* is the optimal value function of $\mathcal{M}_i$, $i \in \{1, 2\}$. Policy π_i^* can be, e.g., a *greedy* policy w.r.t. J_i^* (cf. Theorem 6). Then, by using the Bellman equation in the first step and Lemma 14 in the second, we have

$$\forall x \in \mathbb{X} : |J_1^*(x) - J_2^*(x)| =$$

$$= \Bigg| \min_{a \in \mathcal{A}(x)} \Big[g_1(x,a) + \alpha_1 \sum_{y \in \mathbb{X}} p_1(y \mid x, a)\, J_1^*(y) \Big] -$$

$$- \min_{a \in \mathcal{A}(x)} \Big[g_2(x,a) + \alpha_2 \sum_{y \in \mathbb{X}} p_2(y \mid x, a)\, J_2^*(y) \Big] \Bigg| \leq$$

$$\leq \Bigg| g_1(x, \hat{\pi}(x)) + \alpha_1 \sum_{y \in \mathbb{X}} p_1(y \mid x, \hat{\pi}(x))\, J_1^*(y) -$$

$$- g_2(x, \hat{\pi}(x)) - \alpha_2 \sum_{y \in \mathbb{X}} p_2(y \mid x, \hat{\pi}(x))\, J_2^*(y) \Bigg| =$$

$$= \left| J_1^{\hat{\pi}}(x) - J_2^{\hat{\pi}}(x) \right|.$$

Therefore, with the help of Lemma 14, we reduced the case of changes in the optimal value function to the case of changes in the value function of a *fixed* policy, namely $\hat{\pi}$. This result is precisely stated in Corollary 15. It allows us to extend Theorems 9, 10, 11 and 12 to the case of optimal value functions, as well, which consequence is summarized by Corollary 16.

Corollary 15 *Assume that we have two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, which may differ in their transition functions, cost functions and discount factors. Let us denote the optimal value functions of $\mathcal{M}_1$ and $\mathcal{M}_2$ by J_1^* and J_2^*, respectively. Then, there exists a (stationary, Markovian, deterministic) control policy π such that $\|J_1^* - J_2^*\|_\infty \leq \|J_1^\pi - J_2^\pi\|_\infty$.*

Since deterministic control policies are special cases of randomized ones, we also have:

Corollary 16 *Assume that we have two discounted MDPs which differ only in the transition-probability functions or only in the immediate-cost functions or only in the discount factors. Let the corresponding optimal value functions be denoted by J_1^* and J_2^*, respectively. Then, the bounds for $\|J_1^\pi - J_2^\pi\|_\infty$ of Theorems 9, 10, 11 and 12 are also bounds for $\|J_1^* - J_2^*\|_\infty$.*

It is straightforward to see that these results can be extended to optimal *action-value* functions, as well. One should apply the same "trick" as before and introduce an auxiliary control policy by which the difference of the optimal action-value functions can be estimated from above. In fact, the same control policy, namely $\hat{\pi}$, can be applied to achieve this.

Corollary 17 *Assume that we have two discounted MDPs which differ only in the transition-probability functions or only in the immediate-cost functions or only in the discount factors. The corresponding optimal action-value functions are denoted by Q_1^* and Q_2^*, respectively. Then, the value function bounds for $\|J_1^\pi - J_2^\pi\|_\infty$ are also bounds for $\|Q_1^* - Q_2^*\|_\infty$.*

3.1.6 Further Remarks

In the previous parts of the section we saw that value functions of discounted MDPs depend smoothly on the transition-probability function, the immediate-cost function and the discount rate. This dependence is of Lipschitz type in the first two cases and non-Lipschitz for discount rates. Later, we will see that changes in the discount factor can be traced back to changes in the immediate-cost function (Lemma 24), therefore, it is sufficient to consider transition and cost changes. The following corollary summarizes some of the results.

Corollary 18 *Assume that two discounted MDPs, $\mathcal{E}$ and $\mathcal{M}$, differ only in their transition-probability functions and their immediate-cost functions. The corresponding transition and cost functions are denoted by $p_\mathcal{E}$, $p_\mathcal{M}$ and $g_\mathcal{E}$, $g_\mathcal{M}$. Let the value functions of a (stationary, Markovian, randomized) control policy π be denoted by $J_\mathcal{E}^\pi$ and $J_\mathcal{M}^\pi$, respectively. Then,*

$$\|J_\mathcal{E}^\pi - J_\mathcal{M}^\pi\|_\infty \leq \frac{\|g_\mathcal{E} - g_\mathcal{M}\|_\infty}{1-\alpha} + \frac{c\,\alpha\,\|p_\mathcal{E} - p_\mathcal{M}\|_1}{(1-\alpha)^2},$$

where $c = \min\{\|g_\mathcal{E}\|_\infty, \|g_\mathcal{M}\|_\infty\}$ and $\alpha \in [0,1)$ is, as usually, the discount factor.

The proof simply follows from Theorems 10 and 11 and from the triangle inequality. If we treat one of the MDPs in the previous theorems as a system which describes the "real" behavior of the environment and the other MDP as our model relatedly, then these results show that even if the model used is slightly inaccurate or there were changes in the environment, the optimal value function based on the model cannot be arbitrarily wrong from the viewpoint of the environment. These theorems are of special interest because in "real world" problems the transitions and the costs are mostly estimated only, e.g., by statistical methods from historical data, the exact values are unknown and may even change over time.

Now, we investigate few supplementary results of the presented statements, e.g., some connections with other results as well as two counterexamples for the undiscounted case.

Average Cost Case

Though, the largest part of the MDP related research studies the expected total discounted cost optimality criterion, in some cases discounting is inappropriate and, therefore, there are alternative optimality approaches, as well. A popular alternative approach is to optimize the *expected average cost* (Bertsekas, 2001). In this case the value function is defined as

$$J^{\pi}(x) = \limsup_{N\to\infty} \frac{1}{N}\, \mathbb{E}\left[\sum_{t=0}^{N-1} g(X_t, A_t^{\pi}) \;\middle|\; X_0 = x \right],$$

where the notations are the same as previously, e.g., as applied in equation (1.1).

Regarding the validity of Corollary 16 concerning MDPs with the average cost minimization objective, we can recall the well-known result that, in the case of finite MDPs, discounted cost offers a good approximation to the other optimality criterion. More precisely, it can be shown that there exists a large enough $\alpha_0 < 1$ such that $\forall\, \alpha \in (\alpha_0, 1)$ optimal control policies for the discounted cost problem are also optimal for the average cost problem (Feinberg and Shwartz, 2002). These policies are called *Blackwell optimal.*

Simulation Lemma

The results presented in this section have some similarities with the "Simulation Lemma" of Kearns and Singh (2002). They apply that lemma to deduce polynomial time bounds to achieve near-optimal return in MDPs. The Simulation Lemma states that if two MDPs differ only in their transition functions and their cost functions and we want to approximate (w.r.t. supremum norm) the value function of a fixed control policy concerning one of the MDPs in the other MDP, then how close should we choose the transition-probabilities and the immediate-costs to the original MDP relative to the mixing time (in the undiscounted case) or the horizon time (in the discounted case). Nonetheless, they arrive at a different kind of bound than we did and, additionally, they only consider the case when the distances of both the transition and the cost functions are determined by the same parameter.

State and Action Changes

In theory, changes in the state space can be traced back to changes in the control action space, because, for example, the deletion of a state can be treated as the deletion of all actions which lead to it, and moreover, adding a new state can be treated as adding some new actions that lead to a state that was previously unreachable. Additionally, changes in the control space can be traced back to changes in the cost function or in the transition-probability function. We can treat, e.g., the costs of all controls which are unavailable as very-high (quasi-infinity) or actions could have the same transition-probability but, after changes, they could become different mimicking the effect of appearing new control actions.

However, in these cases the presented theorems do not provide efficient estimations for measuring the changes in the value function of a policy. It is easy to construct examples (analogously to the examples presented in Section 3.1.6) which show that inserting a single new state or action may change the optimal cost-to-go function radically, e.g., if the cost of the new action is much less than the costs of the actions already in the system. We do not concern with these cases in the thesis and leave their investigation for further work.

Counterexamples

The previously presented theorems, lemmas and corollaries are only valid in case we consider *discounted* MDPs, when $\alpha \in [0, 1)$. On the other hand, it is easy to construct examples which demonstrate that in case of using *undiscounted* MDPs, an arbitrarily small change either in the transition-probabilities or in the immediate-costs may lead to arbitrarily large changes in the optimal cost-to-go function. Now, we demonstrate this phenomenon by two examples.

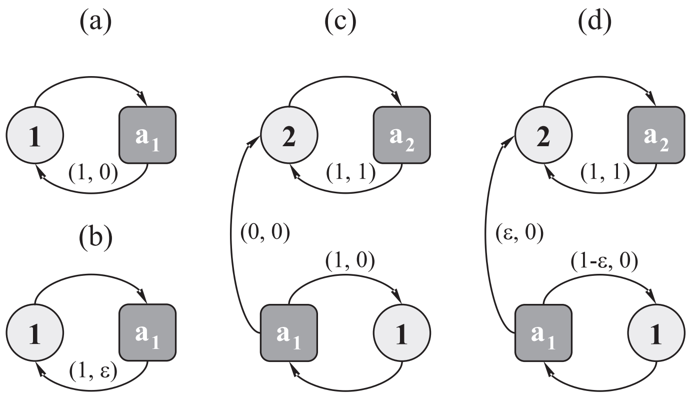


Figure 3.1: Demonstrative examples. The notation (p, g) on the arrows shows that the control action takes to the pointed state with probability p and cost g.

First, let $\mathbb{X} = \{x_1\}$, $\mathbb{A} = \{a_1\}$, $\mathcal{A}(x_1) = \{a_1\}$, $p(x_1 \mid x_1, a_1) = 1$ and $g_1(x_1, a_1) = 0$. In this case $J_1^*(x_1) = 0$. If we define $g_2(x_1, a_1) = \varepsilon$ for an $\varepsilon > 0$, then $\|g_1 - g_2\|_\infty \leq \varepsilon$, but it is easy to see that $J_2^*(x_1) = +\infty$. Take a look at parts (a) and (b) of Figure 3.1.

Now, for the case of probability transition functions, let $\mathbb{X} = \{x_1, x_2\}$; $\mathbb{A} = \{a_1, a_2\}$; $\mathcal{A}(x_1) = \{a_1\}$, $\mathcal{A}(x_2) = \{a_2\}$; $p_1(x_1 \mid x_1, a_1) = 1$, $p_1(x_2 \mid x_2, a_2) = 1$; $g(x_1, a_1) = 0$, $g(x_2, a_2) = 1$. In this case $J_1^*(x_1) = 0$. If we define the other transition-probability function as $p_2(x_1 \mid x_1, a_1) = 1 - \varepsilon$ and $p_2(x_2 \mid x_1, a_1) = \varepsilon$ for an $\varepsilon \in (0, 1]$, then $\|p_1 - p_2\|_\infty \leq \varepsilon$, but $J_2^*(x_1) = +\infty$. Take a look at parts (c) and (d) of Figure 3.1 for an illustration.

This undesirable feature of undiscounted MDPs is an effect of the infinite cost cycles appearing in the system. Nevertheless, if the transition graph of an undiscounted MDP is *acyclic* (viz., for each state the probability of returning to that state is zero), it is straightforward to prove that the optimal value function of the MDP also depends Lipschitz continuously on the transition-probability function and the immediate-cost function. Additionally, the same statement is valid even if an undiscounted MDP has cycles but it has *finite horizon*, since any change can be propagated only in finite number of steps.

3.2 Learning in Varying Environments

Now, we turn our attention to learning algorithms acting in changing environments. First, a unified framework to analyze value function based methods is presented followed by the description of (ε, δ)-MDPs, which is a class of non-stationary MDPs. Finally, we present a relaxed convergence theorem for time-dependent stochastic iterative algorithms.

3.2.1 Unified Learning Framework

In this section we will briefly overview a unified framework to analyze value function based reinforcement learning algorithms. We will use this approach later when we prove convergence properties in changing environments. The theory presented in this section (3.2.1) was developed by Szepesvári and Littman (1999) and was extended by Szita et al. (2002).

Generalized Value Functions

Throughout the thesis we denote the set of value functions by $\mathcal{V}$ which contains, in general, all bounded real-valued functions over an arbitrary set $\mathcal{X}$, e.g., $\mathcal{X} = \mathbb{X}$, in the case of state-value functions, or $\mathcal{X} = \mathbb{X} \times \mathbb{A}$, in the case of action-value functions. Note that the set of value functions, $\mathcal{V} = \mathcal{B}(\mathcal{X})$, where $\mathcal{B}(\mathcal{X})$ denotes the set of all bounded real-valued functions over set $\mathcal{X}$, is a normed space, for example, with the supremum norm. Naturally, bounded functions constitute no real restriction in case of analyzing finite MDPs.

Kappa Approximation

In order to study convergence properties in changing, *non-stationary* environments, we will apply a relaxed form of the classical *almost sure* or *probability one* converge. This relaxed convergence concept is called *κ-approximation* and defined as follows (Szita et al., 2002).

Definition 19 *We say that a sequence of random variables, denoted by X_t, κ-approximates random variable X with $\kappa > 0$ if for all $\varepsilon > 0$, there exits an index t_0 such that*

$$\mathbb{P}\left(\sup_{t>t_0} (\|X_t - X\| \leq \kappa) \right) > 1 - \varepsilon. \tag{3.2}$$

An equivalent definition of κ-approximation can be given as $\limsup_{t\to\infty} \|X_t - X\| \leq \kappa$ with probability one. Hence, the "meaning" of this definition is that sequence X_t converges almost surely to an environment of X and the radius of this environment is less than or equal to a given κ. Note that this definition is weaker (more general) than the probability one convergence, because parameter κ is fixed. If we required inequality (3.2) for all $\kappa > 0$, then we would get back to the classical probability one (almost sure) convergence.

Generalized Value Iteration

A general form of value iteration type algorithms can be given as follows,

$$V_{t+1} = H_t(V_t, V_t),$$

where H_t is a random operator on $\mathcal{V} \times \mathcal{V} \to \mathcal{V}$ (Szepesvári and Littman, 1999). Consider, e.g., the SARSA (State-Action-Reward-State-Action) algorithm which is a model-free policy evaluation method. It aims at finding Q^π for a given policy π and it is defined as

$$Q_{t+1}(x,a) = (1 - \gamma_t(x,a))\, Q_t(x,a) + \gamma_t(x,a)(g(x,a) + \alpha\, Q_t(Y,B)),$$

where $\gamma_t(x,a)$ denotes the stepsize associated with state x and action a at time t; Y and B are random variables, Y is generated from the pair (x,a) by simulation, that is, according to the distribution $p(x,a)$, and the distribution of B is $\pi(Y)$. In this case, H_t is defined as

$$H_t(Q_a, Q_b)(x,a) = (1 - \gamma_t(x,a))\, Q_a(x,a) + \gamma_t(x,a)(g(x,a) + \alpha\, Q_b(Y,B)), \tag{3.3}$$

for all x and a. Therefore, the SARSA algorithm takes the form $Q_{t+1} = H_t(Q_t, Q_t)$.

Definition 20 *We say that the operator sequence H_t κ-approximates operator $H : \mathcal{V} \to \mathcal{V}$ at $V \in \mathcal{V}$ if for any initial $V_0 \in \mathcal{V}$ the sequence $V_{t+1} = H_t(V_t, V)$ κ-approximates HV.*

Asymptotic Convergence Bounds

The next theorem (Szita et al., 2002) will be an important tool for proving convergence results for value function based RL algorithms working in varying environments.

Theorem 21 *Let H be an arbitrary mapping with fixed point V^*, and let H_t κ-approximate H at V^* over set $\mathcal{X}$. Additionally, assume that there exist random functions $0 \leq F_t(x) \leq 1$ and $0 \leq G_t(x) \leq 1$ satisfying the four conditions below with probability one*

1. *For all $V_1, V_2 \in \mathcal{V}$ and for all $x \in \mathcal{X}$,*

$$|H_t(V_1, V^*)(x) - H_t(V_2, V^*)(x)| \leq G_t(x)\, |V_1(x) - V_2(x)|\,.$$

2. *For all $V_1, V_2 \in \mathcal{V}$ and for all $x \in \mathcal{X}$,*

$$|H_t(V_1, V^*)(x) - H_t(V_1, V_2)(x)| \leq F_t(x)\, \|V^* - V_2\|_\infty\,.$$

3. *For all $k > 0$, $\prod_{t=k}^{n} G_t(x)$ converges to zero uniformly in x as n increases.*

4. *There exist $0 \leq \xi < 1$ such that for all $x \in \mathcal{X}$ and sufficiently large t,*

$$F_t(x) \leq \xi\,(1 - G_t(x)).$$

Then, $V_{t+1} = H_t(V_t, V_t)$ κ'-approximates V^* over $\mathcal{X}$ for any $V_0 \in \mathcal{V}$, where $\kappa' = 2\,\kappa/(1-\xi)$.

Usually, functions F_t and G_t can be interpreted as the ratio of mixing the two arguments of operator H_t. In the case of the SARSA algorithm, described above by (3.3), $\mathcal{X} = \mathbb{X} \times \mathbb{A}$, $G_t(x, a) = (1 - \gamma_t(x, a))$ and $F_t(x, a) = \alpha\, \gamma_t(x, a)$ would be a suitable choice.

One of the most important aspects of this theorem is that it shows how to reduce the problem of approximating V^* with $V_t = H_t(V_t, V_t)$ type operators to the problem of approximating it with a $V_t' = H_t(V_t', V^*)$ sequence, which is, in many cases, much easier to be dealt with. This makes, e.g., the convergence of Watkins' Q-learning a consequence of the classical Robbins-Monro theory (Szepesvári and Littman, 1999; Szita et al., 2002).

3.2.2 Varying Markov Decision Processes

Section 3.2 aims at analyzing how learning algorithms can act in environments which may change over time, namely, in the transition-probability function and the immediate-cost function. However, without any restrictions, this approach would be too general to establish convergence results. Therefore, we restrict ourselves to the case when the cumulative changes remain bounded over time. In order to precisely define this concept, the idea of (ε, δ)-MDPs is introduced, which is a generalization of classical MDPs and ε-MDPs.

First, we recall the definition of ε-MDPs (Kalmár et al., 1998; Szita et al., 2002).

Definition 22 *A sequence of MDPs $(\mathcal{M}_t)_{t=1}^{\infty}$ is called an ε-MDP with $\varepsilon > 0$ if the MDPs differ only in their transition-probability functions, denoted by p_t for $\mathcal{M}_t$, and there exists an MDP with transition function p, called the base MDP, such that $\sup_t \|p - p_t\| \leq \varepsilon$.*

Now, we extend the idea described above. The following definition of (ε, δ)-MDPs generalizes the concept of ε-MDPs in two ways. First, we also allow the cost function to change over time and, additionally, we require the changes to remain bounded by parameters ε and δ only asymptotically, in the limit. A finite number of large deviations is tolerated.

Definition 23 *A tuple $\langle \mathbb{X}, \mathbb{A}, \mathcal{A}, \{p_t\}_{t=1}^{\infty}, \{g_t\}_{t=1}^{\infty}, \alpha \rangle$ is an (ε, δ)-MDP with $\varepsilon, \delta \geq 0$, if there exists an MDP $\langle \mathbb{X}, \mathbb{A}, \mathcal{A}, p, g, \alpha \rangle$, called the base MDP (take a look at Figure 3.2), such that*

1. $\limsup_{t \to \infty} \|p - p_t\| \leq \varepsilon$
2. $\limsup_{t \to \infty} \|g - g_t\| \leq \delta$

The optimal value function of the base MDP and of the current MDP at time t (which MDP has transition function p_t and cost function g_t) are denoted by J^* and J_t^*, respectively.

In order to keep the analysis as simple as possible, we do not allow the discount rate parameter α to change over time; not only because, e.g., with Theorem 12 at hand, it would be straightforward to extend the results to the case of changing discount factors, but even more because, as Lemma 24 demonstrates, the effects of changes in the discount rate can be incorporated into the immediate-cost function, which is allowed to change in (ε, δ)-MDPs.

Lemma 24 *Assume that two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, differ only in the discount factors, denoted by α_1 and α_2. Let the value function of a (stationary, Markovian, randomized) control policy π be denoted by J_i^π related to $\mathcal{M}_i$. The optimal value function of $\mathcal{M}_i$ is denoted by J_i^*. We will treat both cases simultaneously, thus, let us fix hyper-parameter μ to either π or $*$. Then, there exists an MDP, denoted by $\mathcal{M}_3$, such that it differs only in the immediate-cost function from $\mathcal{M}_1$ and $J_2^\mu = J_3^\mu$. The immediate-cost function of $\mathcal{M}_3$ is*

$$\widehat{g}(x,a) = g(x,a) + (\alpha_2 - \alpha_1)\sum_{y \in \mathbb{X}} p(y \,|\, x, a) J_2^\mu(y),$$

where p is the transition-probability function of $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$; g is the immediate-cost function of $\mathcal{M}_1$ and $\mathcal{M}_2$; and $J_2^\mu(y)$ denotes a value function of $\mathcal{M}_2$, where $\mu \in \{\pi, *\}$.

On the other hand, we can notice that changes in the cost function cannot be traced back to changes in the transition function. Consider, e.g., an MDP with a constant zero cost function. Then, no matter what the transition-probabilities are, the optimal value function remains zero. However, we may achieve non-zero optimal value function values if we change the immediate-cost function. Therefore, (ε, δ)-MDPs cannot be traced back to ε-MDPs.

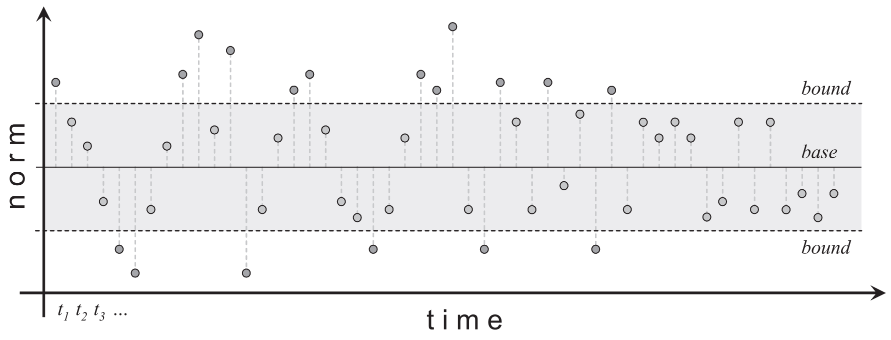


Figure 3.2: The transition-probabilities and the costs of an (ε, δ)-MDP may vary arbitrarily, as long as their distances from the base values remain asymptotically bounded.

Now, we briefly investigate the applicability of (ε, δ)-MDPs and a possible motivation behind them. When we model a "real world" problem as an MDP, then we typically take only the *major characteristics* of the system into account, but there could be many *hidden parameters*, as well, which may affect the transition-probabilities and the immediate-costs, however, which are not explicitly included in the model. For example, if we model a production control system as an MDP (Csáji and Monostori, 2006b), then the workers' fatigue, mood or the quality of the materials may affect the durations of the tasks, but these characteristics are usually not included in the model. Additionally, the values of these hidden parameters may change over time. In these cases, we could either try to incorporate as many aspects of the system as possible into the model, which approach would most likely lead to *computationally intractable* results, or we could model the system as an (ε, δ)-MDP instead, which would result in a simplified model and, presumably, in a more tractable system.

3.2.3 Stochastic Iterative Algorithms

In this section we present a general relaxed convergence theorem for a large class of stochastic iterative algorithms. Later, we will apply this theorem to investigate the convergence properties of value function based reinforcement learning methods in (ε, δ)-MDPs.

Many learning and optimization methods can be written in a general form as a stochastic iterative algorithm (Bertsekas and Tsitsiklis, 1996). In general, these algorithms are defined by the iteration $r_{t+1} = (1 - \gamma_t)r_t + \gamma_t s_t$, where the updates are performed on r_t, γ_t is a nonnegative stepsize parameter and s_t is usually a function of r_t, for example, $s_t = H_t r_t$.

One of the simplest such algorithm is the *Robbins-Monro stochastic approximation algorithm*, defined as follows. Let q_t be a sequence of independent identically-distributed (i.i.d.) random variables with unknown mean μ and finite variance. Let us define sequence r_t with the iteration $r_{t+1} = (1 - \gamma_t)r_t + \gamma_t q_t$. Then, sequence r_t converges almost surely to μ if suitable assumptions on the stepsizes are made, cf. with Assumption 2 of this section.

Another classical example of a stochastic iterative algorithm is the *stochastic gradient descent algorithm* which aims at minimizing cost function f and is described by

$$r_{t+1} = (1 - \gamma_t)r_t + \gamma_t(r_t - \nabla f(r_t) + w_t),$$

where w_t is a noise parameter, cf. Assumption 1, and ∇f denotes the gradient of f.

Time-Dependent Update

During Section 3.2.3 we study time-dependent stochastic iterative algorithms of the form

$$V_{t+1}(x) = (1 - \gamma_t(x))V_t(x) + \gamma_t(x)((K_t V_t)(x) + W_t(x)), \tag{3.4}$$

where $V_t \in \mathcal{V}$, operator $K_t : \mathcal{V} \to \mathcal{V}$ acts on value functions, each $\gamma_t(x)$ is a random variable which determines the stepsize and $W_t(x)$ is also a random variable, a noise parameter.

Regarding reinforcement learning algorithms, for example, (asynchronous) value iteration, Gauss-Seidel methods, Q-learning, SARSA and TD(λ) can be formulated this way. We will show that under suitable conditions these algorithms work in (ε, δ)-MDPs, more precisely, κ-approximation to the optimal value function of the base MDP will be proven.

Main Assumptions

Now, in order to provide our relaxed convergence result, we introduce three assumptions on the noise parameters, the stepsize parameters and the value function operator.

Definition 25 *We denote the history of the algorithm until time t by $\mathcal{F}_t$, defined as*

$$\mathcal{F}_t = \{V_0, \dots, V_t, W_0, \dots, W_{t-1}, \gamma_0, \dots, \gamma_t\}.$$

The sequence $\mathcal{F}_0 \subseteq \mathcal{F}_1 \subseteq \mathcal{F}_2 \subseteq \dots$ can be seen as a *filtration*, viz., as an increasing sequence of σ-fields. The set $\mathcal{F}_t$ represents the information available at each time t.

Assumption 1 *There exits a constant $C > 0$ such that for all state x and time t, we have*

$$\mathbb{E}\left[W_t(x) \,|\, \mathcal{F}_t\right] = 0 \qquad \textit{and} \qquad \mathbb{E}\left[W_t^2(x) \,|\, \mathcal{F}_t\right] < C < \infty.$$

Regarding the stepsize parameters, γ_t, we make the "usual" stochastic approximation assumptions. Note that there is a separate stepsize parameter for each possible state.

Assumption 2 *For all x and t, $0 \leq \gamma_t(x) \leq 1$, and we have with probability one*

$$\sum_{t=0}^{\infty} \gamma_t(x) = \infty \qquad \textit{and} \qquad \sum_{t=0}^{\infty} \gamma_t^2(x) < \infty.$$

Assumption 3 *For all t, operator $K_t : \mathcal{V} \to \mathcal{V}$ is a supremum norm contraction mapping with Lipschitz constant $\beta_t < 1$ and with fixed point V_t^*. Formally, for all $V_1, V_2 \in \mathcal{V}$,*

$$\|K_t V_1 - K_t V_2\|_\infty \leq \beta_t \|V_1 - V_2\|_\infty .$$

Let us introduce a common Lipschitz constant $\beta_0 = \limsup_{t\to\infty} \beta_t$, and assume that $\beta_0 < 1$.

Because our aim is to analyze changing environments, each K_t operator can have different fixed points and different Lipschitz constants. However, to avoid the progress of the algorithm to "slow down" infinitely, we should require that $\limsup_{t\to\infty} \beta_t < 1$. In the next section, when we apply this theory to the case of (ε, δ)-MDPs, each value function operator can depend on the current MDP at time t and, thus, can have different fixed points.

Approximate Convergence

Now, we present a theorem (its proof can be found in the appendix) that shows how the function sequence generated by iteration (3.4) can converge to an environment of a function.

Theorem 26 *Suppose that Assumptions 1-3 hold and let V_t be the sequence generated by iteration (3.4). Then, for any $V^*, V_0 \in \mathcal{V}$, the sequence V_t κ-approximates function V^* with*

$$\kappa = \frac{4\varrho}{1 - \beta_0} \qquad \textit{where} \qquad \varrho = \limsup_{t\to\infty} \|V_t^* - V^*\|_\infty .$$

This theorem is very general, it is valid even in the case of non-finite MDPs. Notice that V^* can be an *arbitrary* function but, naturally, the radius of the environment of V^*, which the sequence V_t almost surely converges to, depends on $\limsup_{t\to\infty} \|V_t^* - V^*\|_\infty$.

If we take a closer look at the proof, we can notice that the theorem is still valid if each K_t is only a *pseudo-contraction* but, additionally, it also attracts points to V^*. Formally, it is enough if we assume that for all $V \in \mathcal{V}$, we have $\|K_t V - K_t V_t^*\|_\infty \leq \beta_t \|V - V_t^*\|_\infty$ and $\|K_t V - K_t V^*\|_\infty \leq \beta_t \|V - V^*\|_\infty$ for a suitable $\beta_t < 1$. This remark could be important in case we want to apply Theorem 26 to changing *stochastic shortest path (SSP)* problems.

An Alternating Example

Consider a one dimensional stochastic process characterized by the iteration

$$v_{t+1} = (1 - \gamma_t)v_t + \gamma_t(K_t(v_t) + w_t), \tag{3.5}$$

where γ_t is the learning rate and w_t is a noise term. Let us suppose we have n alternating operators k_i with Lipschitz constants $b_i < 1$ and fixed points v_i^* where $i \in \{0, \dots, n-1\}$,

$$k_i(v) = v + (1 - b_i)(v_i^* - v).$$

The current operator at time t is $K_t = k_i$ (thus, $V_t^* = v_i^*$ and $\beta_t = b_i$) if $i \equiv t \pmod n$. Figure 3.3 shows that the trajectories remained close to the fixed points. The figure illustrates the case of two (-1 and 1) and six ($-3, -2, -1, 1, 2, 3$) alternating fixed points.

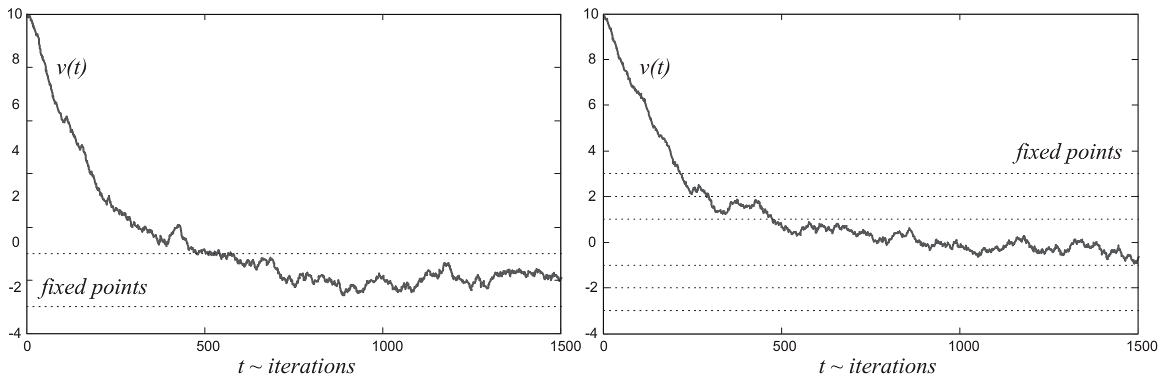


Figure 3.3: Trajectories generated by (3.5) with two (left) and six (right) fixed points.

A Pathological Example

During this example we will restrict ourselves to deterministic functions. According to the *Banach fixed point theorem*, if we have a contraction mapping f over a complete metric space with fixed point $v^* = f(v^*)$, then, for any initial v_0 the sequence $v_{t+1} = f(v_t)$ converges to v^*. It could be thought that this result can be easily generalized to the case of alternating operators. For example, suppose we have n alternating contraction mappings k_i with Lipschitz constants $b_i < 1$ and fixed points v_i^*, respectively, where $i \in \{0, \dots, n-1\}$, and we apply them iteratively starting from an arbitrary v_0, viz., $v_{t+1} = K_t(v_t)$, where $K_t = k_i$ if $i \equiv t \pmod n$. One may think that since each k_i attracts the point towards its fixed point, the sequence v_t converges to the *convex hull* of the fixed points. However, as the following example demonstrates, this is not the case, since it is possible that the point moves away from the convex hull and, in fact, it gets farther and farther after each iteration.

Now, let us consider two one-dimensional functions, $k_i : \mathbb{R} \to \mathbb{R}$, where $i \in \{a, b\}$, defined below by equation (3.6). It can be easily proven that these functions are contractions with fixed points v_i^* and Lipschitz constants b_i (in Figure 3.4, $v_a^* = 1$, $v_b^* = -1$ and $b_i = 0.9$).

$$k_i(v) = \begin{cases} v + (1 - b_i)(v_i^* - v) & \text{if } sgn(v_i^*) = sgn(v - v_i^*), \\ \\ v_i^* + (v_i^* - v) + (1 - b_i)(v - v_i^*) & \text{otherwise}, \end{cases} \tag{3.6}$$

where $sgn(\cdot)$ denotes the signum function. Figure 3.4 demonstrates that even if the iteration starts from the middle of the convex hull (from the center of mass), $v_0 = 0$, it starts getting farther and farther from the fixed points in each step when we apply k_a and k_b after each other. Nevertheless, the following argument shows that sequence v_t cannot get arbitrarily far

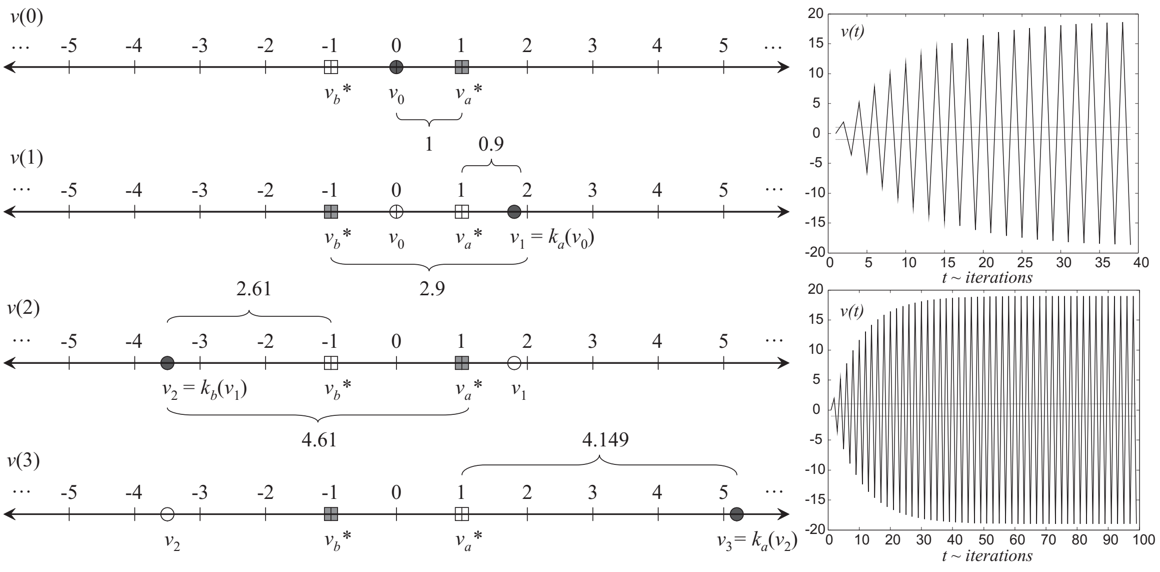


Figure 3.4: A deterministic pathological example, generated by the iterative application of equation (3.6). The left part demonstrates the first steps, while the two images on the right-hand side show the behavior of the trajectory in the long run.

from the fixed points. Let us denote the *diameter* of the convex hull of the fixed points by ϱ. Since this convex hull is a polygon (where the vertices are fixed points) $\varrho = \max_{i,j} \|v_i^* - v_j^*\|$. Furthermore, let β_0 be defined as $\beta_0 = \max_i b_i$ and d_t as $d_t = \min_i \|v_i^* - v_t\|$. Then, it can be proven that for all t, we have $d_{t+1} \leq \beta_0(2\varrho + d_t)$. If we assume that $d_{t+1} \geq d_t$, then it follows that $d_t \leq d_{t+1} \leq \beta_0(2\varrho + d_t)$. After rearrangement, we get the following inequality

$$d_t \leq \frac{2\,\beta_0\,\varrho}{1 - \beta_0} = \phi(\beta_0, \varrho).$$

Therefore, $d_t > \phi(\beta_0, \varrho)$ implies that $d_{t+1} < d_t$. Consequently, if v_t somehow got farther than $\phi(\beta_0, \varrho)$, in the next step it would inevitably be attracted towards the fixed points. It is easy to see that this argument is valid in an arbitrary normed space, as well.

3.2.4 Learning in Varying MDPs

In case we consider finite (ε, δ)-MDPs, we can formulate a relaxed convergence theorem for value function based reinforcement learning algorithms, as a corollary of Theorem 26. Suppose that $\mathcal{V}$ consists of state-value functions, namely, $\mathcal{X} = \mathbb{X}$. Then, we have

$$\limsup_{t\to\infty} \|J^* - J_t^*\|_\infty \leq d(\varepsilon, \delta),$$

where J_t^* is the optimal value function of the MDP at time t and J^* is the optimal value function of the base MDP. In order to calculate $d(\varepsilon, \delta)$, Theorems 10 (or 9), 11 and the triangle inequality could be applied. Assume, e.g., that we use the supremum norm, $\|\cdot\|_\infty$, for cost functions and $\|\cdot\|_1$, defined by equation (3.1), for transition functions. Then,

$$d(\varepsilon, \delta) = \frac{\varepsilon\,\alpha\,\|g\|_\infty}{(1-\alpha)^2} + \frac{\delta}{1-\alpha},$$

where g is the immediate-cost function of the base MDP. Now, as an immediate consequence of Theorem 26, we can formulate the following relaxed convergence theorem.

Corollary 27 *Suppose that we have an (ε, δ)-MDP and Assumptions 1-3 hold. Let V_t be the sequence generated by iteration (3.4). Furthermore, assume that the fixed point of each operator K_t is J_t^*. Then, for any initial $V_0 \in \mathcal{V}$, the sequence V_t κ-approximates J^* with*

$$\kappa = \frac{4\,d(\varepsilon, \delta)}{1-\beta_0}.$$

Notice that as parameters ε and δ go to zero, we get back to a classical convergence theorem for this kind of stochastic iterative algorithm (still in a little bit generalized form, since β_t might still change over time). Now, with the help of these results, we will investigate the convergence of some classical reinforcement learning algorithms in (ε, δ)-MDPs.

Asynchronous Value Iteration

The method of value iteration is one of the simplest reinforcement learning algorithms. In ordinary MDPs it is defined by the iteration $J_{t+1} = TJ_t$, where T is the Bellman operator. It is known that the sequence J_t converges in the supremum norm to J^* for any initial J_0 (Bertsekas and Tsitsiklis, 1996). The asynchronous variant of value iteration arises when the states are updated asynchronously, e.g., only one state in each iteration. In the case of (ε, δ)-MDPs a small stepsize variant of asynchronous value iteration can be defined as

$$J_{t+1}(x) = (1 - \gamma_t(x))J_t(x) + \gamma_t(x)(T_tJ_t)(x),$$

where T_t is the Bellman operator of the current MDP at time t. Since there is no noise term in the iteration, Assumption 1 is trivially satisfied. Assumption 3 follows from the fact that each T_t operator is an α contraction where α is the discount factor. Therefore, if the stepsizes satisfy Assumption 2 then, by applying Corollary 27, we have that the sequence J_t κ-approximates J^* for any initial value function J_0 with $\kappa = (4\,d(\varepsilon, \delta))/(1-\alpha)$.

Q-learning

The Q-learning method of Watkins (1989) is a very popular off-policy, model-free reinforcement learning algorithm (Even-Dar and Mansour, 2003). Its generalized version in ε-MDPs was studied by Szita et al. (2002). Q-learning works with action-value functions, therefore, $\mathcal{X} = \mathbb{X} \times \mathbb{A}$, and the one-step Q-learning rule in (ε, δ)-MDPs can be defined as follows

$$Q_{t+1}(x,a) = (1 - \gamma_t(x,a))Q_t(x,a) + \gamma_t(x,a)(\widetilde{T}_t Q_t)(x,a), \tag{3.7}$$

$$(\widetilde{T}_t Q_t)(x,a) = g_t(x,a) + \alpha \min_{B \in \mathcal{A}(Y)} Q_t(Y,B),$$

where g_t is the immediate-cost function of the current MDP at time t and Y is a random variable generated from the pair (x,a) by simulation, that is, according to the probability distribution $p_t(x,a)$, where p_t is the transition function of the current MDP at time t.

Operator $\widetilde{T}_t$ is randomized, but as it was shown by Bertsekas and Tsitsiklis (1996) in their convergence theorem for Q-learning, it can be rewritten in a form as follows

$$(\widetilde{T}_t Q)(x,a) = (\widetilde{K}_t Q)(x,a) + \widetilde{W}_t(x,a),$$

where $\widetilde{W}_t(x,a)$ is a noise term with zero mean and finite variance, and $\widetilde{K}_t$ is defined as

$$(\widetilde{K}_t Q)(x,a) = g_t(x,a) + \alpha \sum_{y \in \mathbb{X}} p_t(y \mid x,a) \min_{b \in \mathcal{A}(y)} Q(y,b).$$

Let us denote the optimal action-value function of the current MDP at time t and the base MDP by Q_t^* and Q^*, respectively. By using the fact that $J^*(x) = \min_a Q^*(x,a)$, it is easy to see that for all t, Q_t^* is the fixed point of operator $\widetilde{K}_t$ and, moreover, each $\widetilde{K}_t$ is an α contraction. Therefore, if the stepsizes satisfy Assuption 2, then the Q_t sequence generated by iteration (3.7) κ-approximates Q^* for any initial Q_0 with $\kappa = (4\,d(\varepsilon,\delta))/(1-\alpha)$.

In some situations the immediate costs are randomized, however, even in this case the relaxed convergence of Q-learning would follow as long as the random immediate costs had finite expected value and variance, which is required for satisfying Assumption 1.

Temporal Difference Learning

Temporal difference learning, or for short TD-learning, is a policy evaluation algorithm. It aims at finding the corresponding value function J^π for a given policy π. It can also be used for approximating the optimal value function, e.g., if we apply it together with the policy iteration algorithm (Bertsekas and Tsitsiklis, 1996; Sutton and Barto, 1998).

First, we briefly overview the off-line first-visit variant of TD(λ) in case of ordinary MDPs. It can be shown that the value function of a policy π can be rewritten in a form as

$$J^\pi(x) = \mathbb{E}\left[\sum_{m=0}^{\infty} (\alpha\lambda)^m D_{\alpha,m}^\pi \;\middle|\; X_0 = x\right] + J^\pi(x),$$

where $\lambda \in [0,1)$ and $D_{\alpha,m}^\pi$ denotes the "temporal difference" coefficient at time m,

$$D_{\alpha,m}^\pi = g(X_m, A_m^\pi) + \alpha J^\pi(X_{m+1}) - J^\pi(X_m),$$

where X_m, X_{m+1} and A^π_m are random variables, X_{m+1} has $p(X_m, A^\pi_m)$ distribution and A^π_m is a random variable for actions, it is selected according to the distribution $\pi(X_m)$.

Based on this observation, we can define a stochastic approximation algorithm as follows. Let us suppose that we have a generative model of the environment, e.g., we can perform simulations in it. Each simulation produces a state-action-reward trajectory. We can assume that all simulations eventually end, e.g., there is an absorbing termination state or we can stop the simulation after a given number of steps. Note that even in this case we can treat each trajectory as infinitely long, viz., we can define all costs after the termination as zero. The off-line first-visit TD(λ) algorithm updates the value function after each simulation,

$$J_{t+1}(x^t_k) = J_t(x^t_k) + \gamma_t(x^t_k) \sum_{m=k}^{\infty} (\alpha\lambda)^{m-k} d_{\alpha,m,t}, \tag{3.8}$$

where x^t_k is the state at step k in trajectory t and $d_{\alpha,m,t}$ is the temporal difference coefficient,

$$d_{\alpha,m,t} = g(x^t_m, a^t_m) + \alpha J_t(x^t_{m+1}) - J_t(x^t_m).$$

For the case of ordinary MDPs it is known that TD(λ) converges almost surely to J^π for any initial J_0 provided that each state is visited by infinitely many trajectories and the stepsizes satisfy Assumption 2. The proof (Bertsekas and Tsitsiklis, 1996) is based on the observation that iteration (3.8) can be seen as a Robbins-Monro type stochastic iterative algorithm for finding the fixed point of $J^\pi = HJ^\pi$, where H is a contraction mapping with Lipschitz constant α. The only difference in the case of (ε, δ)-MDPs is that the environment may change over time and, therefore, operator H becomes time-dependent. However, each H_t is still an α contraction, but they potentially have different fixed points. Therefore, we can apply Theorem 26 to achieve a relaxed convergence result for off-line first-visit TD(λ) in changing environments under the same conditions as in the case of ordinary MDPs.

The convergence of the on-line every-visit variant can be proven in the same way as in the case of ordinary MDPs, viz., by showing that the difference between the two variants is of second order in the size of γ_t and hence inconsequential as γ_t diminishes to zero.

Chapter 4

Experimental Results

In the previous chapters we investigated machine learning based resource allocation techniques. We argued that the suggested approach was not only capable of computing good solutions in reasonable amount of time, but it could also deal with uncertainties and changes. We presented a few theorems to support our claims, e.g., in Chapter 3 we theoretically studied changing environments. However, several parts of the solution were not supported by formal results. In order to *confirm* these parts and to *verify* the others, a simulation environment was developed in C++. This chapter aims at presenting our experimental results concerning stochastic resource allocation and adaptation to environmental changes. The performed simulation experiments highlight some typical characteristics of our approach.

4.1 Stochastic Resource Allocation

In this section some experimental results concerning machine learning based stochastic resource allocation on both benchmark and industry-related problems are presented.

4.1.1 Testing Methodology

During our experiments, we applied FQL and, in most of the cases, SVRs which were realized by the LIBSVM free library for support vector machines (Chang and Lin, 2001). After centering and scaling the data into interval $[0, 1]$, we used Gaussian kernels and shrinking techniques. We always applied rollout algorithms and action decomposition, but clustering was only used in tests presented in Section 4.1.5, furthermore, distributed sampling was only applied in test shown in Section 4.1.3. In both of the latter cases (tests for clustering and distributed sampling) we used hash tables with approximately 256Mb hash memory.

The performance of the algorithm was measured as follows. Testing took place in two main fields: the first one was a benchmark scheduling dataset of hard problems, the other one was a simulation of a "real world" production control problem. In the first case the best solution, viz., the optimal value of the (aggregated) initial state, $J^*(x_0) = \min_a Q^*(x_0, a)$, was known for most of the test instances. Some "very hard" instances occurred for which only lower and upper bounds were known, e.g., $J_1^*(x_0) \leq J^*(x_0) \leq J_2^*(x_0)$. In these cases we assumed that $J^*(x_0) \approx (J_1^*(x_0) + J_2^*(x_0))/2$. Since these estimations were "good" (viz., the length of the intervals were short), this simplification might not introduce considerable error to our performance estimations. In the latter test case we have generated the problems with a generator in a way that $J^*(x_0)$ was known concerning the constructed problems.

The performance presented in the tables of the section, more precisely the average, $\overline{E}_i$, and the standard deviation, $\sigma(E_i)$, of the error in iteration i were computed as follows

$$\overline{E}_i = \frac{1}{N}\sum_{j=1}^{N}\left[G_j^i - J^*(x_0)\right], \qquad \text{and} \qquad \sigma(E_i) = \sqrt{\frac{1}{N}\sum_{j=1}^{N}\left[G_j^i - J^*(x_0) - \overline{E}_i\right]^2},$$

where G_j^i denotes the cumulative incurred costs in iteration i of sample j and N is the sample size. Unless indicated otherwise, the sample contained the results of 100 simulation trials for each parameter configuration (which is associated with the rows of the tables).

As it was shown RAP-MDPs are acyclic, moreover, they have the APP property, therefore, discounting is not necessary to achieve a well-defined problem. However, in order to enhance learning, it is still advised to apply discounting, therefore, to give less credit to events which are farther from the current decision point. Heuristically, we suggest applying $\alpha = 0.95$ for middle-sized RAPs, such as the problems of the benchmark dataset, and $\alpha = 0.99$ for large-scale RAPs, such as the problems of the industry-related experiments.

4.1.2 Benchmark Datasets

The ADP based resource control approach was tested on Hurink's benchmark dataset (Hurink et al., 1994). It contains *flexible job-shop scheduling problems* (FJSPs) with 6–30 jobs (30–225 tasks) and 5–15 machines. The applied performance measure is the maximum completion time of the tasks (makespan). These problems are "hard", which means, e.g., that standard dispatching rules or heuristics perform poorly on them. This dataset consists of four subsets, each subset contains about 60 problems. The subsets (sdata, edata, rdata, vdata) differ in the ratio of machine interchangeability (flexibility), which is shown in the "flex(ib)" columns in Tables 4.1 and 4.2. The columns with label "n iters" (and "avg err") show the average error after carrying out altogether "n" iterations. The "std dev" columns in all of the tables of this chapter contain the standard deviation of the sample.

benchmark		**1000 iterations**		**5000 iterations**		**10 000 iterations**	
dataset	**flexib**	**avg err**	**std dev**	**avg err**	**std dev**	**avg err**	**std dev**
sdata	1.0	8.54 %	5.02 %	5.69 %	4.61 %	3.57 %	4.43 %
edata	1.2	12.37 %	8.26 %	8.03 %	6.12 %	5.26 %	4.92 %
rdata	2.0	16.14 %	7.98 %	11.41 %	7.37 %	7.14 %	5.38 %
vdata	5.0	10.18 %	5.91 %	7.73 %	4.73 %	3.49 %	3.56 %
average	2.3	11.81 %	6.79 %	8.21 %	5.70 %	4.86 %	4.57 %

Table 4.1: Summarized performance (average error and deviation) on benchmark datasets.

In Table 4.1 the summarized performance on the benchmark datasets is shown. Table 4.2 illustrates the performance on some typical dataset instances and also gives some details on them, e.g., the number of machines and jobs (columns with labels "mcs" and "jbs").

benchmark configuration					**average error (standard deviation)**		
dataset	**inst**	**mcs**	**jbs**	**flex**	**1000 iters**	**5000 iters**	**10 000 iters**
sdata	mt06	6	6	1	1.79 (1.01) %	0.00 (0.00) %	0.00 (0.00) %
sdata	mt10	10	10	1	9.63 (4.59) %	8.83 (4.37) %	7.92 (4.05) %
sdata	la09	5	15	1	5.67 (2.41) %	3.87 (1.97) %	3.05 (1.69) %
sdata	la19	10	10	1	11.65 (5.21) %	6.44 (3.41) %	3.11 (1.74) %
sdata	la39	15	15	1	14.61 (7.61) %	12.74 (5.92) %	11.92 (5.63) %
sdata	la40	15	15	1	10.98 (5.04) %	8.87 (4.75) %	8.39 (4.33) %
edata	mt06	6	6	1.15	0.00 (0.00) %	0.00 (0.00) %	0.00 (0.00) %
edata	mt10	10	10	1.15	18.14 (8.15) %	12.51 (6.12) %	9.61 (4.67) %
edata	la09	5	15	1.15	7.51 (3.33) %	5.23 (2.65) %	2.73 (1.89) %
edata	la19	10	10	1.15	8.04 (4.64) %	4.14 (2.81) %	1.38 (1.02) %
edata	la39	15	15	1.15	22.80 (9.67) %	17.32 (8.29) %	12.41 (6.54) %
edata	la40	15	15	1.15	14.78 (7.14) %	8.08 (4.16) %	6.68 (4.01) %
rdata	mt06	6	6	2	6.03 (3.11) %	0.00 (0.00) %	0.00 (0.00) %
rdata	mt10	10	10	2	17.21 (8.21) %	12.68 (6.81) %	7.87 (4.21) %
rdata	la09	5	15	2	7.08 (3.23) %	6.15 (2.92) %	3.80 (2.17) %
rdata	la19	10	10	2	18.03 (8.78) %	11.71 (5.78) %	8.87 (4.38) %
rdata	la39	15	15	2	24.55 (9.59) %	18.90 (8.05) %	13.06 (7.14) %
rdata	la40	15	15	2	23.90 (7.21) %	18.91 (6.92) %	14.08 (6.68) %
vdata	mt06	6	6	3	0.00 (0.00) %	0.00 (0.00) %	0.00 (0.00) %
vdata	mt10	10	10	5	8.76 (4.65) %	4.73 (2.23) %	0.45 (0.34) %
vdata	la09	5	15	2.5	9.92 (5.32) %	7.97 (3.54) %	4.92 (2.60) %
vdata	la19	10	10	5	14.42 (7.12) %	11.61 (5.76) %	6.54 (3.14) %
vdata	la39	15	15	7.5	16.16 (7.72) %	12.25 (6.08) %	9.02 (4.48) %
vdata	la40	15	15	7.5	5.86 (3.11) %	4.08 (2.12) %	2.43 (1.83) %

Table 4.2: Performance (average error and deviation) on some typical benchmark problems.

The best performance was achieved by (Mastrolilli and Gambardella, 2000) on these benchmark FJSP datasets. Though, their algorithm performs slightly better than ours, their solution exploits the (unrealistic) specialties of the dataset, e.g., the durations do not depend on the resources; the tasks are linearly ordered in the jobs; each job consists of the same number of tasks. Moreover, it cannot be easily generalized to stochastic resource control problem our algorithm faces. Therefore, the comparison of the two solutions is hard.

4.1.3 Distributed Sampling

The possible parallelizations of the presented method was also investigated, i.e., the *speedup* of the system relative to the number of processors (in practise, the multiprocessor environment was emulated on a single processor, only). The average number of iterations was studied, until the system could reach a solution with less than 5 % error on Hurink's dataset. The average speed of a single processor was treated as a unit, for comparison.

In Figure 4.1 two cases are shown: in the first case (rear dark bars) each processor could access a common global value function. It means that each processor could read and write the same global value function, but otherwise, they searched (sampled the search space) independently. Figure 4.1 demonstrates that in this case the speedup was almost linear.

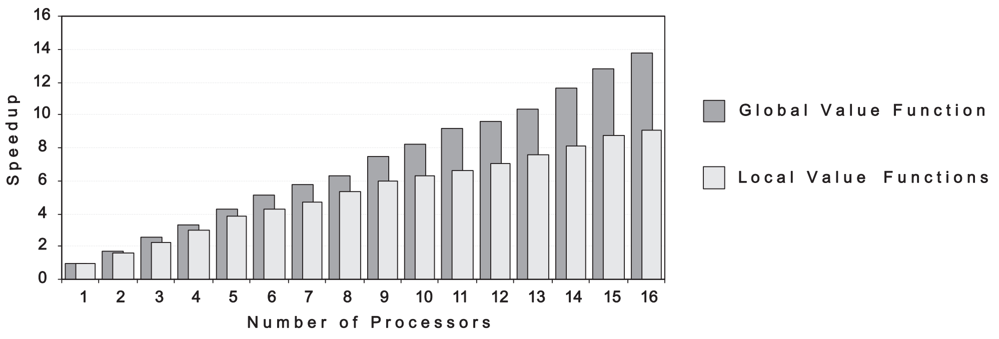


Figure 4.1: Average speedup relative to the number of processors.

In the second case (front light bars) each processor had its own (local) value function (which is more realistic in a strongly distributed system, such as a GRID) and, after the search had been finished, these individual value functions were compared. Therefore, all of the processors had estimations of their own, and after the search, the local solution of the best performing processor was selected. Figure 4.1 shows the achieved speedup in case we stopped the simulation if any of the processors achieved a solution with less than 5 % error.

The experiments show that the computation of the resource allocator function can be effectively distributed, even if there is not a commonly accessible value function available.

4.1.4 Industry Related Tests

We also initiated numerical experiments on a simulated factory by modeling the structure of a real plant producing customized mass-products, especially, light bulbs. These industrial data came from a huge national industry-academia project for research and development of solutions which support manufacturing enterprises in coping with the requirements of adaptiveness, realtimeness and cooperativeness (Monostori et al., 2008).

Since, we did not have access to historical data concerning past orders, we used randomly generated orders (jobs) with random due dates. The tasks and the process-plans of the jobs, however, covered real products; as well as, the resources covered real machine types. In this plant the machines require product-type dependent setup times, and there are some special tasks that have durations but that do not require any resources to be processed, for example, cooling down. Another feature of the plant is that at some previously given time points preemptions are allowed, e.g., at the end of a work shift. The applied performance measure was to minimize the *number of late jobs*, viz., jobs that are finished after their due dates, and an additional secondary measure was to minimize the total cumulative *lateness*, which can be applied to compare two schedules having the same number of late jobs.

optimal	**1000 iterations**		**5000 iterations**		**10 000 iterations**	
slack ratio	**avg err**	**std dev**	**avg err**	**std dev**	**avg err**	**std dev**
50 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %	0.00 %
40 %	0.12 %	0.10 %	0.00 %	0.00 %	0.00 %	0.00 %
30 %	0.52 %	0.71 %	0.24 %	0.52 %	0.13 %	0.47 %
20 %	1.43 %	1.67 %	1.11 %	1.58 %	1.05 %	1.49 %
10 %	5.28 %	3.81 %	4.13 %	3.53 %	3.91 %	3.48 %
0 %	8.89 %	5.17 %	7.56 %	5.04 %	6.74 %	4.83 %

Table 4.3: Summarized performance relative to the optimal slack ratio of the system.

During these experiments the jobs and their due dates were generated by a special parameterizable generator in a way that optimally none of the jobs were late. Therefore, it was known that $J^*(x_0) = 0$ and the error of the algorithm was computed accordingly.

In the first case, shown in Table 4.3, we applied 16 machines and 100 random jobs, which altogether contained more than 200 tasks. The convergence properties were studied relative to the optimal *slack ratio*. In the deterministic case, e.g., the slack ratio of a solution is

$$\Phi(\varrho) = \frac{1}{n} \sum_{i=1}^{n} \frac{B(J_i) - F(J_i)}{B(J_i) - A(J_i)},$$

where n is the number of jobs; $A(J)$ and $B(J)$ denote the release and due date of job J, respectively; $F(J)$ is the finish time of job J relative to solution ϱ, namely, the latest finish time of the tasks in the job. Roughly, the slack ratio measures the tightness of the solution, for example, if $\Phi(\varrho) > 0$, then it shows that the jobs were, on the average, finished before their due dates and if $\Phi(\varrho) < 0$, then it indicates that, approximately, many jobs were late. If $\Phi(\varrho) = 0$, then it shows that if all the jobs meet their due dates, each job was finished just in time, there were no spare (residual) times. Under the *optimal* slack ratio we mean the maximal achievable slack ratio (by an optimal solution). During the experiments these values were known because of the special construction of the test problem instances. We applied the optimal slack ratio to measure how "hard" a problem is. The first column of Table 4.3 shows the optimal slack ratio in percentage, e.g., 30 % means a 0.3 slack ratio.

In the second case, shown in Table 4.4, we have fixed the optimal slack ratio of the system to 10 % and investigated the convergence speed relative to the plant size (number of machines) and the number of tasks. In the last two experiments (configuration having

2000 and 10 000 tasks) only 10 samples were generated, because of the long runtime. The computation of 10 000 iterations took approximately 30 minutes for the 50 machines & 2000 tasks configuration and 3 hours for the 100 machines & 10000 tasks configuration[1].

The results demonstrate that the ADP and adaptive sampling based solution scales well with both the slack ratio and the size (the number of machines and task) of the problem.

configuration		**1000 iterations**		**5000 iterations**		**10 000 iterations**	
machs	**tasks**	**avg err**	**std dev**	**avg err**	**std dev**	**avg err**	**std dev**
6	30	4.01 %	2.24 %	3.03 %	1.92 %	2.12 %	1.85 %
16	140	4.26 %	2.32 %	3.28 %	2.12 %	2.45 %	1.98 %
25	280	7.05 %	2.55 %	4.14 %	2.16 %	3.61 %	2.06 %
30	560	7.56 %	3.56 %	5.96 %	2.47 %	4.57 %	2.12 %
50	2000	8.69 %	7.11 %	7.24 %	5.08 %	6.04 %	4.53 %
100	10000	15.07 %	11.89 %	10.31%	7.97 %	9.11 %	7.58 %

Table 4.4: Summarized performance relative to the number of machines and tasks.

4.1.5 Clustering Experiments

The effectiveness of clustering on industry-related data was also studied. We considered a system with 60 resources and 1000 random tasks distributed among 400–500 jobs (there were approximately 1000–2000 precedence constraints). The tasks were generated in a way that, in the optimal case, none of them are late and the slack ratio is about 20 %.

First, the tasks were ordered according to their slack times and then they were clustered. We applied 10^4 iterations on each cluster. The computational time in case of using only one cluster was treated as a unit. In Table 4.5 the average and the standard deviation of the error and the computational speedup are shown relative to the number tasks in a cluster.

configuration		**performance after 10 000 iterations per cluster**				
clusters	**tasks**	**late jobs**	**avg error**	**std dev**	**speed**	**speedup**
1	1000	28.1	6.88 %	2.38 %	423 s	1.00 ×
5	200	22.7	5.95 %	2.05 %	275 s	1.54 ×
10	100	20.3	4.13 %	1.61 %	189 s	2.24 ×
20	50	13.9	3.02 %	1.54 %	104 s	3.28 ×
30	33	14.4	3.15 %	1.51 %	67 s	6.31 ×
40	25	16.2	3.61 %	1.45 %	49 s	8.63 ×
50	20	18.7	4.03 %	1.43 %	36 s	11.65 ×

Table 4.5: Speedup and performance relative to the number of tasks in a cluster.

1. The tests were performed on a Centrino (Core-Duo) 1660Mhz CPU ($\approx$ P4 3GHz) with 1Gb RAM.

The results demonstrate that partitioning the search space not only results in a greater speed, but it is often accompanied by better solutions. The latter phenomenon can be explained by the fact that using smaller sample trajectories generates smaller variance that is preferable for learning. On the other hand, making too small clusters may decrease the performance (e.g., making 50 clusters with 20 tasks in the current case). In our particular case applying 20 clusters with approximately 50 tasks in each cluster balances good performance (3.02 % error on the average) with remarkable speedup (approximately $3.28\times$).

As clustering the tasks represents a considerable help in dealing with large-scale RAPs, their further theoretical and experimental investigation would be very promising.

4.2 Varying Environments

In this section we present two numerical experiments. The first one demonstrates the effects of environmental changes during Q-learning based *scheduling*. The second one presents a parameter analysis concerning the effectiveness of SARSA in (ε, δ)-type *grid world* domains.

4.2.1 Adaptation to Disturbances

In order to verify the algorithm in changing environments, experiments were carried out on random JSPs with the aim of minimizing the makespan. The adaptive features of the system were tested by confronting it with unexpected events, such as: resource breakdowns, new resource availability (Figure 4.2), new job arrivals or job cancellations (Figure 4.3). In Figures 4.2 and 4.3 the horizontal axis represents time, while the vertical one, the achieved performance measure. The figures were made by averaging hundred random samples. In these tests 20 machines were used with few dozens of jobs. In each case there was an unexpected event at time $t = 100$. After the change took place, we considered two possibilities: we either restarted the iterative scheduling process from scratch or continued the learning, using the current (obsolete) value function. We experienced that the latter approach is much more efficient. This was one of the reasons why we started to study how the value function of a control policy depends on the dynamics of the underlying Markov process.

The results, black curves, show the case when the obsolete value function approximation was applied after the change took place. The performance which would arise if the system recomputed the whole schedule from scratch is drawn in gray in part (a) of Figure 4.2.

One can notice that even if the problem became "easier" after the change in the environment (at time $t = 100$), for example, a new resource was available (part (b) of Figure 4.2) or a job was cancelled (part (b) of Figure 4.3), the performance started to slightly decrease (κ started to slightly increase) after the event. This phenomena can be explained by the fact that even in these special cases the system had to "explore" the new configuration.

Recall that Theorems 9 and 11 measure the amount of the possible change in the value function of a control policy in case there were changes in the MDP, but since they apply supremum norm, they only provide bounds for *worst case* situations. However, the results of our numerical experiments, shown in Figures 4.2 and 4.3, are indicative of the phenomenon that in an *average case* the change is much less. Therefore, applying the obsolete value function after a change took place is preferable over restarting the optimization from scratch.

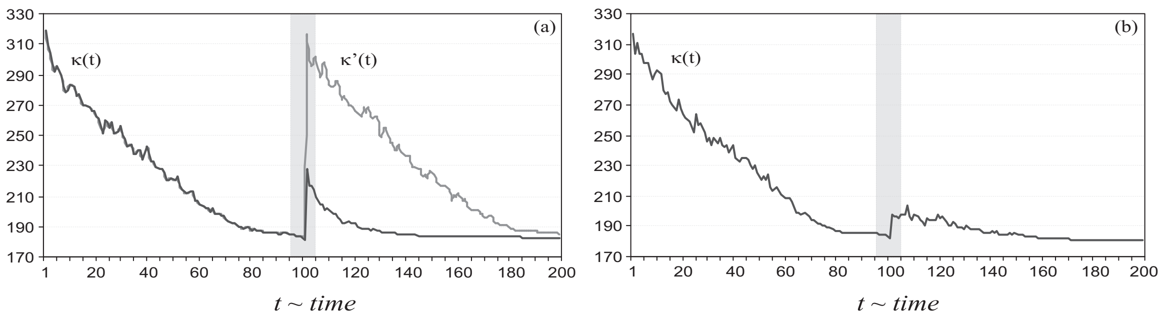


Figure 4.2: The black curves, $\kappa(t)$, show the performance measure in case there was a resource breakdown (a) or a new resource availability (b) at $t = 100$; the gray curve, κ'(t), illustrates the case the policy would be recomputed from scratch.

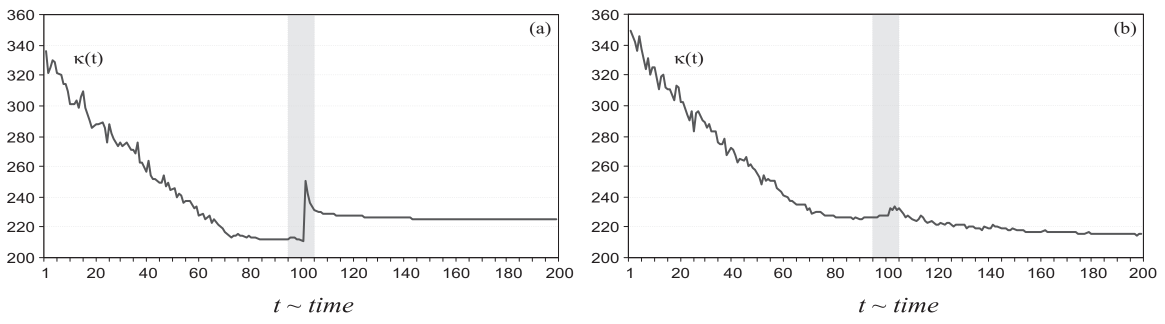


Figure 4.3: The black curves, $\kappa(t)$, show the performance measure during resource control in case there was a new job arrival (a) or a job cancellation (b) at time $t = 100$.

4.2.2 Varying Grid Worlds

We also performed numerical experiments on a variant of the classical *grid world* problem (Sutton and Barto, 1998). The original version of this problem can be briefly described as follows: an agent wanders in a rectangular world starting from a random *initial* state with the aim of finding the *goal* state. In each state the agent is allowed to choose from four possible actions: "north", "south", "east" and "west". After an action was selected, the agent moves one step in that direction. There are some *mines* on the field, as well, that the agent should avoid. An *episode* ends if the agent finds the goal state or hits a mine. During our experiments, we have applied randomly generated 10×10 grid worlds (therefore, these MDPs had 100 states) with 10 mines. The (immediate) *cost* of taking a (non-terminating) step was 5, a cost of hitting a mine was 100 and the cost of finding the goal state was -100.

In order to perform the experiment described by Table 4.6, we applied the "RL-Glue" framework[2] which consists of open source softwares and aims at being a standard protocol for benchmarking and interconnecting reinforcement learning agents and environments.

We have analyzed an (ε, δ)-type version of grid world, where the problem formed an (ε, δ)-MDP. More precisely, we have investigated the case when for all time t, the transition-probabilities could vary by at most $\varepsilon \geq 0$ around the base transition-probability values and the immediate-costs could vary by at most $\delta \geq 0$ around the base cost values.

During our numerical experiments, the environment changed at each time-step. These changes were generated as follows. First, changes concerning the transition-probabilities are described. In our randomized grid worlds the agent was taken to a random surrounding state (no matter what action it chose) with probability η and this probability *changed* after each step. The new η was computed according to the *uniform* distribution, but its possible values were *bounded* by the values described in the first row of Table 4.6.

Similarly, the immediate-costs of the base MDP (cf. the first paragraph) were *perturbed* with a uniform random variable that changed at each time-step. Again, its (absolute) value was bounded by δ, which is presented in the first column of the table. The values shown were divided by 100 to achieve the same scale as the transition-probabilities have.

$\Delta \|\|g\|\|$	the bounds for the varying probability of arriving at random states $\sim \varepsilon$										
$\delta/100$	**0.0**	**0.1**	**0.2**	**0.3**	**0.4**	**0.5**	**0.6**	**0.7**	**0.8**	**0.9**	**1.0**
0.0	-55.5	-48.8	-41.4	-36.7	-26.7	-16.7	-8.5	2.1	14.2	31.7	46.0
0.1	-54.1	-46.1	-41.2	-34.5	-25.8	-15.8	-6.0	3.7	16.5	32.3	46.3
0.2	-52.5	-44.8	-40.1	-34.4	-25.3	-15.4	-5.8	4.0	17.6	33.1	48.1
0.3	-49.7	-42.1	-36.3	-31.3	-23.9	-14.2	-5.3	8.0	18.1	37.2	51.6
0.4	-47.4	-41.5	-34.7	-30.7	-22.2	-12.2	-2.3	8.8	20.2	38.3	52.0
0.5	-42.7	-41.0	-34.5	-24.8	-21.1	-10.1	-1.3	11.2	25.7	39.2	52.1
0.6	-36.1	-36.5	-29.7	-24.0	-16.8	-7.9	1.1	17.0	31.3	43.9	54.1
0.7	-30.2	-29.3	-29.3	-19.1	-13.4	-6.0	7.4	18.9	26.9	47.2	60.9
0.8	-23.1	-27.0	-21.4	-18.8	-10.9	-2.6	8.9	22.5	31.3	50.0	64.2
0.9	-14.1	-19.5	-21.0	-12.4	-7.5	0.7	13.2	23.2	38.9	52.2	68.1
1.0	-6.8	-10.7	-14.5	-7.1	-5.3	6.6	15.7	26.4	39.8	57.3	68.7

Table 4.6: The (average) cumulative costs gathered by SARSA in varying grid worlds.

Table 4.6 was generated using an (optimistic) SARSA algorithm, namely, the actual policy was evaluated by SARSA, then the policy was (optimistically) improved, more precisely, the *greedy* policy with respect to the achieved evaluation was calculated. That policy was also *soft*, namely, it made random *explorations* with probability 0.05. We have generated 1000 random grid worlds for each parameter pairs and performed 10 000 episodes in each of

2. http://rlai.cs.ualberta.ca/RLBB/top.html

these generated worlds. The results presented in the table were calculated by averaging the cumulative costs over all episodes and over all generated sample worlds.

The parameter analysis shown in Table 4.6 is indicative of the phenomenon that changes in the transition-probabilities have a much higher impact on the performance. Even large perturbations in the costs were tolerated by SARSA, but large variations in the transition-probabilities caused a high decrease in the performance. An explanation could be that large changes in the transitions cause the agent to loose control over the events, since it becomes very hard to predict the effects of the actions and, hence, to estimate the expected costs.

Chapter 5

Conclusion

Efficient allocation of scarce, reusable resources over time to interconnected tasks is an important problem that arises in many “real world” domains. Though, in the past decades much effort was spent on designing efficient resource allocation algorithms, most of these approaches investigated deterministic and static problems. However, in practise there is a significant amount of uncertainty concerning resource allocation, for example, the execution times of the tasks are usually not known exactly. Moreover, there could be disturbances and changes in the environment, as well. These issues seriously limit the applicability of classical solution methods. Unfortunately, it is not trivial the extend these algorithms, such as branch and cut or constraint satisfaction, to handle stochastic problems. Instead of this direction, the thesis took a machine learning (ML) approach to resource allocation to face these additional challenges. In this concluding chapter we briefly summarize the solution concerning both managing uncertainties during resource allocation and dealing with the changes of the environment. We also take the opportunity to highlight a few advantages of the approach and, finally, we present some possible further research directions, as well.

5.1 Managing Uncertainties

In order to define an efficient reactive solution, first, a general resource allocation framework was presented and it was reformulated as a stochastic shortest path problem, a special Markov decision process (MDP). It was shown that this reformulation has several favorable properties, such as it has finite state and action spaces, it is acyclic, hence all policies are proper and the space of control policies can be safely restricted. The possibility of achieving proactive solutions with this approach was also investigated and proactive solutions were formulated as control policies of suitably defined partially observable MDPs (POMDPs).

The core idea of the proposed solution was the application of simulation based reinforcement learning (RL) techniques together with other machine learning methods. The exploration and exploitation ratio of the system was controlled by a Boltzmann formula, combined with a Metropolis algorithm. Regarding value function representations, two approaches were studied: hash table and support vector regression (SVR). Afterwards, several additional improvements, such as the application of rollout algorithms, action space decomposition, task clustering and distributed sampling, were suggested for speeding up the computation of a good policy. Finally, the effectiveness of the approach was demonstrated by results of simulation experiments on both benchmark and industry-related data.

There are several advantages why ML based resource allocation is preferable to other kinds of RAP solutions, e.g., classical approaches. These favorable features are as follows:

1. The presented RAP framework is very *general*, it can model several resource management problems that appear in practise, such as scheduling problems, transportation problems, inventory management problems or maintenance and repair problems.

2. RL based methods essentially face the problem under the presence of *uncertainties*, since their theoretical foundation is provided by MDPs. Moreover, they can adapt to unexpected *changes* in the environmental dynamics, such as breakdowns.

3. Additionally, for most algorithms theoretical *guarantees* of finding (approximately) optimal solutions, at least in the limit, are known. As demonstrated by our experiments, the actual *convergence speed* for RAPs is usually high, especially in the case of applying the described improvements, such as clustering or distributed sampling.

4. The simulation experiments on industrial data also demonstrate that RL based solutions *scale well* with the workload and the size of the problem and, therefore, they can be effectively applied to handle real world problems, such as production control.

5. *Domain specific knowledge* can also be incorporated into the solution. The base policy of the rollout algorithm, for example, can reflect a priori knowledge about the structure of the problem; later this knowledge may appear in the exploration strategy.

6. Finally, the proposed method constitutes an *any-time* solution, since the sampling can be stopped after any number of iterations. By this way, the amount of computational time can be controlled, which is also an important practical advantage.

Consequently, ML approaches have great potentials in dealing with real world RAPs, since they can handle large-scale problems even in dynamic and uncertain environments.

5.2 Dealing with Changes

The theory of MDPs provide a general framework for modeling decision making in stochastic dynamic systems, if we know a function that describes the dynamics or we can simulate it, for example, with a suitable program. In some situations, however, the dynamics of the system may change, too. In theory, this change can be modeled with another (higher level) MDP, as well, but doing so would lead to models which are practically intractable.

In the dissertation we have argued that the value function of a (stationary, Markovian, randomized) control policy in a (discounted or acyclic undiscounted) MDP Lipschitz continuously depends on the transition-probability function and the immediate-cost function, therefore, small changes in the environment result only in small changes in the value function. A similar result was already known for optimal value functions in the case of transition-probabilities, but we have presented an improved estimation for that case, as well. A bound for changes in the discount factor was also proven, and it was demonstrated that this dependence was not Lipschitz continuous. Additionally, it was proven that changes in the discount rate could be traced back to changes in the cost function. Though, optimal policies may also change if the environment had changed, we showed that the previous results can be extended to optimal value functions, as well. The application of the Lipschitz property helps the theoretical treatment of changing environments or inaccurate models, e.g., if the transition-probabilities or the immediate-costs are estimated statistically, only.

In order to theoretically analyze environmental changes, the framework of (ε, δ)-MDPs was introduced as a generalization of classical MDPs and ε-MDPs. In this quasi-stationary model the transition-probability function and the immediate-cost function may change over time, but the cumulative changes must remain bounded by ε and δ, asymptotically.

Afterwards, we have investigated how RL methods could work in this kind of changing environment. We have presented a general theorem that estimated the asymptotic distance of a value function sequence from a fixed value function. This result was applied to deduce a convergence theorem for value function based algorithms that work in (ε, δ)-MDPs.

In order to demonstrate our approach, we have presented some numerical experiments, too. First, two simple iterative processes were shown, a “well-behaving” stochastic process and a “pathological”, oscillating deterministic process. Later, the effects of environmental changes on Q-learning based flexible job-shop scheduling was experimentally studied. Finally, we have analyzed how SARSA could work in varying (ε, δ)-type grid world domains.

We can conclude that value function based RL algorithms can work in varying environments, at least if the changes remain bounded in the limit. The asymptotic distance of the generated value function sequence from the optimal value function of base MDP of the changing environment is bounded for a large class of stochastic iterative algorithms. Moreover, this bound is proportional to the diameter of this set, e.g., to parameters ε and δ in the case of (ε, δ)-MDPs. These results were illustrated through three classical RL methods: asynchronous value iteration, Q-learning and temporal difference learning policy evaluation.

5.3 Further Research Directions

Several further research directions are possible to enhance the proposed ML based resource allocation approach. Now, as a conclusion to the thesis, we highlight some of them.

The suggested improvements, such as *clustering* and *distributed sampling*, should be further investigated, both theoretically and experimentally, since they resulted in considerable speedup. The guidance of reinforcement learning with *rollout* algorithms might be effectively applied in other applications, as well. Regarding *proactive* solutions, efficient algorithms to handle partially observable MDPs (POMDPs) should also be studied. The theoretical analysis of the *average* effects of environmental changes on the value functions of a policy could result in new approaches to handle disturbances. Another promising direction would be to extend the solution in a way which also takes *risk* into account and, for example, minimizes not only the expected value of the total costs but also the *deviation*, as a secondary optimization criterion. Finally, trying to apply the solution in a *pilot* project to control a real plant would be interesting and could motivate many more further research directions.

Concerning changes in the dynamics, a promising direction could be to investigate environments with non-bounded changes, e.g., when the environment might *drift* over time. Naturally, this drift should also be sufficiently slow in order to give the opportunity to the learning algorithm to *track* the changes. Another possible direction could be the analysis of the convergence results in case of applying *value function approximation.* The classical problem of *exploration* and *exploitation* should also be reinvestigated in changing environments. Finally, finding *finite time bounds* for the convergence of stochastic iterative algorithms for (a potentially restricted class of) non-stationary environments would also be important.

Appendix: Proofs

In this appendix we present the proofs of all theorems and lemmas we gave. Naturally, we do not prove those statements which were taken from other sources. In the latter cases adequate references are given in the text concerning the locations of the proofs.

Theorem 7 *Consider a POMDP and its fully observable MDP counterpart, which system has the same state space, action space, transition-probabilities and costs as the original POMDP, only the observability is different. The optimal cost-to-go functions of the POMDP and the MDP are denoted by $\tilde{J}^*$ and J^*, respectively. Then, for all belief state b, we have*

$$\sum_{x \in \mathbb{X}} b(x)\, J^*(x) \le \tilde{J}^*(b) \le \sum_{x \in \mathbb{X}} b(x)\, J^*(x) + \frac{c}{1-\alpha},$$

where $c = (g_{max} - g_{min})$; $g_{max} = \max\{g(x,a) \mid x \in \mathbb{X}, a \in \mathbb{A}\}$ and similarly for g_{min}. If the immediate-cost function g is not constant ($c \neq 0$), then the second inequality is strict.

Proof We will prove the theorem in two steps. First, we will show that for all belief states, $\sum_{x \in \mathbb{X}} b(x)\, J^*(x) \le \tilde{J}^*(b)$, then we will prove that $\tilde{J}^*(b) \le \sum_{x \in \mathbb{X}} b(x)\, J^*(x) + c/(1-\alpha)$, where $c = (g_{max} - g_{min})$. The first part of the theorem will be proven by induction. We can treat our infinite horizon problem as the limit of the finite horizon problems and, thus, we can apply induction on the horizon. The Bellman equation for the N-stage problem is

$$J^*_{k+1}(x) = \min_{a \in \mathcal{A}(x)} \left[g(x,a) + \alpha \sum_{y \in \mathbb{X}} p(y \mid x,a)\, J^*_k(y) \right], \tag{5.1}$$

for all $k \in \{0, \dots, N-1\}$ and $x \in \mathbb{X}$. Note that $J^*_0 = 0$ by definition (we do not have to consider terminal cost). It is known, (see Bertsekas and Tsitsiklis, 1996), that for all $x \in \mathbb{X}$

$$J^*(x) = J^*_\infty(x) = \lim_{N \to \infty} J^*_N(x) \tag{5.2}$$

It is obvious that $\sum_{x \in \mathbb{X}} J^*_N(x)\, b(x) \le \tilde{J}^*_N(b)$ holds if $N = 0$, since $J^*_0 \equiv \tilde{J}^*_0 \equiv 0$ (they are both constant zero functions). Now, we assume that it holds for N and prove it for $N+1$:

$$\tilde{J}^*_{N+1}(b) = \min_{a \in \mathbb{A}} \left[g(b,a) + \alpha \sum_{z \in \mathbb{O}} p(z \mid b,a)\, \tilde{J}^*_N(\tau(b,a,z)) \right] \ge$$

$$\geq \min_{a\in\mathbb{A}} \left[g(b,a) + \alpha \sum_{z\in\mathbb{O}} p(z \mid b,a) \sum_{y\in\mathbb{X}} \tau(b,a,z)(y)\, J^*_N(y) \right] =$$

$$= \min_{a\in\mathbb{A}} \left[\sum_{x\in\mathbb{X}} b(x)\, g(x,a) + \alpha \sum_{z\in\mathbb{O}} p(z \mid b,a) \sum_{y\in\mathbb{X}} \frac{\sum_{x\in\mathbb{X}} p(z,y \mid x,a)\, b(x)}{p(z \mid b,a)} J^*_N(y) \right] =$$

$$= \min_{a\in\mathbb{A}} \left[\sum_{x\in\mathbb{X}} b(x)\, g(x,a) + \alpha \sum_{z\in\mathbb{O}} \sum_{y\in\mathbb{X}} \sum_{x\in\mathbb{X}} p(z,y \mid x,a)\, b(x)\, J^*_N(y) \right] =$$

$$= \min_{a\in\mathbb{A}} \left[\sum_{x\in\mathbb{X}} b(x) \left[g(x,a) + \alpha \sum_{z\in\mathbb{O}} \sum_{y\in\mathbb{X}} p(z \mid y,a)\, p(y \mid x,a)\, J^*_N(y) \right] \right] =$$

$$= \min_{a\in\mathbb{A}} \left[\sum_{x\in\mathbb{X}} b(x) \left[g(x,a) + \alpha \sum_{y\in\mathbb{X}} p(y \mid x,a)\, J^*_N(y) \right] \right] \geq$$

now, we apply *Jensen's inequality* with the remark that min is a concave function

$$\geq \sum_{x\in\mathbb{X}} b(x) \min_{a\in\mathbb{A}} \left[g(x,a) + \alpha \sum_{y\in\mathbb{X}} p(y \mid x,a)\, J^*_N(y) \right] = \sum_{x\in\mathbb{X}} b(x)\, J^*_{N+1}(x)$$

for all $b \in \mathbb{B}$. Now, we turn to the second part of the theorem and show that for all $b \in \mathbb{B} : \tilde{J}^*(b) \leq \sum_{x\in\mathbb{X}} b(x)\, J^*(x) + (g_{max} - g_{min})/(1-\alpha)$. In possession of the knowledge that the first part of the theorem holds, the second part can be proven as follows. We can notice that at each step the maximum difference of the incurred cost between the convex combination of the optimal cost-to-go functions of the fully observable and the non-observable (belief state) MDPs (that also uses a convex combination for cost) is $g_{max} - g_{min}$, which is denoted by c and it is discounted by α. This simply builds a geometric series: $c + \alpha c + \alpha^2 c + \cdots = c/(1-\alpha)$. To show that this inequality is strict, suppose (indirectly) that $\tilde{J}^*(b) = \sum_x b(x)\, J^*(x) + c/(1-\alpha)$. It means that, at each step the difference is $g_{max} - g_{min}$, even at the first step. Thus, it must be the case that $\sum_x J^*_1(x)\, b(x) = g_{min}$, therefore, for all state x such that $b(x) \neq 0$ there must be an action a such that $g(x,a) = g_{min}$. Additionally, in order to satisfy the equality, $\tilde{J}^*_1(b) = g_{max}$ must hold. It means that the best action a in b has the cost $g(b,a) = \sum_x b(x)\, g(x,a) = g_{max}$, which cannot be the best action (if $c \neq 0$), since for each state x (such that $b(x) \neq 0$) there is an action a that has minimal cost, $g(x,a) = g_{min}$. We have reached a contradiction, therefore, the inequality is strict. ■

Lemma 8 *(1) There exists a POMDP and a corresponding fully observable MDP such that, for a belief state b: $\tilde{J}^*(b) = \langle J^*, b\rangle$; (2) for all $\varepsilon > 0$ there exists a POMDP and a corresponding MDP such that for a belief state b: $|\tilde{J}^*(b) - \langle J^*, b\rangle - c/(1-\alpha)| < \varepsilon$.*

Proof The lemma is trivial if constant immediate-cost functions are allowed. Let us now assume that we have a non-constant cost function. We can also assume, without loss of generality, that $g_{min} = 0$ and $g_{max} = 1$. Then, the first statement of the lemma is still straightforward to prove, since any POMDP which is deterministic or which is fully observable (e.g., the observations are the real states) can do with a $b \in \mathbb{B}$ such that $b(x) = 1$ for an $x \in \mathbb{X}$. If the system is deterministic, then this result is valid for NOMDPs, as well.

The second part of the lemma can be shown through an example, moreover, the applied POMDP will be a NOMDP (viz., its observation space contains only one element). Consider a NOMDP with $\mathbb{X} = \{x_1, \dots, x_n\}$, $\mathbb{A} = \{a_1, \dots, a_n\}$, for all $x_i : \mathcal{A}(x_i) = \{a_1, \dots, a_n\}$. For all x_i and a_j the immediate-cost $g(x_i, a_j) = 0$ if $i = j$ and $g(x_i, a_j) = 1$ otherwise. The transition-probabilities for all $x, y \in \mathbb{X}$ and $a \in \mathbb{A}$: $p(y \mid x, a) = 1/n$. Then, if the system was fully observable, the optimal cost-to-go function would be $J^*(x) = 0$ for all $x \in \mathbb{X}$, because in each state x_i one could choose an action (a_i) that did not have any cost. However, consider a belief state b such that $b(x) = 1/n$ for all $x \in \mathbb{X}$. In this case for each action $a \in \mathbb{A}$ its immediate-cost is $g(b, a) = \sum_x b(x) g(x, a) = (n-1)/n$, therefore, the optimal control policy in b also incur $(n-1)/n$ cost before it arrives back to belief state b. In the long run, $\tilde{J}^*(b) = (1/(1-\alpha)) \cdot ((n-1)/n)$. Because $c = (g_{max} - g_{min}) = 1$ and for any $\varepsilon > 0$ we can choose a large enough n such that $1 - (n-1)/n < \varepsilon$, consequently, $\tilde{J}^*(b)$ can be arbitrary close to $\langle J^*, b \rangle + c/(1-\alpha) = 1/(1-\alpha)$. Thus, we proved both parts. ■

Theorem 9 *Assume that two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, differ only in their transition-probability functions, and let these two functions be denoted by p_1 for $\mathcal{M}_1$ and p_2 for $\mathcal{M}_2$. Let the value functions of a (stationary, Markovian, randomized) control policy π be denoted by J_1^π related to $\mathcal{M}_1$ and J_2^π related to $\mathcal{M}_2$. Then, the following inequality holds*

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{\alpha\, |\mathbb{X}|\, \|g\|_\infty}{(1-\alpha)^2} \|p_1 - p_2\|_\infty .$$

Proof Using the Bellman equation in the first step, the absolute value of the difference of the two cost-to-go functions in any state can be estimated as follows

$$\forall x \in \mathbb{X} : |J_1^\pi(x) - J_2^\pi(x)| =$$

$$= \Bigg| \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big[g(x,a) + \alpha \sum_{y \in \mathbb{X}} p_1(y \mid x,a)\, J_1^\pi(y) \Big] -$$

$$- \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big[g(x,a) + \alpha \sum_{y \in \mathbb{X}} p_2(y \mid x,a)\, J_2^\pi(y) \Big] \Bigg| \leq$$

$$\leq \alpha \sum_{a \in \mathcal{A}(x)} \pi(x,a) \left| \sum_{y \in \mathbb{X}} p_1(y \mid x,a)\, J_1^\pi(y) - \sum_{y \in \mathbb{X}} p_2(y \mid x,a)\, J_2^\pi(y) \right| =$$

$$= \alpha \sum_{a \in \mathcal{A}(x)} \pi(x,a) \left| \sum_{y \in \mathbb{X}} (p_1(y \mid x,a) - p_2(y \mid x,a))\, J_1^\pi(y) + \sum_{y \in \mathbb{X}} p_2(y \mid x,a)\, (J_1^\pi(y) - J_2^\pi(y)) \right|,$$

where we have rewritten $p_1(y\,|\,x,a)J_1^\pi(y) - p_2(y\,|\,x,a)J_2^\pi(y)$ as

$$p_1(y\,|\,x,a)J_1^\pi(y) - p_2(y\,|\,x,a)J_2^\pi(y) =$$

$$= p_1(y\,|\,x,a)J_1^\pi(y) - p_2(y\,|\,x,a)J_1^\pi(y) + p_2(y\,|\,x,a)J_1^\pi(y) - p_2(y\,|\,x,a)J_2^\pi(y) =$$

$$= (p_1(y\,|\,x,a) - p_2(y\,|\,x,a))J_1^\pi(y) + p_2(y\,|\,x,a)(J_1^\pi(y) - J_2^\pi(y)).$$

Now, we can continue the estimation using the fact that $\forall y : J_1^\pi(y) \leq \|g\|_\infty / (1-\alpha)$.

$$\alpha \sum_{a\in\mathcal{A}(x)} \pi(x,a) \left| \sum_{y\in\mathbb{X}} (p_1(y \mid x,a) - p_2(y \mid x,a))\, J_1^\pi(y) + \sum_{y\in\mathbb{X}} p_2(y \mid x,a)\,(J_1^\pi(y) - J_2^\pi(y)) \right| \leq$$

$$\leq \alpha \sum_{a\in\mathcal{A}(x)} \pi(x,a) \left| \sum_{y\in\mathbb{X}} (p_1(y \mid x,a) - p_2(y \mid x,a)) \frac{\|g\|_\infty}{1-\alpha} + \sum_{y\in\mathbb{X}} p_2(y \mid x,a)\, \|J_1^\pi - J_2^\pi\|_\infty \right| \leq$$

$$\leq \alpha \sum_{a\in\mathcal{A}(x)} \pi(x,a) \left| \sum_{y\in\mathbb{X}} \|p_1 - p_2\|_\infty \frac{\|g\|_\infty}{1-\alpha} + \|J_1^\pi - J_2^\pi\|_\infty \right| \leq$$

$$\leq \alpha \sum_{a\in\mathcal{A}(x)} \pi(x,a) \left| |\mathbb{X}|\, \|p_1 - p_2\|_\infty \frac{\|g\|_\infty}{1-\alpha} + \|J_1^\pi - J_2^\pi\|_\infty \right| \leq$$

$$\alpha \left| |\mathbb{X}|\, \|p_1 - p_2\|_\infty \frac{\|g\|_\infty}{1-\alpha} + \|J_1^\pi - J_2^\pi\|_\infty \right| =$$

$$\alpha\, |\mathbb{X}|\, \|p_1 - p_2\|_\infty \frac{\|g\|_\infty}{1-\alpha} + \alpha\, \|J_1^\pi - J_2^\pi\|_\infty.$$

Since we have this estimation for all x, we also have

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \alpha\, |\mathbb{X}|\, \|p_1 - p_2\|_\infty \frac{\|g\|_\infty}{1-\alpha} + \alpha\, \|J_1^\pi - J_2^\pi\|_\infty,$$

after rearrangement, we get

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{\alpha\, |\mathbb{X}|\, \|g\|_\infty}{(1-\alpha)^2} \|p_1 - p_2\|_\infty.$$

■

Theorem 10 *Assume that two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, differ only in their transition-probability functions, and let these two functions be denoted by p_1 for $\mathcal{M}_1$ and p_2 for $\mathcal{M}_2$. Let the value functions of a (stationary, Markovian, randomized) control policy π be denoted by J_1^π related to $\mathcal{M}_1$ and J_2^π related to $\mathcal{M}_2$. Then, the following inequality holds*

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{\alpha\, \|g\|_\infty}{(1-\alpha)^2} \|p_1 - p_2\|_1\,,$$

where $\|\cdot\|_1$ *is a norm on* $f : \mathbb{X} \times \mathbb{A} \times \mathbb{X} \to \mathbb{R}$ *type functions, e.g.,* $f(x, a, y) = p(y \,|\, x, a)$,

$$\|f\|_1 = \max_{x,\, a} \sum_{y \in \mathbb{X}} |\, f(x, a, y) \,| \,.$$

Proof Using the Bellman equation in the first step, $\|J_1^\pi - J_2^\pi\|_\infty$ can be estimated as

$$\forall x \in \mathbb{X} : |J_1^\pi(x) - J_2^\pi(x)| =$$

$$= \left| \sum_{a \in \mathcal{A}(x)} \pi(x, a) \Big[g(x, a) + \alpha \sum_{y \in \mathbb{X}} p_1(y \mid x, a)\, J_1^\pi(y) \Big] - \right.$$

$$\left. \sum_{a \in \mathcal{A}(x)} \pi(x, a) \Big[g(x, a) + \alpha \sum_{y \in \mathbb{X}} p_2(y \mid x, a)\, J_2^\pi(y) \Big] \right| \leq$$

$$\leq \alpha \sum_{a \in \mathcal{A}(x)} \pi(x, a) \left| \sum_{y \in \mathbb{X}} p_1(y \mid x, a) J_1^\pi(y) - p_2(y \mid x, a) J_2^\pi(y) \right| =$$

$$= \alpha \sum_{a \in \mathcal{A}(x)} \pi(x, a) \left| \sum_{y \in \mathbb{X}} (p_1(y \mid x, a) - p_2(y \mid x, a)) J_1^\pi(y) + \sum_{y \in \mathbb{X}} p_2(y \mid x, a)(J_1^\pi(y) - J_2^\pi(y)) \right| \leq$$

$$\leq \alpha \sum_{a \in \mathcal{A}(x)} \pi(x, a) \left[\sum_{y \in \mathbb{X}} |(p_1(y \mid x, a) - p_2(y \mid x, a)) J_1^\pi(y)| + \sum_{y \in \mathbb{X}} |p_2(y \mid x, a)(J_1^\pi(y) - J_2^\pi(y))| \right],$$

where we also applied the reformulation of $p_1(y \,|\, x, a) J_1^\pi(y) - p_2(y \,|\, x, a) J_2^\pi(y)$ as

$$p_1(y \,|\, x, a) J_1^\pi(y) - p_2(y \,|\, x, a) J_2^\pi(y) =$$

$$= p_1(y \,|\, x, a) J_1^\pi(y) - p_2(y \,|\, x, a) J_1^\pi(y) + p_2(y \,|\, x, a) J_1^\pi(y) - p_2(y \,|\, x, a) J_2^\pi(y) =$$

$$= (p_1(y \,|\, x, a) - p_2(y \,|\, x, a)) J_1^\pi(y) + p_2(y \,|\, x, a)(J_1^\pi(y) - J_2^\pi(y)).$$

Now, let us recall (a special form of) *Hölder's inequality*: let v_1, v_2 be two vectors and $1 \leq q, r \leq \infty$ with $1/q + 1/r = 1$. Then, we have $\|v_1\, v_2\|_{(1)} \leq \|v_1\|_{(q)}\, \|v_2\|_{(r)}$, where $\|\cdot\|_{(q)}$ denotes *vector* norm, e.g., $\|v\|_{(q)} = (\sum_i |v_i|^q)^{1/q}$ and $\|v\|_{(\infty)} = \max_i |v_i| = \|v\|_\infty$. Here, we applied the unusual "(q)" notation to avoid confusion with the applied matrix norm. Notice that the first sum in the rectangular brackets can be treated as the (1)-norm of $v_1\, v_2$, where

$$v_1(y) = p_1(y \mid x, a) - p_2(y \mid x, a)) \qquad \text{and} \qquad v_2(y) = J_1^\pi(y),$$

after which Hölder's inequality can be applied with $q = 1$ and $r = \infty$ to estimate the sum. A similar argument can be repeated in the case of the second sum in the brackets with

$$v_1(y) = p_2(y \mid x, a) \qquad \text{and} \qquad v_2(y) = J_1^\pi(y) - J_2^\pi(y).$$

Then, after the two applications of Hölder's inequality, we have the following estimation

$$\forall x \in \mathbb{X} : |J_1^\pi(x) - J_2^\pi(x)| \leq$$

$$\leq \alpha \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big(\|p_1(\,\cdot\,|\,x,a) - p_2(\,\cdot\,|\,x,a)\|_{(1)} \, \|J_1^\pi\|_\infty + \|p_2(\,\cdot\,|\,x,a)\|_{(1)} \, \|J_1^\pi - J_2^\pi\|_\infty \Big) \leq$$

$$\leq \alpha \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big(\|p_1(\,\cdot\,|\,x,a) - p_2(\,\cdot\,|\,x,a)\|_{(1)} \, \frac{\|g\|_\infty}{1-\alpha} + \|J_1^\pi - J_2^\pi\|_\infty \Big),$$

where in the last step we applied that $\|J_1^\pi\|_\infty \leq \|g\|_\infty / (1-\alpha)$ and $\|p_2(\,\cdot\,|\,x,a)\|_{(1)} = 1$. The last formula can be overestimated by taking the maximum over all states and actions,

$$\forall x \in \mathbb{X} : |J_1^\pi(x) - J_2^\pi(x)| \leq$$

$$\leq \alpha \sum_{a \in \mathcal{A}(x)} \pi(x,a) \left(\max_{(z,b) \in \mathbb{X} \times \mathbb{A}} \sum_{y \in \mathbb{X}} |\, p_1(y\,|\,z,b) - p_2(y\,|\,z,b)\,| \, \frac{\|g\|_\infty}{1-\alpha} + \alpha \, \|J_1^\pi - J_2^\pi\|_\infty \right) \leq$$

$$\leq \alpha \sum_{a \in \mathcal{A}(x)} \pi(x,a) \left(\frac{\|g\|_\infty}{1-\alpha} \, \|p_1 - p_2\|_1 + \alpha \, \|J_1^\pi - J_2^\pi\|_\infty \right) =$$

$$= \frac{\alpha \, \|g\|_\infty}{1-\alpha} \, \|p_1 - p_2\|_1 + \alpha \, \|J_1^\pi - J_2^\pi\|_\infty \, ,$$

since we have this estimation for all state x, we also have the following inequality

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{\alpha \, \|g\|_\infty}{1-\alpha} \, \|p_1 - p_2\|_1 + \alpha \, \|J_1^\pi - J_2^\pi\|_\infty \, ,$$

from which the statement of the theorem immediately follows after rearrangement,

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{\alpha \, \|g\|_\infty}{(1-\alpha)^2} \, \|p_1 - p_2\|_1 \, .$$

■

Theorem 11 *Assume that two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, differ only in their immediate-cost functions, and let these two functions be denoted by g_1 for $\mathcal{M}_1$ and g_2 for $\mathcal{M}_2$, respectively. Let the value functions of a (stationary, Markovian, randomized) policy π be denoted by J_1^π related to $\mathcal{M}_1$ and J_2^π related to $\mathcal{M}_2$. Then, the following inequality holds*

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{1}{1-\alpha} \, \|g_1 - g_2\|_\infty \, .$$

Proof Using the Bellman equation in the first step, the absolute value of the difference of the two cost-to-go functions in any state can be estimated as follows

$$\forall x \in \mathbb{X} : |J_1^\pi(x) - J_2^\pi(x)| =$$

$$= \Bigg| \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big[g_1(x,a) + \alpha \sum_{y \in \mathbb{X}} p(y \mid x,a)\, J_1^\pi(y) \Big] -$$

$$- \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big[g_2(x,a) + \alpha \sum_{y \in \mathbb{X}} p(y \mid x,a)\, J_2^\pi(y) \Big] \Bigg| \leq$$

$$\leq \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big(|g_1(x,a) - g_2(x,a)| + \alpha \sum_{y \in \mathbb{X}} p(y \mid x,a)\, |J_1^\pi(y) - J_2^\pi(y)| \Big) \leq$$

$$\leq \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big(\|g_1 - g_2\|_\infty + \alpha \sum_{y \in \mathbb{X}} p(y \mid x,a)\, \|J_1^\pi - J_2^\pi\|_\infty \Big) =$$

$$= \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Big(\|g_1 - g_2\|_\infty + \alpha\, \|J_1^\pi - J_2^\pi\|_\infty \Big) =$$

$$= \|g_1 - g_2\|_\infty + \alpha\, \|J_1^\pi - J_2^\pi\|_\infty \,.$$

It is easy to see that if

$$\forall x \in \mathbb{X} : |J_1^\pi(x) - J_2^\pi(x)| \leq \|g_1 - g_2\|_\infty + \alpha\, \|J_1^\pi - J_2^\pi\|_\infty \,,$$

then

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \|g_1 - g_2\|_\infty + \alpha\, \|J_1^\pi - J_2^\pi\|_\infty \,,$$

after rearrangement

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{1}{1-\alpha}\, \|g_1 - g_2\|_\infty \,.$$

■

Theorem 12 *Assume that two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, differ only in the discount factors, denoted by $\alpha_1, \alpha_2 \in [0,1)$. Let the value functions of a (stationary, Markovian, randomized) policy π be denoted by J_1^π related to $\mathcal{M}_1$ and J_2^π related to $\mathcal{M}_2$. Then, we have*

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{|\alpha_1 - \alpha_2|}{(1-\alpha_1)(1-\alpha_2)}\, \|g\|_\infty \,.$$

Proof For any state x the difference of the two value functions can be estimated as follows,

$$|J_1^\pi(x) - J_2^\pi(x)| =$$

$$= \Bigg| \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Bigg[g(x,a) + \alpha_1 \sum_{y \in \mathbb{X}} p(y \mid x,a)\, J_1^\pi(y) \Bigg] -$$

$$- \sum_{a \in \mathcal{A}(x)} \pi(x,a) \Bigg[g(x,a) + \alpha_2 \sum_{y \in \mathbb{X}} p(y \mid x,a)\, J_2^\pi(y) \Bigg] \Bigg| \leq$$

$$\leq \sum_{a \in \mathcal{A}(x)} \pi(x,a) \left| \sum_{y \in \mathbb{X}} p(y \mid x,a) \left(\alpha_1 J_1^\pi(y) - \alpha_2 J_2^\pi(y) \right) \right| \leq$$

$$\leq |\alpha_1 - \alpha_2| \frac{1}{1-\alpha_1} \|g\|_\infty + \alpha_2 \|J_1^\pi - J_2^\pi\|_\infty \,,$$

where in the last step we used the following estimation of $|\alpha_1 J_1^\pi(y) - \alpha_2 J_2^\pi(y)|$,

$$|\alpha_1 J_1^\pi(y) - \alpha_2 J_2^\pi(y)| = |\alpha_1 J_1^\pi(y) - \alpha_2 J_1^\pi(y) + \alpha_2 J_1^\pi(y) - \alpha_2 J_2^\pi(y)| \leq$$

$$\leq |\alpha_1 - \alpha_2| \, |J_1^\pi(y)| + \alpha_2 \, |J_1^\pi(y) - J_2^\pi(y)| \leq |\alpha_1 - \alpha_2| \frac{1}{1-\alpha_1} \|g\|_\infty + \alpha_2 \|J_1^\pi - J_2^\pi\|_\infty \,,$$

where we applied the fact that for any state y we have,

$$|J_1^\pi(y)| \leq \sum_{t=0}^{\infty} \alpha_1^t \|g\|_\infty = \frac{1}{1-\alpha_1} \|g\|_\infty \,.$$

Because the estimation of $|J_1^\pi(x) - J_2^\pi(x)|$ is valid for all x, we have the following result

$$\|J_1^\pi - J_2^\pi\|_\infty \leq |\alpha_1 - \alpha_2| \frac{1}{1-\alpha_1} \|g\|_\infty + \alpha_2 \|J_1 - J_2\|_\infty \,,$$

from which the statement of the theorem immediately follows after rearrangement,

$$\|J_1^\pi - J_2^\pi\|_\infty \leq \frac{|\alpha_1 - \alpha_2|}{(1-\alpha_1)(1-\alpha_2)} \|g\|_\infty \,.$$

■

Lemma 13 *Assume that we have two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, which differ only in the transition-probability functions or only in the immediate-cost functions or only in the discount factors. Let π denote an arbitrary (stationary, Markovian, randomized) control policy. The (state-) value and action-value functions of control policy π are denoted by J_1^π, Q_1^π and J_2^π, Q_2^π for $\mathcal{M}_1$ and $\mathcal{M}_2$, respectively. Then, the previously given value function bounds for $\|J_1^\pi - J_2^\pi\|_\infty$ of Theorems 9, 10, 11 and 12 are also bounds for $\|Q_1^\pi - Q_2^\pi\|_\infty$.*

Proof We will prove the theorem in three parts, depending on the changing components. Since Theorem 9 follows from Theorem 10 we will only prove the latter statement.

Case 1: Assume that the MDPs differ only in the transition functions, denoted by p_1 and p_2. We will prove the same estimation as in the case of Theorem 10, more precisely, that

$$\|Q_1^\pi - Q_2^\pi\|_\infty \leq \frac{\alpha \|g\|_\infty}{(1-\alpha)^2} \|p_1 - p_2\|_1 \,.$$

For all state-action pair (x,a) we can estimate the absolute difference of Q_1^π and Q_2^π as

$$|Q_1^\pi(x,a) - Q_2^\pi(x,a)| =$$

$$= \left| g(x,a) + \alpha \sum_{y\in\mathbb{X}} p_1(y\,|\,x,a) J_1^\pi(y) - g(x,a) - \alpha \sum_{y\in\mathbb{X}} p_2(y\,|\,x,a) J_2^\pi(y) \right| \leq$$

$$\leq \left| \alpha \sum_{y\in\mathbb{X}} \left(p_1(y\,|\,x,a) J_1^\pi(y) - p_2(y\,|\,x,a) J_2^\pi(y) \right) \right|,$$

from which the proof continues in the same way as the proof of Theorem 10.

Case 2: Assume that the MDPs differ only in the immediate-cost functions, denoted by g_1 and g_2. We will prove the same estimation as in the case of Theorem 11, more precisely,

$$\|Q_1^\pi - Q_2^\pi\|_\infty \leq \frac{1}{1-\alpha} \|g_1 - g_2\|_\infty .$$

For all state-action pair (x,a) we can estimate the absolute difference of Q_1^π and Q_2^π as

$$|Q_1^\pi(x,a) - Q_2^\pi(x,a)| =$$

$$= \left| g_1(x,a) + \alpha \sum_{y\in\mathbb{X}} p(y\,|\,x,a) J_1^\pi(y) - g_2(x,a) - \alpha \sum_{y\in\mathbb{X}} p(y\,|\,x,a) J_2^\pi(y) \right| \leq$$

$$\leq \|g_1 - g_2\|_\infty + \left| \alpha \sum_{y\in\mathbb{X}} p(y\,|\,x,a) (J_1^\pi(y) - J_2^\pi(y)) \right| \leq \|g_1 - g_2\|_\infty + \alpha \|J_1^\pi - J_2^\pi\|_\infty .$$

The statement immediately follows after we apply Theorem 11 to estimate $\|J_1^\pi - J_2^\pi\|_\infty$.

Case 3: Assume that the MDPs differ only in the discount rates, denoted by α_1 and α_2. We will prove the same estimation as in the case of Theorem 12, more precisely, that

$$\|Q_1^\pi - Q_2^\pi\|_\infty \leq \frac{|\alpha_1 - \alpha_2|}{(1-\alpha_1)(1-\alpha_2)} \|g\|_\infty .$$

For all state-action pair (x,a) we can estimate the absolute difference of Q_1^π and Q_2^π as

$$|Q_1^\pi(x,a) - Q_2^\pi(x,a)| =$$

$$= \left| g(x,a) + \alpha_1 \sum_{y\in\mathbb{X}} p(y\,|\,x,a) J_1^\pi(y) - g(x,a) - \alpha_2 \sum_{y\in\mathbb{X}} p(y\,|\,x,a) J_2^\pi(y) \right| \leq$$

$$\leq \left| \alpha_1 \sum_{y\in\mathbb{X}} p(y\,|\,x,a) J_1^\pi(y) - \alpha_2 \sum_{y\in\mathbb{X}} p(y\,|\,x,a) J_2^\pi(y) \right| \leq |\alpha_1 - \alpha_2| \frac{1}{1-\alpha_1} \|g\|_\infty + \alpha_2 \|J_1^\pi - J_2^\pi\|_\infty ,$$

where in the last step we applied the same estimation as in the proof of Theorem 12. The statement immediately follows after we apply Theorem 12 to estimate $\|J_1^\pi - J_2^\pi\|_\infty$. ■

Lemma 14 *For all $f_1, f_2 : \mathcal{X} \to \mathbb{R}$ bounded functions such that $\min_x f_1(x) \leq \min_x f_2(x)$ and $\hat{x} = \arg\min_x f_1(x)$, we have the inequality $|\min_x f_1(x) - \min_x f_2(x)| \leq |f_1(\hat{x}) - f_2(\hat{x})|$.*

Proof First, we may assume w.l.o.g. that $\forall x \in \mathcal{X} : f_1(x), f_2(x) \geq 0$, e.g., we can shift them in order to achieve this (since they are bounded and a parallel translation does not change the distance between them). Now, assume that $\min_x f_1(x) \leq \min_x f_2(x)$ and $\hat{x} = \arg\min_x f_1(x)$. Applying the above assumptions, we have $|\min_x f_1(x) - \min_x f_2(x)| = \min_x f_2(x) - \min_x f_1(x)$ and since we have assumed that $\min_x f_1(x) \leq \min_x f_2(x) \leq f_2(\hat{x})$, we also have that $|\min_x f_1(x) - f_2(\hat{x})| = f_2(\hat{x}) - \min_s f_1(x)$.

Now, we can assume *indirectly* that $|\min_x f_1(x) - \min_x f_2(x)| > |f_1(\hat{x}) - f_2(\hat{x})|$. Then:

$$\left|\min_x f_1(x) - \min_x f_2(x)\right| > |f_1(\hat{x}) - f_2(\hat{x})|$$

$$\min_x f_2(x) - \min_x f_1(x) > |f_1(\hat{x}) - f_2(\hat{x})|$$

$$\min_x f_2(x) - \min_x f_1(x) > \left|\min_x f_1(x) - f_2(\hat{x})\right|$$

$$\min_x f_2(x) - \min_x f_1(x) > f_2(\hat{x}) - \min_x f_1(x)$$

$$\min_x f_2(x) > f_2(\hat{x})$$

We have reached a *contradiction*, therefore, the original statement was true. ■

Lemma 24 *Assume that two discounted MDPs, $\mathcal{M}_1$ and $\mathcal{M}_2$, differ only in the discount factors, denoted by α_1 and α_2. Let the value function of a (stationary, Markovian, randomized) control policy π be denoted by J_i^π related to $\mathcal{M}_i$. The optimal value function of $\mathcal{M}_i$ is denoted by J_i^*. We will treat both cases simultaneously, thus, let us fix hyper-parameter μ to either π or $*$. Then, there exists an MDP, denoted by $\mathcal{M}_3$, such that it differs only in the immediate-cost function from $\mathcal{M}_1$ and $J_2^\mu = J_3^\mu$. The immediate-cost function of $\mathcal{M}_3$ is*

$$\widehat{g}(x, a) = g(x, a) + (\alpha_2 - \alpha_1) \sum_{y \in \mathbb{X}} p(y \,|\, x, a) J_2^\mu(y),$$

where p is the transition-probability function of $\mathcal{M}_1$, $\mathcal{M}_2$ and $\mathcal{M}_3$; g is the immediate-cost function of $\mathcal{M}_1$ and $\mathcal{M}_2$; and $J_2^\mu(y)$ denotes a value function of $\mathcal{M}_2$, where $\mu \in \{\pi, *\}$.

Proof First of all, let us overview some general statements that will be used in the proof.

Recall (Bertsekas and Tsitsiklis, 1996) that we can treat the optimal value function or the value function of a control policy regarding the infinite horizon problem as the limit of finite horizon value functions. The Bellman equations for the n-stage problems are

$$J_k^*(x) = \min_{a \in \mathcal{A}(x)} \Big[g(x, a) + \alpha \sum_{y \in \mathbb{X}} p(y \mid x, a)\, J_{k-1}^*(y) \Big],$$

$$J_k^\pi(x) = \sum_{a \in \mathcal{A}(x)} \pi(x, a) \Big[g(x, a) + \alpha \sum_{y \in \mathbb{X}} p(y \mid x, a)\, J_{k-1}^\pi(y) \Big],$$

for all $k \in \{1, \dots, n\}$ and $x \in \mathbb{X}$. Note that $J_0^*(x) = 0$ and $J_0^\pi(x) = 0$. Moreover,

$$\forall x \in \mathbb{X} : J^\mu(x) = J^\mu_\infty(x) = \lim_{n \to \infty} J^\mu_n(x),$$

where μ is a hyper-parameter fixed to either $*$ or π. The application of μ allows us to treat both cases simultaneously. Also recall that the n-stage action-value function is defined as

$$Q^\mu_k(x,a) = g(x,a) + \alpha \sum_{y \in \mathbb{X}} p(y \,|\, x,a) J^\mu_{k-1}(y),$$

for all x, a and $k \in \{1, \dots, n\}$ and we also have $Q^\mu_0(x,a) = 0$. We introduce a new operator M, as well, that transforms action-value functions to state-value functions as follows

$$(MQ^\mu_k)(x) = \begin{cases} \sum\limits_{a \in \mathcal{A}(x)} \pi(x,a)\, Q^\mu_k(x,a) & \text{if } \mu = \pi \\[2ex] \min\limits_{a \in \mathcal{A}(x)} Q^\mu_k(x,a) & \text{if } \mu = * \end{cases}$$

During the proof we apply the solutions of suitable finite horizon problems, thus, in order to avoid notational confusions, let us denote the state and action value functions of $\mathcal{M}_2$ and $\mathcal{M}_3$ by J^μ, Q^μ and $\hat{J}^\mu$, $\widehat{Q}^\mu$, respectively. The corresponding finite horizon value functions are denoted by J^μ_n, Q^μ_n and $\hat{J}^\mu_n$, $\widehat{Q}^\mu_n$, respectively, where n is the length of the horizon. We will show that for all state x and action a we have $Q^\mu(x,a) = \widehat{Q}^\mu(x,a)$, from which $J^\mu = \hat{J}^\mu$ follows. Let us define the new immediate-cost function $\widehat{g}_n$ for all $n > 0$ by

$$\widehat{g}_n(x,a) = g(x,a) + (\alpha_2 - \alpha_1) \sum_{y \in \mathbb{X}} p(y \,|\, x,a) J^\mu_{n-1}(y).$$

We will apply induction on n. For the case of $n = 0$ we trivially have $Q^\mu_0 = \widehat{Q}^\mu_0$, since both of them are constant zero functions. Now, assume that $Q^\mu_k = \widehat{Q}^\mu_k$ for $k \leq n$, then

$$\widehat{Q}^\mu_{n+1}(x,a) = \widehat{g}_{n+1}(x,a) + \alpha_1 \sum_{y \in \mathbb{X}} p(y \,|\, x,a) \hat{J}^\mu_n(y) =$$

$$= g(x,a) + (\alpha_2 - \alpha_1) \sum_{y \in \mathbb{X}} p(y \,|\, x,a) J^\mu_n(y) + \alpha_1 \sum_{y \in \mathbb{X}} p(y \,|\, x,a) \hat{J}^\mu_n(y) =$$

$$= g(x,a) + \alpha_2 \sum_{y \in \mathbb{X}} p(y \,|\, x,a) J^\mu_n(y) + \alpha_1 \sum_{y \in \mathbb{X}} p(y \,|\, x,a) \left(\hat{J}^\mu_n(y) - J^\mu_n(y) \right) =$$

$$= g(x,a) + \alpha_2 \sum_{y \in \mathbb{X}} p(y \,|\, x,a) J^\mu_n(y) + \alpha_1 \sum_{y \in \mathbb{X}} p(y \,|\, x,a) \left((M\widehat{Q}^\mu_n)(y) - (MQ^\mu_n)(y) \right) =$$

$$= g(x,a) + \alpha_2 \sum_{y \in \mathbb{X}} p(y \,|\, x,a) J^\mu_n(y) = Q^\mu_{n+1}(x,a).$$

We have proved that for all n: $Q^\mu_n = \widehat{Q}^\mu_n$. Consequently, $Q^\mu(x,a) = \lim_{n \to \infty} Q^\mu_n(x,a) = \lim_{n \to \infty} \widehat{Q}^\mu_n(x,a) = \widehat{Q}^\mu(x,a)$ and, thus, $J^\mu(x) = \min_a Q^\mu(x,a) = \min_a \widehat{Q}^\mu(x,a) = \hat{J}^\mu(x)$. Finally, note that for the case of the infinite horizon problem $\widehat{g}(x,a) = \lim_{n \to \infty} \widehat{g}_n(x,a)$. ■

Theorem 26 *Suppose that Assumptions 1-3 hold and let V_t be the sequence generated by*

$$V_{t+1}(x) = (1 - \gamma_t(x))V_t(x) + \gamma_t(x)((K_t V_t)(x) + W_t(x)),$$

then, for any $V^*, V_0 \in \mathcal{V}$, the sequence V_t κ-approximates function V^* with

$$\kappa = \frac{4\varrho}{1-\beta_0} \qquad \textit{where} \qquad \varrho = \limsup_{t\to\infty} \|V_t^* - V^*\|_\infty.$$

The applied three main assumptions are as follows

Assumption 1 *There exits a constant $C > 0$ such that for all state x and time t, we have*

$$\mathbb{E}\left[W_t(x) \,|\, \mathcal{F}_t\right] = 0 \qquad \textit{and} \qquad \mathbb{E}\left[W_t^2(x) \,|\, \mathcal{F}_t\right] < C < \infty.$$

Assumption 2 *For all x and t, $0 \leq \gamma_t(x) \leq 1$, and we have with probability one*

$$\sum_{t=0}^{\infty} \gamma_t(x) = \infty \qquad \textit{and} \qquad \sum_{t=0}^{\infty} \gamma_t^2(x) < \infty.$$

Assumption 3 *For all t, operator $K_t : \mathcal{V} \to \mathcal{V}$ is a supremum norm contraction mapping with Lipschitz constant $\beta_t < 1$ and with fixed point V_t^*. Formally, for all $V_1, V_2 \in \mathcal{V}$,*

$$\|K_t V_1 - K_t V_2\|_\infty \leq \beta_t \|V_1 - V_2\|_\infty.$$

Let us introduce a common Lipschitz constant $\beta_0 = \limsup_{t\to\infty} \beta_t$, and assume that $\beta_0 < 1$.

Proof During the proof, our main aim will be to apply Theorem 21, thus, we have to show that the assumptions of the theorem hold. Let us define operator H_t for all $V_a, V_b \in \mathcal{V}$ by

$$H_t(V_a, V_b)(x) = (1 - \gamma_t(x))V_a(x) + \gamma_t(x)((K_t V_b)(x) + W_t(x)).$$

Applying this definition, first, we will show that $V'_{t+1} = H_t(V'_t, V^*)$ κ-approximates V^* for all V'_0. Because $\beta_t < 1$ for all t and $\limsup_{t\to\infty} \beta_t = \beta_0 < 1$, it follows that $\sup_t \beta_t = \widetilde{\beta} < 1$ and each K_t is $\widetilde{\beta}$ contraction. We know that $\limsup_{t\to\infty} \|V^* - V_t^*\|_\infty = \varrho$, therefore, for all $\delta > 0$, there is an index t_0 such that for all $t \geq t_0$, we have that $\|V^* - V_t^*\|_\infty \leq \varrho + \delta$. Using these observations, we can estimate $\|K_t V^*\|_\infty$ for all $t > t_0$, as follows

$$\|K_t V^*\|_\infty = \|K_t V^* - V^* + V^*\|_\infty \leq \|K_t V^* - V^*\|_\infty + \|V^*\|_\infty \leq$$
$$\leq \|K_t V^* - V_t^* + V_t^* - V^*\|_\infty + \|V^*\|_\infty \leq \|K_t V^* - V_t^*\|_\infty + \|V_t^* - V^*\|_\infty + \|V^*\|_\infty \leq$$
$$\leq \|K_t V^* - K_t V_t^*\|_\infty + \varrho + \delta + \|V^*\|_\infty \leq \widetilde{\beta}\,\|V^* - V_t^*\|_\infty + \varrho + \delta + \|V^*\|_\infty \leq$$
$$\leq (1 + \widetilde{\beta})\varrho + (1 + \widetilde{\beta})\delta + \|V^*\|_\infty \leq (1 + \widetilde{\beta})\varrho + 2\delta + \|V^*\|_\infty.$$

If we apply $\delta = (1 - \widetilde{\beta})\varrho/2$, then for sufficiently large t $(t \geq t_0)$ we have that

$$\|K_t V^*\|_\infty \leq 2\varrho + \|V^*\|_\infty.$$

Now, we can upper estimate $V'_{t+1} = H_t(V'_t, V^*)$, for all $x \in \mathcal{X}$, $V'_0 \in \mathcal{V}$ and $t \geq t_0$ by

$$V'_{t+1}(x) = H_t(V'_t, V^*)(x) = (1 - \gamma_t(x))V'_t(x) + \gamma_t(x)((K_t V^*)(x) + W_t(x)) \leq$$
$$\leq (1 - \gamma_t(x))V'_t(x) + \gamma_t(x)(\|K_t V^*\|_\infty + W_t(x)) \leq$$
$$\leq (1 - \gamma_t(x))V'_t(x) + \gamma_t(x)(\|V^*\|_\infty + 2\varrho + W_t(x))$$

Let us define a new sequence for all $x \in \mathcal{X}$ by

$$\widetilde{V}_{t+1}(x) = \begin{cases} (1 - \gamma_t(x))\widetilde{V}_t(x) + \gamma_t(x)(\|V^*\|_\infty + 2\varrho + W_t(x)) & \text{if } t \geq t_0 \\ \\ V'_t(x) & \text{if } t < t_0 \end{cases}$$

It is easy to see (for example, by induction from t_0) that for all time t and state x we have that $V'_t(x) \leq \widetilde{V}_t(x)$ with probability one, more precisely, for almost all $\omega \in \Omega$, where $\omega = \langle \omega_1, \omega_2, \dots \rangle$ drives the noise parameter $W_t(x) = w_t(x, \omega_t)$ in both V'_t and $\widetilde{V}_t$. Note that Ω is the sample space of the underlying probability measure space $\langle \Omega, \mathcal{F}, \mathbb{P} \rangle$.

Applying the "Conditional Averaging Lemma" of Szepesvári and Littman (1999), which is a variant of the Robbins-Monro Theorem and requires Assumptions 1 and 2, we get that $\widetilde{V}_t(x)$ converges with probability one to $2\varrho + \|V^*\|_\infty$ for all $\widetilde{V}_0 \in \mathcal{V}$ and $x \in \mathcal{X}$. Therefore, because $V'_t(x) \leq \widetilde{V}_t(x)$ for all x and t with probability one, we have that the sequence $V'_t(x)$ κ-approximates $V^*(x)$ with $\kappa = 2\varrho$ for all $V'_0 \in \mathcal{V}$ and $x \in \mathcal{X}$.

Now, let us turn to Conditions 1-4 of Theorem 21. For all x and t, we define functions $F_t(x)$ and $G_t(x)$ as $F_t(x) = \beta_t \gamma_t(x)$ and $G_t(x) = (1 - \gamma_t(x))$. By Assumption 2, functions $F_t(x), G_t(x) \in [0, 1]$ for all x and t. Condition 1 trivially follows from the definitions of G_t and H_t. For the proof of Condition 2, we need Assumption 3, namely that each operator K_t is a contraction with respect to β_t. Condition 3 is a consequence of Assumption 2 and the definition of G_t. Finally, we have Condition 4 for any $\varepsilon > 0$ and sufficiently large t by defining $\xi = \beta_0 + \varepsilon$. Applying Theorem 21 with $\kappa = 2\varrho$, we get that V_t κ'-approximates V^* with $\kappa' = 4\varrho/(1 - \beta_0 - \varepsilon)$. In the end, letting ε go to zero proves our statement. ■

Abbreviations

Acronym	**Meaning**
ADP	approximate dynamic programming
AI	artificial intelligence
APP	all policies (are) proper
AVI	approximate value iteration
CAS	complex adaptive system
CLP	container loading problem
CoG	center of gravity
CS	constraint satisfaction
DP	dynamic programming
ERM	empirical risk minimization
FJSP	flexible job-shop scheduling problem
FOMDP	fully observable Markov decision process
FPI	fitted policy iteration
FQL	fitted Q-learning
FVI	fitted value iteration
ICA	independent component analysis
IMS	intelligent manufacturing system
JSP	job-shop scheduling problem
KKT	Karush-Kuhn-Tucker
LP	linear programming
MAS	multi-agent system
MCMC	Markov chain Monte Carlo
MDP	Markov decision process
ML	machine learning
MLP	multi-layer perceptron
NOMDP	non-observable Markov decision process
PCA	principal component analysis
POMDP	partially observable Markov decision process
RAP	resource allocation problem
RBF	radial basis function
RCPSP	resource constrained project scheduling problem

Acronym	**Meaning**
RL	reinforcement learning
RTDP	real-time dynamic programming
SARSA	state-action-reward-state-action
SSP	stochastic shortest path
SVM	support vector machine
SVR	support vector regression
TD	temporal difference learning
TSP	traveling salesman problem
UMA	uniform memory access

Notations

General Notations

$\mathbb{N}$	set of natural numbers
$\mathbb{R}$	set of real numbers
$\mathbb{F}^{n\times m}$	matrix over field $\mathbb{F}$ with n rows and m columns
$\mathbb{P}(A)$	probability of event A
$\mathbb{E}(X)$	expected value of random variable X
$\mathbb{P}(A \mid B)$	conditional probability; the probability of A, given B
$\mathbb{E}(X \mid A)$	conditional expectation; the conditional expected value of random variable X given the occurrence of event A
$\Delta(S)$	set of all probability distributions over set S
$X \sim F$	random variable X is of distribution F
$\arg\min$	argument of the minimum
$\exp$	exponential function
$f(\cdot)$	function f takes one argument
$f\vert_S$	function f with domain set restriction to set S
$dom(\cdot)$	domain set (of a binary relation)
$rng(\cdot)$	range set (of a binary relation)
$\rightarrow$	(total) function
$\hookrightarrow$	partial function
$\times$	direct or Descartes product
$\langle \cdots \rangle$	tuple, ordered list
$dim(\cdot)$	dimension of a vector (or tuple)
$\vert\cdot\vert$	cardinality of a set
$\Vert\cdot\Vert_p$	p-norm (supremum norm, if $p = \infty$)
$\Vert\cdot\Vert_{\mathcal{X}}$	norm of normed space $\mathcal{X}$

Markov Decision Processes

$\mathbb{X}$	state space
$\mathbb{A}$	action space
$\mathbb{O}$	observation space
$\mathbb{B}$	belief space
$\mathcal{A}(\cdot)$	action-constraint function

$p(\cdot \mid \cdot, \cdot)$	transition-probability function
$q(\cdot \mid \cdot, \cdot)$	observation-probability function
$g(\cdot, \cdot)$	immediate-cost function
α	discount factor
$\tau(\cdot, \cdot)$	successor belief state function
$\pi(\cdot, \cdot)$	randomized control policy
$\pi(\cdot)$	deterministic control policy
$\widetilde{x}_0$	initial probability distribution
X_t	state of the system at time t
A_t^π	action taken at time t according to π
x_j^i	state at time t in episode i
N	length of the horizon
$J^\pi(\cdot)$	value function of policy π
$Q^\pi(\cdot, \cdot)$	action-value function of policy π
$J^*(\cdot)$	optimal value function
$Q^*(\cdot, \cdot)$	optimal action-value function
$\tilde{J}^\pi(\cdot)$	value function of policy π on belief states
$\tilde{J}^*(\cdot)$	optimal value function on belief states
$T^\pi(\cdot)$	Bellman operator for policy π
$T^*(\cdot)$	Bellman optimality operator
μ	hyper-parameter ("$*$" or "π")

Formal Logic

$\wedge$	logical "and"
$\vee$	logical "or"
$\neg$	logical "not"
$\Rightarrow$	(material) "implication"
$\forall$	universal quantifier
$\exists$	existential quantifier
$\exists!$	unique existence

Resource Allocation

$\mathcal{R}$	set of resources
$\mathcal{S}$	set of resource states
$\mathcal{O}$	set of operations
$\mathcal{T}$	set of tasks
$\mathcal{C}$	precedence constraints
$d(\cdot, \cdot)$	duration function
$D(\cdot, \cdot)$	stochastic duration function
$e(\cdot, \cdot)$	operation effect function

$E(\cdot,\cdot)$	stochastic effect function
$i(\cdot)$	initial state function
$I(\cdot)$	stochastic initial state function
$s(\cdot,\cdot)$	resource state function
$\varrho(\cdot,\cdot)$	resource allocator function
$\Phi(\cdot)$	slack ratio function
$\kappa(\cdot)$	performance measure
$\gamma(\cdot,\cdot)$	composition function

Stochastic Iterative Algorithms

$V_t(\cdot)$	generalized value function (at time t)
$\mathcal{V}$	set of generalized value functions
$\mathcal{F}_t$	history of the algorithm (until time t)
$K_t(\cdot)$	value function update operator
$H_t(\cdot,\cdot)$	generalized value iteration operator
$\gamma_t(\cdot)$	learning rate
$W_t(\cdot)$	noise parameter
τ_t	Boltzmann temperature

Support Vector Regression

l	sample size
$R\,[\cdot]$	risk function
$R^{\varepsilon}_{emp}[\cdot]$	empirical risk function
$l(\cdot,\cdot,\cdot)$	loss function
$\lvert\cdot\rvert_{\varepsilon}$	ε-insensitive loss function
ε	insensitivity parameter
ν	support vector control parameter
C	flatness trade-off constant
w_i	regression weight
b	bias parameter
$\phi_i(\cdot)$	basis function
α_i	Lagrange multiplier
ξ_i, ξ_i^*	slack variables
$K(\cdot,\cdot)$	kernel function
$\langle\cdot,\cdot\rangle$	inner product

Bibliography

Aberdeen, D. (2003). A (revised) survey of approximate methods for solving partially observable Markov decision processes. Technical report, National ICT Australia, Canberra.

Andrieu, C., Freitas, N. D., Doucet, A., and Jordan, M. I. (2003). An introduction to MCMC (Markov Chain Monte Carlo) for machine learning. *Machine Learning*, 50:5–43.

Åström, K. (1965). Optimal control of Markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications (JMAA)*, 10:174–205.

Aydin, M. E. and Öztemel, E. (2000). Dynamic job-shop scheduling using reinforcement learning agents. *Robotics and Autonomous Systems*, 33:169–178.

Baker, A. D. (1998). A survey of factory control algorithms that can be implemented in a multi-agent heterarchy: dispatching, scheduling, and pull. *Journal of Manufacturing Systems*, 17(4):297–320.

Beck, J. C. and Wilson, N. (2007). Proactive algorithms for job shop scheduling with probabilistic durations. *Journal of Artificial Intelligence Research*, 28:183–232.

Bertsekas, D. P. (2001). *Dynamic Programming and Optimal Control.* Athena Scientific, Belmont, Massachusetts, 2nd edition.

Bertsekas, D. P. (2005). Dynamic programming and suboptimal control: A survey from ADP to MPC. *European Journal of Control*, 11(4–5):310–334.

Bertsekas, D. P. and Tsitsiklis, J. N. (1996). *Neuro-Dynamic Programming.* Athena Scientific, Belmont, Massachusetts.

Bulitko, V. and Lee, G. (2006). Learning in real-time search: A unifying framework. *Journal of Artificial Intelligence Research*, 25:119–157.

Chang, C. C. and Lin, C. J. (2001). LIBSVM: A library for support vector machines. Software available on-line at http://www.csie.ntu.edu.tw/∼cjlin/libsvm.

Csáji, B. Cs., Kádár, B., and Monostori, L. (2003). Improving multi-agent based scheduling by neurodynamic programming. In *Proceedings of the 1st International Conference on Holonic and Mult-Agent Systems for Manufacturing (HoloMAS), September 1–3, Prague, Czech Republic*, volume 2744 of *Lecture Notes in Artificial Intelligence*, pages 110–123.

Csáji, B. Cs. and Monostori, L. (2005a). Stochastic approximate scheduling by neurodynamic learning. In *Proceedings of the 16th IFAC (International Federation of Automatic Control) World Congress, July 3–8, Prague, Czech Republic.*

Csáji, B. Cs. and Monostori, L. (2005b). Stochastic reactive production scheduling by multi-agent based asynchronous approximate dynamic programming. In *Proceedings of the 4th International Central and Eastern European Conference on Multi-Agent Systems (CEEMAS 2005)*, volume 3690 of *Lecture Notes in Artificial Intelligence*, pages 388–397.

Csáji, B. Cs. and Monostori, L. (2006a). Adaptive algorithms in distributed resource allocation. In *Proceedings of the 6th International Workshop on Emergent Synthesis (IWES-06), August 18–19, The University of Tokyo, Japan*, pages 69–75.

Csáji, B. Cs. and Monostori, L. (2006b). Adaptive sampling based large-scale stochastic resource control. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006), July 16–20, Boston, USA*, pages 815–820.

Csáji, B. Cs., Monostori, L., and Kádár, B. (2004). Learning and cooperation in a distributed market-based production control system. In *Proceedings of the 5th International Workshop on Emergent Synthesis (IWES 2004), May 24–25, Budapest, Hungary*, pages 109–116.

Csáji, B. Cs., Monostori, L., and Kádár, B. (2006). Reinforcement learning in a distributed market-based production control system. *Advanced Engineering Informatics*, 20:279–288.

Davies, A. and Bischoff, E. (1999). Weight distribution considerations in container loading. *European Journal of Operational Research*, 114:509–527.

Dolgov, D. A. and Durfee, E. H. (2006). Resource allocation among agents with MDP-induced preferences. *Journal of Artificial Intelligence Research*, 27:505–549.

Even-Dar, E. and Mansour, Y. (2003). Learning rates for Q-learning. *Journal of Machine Learning Research (JMLR)*, 5:1–25.

Favero, G. and Runggaldier, W. J. (2002). A robustness result for stochastic control. *Systems and Control Letters*, 46:91–66.

Feinberg, E. A. and Shwartz, A., editors (2002). *Handbook of Markov Decision Processes: Methods and Applications.* Kluwer Academic Publishers.

Gersmann, K. and Hammer, B. (2005). Improving iterative repair strategies for scheduling with the SVM. *Neurocomputing*, 63:271–292.

Gordienko, E. and Salem, F. S. (2000). Estimates of stability of Markov control processes with unbounded cost. *Kybernetika*, 36:195–210.

Hadeli, Valckenaers, P., Kollingbaum, M., and Brussel, H. V. (2004). Multi-agent coordination and control using stigmergy. *Computers in Industry*, 53:75–96.

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their application. *Biometrika*, 57:97–109.

Hatvany, J. and Nemes, L. (1978). Intelligent manufacturing systems - a tentative forecast. In Niemi, A., editor, *A link between science and applications of automatic control; Proceedings of the 7th IFAC World Congress*, volume 2, pages 895–899.

Hauskrecht, M. (2000). Value-function approximations for partially observable Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 13:33–91.

Holland, J. (1992). Complex adaptive systems. *Daedalus*, 121:17–30.

Holland, J. (1995). *Hidden Order: How Adaptation Builds Complexity.* Helix Books, Addison-Wesley, New York.

Hurink, E., Jurisch, B., and Thole, M. (1994). Tabu search for the job shop scheduling problem with multi-purpose machines. *Operations Research Spektrum*, 15:205–215.

Hyvärinen, A., Karhunen, J., and Oja, E. (2001). *Independent Component Analysis.* Wiley.

Kalmár, Zs., Szepesvári, Cs., and Lőrincz, A. (1998). Module-based reinforcement learning: Experiments with a real robot. *Machine Learning*, 31:55–85.

Kearns, M. and Singh, S. (2002). Near-optimal reinforcement learning in polynomial time. *Machine Learning*, 49:209–232.

Kennedy, J. and Eberhart, R. C. (1995). Particle swarm optimization. In *Proceeding of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220(4598):671–680.

Kovács, A. and Beck, J. C. (2007). A global constraint for total weighted completion time. In *Proceeding of the 4th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR)*, volume 4510 of *Lecture Notes in Computer Science*, pages 112–126.

Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., and Shmoys, D. B. (1993). Sequencing and scheduling: algorithms and complexity. *Handbooks in Operations Research and Management Science.*

Lovász, L. and Gács, P. (1999). *Complexity of Algorithms.* Lecture Notes, Boston University, Yale University.

Márkus, A., Kis, T., Váncza, J., and Monostori, L. (1996). A market approach to holonic manufacturing. *Annals of the CIRP – Manufacturing Technology*, 45(1):433–436.

Mastrolilli, M. and Gambardella, L. M. (2000). Effective neighborhood functions for the flexible job shop problem. *Journal of Scheduling*, 3(1):3–20.

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chem. Physics*, 21:1087–1092.

Modi, P. J., Hyuckchul, J., Tambe, M., Shen, W., and Kulkarni, S. (2001). Dynamic distributed resource allocation: Distributed constraint satis-faction approach. In *Pre-proceedings of the 8th In-ternational Workshop on Agent Theories, Archi-tectures, and Languages*, pages 181–193.

Monostori, L. (2003). AI and machine learning techniques for managing complexity, changes and uncertainties in manufacturing. *Engineering Applications of Artificial Intelligence*, 16(4):277–291.

Monostori, L. and Csáji, B. Cs. (2006). Stochastic dynamic production control by neurodynamic programming. *Annals of the CIRP – Manufacturing Technology*, 55(1):473–478.

Monostori, L., Csáji, B. Cs., and Kádár, B. (2004). Adaptation and learning in distributed production control. *Annals of the CIRP – Manufacturing Technology*, 53(1):349–352.

Monostori, L., Kis, T., Kádár, B., Váncza, J., and Erdős, G. (2008). Real-time cooperative enterprises for mass-customized production. *International Journal of Computer Integrated Manufacturing*, pages (to appear).

Monostori, L., Váncza, J., and Kumara, S. R. T. (2006). Agent-based systems for manufacturing. *Annals of the CIRP – Manufacturing Technology*, 55(2):697–720.

Montes de Oca, R., Sakhanenko, A., and Salem, F. (2003). Estimates for perturbations of general discounted Markov control chains. *Applied Mathematics*, 30:287–304.

Moyson, F. and Manderick, B. (1988). The collective behaviour of ants : an example of self-organization in massive parallelism. In *Proceedings of AAAI Spring Symposium on Parallel Models of Intelligence, Stanford, California.*

Müller, A. (1996). How does the solution of a Markov decision process depend on the transition probabilities? Technical report, Institute for Economic Theory and Operations Research, University of Karlsruhe.

Neumann, J. (1948). *Papers of John von Neumann on Computing and Computer Theory (1987)*, chapter The general and logical theory of automata, pages 391–431. The MIT Press, Cambridge. (Originally presented at the "Hixon Symposium" on September 20, 1948, at the California Institute of Technology).

Papadimitriou, C. H. (1994). *Computational Complexity.* Addison-Wesley.

Perkins, J. R., Humes, C., and Kumar, P. R. (1994). Distributed scheduling of flexible manufacturing systems: Stability and performance. *IEEE Transactions on Robotics and Automation*, 10:133–141.

Pinedo, M. (2002). *Scheduling: Theory, Algorithms, and Systems.* Prentice-Hall.

Powell, W. B. and Van Roy, B. (2004). *Handbook of Learning and Approximate Dynamic Programming*, chapter Approximate Dynamic Programming for High-Dimensional Resource Allocation Problems, pages 261–283. IEEE Press, Wiley-Interscience.

Riedmiller, S. and Riedmiller, M. (1999). A neural reinforcement learning approach to learn local dispatching policies in production scheduling. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI), Stockholm, Sweden*, pages 764Ű–771.

Schneider, J. G., Boyan, J. A., and Moore, A. W. (1998). Value function based production scheduling. In *Proceedings of the 15th International Conference on Machine Learning*, pages 522–530. Morgan Kaufmann, San Francisco, California.

Schölkopf, B., Smola, A., Williamson, R. C., and Bartlett, P. L. (2000). New support vector algorithms. *Neural Computation*, 12:1207–1245.

Schuh, G., Monostori, L., Csáji, B. Cs., and Döring, S. (2008). Complexity-based modeling of reconfigurable collaborations in production industry. *Annals of the CIRP – Manufacturing Technology*, 57(1):(to appear).

Singh, S. and Bertsekas, D. (1997). Reinforcement learning for dynamic channel allocation in cellular telephone systems. In *Advances in Neural Information Processing Systems*, volume 9, pages 974–980. The MIT Press.

Singh, S., Jaakkola, T., Littman, M., and Szepesvári, Cs. (2000). Convergence results for single-step on-policy reinforcement-learning algorithms. *Machine Learning*, 38:287–308.

Smallwood, R. D. and Sondik, E. J. (1973). The optimal control of partially observable Markov decision processes over a finite horizon. *Operations Research*, 21:1071–1088.

Sontag, E. D. (1998). *Mathematical Control Theory: Deterministic Finite Dimensional Systems*. Springer, New York.

Sutton, R. S. and Barto, A. G. (1998). *Reinforcement learning*. The MIT Press.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (2000). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, 12:1057–1063.

Szepesvári, Cs. and Littman, M. L. (1999). A unified analysis of value-function-based reinforcement learning algorithms. *Neural Computation*, 11(8):2017–2060.

Szepesvári, Cs. and Munos, R. (2005). Finite time bounds for sampling based fitted value iteration. In *22nd International Conference on Machine Learning*, pages 881–886.

Szita, I., Takács, B., and Lőrincz, A. (2002). ε-MDPs: Learning in varying environments. *Journal of Machine Learning Research (JMLR)*, 3:145–174.

Topaloglu, H. and Powell, W. B. (2005). A distributed decision-making structure for dynamic resource allocation using nonlinear function approximators. *Operations Research*, 53(2):281–297.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 59(236):4–30.

Ueda, K., Márkus, A., Monostori, L., Kals, H. J. J., and Arai, T. (2001). Emergent synthesis methodologies for manufacturing. *Annals of the CIRP – Manufacturing Technology*, 50(2):535–551.

Van Brussel, H., Wyns, J., Valckenaers, P., Bongaerts, L., and Peeters, P. (1998). Reference architecture for holonic manufacturing systems: PROSA. *Computers in Industry*, 37(3):255–274.

Van Roy, B., Bertsekas, D., Lee, Y., and Tsitsiklis, J. (1996). A neuro-dynamic programming approach to retailer inventory management. Technical report, Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards*. PhD thesis, King's College, Cambridge University, United Kingdom.

Watkins, C. J. C. H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3):279–292.

White, C. (1976). Application of two inequality results for concave functions to a stochastic optimization problem. *Journal of Mathematical Analysis and Applications*, 55:347–350.

Wu, T., Ye, N., and Zhang, D. (2005). Comparison of distributed methods for resource allocation. *International Journal of Production Research*, 43:515–536.

Zhang, W. and Dietterich, T. (1995). A reinforcement learning approach to job-shop scheduling. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1114–1120. Morgan Kauffman.