Learning and Optimizing Probabilistic Models for Planning Under Uncertainty

by

R. van Bekkum

to obtain the degree of Master of Science
in Computer Science
at the Delft University of Technology,
to be defended publicly on Wednesday September 27, 2017 at 14:00.

Student number: 4210816
Thesis committee: Dr. M. T. J. Spaan, TU Delft, supervisor
Dr. Ing. J. Kober, TU Delft
Dr. M. Loog, TU Delft

An electronic version of this thesis is available at http://repository.tudelft.nl/.

Algorithmics Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands

Abstract

Decision-theoretic planning techniques are increasingly being used to obtain (optimal) plans for domains involving uncertainty, which may be present in the form of the controlling agent's actions, its percepts, or exogenous factors in the domain. These techniques build on detailed probabilistic models of the underlying system, for which Markov Decision Processes (MDPs) have become the *de facto* standard formalism. However, handcrafting these probabilistic models is usually a daunting and error-prone task, requiring expert knowledge on the domain under consideration. Therefore, it is desirable to automate the process of obtaining these models by means of learning algorithms presented with a set of execution traces from the system. Although some work has already been done on crafting such learning algorithms, the state of the art lacks an automated method of configuring their hyperparameters, so to maximize the performance yielded from executing the derived plans. In this work we present a method that employs the Bayesian Optimization (BO) framework to learn MDPs autonomously from a set of execution traces, optimizing the expected value and performance in simulations over a set of tasks the underlying system is expected to perform. The approach has been tested on learning MDPs for mobile robot navigation, motivated by the significant uncertainty accompanying the robots' actions in this domain.

Preface

In the last year of my master studies, I have had the opportunity to explore and gain insight into fields of research including planning, reinforcement learning and optimization that I was not familiar with at the start of this project. This report documents the work on identifying the possibilities of automating the task of learning performance-maximizing Markov Decision Processes from data effectively, performed as a master thesis project in the Algorithmics Group of the Department of Software Technology of the EEMCS Faculty at the Delft University of Technology.

I would like to thank Bruno Lacerda, University of Birmingham, for the tips I got at the start of the project on the usage of the robot simulation software. Especially, I would like to thank Matthijs Spaan for being my supervisor during this project in which the weekly meetings in which we could exchange ideas were very helpful.

R. van Bekkum
Delft, September 2017

Contents

List of Figures

1 Introduction

In numerous real-world settings, such as traffic light control [80], mobile robot navigation [49] or decision making in medical treatment [67], it is quite common to be faced with uncertainty about the effect of an action which may have been performed on the basis of incomplete or faulty observations. Therefore, when plans are provided to impose a course of action for such settings, they need to be more sophisticated, anticipating for the observations that might be made and being able to act accordingly. That is, a proper trade-off should be made between the different possible outcomes and the costs of executing a plan.

A general approach for *planning under uncertainty* is to model the dynamics of the system under consideration, defining transition probabilities for moving between attainable states through the selection of actions. The resulting *probabilistic models* are used in offline planning to compute policies that can be leveraged to automate the sequential decision making in the underlying system. In this thesis our main objective is to automate the development of probabilistic models which maximize the performance of underlying system's in the execution of their tasks. In this chapter we present our motivations for investigating techniques for developing probabilistic models and the overall problem it poses. Accordingly, we identify and rationalize the research questions that we aim to answer in this thesis. Further, we summarize our main contributions, define the scope and limitations of our research, and finally present an outline for the remainder of this thesis.

1.1. Motivation

There exist several practical applications in which the (sequential) actions of systems are coordinated by decision makers or *agents* to achieve long-term goals. In particular these agents need to take into account the uncertainty in these systems that may be present in the form of action failures, exogenous events and noisy observations. To devise optimal plans for those systems whose dynamics are stochastic, *Decision-Theoretic Planning (DTP)* aims to account for uncertainty by exploiting the considerable structure these systems pose through the development of probabilistic models which reflect this uncertainty. These probabilistic models serve as a system representation which describe a system's state and its evolution over time after a sequence of actions has been executed. The advantage of having such (accurate) probabilistic models at one's disposal is that agents can act according to different policies derived from one and the same model to perform multiple tasks.

In recent years, particularly *Markov Decision Processes (MDPs)* have become a significant popular formalism for modeling DTP problems [8]. That is, first of all, due to their firm foundation in decision theory and successes of Markovian approaches in speech recognition [3, 27, 64] and the closely-related field of Reinforcement Learning (RL) [9, 41]. Furthermore, over the years various computationally efficient solution techniques have been devised for obtaining optimal plans for MDP models which maximize expected value (e.g., [40, 63]). Considering the expediency of MDPs, it seems worthwhile to investigate methods of developing accurate probabilistic models for the purpose of planning under uncertainty.

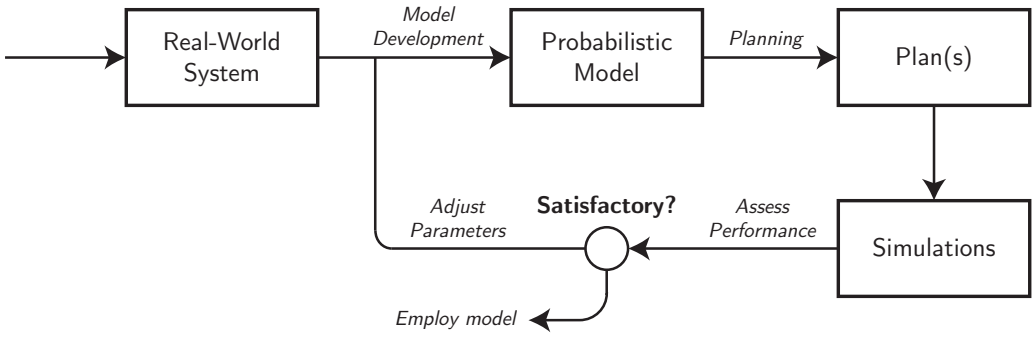


Figure 1.1: Diagram showing a typical iterative procedure followed in the development of probabilistic models for planning under uncertainty. Given a problem description for the real-world system under consideration, a designer refines the model until satisfactory performance is observed.

1.2. Problem Statement

The problem this thesis is concerned with is the development of probabilistic models, which are used for obtaining plans for automated control of systems whose dynamics are stochastic. Setting up such models is an important, though difficult and time-costly, process for a human designer that might not always have (enough) expertise to craft a suitable, well-generalizing model that can be used to plan for the execution of different tasks. In practice, this problem is tackled by domain experts by iteratively tweaking the model parameters until the desired performance is achieved.

Figure 1.1 shows a typical procedure followed in the design of a probabilistic model. In this procedure, the human designer is presented a problem description and handcrafts an initial model for the system. The model is then evaluated by performing tasks following the plans that are derived from the model in simulations or even in the real world. The assessed performance steers the designer towards what model parameters appear to work well. The designer uses the obtained knowledge to iteratively refine the model parameters until ultimately satisfactory performance is yielded in the execution of the derived plans.

An alternative that one ought to consider, is that of applying RL techniques rather than planning algorithms. However, although ideas from planning and RL are interchangeable, these techniques typically demand direct interaction with the environment which is something that cannot always be readily offered. That is, RL techniques turn out as time-consuming and sometimes even riskful or dangerous when applied in real-world environments. However, one should note that at one point potential risks should be accounted for when the system is employed in the real world, and that algorithms, such as risk-sensitive RL, do exist for this purpose [28, 52]. Those domains for which plans need to be formulated offline, demand the acquisition of probabilistic models which accurately define what constitutes the state of the system, reflect uncertainty through transition probabilities and specify its goals.

An attractive approach of bypassing the daunting and error-prone task of handcrafting probabilistic models, is that of automating the model development process by applying learning algorithms on data that describes the dynamics of the system. For this approach the data is typically presented in the form of execution traces of the system operating in a real-world environment, which is collected in an exploration phase prior to the actual planning. However, although one could obtain a model of the system by applying learning algorithms, the corresponding plans that are inferred from this model might not be effective when applied in a real-world environment. First of all, this might be due to the parameters of the learning algorithm not being set properly, signifying the need for proper adjustment of these parameters. Another possibility is that the gathered data is incomplete and so does not accurately describe the dynamics of the system. In this case, one could choose to augment the data for those areas for which the training data is inadequate, although one should be aware that this is accompanied by a more cost-expensive model learning process. Therefore, in order to learn accurate probabilistic models, we are in need of a way of assessing the performance of a model accompanied by a method for identifying inadequacies due to incomplete data, while taking into account the corresponding cost of learning and evaluating these system models.

1.3. Research Questions

The problem statement as presented in the previous section yields the following main research question for this thesis:

Main Research Question. How can the task of obtaining a (discrete) MDP that maximizes the yielded performance of executing plans that are derived from it, given a dataset about the system under consideration, be automated?

To answer this main research question, four research questions have been identified which are presented below. The relevance of each of these research questions is discussed accordingly in the next paragraphs.

Research Question 1. Which learning algorithms exist that can be employed for learning MDPs from data for systems involving uncertainty that require plans for automated control?

First off, to facilitate the formulation of plans for a system involving uncertainty, we need to obtain an MDP from the provided dataset. Therefore, we should know what learning algorithms exist that can be employed to learn the parameters of an MDP. Accordingly, we should identify the applications for which each of these algorithms are suited, but also what the shortcomings or vulnerabilities of each of these algorithms are.

Research Question 2. How should a performance measure be defined which can be used to fairly compare the value of different MDPs?

As various MDPs can be obtained for different parameter settings of the model learning algorithms, the need for a measure of performance for different MDPs emerges. That is, to establish which parameter settings yield an MDP that best reflects the underlying system for the tasks to be executed, we require a way to express and fairly compare the value of the learned MDPs.

Research Question 3. How can the parameter space of model learning algorithms cost-effectively be explored towards a global maximizer with only limited knowledge about the system under consideration?

A model learning algorithm may yield different MDPs depending on the selected parameter settings. To establish the most appropriate MDP for the system under consideration, the performance yielded by multiple MDPs should be compared. As evaluating all possible parameter settings would be cost-expensive, we need to investigate other ways of exploring the parameter space more cost-effectively. This is under the assumption that we only have limited to no belief over which parameter settings might work well for the system under consideration.

Research Question 4. How can the hierarchy of different abstraction levels be exploited to find a performance-maximizing MDP in a more cost-effective way?

Assessments of performance can be made at different levels of abstraction of the underlying system. That is, we could examine how well an agent would perform a task only from the perspective of the MDP model, but also from simulations or even the real world. On the one hand, abstracting from the real world typically yields a less accurate representation of the reality, while on the other hand, assessing the performance from a more abstract level is accompanied by smaller computational costs. Therefore, it might be valuable to investigate how this could be exploited to achieve a more cost-effective optimization of the performance.

1.4. Contributions

We propose a framework for automating the development of probabilistic models in the form of MDPs by applying learning algorithms on data describing the dynamics of the system under consideration. The data about the environment that is used by these learning algorithms is gathered prior to the model learning process in an exploration phase. Although various algorithms for learning MDPs offline from a dataset [55, 69, 79] do already exist, the state of the art lacks a method for setting the hyperparameters of these algorithms so to best reflect the underlying system and maximize the performance in the execution of its tasks.

To address this problem, the proposed framework poses the adjustment of the learning algorithm parameters as an optimization task of maximizing the performance that follows from executing plans derived from learned MDPs. The optimization is performed by applying a technique known as *Bayesian Optimization (BO)*, such that we define a probability distribution over functions to model the performance measure and iteratively sample parameter settings towards a global maximizer of the performance. Although algorithms that employ the same framework for optimizing the parameters of probabilistic models [30, 34] or policy search [21, 81] do exist, these are online approaches that do not utilize the available data prior to interacting with the real world environment.

To achieve a more cost-effective optimization, the parameter search space is first narrowed down by a 'pre-processing' phase. In this phase assessments of the model value are made on a more abstract level (i.e., based on the value functions computed from an MDP). The posterior that follows is then used to steer the acquisition in the main optimization phase.

In attempt to further improve the learned models, a 'post-processing' step is performed that aims to identify and fix discrepancies between the model and the real world. This post-processing step increases the resolution for those areas of the state space for which it identifies the outcomes of actions in simulations do not match the learned transition probabilities.

The framework is applied, tested and evaluated in the domain of mobile robot navigation, as the robots in this domain tend to operate under significant uncertainty in their actions. A solution in this context is a policy which maps discretized robot poses into fine-grained navigation actions so to move a mobile robot to a certain goal location. Probabilistic models are acquired by applying unsupervised machine learning algorithms on execution traces (consisting of odometry data describing robot poses) obtained in an exploration phase. The optimization is based on the performance under acquired models in simulations, expressed in terms of the number of discrete time-steps needed to reach multiple goal locations.

1.5. Scope and Limitations

First of all, we restrict ourselves to learning fully observable discrete MDPs from data. However, our framework could possibly be extended to deal with applications where the states cannot be directly observed and should be inferred from observations. That is, the learning algorithm forms an interchangeable component in the framework, which means it is possible to instead employ an algorithm for learning these partially observable models. Though for now, our endeavor is to give a clear view of our methodology for learning MDPs which can, as such, serve as a foundation for future work that employs more complex learning algorithms (e.g., [44, 54, 55, 69]) for partially observable models.

Secondly, we note that all experiments and corresponding results are based on data that is solely obtained from simulations of a mobile robot. Reasons for this are that we make the assumption that the environment is fully observable, which is typically not the case for real-world applications. Another motivation of this is that the framework is not solely intended for the domain of mobile robot navigation and data from simulations is therefore deemed adequate for our experiments.

Finally, we focus on the domain of mobile robot navigation to evaluate and test our approach. One of the important aspects to consider, is which data should be collected to describe the dynamics of the system to learn models from this data. For our implementation we choose to collect data about the robot's poses from internal odometry, but for other applications we might not be able to describe the states and transitions of the system through a geometric model and may need other learning algorithms than the clustering algorithms

we use for our application. An example of such an application may be that of learning MDPs for traffic light control [23, 80], in which, for instance, lane-turn probabilities are learned from a dataset. Note that this application seems particularly appropriate to be approached in combination with RL techniques, so that the system is able to adapt to changes that occur over time.

1.6. Outline

In this chapter we presented our motivations and defined the problem this thesis is concerned with. To approach this problem we propose a framework that is aimed at automating the development of probabilistic models for the purpose of automated control by agents through planning under uncertainty. This section describes the topics of the remaining chapters and how they relate to our problem statement and research questions presented earlier.

In Chapter 2 we discuss the required background for both the planning and optimization aspect of this thesis. First of all, the chapter concretizes the type of problems dealt with in DTP and how probabilistic models are employed in this field to construct plans for these problems. Secondly, the chapter discusses the BO framework which is used optimize cost-expensive functions by sampling new observations effectively based on earlier observations.

In Chapter 3 we explore other existing techniques for the development of MDP models for systems involving uncertainty. Apart from this, we explore methods that takes advantage of techniques from the closely-related field of RL to approach SDM problems where possible. Further, the chapter reviews techniques that overcome the infeasibility of accurately defining transition probabilities by accounting for uncertainty in these probabilities.

A solution to the problem we described, is proposed in Chapter 4, in which we put the theory and algorithms discussed in the earlier chapters together into an optimization framework. This chapter presents a *base framework* and a *multi-phase* extension of this framework, which optimize for a performance-maximizing MDP model to be used for offline planning. The proposed solution is exemplified by a running example problem of path planning for a mobile robot in an uncertain environment.

To test and evaluate our solution, an implementation has been developed for the optimization of probabilistic models for the path planning of a mobile robot in office environments. In Chapter 5 we present the performed experiments and discuss the results obtained from testing the proposed solution with this implementation of the framework.

Finally, in Chapter 6 we summarize and evaluate the proposed solution. In this chapter we revisit our research questions and conclude this thesis with our recommendations and suggestions for future work.

2

Background

In this chapter the background on both the planning and optimization aspect of this thesis is presented. The concepts and techniques discussed in this chapter are employed in the framework we propose in Chapter 4 with the aim of optimizing for a probabilistic model that can be used to obtain plans which maximize the performance of a system that is in need of automated control.

First of all, in Section 2.1, the chapter discusses how Decision-Theoretic Planning (DTP) deals with the central problem of planning under uncertainty using probabilistic models of a system under consideration. In this thesis we put our attention to modeling DTP problems as Markov Decision Processes (MDPs) and accordingly discuss the existing algorithmic techniques for obtaining plans for automated Sequential Decision Making (SDM). We employ these techniques in the framework presented in Chapter 4 to compute optimal plans from the MDP models learned from data, and accordingly assess the value of these models.

Then, in Section 2.2, the chapter elaborates upon a method known as Bayesian Optimization (BO), which is used to automate the process of optimizing the parameters of an unknown objective while minimizing the number of function evaluations. This method is particularly suited for the optimization task dealt with in the framework proposed in Chapter 4, where the parameters of model learning algorithms are adjusted so to yield maximum performance in a system's execution. That is, in our framework we assess the value of different MDP models by performing expensive simulations, which is why we would like to minimize the number of these simulations. Therefore, in this section we describe how the BO framework achieves this and present some of its applications to related SDM problems.

2.1. Decision-Theoretic Planning

Automated Sequential Decision Making (SDM) comprises the central problem of planning under uncertainty. Decision-Theoretic Planning (DTP) is concerned with the design of plans or *policies* for settings in which uncertainty exists about the effects of actions, where the decision maker or *agent* has incomplete information about the environment and its initial conditions, and where trade-offs need to be made between potentially conflicting objectives to determine an optimal course of action. This section gives an introduction to the type of problems faced in DTP and explains how specialized probabilistic models can be used to solve these problems efficiently. First of all, in Section 2.1.1 the goal of DTP and how the problems that are considered are generally approached are discussed. Subsequently, Section 2.1.2 gives an overview of the probabilistic models that are used to make the structure of problems in DTP explicit. Then, Section 2.1.3 discusses the algorithms that enable us to learn these models from a dataset describing the dynamics of the system under consideration. Finally, in Section 2.1.4 some of the most common algorithmic planning techniques are discussed, which either learn a plan directly or through interaction.

2.1.1. General Problem Formulation

The class of problems that are considered in Decision-Theoretic Planning are those that require optimal stochastic control through the actions of decision maker(s), referred to as *agent(s)*, in systems whose dynamics can be modeled as *stochastic processes* [8]. The agent(s) in these systems sequentially need to choose from a set of actions that influence the system's behavior, consequently making the system switch from one state to another. In these settings the system's current state and the agent's choice of action determines the probability distribution over the states the system might reach next. In addition, the agent(s) might be uncertain about the system's current state, implying the need to infer from observations and making decisions based on probabilistic estimates of the system's state.

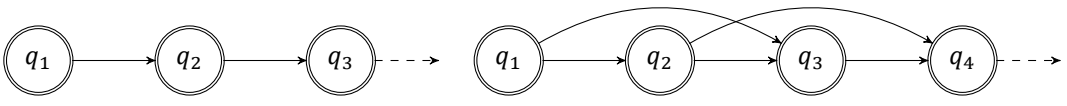
Typically the problems under consideration involve certain objectives to be achieved (e.g., tasks to be fulfilled) or properties to be satisfied (e.g., avoiding certain system states). Therefore, the agent should decide on an optimal plan or *policy* which makes it most likely for the system to reach its targets, while minimizing the risk of producing undesirable states and the accompanied costs of the policy. To find such a policy for Sequential Decision Making problems, a typical approach is to first setup a probabilistic model of the system and then apply a DTP algorithm on this model. This probabilistic model comprises a system representation which defines the state space in terms of a set of multi-valued features, the set of actions the agent may select together with the associated uncertainty defined by transition probabilities, and a goal specification or performance metric typically expressed by means of a reward structure.

Overall DTP aims to devise planning algorithms for planning under uncertainty, a problem that is addressed in numerous different fields of research such as AI planning and control theory. In particular difficulties arise when planning techniques are applied to determine courses of action for real-world settings, such as motion or path planning in robotics which both involve the possibility of action failures and disturbances caused by exogenous events.

2.1.2. System Representations: Markov Models

As the class of problems considered in DTP tend to present considerable structure, there exist various proposed solutions for planning under uncertainty that apply model-based approaches. This type of decision-theoretic planner uses a stochastic model of the environment in which the agent operates, which compasses the uncertainty that is associated with the agent's actions, observations and the exogenous events that might occur. Typically the uncertainty is modeled by establishing a *state space* for the system accompanied by a set of possible *transitions* between the states that might be induced with a certain probability by an agent executing *actions*. The most common types of stochastic models that are used in DTP are called *Markov Models* (sometimes also referred to as *Markovian Models*), which has been motivated by their success in other fields such as speech recognition [3, 27, 64] and Reinforcement Learning (RL) techniques [9, 41]. A Markov Model is a stochastic model in which the future states only depend on a limited number of prior observations. In fact, mostly processes or systems are modeled by Markov Models that satisfy the *Markov Property*, which means that the state transitions are independent of any previous states or agent actions.

In the remainder of this section the most common types of discrete-state Markov Models are discussed, starting from the fundamental models known as *Markov Chains* and their extension known as *Hidden Markov Models (HMMs)*. These fundamental models however only describe the evolution of a system and do not allow for stochastic control. Therefore, this is followed by a discussion of the discrete-state Markov Models known as *Markov Decision Processes (MDPs)* and their extension known as *Partially Observable MDPs (POMDPs)*, which add the possibility of influencing the behavior of the system through sequential decisions.



(a) First-order Markov Chain. (b) Second-order Markov Chain.

Figure 2.1: Graphical representation of Markov Chains of different order.

Markov Chains

The evolution of system or processes can be viewed as so-called *time-series*, in which a set of datapoints can be ordered using an underlying physical dimension, typically time [4]. Formally, time-series can be defined as a series $x_{a:b}$ of datapoints, with $x_{a:b} \equiv x_a, x_{a+1}, \dots, x_b$. By means of these time-series, probabilistic models can be devised for real-world systems or processes, by introducing the notion of *states* as a description of the system at a particular point in time or *stage*.

On the basis of the class of Markov Models lies the simplifying assumption that each state is only dependent on a limited number of previous states. Under this assumption a *Markov Chain* (or *Markov Process*) can be defined as a model of a series $q_{1:T}$ of transitions between states q_i drawn from a state space $\mathcal{S} = \{s_1, \dots, s_n\}$. The initial state q_1 of a Markov Chain typically is either fixed or drawn from $\mathcal{S}$ using a probability distribution over initial states. For the states or variables of a Markov Chain, the earlier mentioned assumption implies the following conditional independence to hold:

$$p(q_t|q_1, \dots, q_{t-1}) = p(q_t|q_{t-L}, \dots, q_{t-1}) \tag{2.1}$$

where L is the so-called *order* of the Markov Chain.

As depicted in Figure 2.1, Markov Chains of different order can be defined. Figure 2.1a exemplifies a first-order Markov Chain in which each state only depends on the previous state, and Figure 2.1b shows a second-order Markov Chain in which each state depends on the two prior states of the Markov Chain. In the special case where the transition distribution is independent of the stage of the system, but solely on the prior state(s), one speaks of a *stationary* or *homogeneous* Markov Chain.

Though, mostly first-order Markov Chains, as depicted in Figure 2.1a, which are said to satisfy the *Markov Property*, are widely applied for modeling stochastic processes, such as physical phenomena and economic time-series [2]. In addition, mostly compact, stationary, discrete-time, finite-space Markov Chains are used, bearing in mind the computational adequacy of the model (i.e., the larger the state space and order, the more computational cost might be incurred). Some concrete examples of practical applications include assessing the reliability and/or safety of appliances in engineering [17, 20, 25], modeling water flows [57], or modeling loan defaults [32] in the financial world (for an overview see [58]).

In these chains the state transition probabilities can be stored in an $n \times n$ transition matrix $\boldsymbol{A} = [a_{ij}]$ with each entry

$$a_{ij} = p(q_{t+1} = s_i|q_t = s_j) \tag{2.2}$$

denoting the probability of state s_i following state s_j. Similarly, the initial state probabilities can be recorded in an $n \times 1$ initial state vector $\boldsymbol{\pi} = [\pi_i]$ with each entry

$$\pi_i = p(q_1 = s_i) \tag{2.3}$$

denoting the probability of state s_i being the initial state of the model. Putting these components together yields Definition 1 below, which formally defines a (discrete-time) stationary first-order Markov Chain.

Definition 1. A stationary first-order *Markov Chain* is a 3-tuple $(\mathcal{S}, \boldsymbol{A}, \boldsymbol{\pi})$ where $\mathcal{S} = \{s_1, \dots, s_n\}$ is a finite set of states, $\boldsymbol{A}$ a transition (probability) matrix and $\boldsymbol{\pi}$ an initial state (probability) vector.

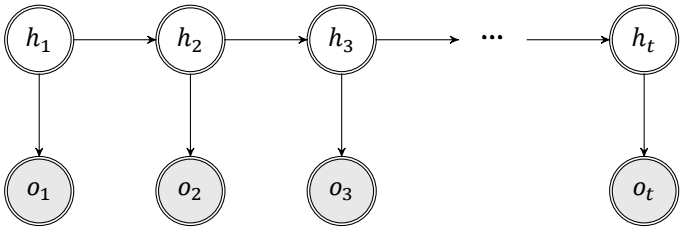


Figure 2.2: Graphical representation of an HMM with hidden states $h_t \in \mathcal{S}$ and observations $o_t \in \mathcal{V}$ for $t = 1 : T$.

Hidden Markov Models

The Markov Chain model discussed in Section 2.1.2 requires the modeled system or stochastic process to be fully observable, meaning that each of its states correspond directly to an observation. However, it is not unusual to encounter real-world systems in which the states are assumed to be unobservable, though are correlated with observable events produced by the system. These systems can be modeled by a Hidden Markov Model (HMM) in which the states, typically referred to as *hidden* or *latent* variables $h_{1:T}$, are unknown. Additionally this model consists of observations, typically referred to as *observed* or *visible* variables $v_{1:T}$, which are dependent on the hidden variables through an emission $p(o_t|h_t)$ graphically depicted in Figure 2.2. From all of this follows Definition 2 below, formally describing a stationary HMM.

Definition 2. A (discrete-state) stationary *Hidden Markov Model (HMM)* is a 5-tuple $(\mathcal{S}, \mathcal{V}, \boldsymbol{\pi}, \boldsymbol{A}, \boldsymbol{B})$ consisting of the following parts:

- a discrete set of n attainable states $\mathcal{S} = \{s_1, \dots, s_n\}$, i.e. the underlying *state space*
- a set of m possible observations, $\mathcal{V} = \{v_1, \dots, v_m\}$, i.e. the *observation space*
- an $n \times n$ transition matrix $\boldsymbol{A} = [a_{ij}]$ defining the models' *transition distribution* in which $a_{ij} = p(h_{t+1} = s_i | h_t = s_j)$ is the probability of state s_i following state s_j $(s_i, s_j \in \mathcal{S})$
- an $m \times n$ emission matrix $\boldsymbol{B} = [b_{ij}]$ defining the models' *emission distribution* in which $b_{ij} = p(o_t = v_i | h_t = s_j)$ the probability of observing v_i from state s_j $(v_i \in \mathcal{V}, s_j \in \mathcal{S})$
- an $n \times 1$ initial state array $\boldsymbol{\pi} = [\pi_i]$ in which $\pi_i = p(h_1 = s_i)$ is the probability of having s_i as the initial state $(s_i \in \mathcal{S})$

In practice HMMs have a wide range of applications. One example is that of object tracking in which inference algorithms for HMMs are used to estimate the (unknown) position of objects by a sequence of observations [12]. Another well-known application example of HMMs is that in automatic speech recognition [64].

Markov Decision Processes

Although Markov Chains and HMMs can be used to model the evolution of stochastic processes or systems, they do not allow for stochastic control through the actions of a decision maker or *agent* which alter the state of the system. The systems of interest in DTP however, involve agents that are assigned the task of influencing the behavior of the stochastic system by making sequential decisions to achieve certain goals.

Markov Decision Processes (MDPs) extend on (stationary) Markov Chains by adding a finite set of actions A available to the agent at each stage or *decision epoch*. Upon the agent choosing to perform an action $a \in A$, a state transition occurs in response to the action. However, due to the uncertainty in the system, the actual transition that occurs might differ from the transition intended by the chosen action. This uncertainty is captured by defining a probabilistic transition function $\delta : \mathcal{S} \times A \times \mathcal{S} \mapsto [0, 1]$ which maps the combination of a current state and action to a probability of ending up in a certain next state.

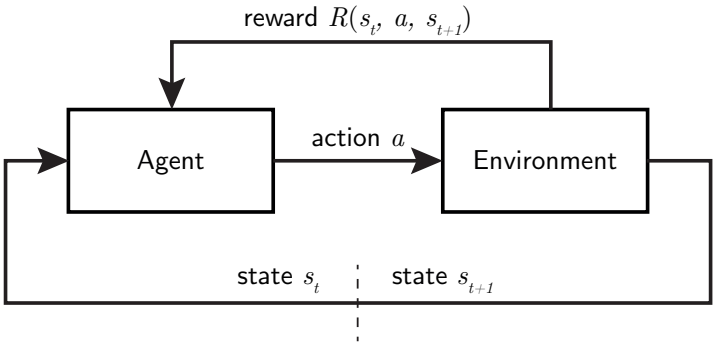


Figure 2.3: Schematic representation of an MDP agent which repeatedly selects an action $a \in A$ based on its current state $s_t \in \mathcal{S}$, after which it ends up in a state $s_{t+1} \in \mathcal{S}$ and receives a reward $R(s_t, a, s_{t+1})$ accordingly.

As the agent of an MDP aims to fulfill certain goals through the selection of actions, it requires some means of assessing which action is the best to pick. The value measure used in MDPs is defined by a mapping $R : \mathcal{S} \times A \times \mathcal{S} \mapsto \mathbb{R}$ from states and actions to real-valued *rewards* (in case of added value) and/or *costs* (in case of lost value). Due to the uncertainty in the modeled system, typically the agent uses the MDP's transition distribution (defined by transition function δ) to compute expected values, and accordingly selects actions that maximize this quantity. Putting these components together results in Definition 3 below.

Definition 3. A *Markov Decision Process (MDP)* is a 5-tuple $(\mathcal{S}, s_0, A, \delta, R)$, where $\mathcal{S}$ is a state space, $s_0 \in \mathcal{S}$ an initial state, $A = \{a_1, \dots, a_m\}$ a finite set of actions to choose from, $\delta : \mathcal{S} \times A \times \mathcal{S} \mapsto [0, 1]$ a probabilistic transition function and $R : \mathcal{S} \times A \times \mathcal{S} \mapsto \mathbb{R}$ a reward function.

In Figure 2.3 a schematic representation is shown of how an agent operates in an environment under the MDP framework. That is, in each decision epoch, the agent selects an action $a \in A$ and executes it in the environment causing a state transition from the current state $s \in \mathcal{S}$ to some state $s' \in \mathcal{S}$ where-after a reward $R(s, a, s')$ is received accordingly.

In the context of DTP, MDPs are used to find an optimal course of action, often referred to as a plan or *policy* $\pi : \mathcal{S} \mapsto A$. For an MDP the optimal policy typically means the policy that when applied by an agent, maximizes the expected value. However, one can also choose to express goals in alternative ways which do not require the specification of a reward function. An example of this can be seen in [6, 45], which replace the reward function of the classic MDP framework by a (co-safe) Linear Temporal Logic (LTL) formula to be satisfied.

Partially Observable MDPs

An extension of the traditional MDP models, are the Partially Observable MDP (POMDP) models which account for uncertainty in the observations that are made by agents. That is, while an MDP is used to model systems that are fully observable, in a POMDP the states are not observable and can only be inferred from the observations that are perceived. In other words, we can intuitively view a POMDP as the combination of a HMM and an MDP. As such a POMDP can be defined as in Definition 4 below.

Definition 4. A *Partially Observable MDP (POMDP)* is a 7-tuple $(\mathcal{S}, s_0, A, \delta, \mathcal{O}, \Omega, R)$ where $\mathcal{S}$ is a state space, $s_0 \in \mathcal{S}$ an initial state, $A = \{a_1, \dots, a_m\}$ a finite set of actions to choose from, $\delta : \mathcal{S} \times A \times \mathcal{S} \mapsto [0, 1]$ a probabilistic transition function, $\mathcal{O}$ an observation space, $\Omega : \mathcal{S} \times A \mapsto \mathcal{O}$ an emission or observation probability function and $R : \mathcal{S} \times A \times \mathcal{S} \mapsto \mathbb{R}$ a reward function.

As the true state of a POMDP is unknown, the transition function δ is defined over beliefs of states, and accordingly a policy π maps beliefs to actions. The observation function Ω defines the probability of observations from state-action pairs in the POMDP and is used to iteratively update the belief state of an agent.

2.1.3. Learning Markov Models

There exist efficient methods for fitting a discrete stationary first-order Markov Chain provided a dataset describing the evolution of a stochastic process either by applying likelihood maximization (e.g., see [2, 4, 46, 75]) or Bayesian inference (e.g., see [4, 46, 51]). These methods estimate the transition distribution to fit a Markov Chain on a collected sequence of time-ordered states. One may obtain such a sequence by applying clustering algorithms such as k-means on the dataset and accordingly make predictions for each of the entries.

The first-mentioned approach of likelihood maximization, works by estimating the transition probabilities by counting the observed transitions in the state sequence. That is, if we let n_{ij} denote the number of observed transitions from state s_j to state s_i, then the maximum-likelihood estimation of the corresponding transition probability is

$$p(q_{t+1} = s_i | q_t = s_j) = \frac{n_{ij}}{\sum_j n_{ij}} \tag{2.4}$$

The second-mentioned approach of Bayesian inference is more suitable for many real-life problems for which state sequences are incomplete, i.e., states are recorded only for certain stages, meaning there might be gaps in-between. This type of approach aims to make an estimation of the transition probabilities by adopting a prior for the transition matrix $\boldsymbol{A}$, a convenient choice being a factorized prior from the product of n independent *Dirichlet* distributions, one for each row $\boldsymbol{A}_j$, such that:

$$p(\boldsymbol{A}) = \prod_j \mathrm{Dir}(\boldsymbol{A}_j | \alpha_j) \tag{2.5}$$

parametrized by a vector $\boldsymbol{\alpha}$ with $\alpha_j > 0$ [4, 58].

Due to the close nature of MDPs these approaches can be applied almost directly to learn transition probabilities of discrete-state MDPs, although more data will be required due to the addition of actions. To learn partially observable models, on the other hand, one requires algorithms that work with a set of observations and estimate emission probabilities. For this thesis, however, we limit ourselves to learning fully observable MDPs, but review these algorithms for learning HMMs and POMDPs in Section 3.1.

2.1.4. Learning Optimal Policies

In SDM problems, the aim, typically, is to compute a policy that maximizes the total expected value. In algorithmic planning techniques, a probabilistic model of the environment, which includes estimates of the transition probabilities and rewards, is used to obtain an optimal policy by exploring its state space towards a goal. On the other hand, in *Reinforcement Learning (RL)* the optimal policy is learned while interacting with the environment and is applied when one does not know the transition probabilities and/or rewards for the system in advance. Both techniques have in common that they iteratively update estimations of a value function to derive a policy. The main difference though is that in planning this progress is carried out based on simulated experience from a model, while in learning techniques this is based on real experience from executing agents in an environment [74]. All in all, planning and RL are closely related and various ideas can be exchanged between the two.

In this section, various solutions that utilize the MDP framework for obtaining optimal policies are discussed. First off, the most common DTP algorithms for (discrete) MDPs are shown and explained. This is followed by a discussion of the most used algorithms in RL.

Model-Based Planning Techniques

In DTP the algorithms compute their plans or policies based on a probabilistic model of the environment. Most common is to prepare an MDP and apply exact dynamic programming solutions to compute an optimal policy for that model. The routine that is typically followed in MDP planning is depicted in Figure 2.4. The state space, transition function and action space are usually handcrafted, based on expertise or trial and error. The goal that should be fulfilled is translated into a reward model, mapping positive highly-valued rewards to desirable states and low rewards/costs to states that are not desirable or need to be avoided.

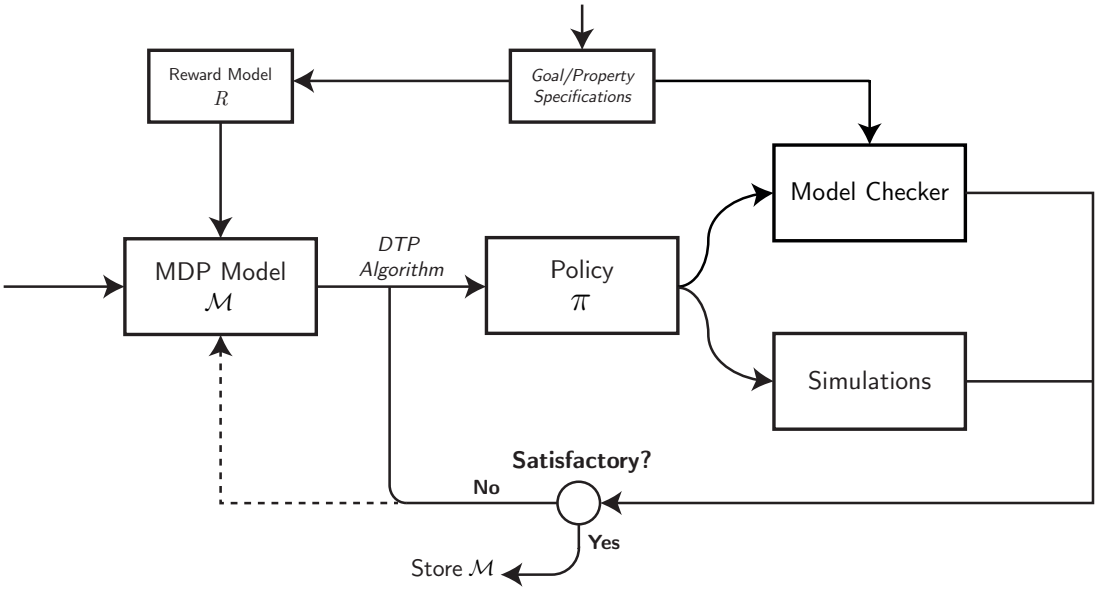


Figure 2.4: Block diagram of the generic routine employed in model-based MDP planning techniques.

Algorithm 1 Value iteration

Require: MDP $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$, Discount factor $\gamma \in [0, 1)$, Threshold $\xi > 0$
Ensure: Optimal policy $\boldsymbol{\pi}^* : \mathcal{S} \mapsto A$

```
Initialize V_t : S ↦ ℝ arbitrarily for t = 0
t ← 0
repeat
    t ← t + 1
    for all s ∈ S do
        V_t(s) ← max_{a∈A} [Σ_{s'∈S} δ(s,a,s') · [R(s,a,s') + γ · V_{t−1}(s')]]
    end for
until ∀s ∈ S : |V_t(s) − V_{t−1}(s)| ≤ ξ
for all s ∈ S do
    π*(s) ← arg max_{a∈A} [Σ_{s'∈S} δ(s,a,s') · [R(s,a,s') + γ · V_t(s')]]
end for
return π*
```

These parts together form the MDP which is fed to a planning algorithm, often referred to as an *MDP solver*. From this solver, a policy π is obtained, which is usually evaluated through simulations or automated model checking tools to check if the policy will satisfy the set of goals. After evaluating the current policy, one may choose to stick with the current policy or adjust the parameter settings of the solver or MDP model and obtain a new policy.

Value Iteration The most well-known algorithm for obtaining optimal policies from MDPs is *Value Iteration (VI)*, which is a synchronous dynamic programming solution which iteratively updates a value function $V : \mathcal{S} \mapsto \mathbb{R}$ until convergence, given an MDP model $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$. The updates are performed through so-called *Bellman backups* based on the following Bellman equation:

$$V^*(s) = \max_{a \in A} \left[\sum_{s' \in \mathcal{S}} P(s'|s,a) \cdot [R(s,a,s') + \gamma \cdot V_t^*(s')] \right] \qquad \forall s \in \mathcal{S} \tag{2.6}$$

where $\gamma \in [0, 1)$ is a discount factor which expresses the magnitude of preference of short-term solutions over long-term solutions. That is, the smaller the factor γ, the more important it is that goals are reached in as few steps as possible.

Algorithm 2 Policy iteration

Require: MDP $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$, Discount factor $\gamma \in [0, 1)$
Ensure: Optimal policy $\boldsymbol{\pi}^* : \mathcal{S} \mapsto A$
1: Initialize $\boldsymbol{\pi}_0$
2: $t \leftarrow 0$
3: **repeat**
4: Solve $\forall s \in \mathcal{S}$: $V_{\boldsymbol{\pi}_t}(s) \leftarrow \sum_{s' \in \mathcal{S}} \delta(s, \boldsymbol{\pi}_t(s), s') \cdot \left[R(s, \boldsymbol{\pi}_t(s), s') + \gamma \cdot V_{\boldsymbol{\pi}_t}(s')\right]$
5: $t \leftarrow t + 1$
6: **for all** $s \in \mathcal{S}$ **do**
7: $\boldsymbol{\pi}_t(s) \leftarrow \arg\max_{a \in A} \left[\sum_{s' \in \mathcal{S}} \delta(s, a, s') \cdot \left[R(s, a, s') + \gamma \cdot V_{\boldsymbol{\pi}_{t-1}}(s')\right]\right]$
8: **end for**
9: **until** $\boldsymbol{\pi}_t = \boldsymbol{\pi}_{t-1}$
10: **return** $\boldsymbol{\pi}_t$

In Algorithm 1, it is shown how the VI algorithm can be used to obtain optimal policies for an MDP (based on the formulation in [61]). Intuitively, VI can be viewed as estimating the values starting from the goal-rewards and working backwards. The initial values of the value function, stored in V_0, can be arbitrarily initialized, but to achieve faster convergence it is customary to do the initialization based on a rough estimation of V^*. In every iteration, the estimates of the value function are updated by the backup operations based on the value in previous operations. After a finite number of iterations the algorithm will converge (as is shown in [63]) and after that point $V(s)$ gives us the maximum to be expected sum of rewards starting from a state $s \in \mathcal{S}$. As can be seen in Algorithm 1, convergence is reached as soon as the change in value gets below a given threshold $\xi > 0$. After such convergence has been reached, the policy can be defined by selecting the action $a \in A$ for each state $s \in \mathcal{S}$ that is most likely to maximize the collected rewards based on the value of $V_t(s)$. All in all, the algorithm works well when the state space is relatively small, but for large state spaces more storage is required and it may take longer to reach convergence.

Note that an alternative formulation of the VI algorithm can be given that makes use of a Q-table instead of a value function V [71]. In this formulation the entries $Q(s, a)$ are updated in each iteration for each state-action pair. The optimal policy can then be obtained directly by selecting the action with the maximum value in the Q-table for each action, though it requires more storage compared to a value function.

Asynchronous Value Iteration An adaptation of the traditional VI algorithm is *asynchronous VI*, which is an asynchronous dynamic programming solution. Almost all of the steps in asynchronous VI are the same except that rather than updating the value function $V : \mathcal{S} \mapsto \mathbb{R}$ for each state in each iteration, the function is updated only for a single state in each iteration in no particular order (or even randomized). In the case of a fixed ordering, this algorithm is usually referred to as *Gauss-Seidel VI*. Compared to the traditional VI algorithm, the asynchronous adaptation requires less space and converges faster, especially when updates occur more often for most relevant states and the update ordering is adjusted carefully.

Real-Time Dynamic Programming (RTDP) [5] is a family of asynchronous VI algorithms which aim to find policies by performing updates and concurrently controlling the MDP (based on the policy corresponding to the latest estimate of the value function). In contrast to traditional VI, where solving MDPs with large state spaces is infeasible, RTDP algorithms often converge without examining all states.

Policy Iteration Another algorithm that is widely applied for obtaining policies from MDPs is known as *Policy Iteration (PI)*. As shown in Algorithm 2 PI starts off with an initial policy vector $\boldsymbol{\pi}_0$ which may be arbitrarily initialized, but preferably by an approximation of an optimal policy for the input MDP $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$ to achieve faster convergence. Then, in each iteration, first the value for each state is computed based on the latest policy $\boldsymbol{\pi}_t$, which comes down to solving a set of linear equations. Solving this set of linear equations can be

Algorithm 3 Backwards induction

Require: MDP $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$, horizon $h \in \mathbb{N}$
Ensure: Optimal policy $\boldsymbol{\pi}^* : \mathcal{S} \mapsto A$
1: $t \leftarrow h$
2: $\forall s \in \mathcal{S} : V_h(s) \leftarrow 0$
3: **repeat**
4: $\quad t \leftarrow t - 1$
5: $\quad$ **for all** $s \in \mathcal{S}$ **do**
6: $\quad\quad V_t(s) \leftarrow \max_{a \in A} \left[\sum_{s' \in \mathcal{S}} \delta(s, a, s') \cdot [R(s, a, s') + V_{t+1}(s')]\right]$
7: $\quad$ **end for**
8: **until** $t = 1$
9: **for all** $s \in \mathcal{S}$ **do**
10: $\quad \boldsymbol{\pi}^*(s) \leftarrow \arg\max_{a \in A} \left[\sum_{s' \in \mathcal{S}} \delta(s, a, s') \cdot [R(s, a, s') + V_1(s')]\right]$
11: **end for**
12: **return** $\boldsymbol{\pi}^*$

done by linear programming in (at most) $O(|\mathcal{S}|^3)$ time [47]. An alternative approach, known as *modified policy iteration*, is to solve these by applying a simplified form of value iteration in which the actions to select are already known for each state from the policy (i.e., $\boldsymbol{\pi}_t(s)$ for each $s \in \mathcal{S}$). This step which is known as *policy evaluation* is followed by a *policy improvement* step in which the policy is greedily updated based on the latest value function $V_{\boldsymbol{\pi}}$. This process is repeated until no improvements are possible and the policy stops changing.

Solving the set of linear equations in each iteration is a time-costly operation, especially for large state spaces, in comparison to the $O(|\mathcal{S}|^2 \cdot |A|)$ time required for an iteration of VI. The main advantage of PI can however be that for a finite space the algorithm converges in a finite number of steps when there is no more strict improvement, because only a finite number of stationary policies exist, while in VI the convergence depends on the threshold.

Backwards Induction In the special case of a finite horizon MDP it is possible to obtain an optimal policy by applying *backwards induction* [15] shown in Algorithm 3. As the name suggests the algorithm works backwards, starting from the last step h and recursively using the value function of step $t + 1$ to compute that of step t. Even though the algorithm works well when dealing with finite horizons, typically real world scenarios more often deal with infinite horizons for planning under uncertainty.

Linear Programming A less frequently applied approach is that of formulating the MDP as a Linear Program (LP) and solving it using so-called *simplex* methods. Following this approach, an LP formulation for an MDP $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$, as explained in [59], is:

$$\begin{aligned} &\min_V \sum_{s' \in \mathcal{S}} \mu_0(s) \cdot V(s) \\ &\text{s.t. } V(s) \geq \sum_{s' \in \mathcal{S}} \delta(s, a, s') \left[R(s, a, s') + \gamma \cdot V(s')\right] \qquad \forall s \in \mathcal{S}, \forall a \in A \end{aligned} \tag{2.7}$$

where $\mu_0 : \mathcal{S} \mapsto [0, 1]$ is a probability distribution over the states.
From the solution V^* of this LP formulation, a policy $\boldsymbol{\pi}^*$ can be obtained by letting

$$\boldsymbol{\pi}^*(s) = \arg\max_{a \in A} \sum_{s' \in \mathcal{S}} \delta(s, a, s') \left[R(s, a, s') + \gamma \cdot V(s')\right]$$

for each state $s \in \mathcal{S}$. This primal formulation optimizes a value function V, but alternatively the policy can be optimized directly by considering the dual formulation (see [47]). Comparing LP algorithms to the specialized VI and PI algorithms, the latter typically hold more promise for efficient solutions than general-purpose LP algorithms, although the LP scale better to larger MDP planning problems.

Algorithm 4 SARSA

Require: State space $\mathcal{S}$, Action space A, Learning rate $\alpha \in (0, 1)$, Discount factor $\gamma \in [0, 1)$
Ensure: Optimal policy $\boldsymbol{\pi}^* : \mathcal{S} \mapsto A$
Initialize $Q : \mathcal{S} \times A \mapsto \mathbb{R}$ arbitrarily
repeat
  Pick $s \in \mathcal{S}$ arbitrarily
  $a \leftarrow \pi[s]$ ▷ π is the policy derived from Q
  repeat
    Perform a and observe r, s'
    $a' \leftarrow \pi[s']$
    $Q(s,a) \leftarrow Q(s,a) + \alpha\left[r + \gamma Q(s',a') - Q(s,a)\right]$
    Update π based on Q
    $s \leftarrow s'$
    $a \leftarrow a'$
  until s is terminal
until Q converged
return $\boldsymbol{\pi}^*$

Reinforcement Learning Techniques

Reinforcement Learning (RL) techniques work on real experience based on which they update the behavior of the agent which is defined by an action-selection policy. To do this they require direct interaction with the environment to obtain experience and update value function estimations accordingly during execution. Although most planning algorithms cannot be used for learning problems, learning algorithms can be used for planning problems as they can make use of simulated experience. There are several techniques that exist, of which the most common ones are discussed in the remainder of this section.

Q-Learning Q-Learning [74, 78] is a model-free reinforcement learning technique which discovers an action-selection policy by learning estimates of the optimal Q-values of an MDP. The technique starts off with a Q-table Q, which is arbitrarily initialized, containing the Q-values $Q(s,a)$ for each state-action pair (s,a). At each point in time the agent for which the Q-Learning technique is applied is assumed to be in a certain state s and chooses a next action a to execute based on the current Q-value estimates. After executing an action the agent ends up in a new state s' and observes a certain reward r. The Q-table is then updated accordingly based on the following update rule:

$$Q(s,a) = Q(s,a) + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right]$$

where $\alpha \in (0,1)$ is the learning rate and $\gamma \in [0,1)$ the discount factor. The learning rate expresses the rate at which newly acquired information overrides the old information in the Q-table. Over time the estimates in the Q-table will improve such that the revenue is maximized. A near-optimal policy can then be constructed by selecting the action with the highest Q-value from the Q-table for each state.

SARSA SARSA [74] is another model-free RL algorithm which is similar to Q-Learning. While Q-Learning is an off-policy method which learns the value of the optimal policy, SARSA is an on-policy method which learns the value of the policy it currently follows in order to iteratively improve this policy. In each iteration, it simulates the action yielded by its policy and observes the reward and next state. Accordingly the Q-table is updated based on these observations as shown in Algorithm 4, which repeats itself with the observed s' and action a' imposed by the policy π that is derived from Q.

2.2. Bayesian Optimization

This section elaborates upon a method known as Bayesian Optimization (BO), which is used to automate the process of optimizing the parameters of an unknown objective. First Section 2.2.1 introduces the optimization tasks this method is aimed at and how these are relevant to SDM problems. Subsequently, in Section 2.2.2 the optimization method is formally described. In Section 2.2.3 the configurable parts of the method are discussed, considering both advantages and disadvantages of available options. Finally, in Section 2.2.4 a number of applications of Bayesian Optimization in the field of planning under uncertainty are discussed, serving as an overview of the method's successes in this field.

2.2.1. Problem Formulation

One of the problems that is faced in the field of optimization is that of maximizing a nonlinear, real valued *objective function* $f : \mathcal{X} \mapsto \mathbb{R}$ on a domain $\mathcal{X} \subset \mathbb{R}^m$ $(m \geq 1)$. Formally, to find a global maximizer $x^* \in \mathcal{X}$ for which:

$$x^* = \arg\max_{x \in \mathcal{X}} f(x) \tag{2.8}$$

In particular the problem turns out to be a common bottleneck when dealing with an objective function that is unknown and expensive to evaluate in terms of the required computational resources. As an example, one could think of finding the hyperparameters for a neural network which maximize the performance, where each single evaluation of a set of parameters requires one to train the neural network and assess the performance on a huge dataset.

Although this problem of optimizing expensive functions can be found in many different contexts, it is foremost a problem in SDM. That is, typically one can only hope to estimate objective functions of SDM problems in AI planning and reinforcement learning through expensive simulations [10]. A particularly relevant optimization problem pops up when considering the problem of learning to control systems that involve uncertainty, in which an important issue is that of faithfully modeling the uncertainty in the outcomes of actions [29]. This particular problem has recently received quite some attention, posing it as a problem of learning and optimizing the parameters of a probabilistic model of the system, such as the model's transition probabilities, reward function or optimal policy [62].

A naive approach for optimizing the objective would be to evaluate a set of (random) combinations of parameters and see which parameter settings seem to give the best results. This approach however, usually requires expert knowledge and might demand a large number of function evaluations that do not necessarily provide new information about the parameter space. The method known as *Bayesian Optimization*, described in Section 2.2.2, improves on these naive approaches by making predictions about which regions of the parameter space are expected to give the best results and hence limiting the number of function evaluations.

2.2.2. Algorithm Description

Bayesian Optimization (BO) is a powerful method for finding the maximum of a typically unknown, expensive, nonlinear objective function, while aiming to minimize the number of objective function evaluations and avoiding local maxima [10]. This method first requires one to set a prior $p(f)$ over the objective function f, representing the belief about the space of plausible objective functions. Then, the algorithm starts off by gathering a small set of initial sample-observation pairs of samples $x \in \mathcal{X}$ and corresponding objective values $y = f(x) + \varepsilon$. These pairs are then stored in a set $\mathcal{D}_{1:t} = \{(x_i, y_i) \mid i = 1 \dots t\}$ (i.e., the *evidence set*) where we let x_i denote the ith sample and $y_i = f(x_i) + \varepsilon_i$ the corresponding ith observation with noise ε_i. The algorithm then derives a posterior distribution $p(f|\mathcal{D}_{1:t})$ which, according to Bayes' Theorem, is said to be proportional to the likelihood $p(\mathcal{D}_{1:t}|f)$ and the prior $p(f)$ for the first t gathered observations, s.t.:

$$p(f|\mathcal{D}_{1:t}) \propto p(\mathcal{D}_{1:t}|f) \cdot p(f) \tag{2.9}$$

This posterior can be viewed as an estimation of the objective function f, sometimes referred to as a *surrogate function*.

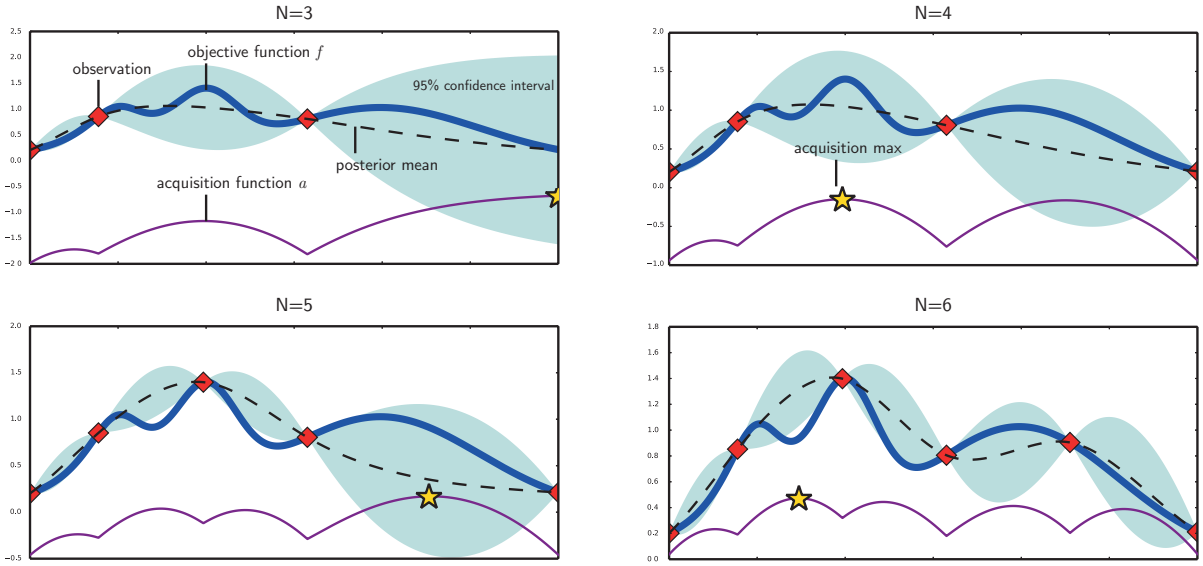


Figure 2.5: An example of using Bayesian Optimization on a toy example problem. The plots show a GP approximation of objective function f over four iterations based on N observations. The plots also show the corresponding GP-UCB acquisition function a for these GP approximations, which yield high utility for those areas of the domain $\mathcal{X}$ with high prediction uncertainty.

Algorithm 5 Bayesian Optimization (General Formulation)

Require: Domain $\mathcal{X} \subset \mathbb{R}^m (m \geq 1)$, prior $p(f)$ and acquisition function $a : \mathcal{X} \mapsto \mathbb{R}$

1: $\mathcal{D}_{1:n} \leftarrow \{(x_i, y_i)\}|_{i=1}^{n}$ ▷ $\mathcal{D}$ is the evidence set with $\mathcal{D}_{1:n}$ the n initial sample-observation pairs
2: **for** $t \leftarrow n+1, n+2, \ldots$ **do**
3: $x_t \leftarrow \arg\max_{x \in \mathcal{X}} a(x|\mathcal{D}_{1:t-1})$ ▷ Acquisition based on posterior $p(f|\mathcal{D}_{1:t})$
4: $y_t \leftarrow f(x_t) + \varepsilon_t$
5: $\mathcal{D}_{1:t} \leftarrow \mathcal{D}_{1:t-1} \cup \{(x_t, y_t)\}$ ▷ Augment $\mathcal{D}$ with the new evidence
6: Update the posterior $p(f|\mathcal{D}_{1:t})$
7: *Break* when satisfactory ▷ Stop condition defined by implementation
8: **end for**
9: **return** $\arg\max_{(x_i, y_i) \in \mathcal{D}} y_i$

To decide on which $x \in \mathcal{X}$ to sample and gather a new observation $f(x)$ from next, a so-called *acquisition function* $a : \mathcal{X} \mapsto \mathbb{R}$ is used, which assigns a certain utility to evaluating f at some particular $x \in \mathcal{X}$, given the evidence set $\mathcal{D}$ at that time. This acquisition function should be defined such that it captures a correct balance between *exploration* (to sample from areas with high uncertainty) and *exploitation* (to sample from areas likely to improve on prior observations). For this reason many different classes of acquisition functions exist, which are discussed in detail in Section 2.2.3.

The procedure is illustrated by a toy example in Figure 2.5 with the goal of retrieving the global maximum of the usually unknown objective function f. The prior and acquisition function in this example are set to some of the most common options (i.e., a zero-mean Gaussian Process and GP-UCB function). The plots show four subsequent iterations which each add a new observation of the objective function f and update the posterior/surrogate function accordingly. The sample for each next iteration is selected at the point with the highest utility in the acquisition function a, in the plots denoted by a star symbol.

The general formulation of the BO framework is presented in Algorithm 5. An implementation of this framework still requires one to define three components, which are the domain $\mathcal{X}$ of f, the prior over f, and last of all the acquisition function a. The next section discusses the various options that exist for the latter two components in detail.

2.2.3. Choice of Prior and Acquisition Function

In order to apply Bayesian Optimization there are two major choices that need to be made, which are those of the prior distribution over the objective and the acquisition function. In this section the different options and corresponding considerations that need to be made for these two components are elaborated upon.

Prior Distributions

In Bayesian Optimization the *de facto* standard for the prior distribution is a Gaussian Process (GP), which is typically well-suited as it accounts for the uncertainty associated with each prediction. Another important aspect that makes a GP a convenient choice for the prior distribution is that it induces a posterior distribution over the objective function that is analytically tractable. Intuitively a GP can be viewed as a prior which assumes that similar inputs result in similar outputs. For an objective function f, the GP defines a Gaussian probability distribution over $f(x)$ for each x, so that a GP can be expressed as a probability distribution over functions:

$$P(f(x)|x) = \mathcal{N}(\mu(x), \sigma^2(x)) \tag{2.10}$$

where $\mathcal{N}$ denotes a normal distribution, while μ and σ denote mean and standard deviation respectively. By this definition, a GP can be viewed as a function that returns the mean and variance of a normal distribution over the possible values of f at x. A GP as prior over an objective function f is typically denoted as:

$$f \sim GP(m(\cdot), K(\cdot,\cdot)) \tag{2.11}$$

so that the GP is completely specified by a mean function m and a *kernel* function K defining the covariance. The mean function μ is typically initialized by a constant mean, usually zero, due to the assumption that all points in the parameter space are equally likely and because the conditional mean can still be flexibly specified by the kernel function K [42].

The choice of the kernel or covariance function of the GP determines the smoothness of the estimations on the performance and confidence intervals of unexplored samples in the parameter space. According to the various literature in the field of Bayesian Optimization the most common kernels are said to be the *squared exponential* (also known as *radial basis function (RBF)* or Gaussian) kernel and Matérn kernel. However, the squared exponential kernel turns out unrealistically smooth for practical applications [72] and therefore would require properly selecting its hyperparameters. The Matérn kernel serves as a more flexible class, which is parameterized by the hyperparameter ν specifying the expected smoothness of the approximated function, i.e., which can be used to tweak the distance at which there are almost no effects from previous samples and the rate at which these effects decrease. As concluded in [48, 65, 68] the most interesting and most commonly used for machine learning applications are the Matérn kernels with $\nu = 3/2$ and $\nu = 5/2$, as can be seen being applied in various works [42, 72, 76].

Acquisition Functions

The next step is defining an acquisition function which is used in the optimization process to efficiently sample observations towards the global optimum. Typically, high utility in the acquisition functions corresponds to potentially high objective function values. However, to avoid getting stuck in local optima, acquisition functions are defined such that a trade-off is made between exploration and exploitation. In each iteration of the BO framework, a new observation is sampled at the point $x \in \mathcal{X}$ where the utility is the highest. Traditionally, the Maximum Probability of Improvement (MPI), Maximum Expected Improvement (MEI) and GP Upper Confidence Bound (GP-UCB) functions are used for Bayesian Optimization [68]. In the discussion of these acquisition functions, let us define $f(x^+)$ as the 'best' observation, corresponding to the sample $x^+ = \arg\max_{x_i \in x_{1:t}} y_i$ when considering the first t samples. In the following overview each of these acquisition functions are defined accompanied by a discussion of their main advantages and disadvantages seen in practice.

Maximum Probability of Improvement (MPI) This acquisition function selects the next sample to maximize the probability of improvement, which is the sample $x \in \mathcal{X}$ so that:

$$\arg\max_{x \in \mathcal{X}} P(f(x) \geq f(x^+) + \xi)$$

where $\xi \geq 0$ is a trade-off parameter. The higher this trade-off parameter ξ, the higher the preference of exploration over exploitation. Note that this means that setting $\xi = 0$ implies that the optimization purely depends on exploitation, acquiring points most likely to yield improvement. One could also choose to update ξ dynamically over time.

Advantages Intuitive acquisition function which can guarantee a minimum improvement of ξ in each iteration. Considers both mean, variance and current best $f(x^+)$ of the surrogate function.

Disadvantages Difficult to tune trade-off parameter and extremely sensitive to the choice of this parameter: a too small ξ may result in an exhaustive search around a local optimum, while a too large ξ results in a slow search (close to random sampling).

Maximum Expected Improvement (MEI) This acquisition function defines its utility based on the magnitude of the improvement to be expected. The function ignores the points that would lead to a decrease compared to the current best $f(x^+)$. The MEI function selects the point $x \in \mathcal{X}$ such that:

$$\arg\max_{x \in \mathcal{X}} \mathbb{E}[\max(0, \mu_f(x) - f(x^+) - \xi)]$$

where $\mu_f(x)$ is the posterior mean at x for the objective f and $\xi \geq 0$ a trade-off parameter.

Advantages Common steps exist for properly setting ξ, with $\xi = 0.01$ appearing to work well in most cases according to [48]. Allows for non-myopic extensions [10].

Disadvantages Points that are likely to bring no improvement are neglected, although the chance of them resulting in actual improvement is usually small.

GP Upper Confidence Bound (GP-UCB) This acquisition function defines its utility by looking at the curve that is β_t standard deviations above the posterior mean μ_f and samples $x \in \mathcal{X}$ such that:

$$\arg\max_{x \in \mathcal{X}} \mu_f(x) + \beta_t \sigma_f(x)$$

where β_t is an $\mathcal{O}(\log t)$ exploration coefficient scheduled over time [18]. This acquisition function relies on the idea of being optimistic in the face of uncertainty.

Advantages Strong bounds exist on the cumulative regret and also guidelines exist to set β_t to achieve optimal regret.

Disadvantages Hyperparameter β_t needs to be set properly.

In some domains, the time that is needed to evaluate the objective function depends on the region from which a point $x \in \mathcal{X}$ is sampled. For example, when again the optimization of the hyperparameters of neural networks is considered, one may observe that for some settings of parameters it takes a lot longer to train a model. To account for this aspect, one could apply cost-sensitive optimization by using the Maximum Expected Improvement Per Second (MEIPS) function.

Maximum Expected Improvement Per Second (MEIPS) This acquisition function, proposed in [72], makes use of another GP-model over the evaluation time of the objective function over the same domain $\mathcal{X}$. The MEIPS selects the next sample $x \in \mathcal{X}$ such that:

$$\arg\max_{x \in \mathcal{X}} \frac{\mathbb{E}[\max(0, \mu_f(x) - f(x^+) - \xi)]}{\mu_s(x)}$$

where $\mu_s(x)$ is the posterior mean at x for the timing GP-model, while μ_f and ξ are defined as in the MEI function.

Advantages Prefers samples that are expected to be evaluated with lower cost.

Disadvantages Requires another timing GP-model accompanied by increased costs.

Further expanding on this, various *portfolio allocation* techniques have been suggested that address the issue that there is no single acquisition function that outperforms the others for all problem instances. One example is the *GP-Hedge* algorithm [39, 68] which holds a portfolio of acquisition functions and employs past performance of each of these functions to predict their future acquisition performance. However, this algorithm fails to account for valuable information gained through exploration, which has led to alternative portfolio algorithms, such as the *Entropy Search Portfolio* [68, 77] which considers the gain of information of each acquisition function towards the optimum.

Finally, one can choose to blend the acquisition functions of a set of GPs with varying hyperparameter settings by computing the *integrated acquisition function* [72]:

$$\hat{a}(x|\mathcal{D}) = \int a(x|\mathcal{D}, \theta_{gp}) p(\theta_{gp}|\mathcal{D}) d\theta_{gp} \approx \frac{1}{K} \sum_{k=1}^{K} a(x|\mathcal{D}, \theta_{gp}^{(k)}) \tag{2.12}$$

where $a(x)$ is the utility at $x \in \mathcal{X}$ in the MEI function for the GP with hyperparameters θ_{gp}, given the evidence set $\mathcal{D}$. Computing $\hat{a}(x)$ is commonly realized by a Monte Carlo integration over the acquisition functions of the GPs. This integrated function is typically used to account for uncertainty in the hyperparameters of the underlying GP in the BO framework. A benefit of this Bayesian treatment of these hyperparameters is that the number of choices to be made is reduced and leads to a more automated optimization solution.

2.2.4. Applications of Bayesian Optimization to SDM Problems

As we consider the optimization of learning probabilistic models for planning under uncertainty in this thesis, it might be valuable to have a look at how Bayesian Optimization is applied to closely-related tasks for planning. Therefore, in this section a number of applications that involve or are closely related to DTP are highlighted, to get an idea of the applicability of Bayesian Optimization and identify the motivations for using this method.

In [49], Bayesian Optimization is applied for a mobile robot that adaptively plans a path while maximizing the information it obtains from observations about its own location and the location of navigation landmarks in the environment. The objective/cost function C is parametrized by a policy vector π and approximations for selected samples (of robot pose and landmark locations) are made by different functions over the belief states of the POMDP (i.e., average mean square error (AMSE), maximum a posteriori square error (MAPSE) and a largest marginal heuristic). Typically, estimating the belief state is an expensive problem, typically carried out through SLAM algorithms in robotics. Therefore, to minimize computational cost, a Bayesian Optimization algorithm is applied with the choice of a GP prior over C and where new samples are acquired using an MEI acquisition function. Applying Bayesian Optimization allows for a more cost-efficient online path planning method for optimal exploration of an environment compared to other approaches.

Another consideration for exploration of environments with stochastic dynamics is that of safety. That is, the assumption of an MDP being *ergodic*, which is when any state is reachable from another, is impractical as systems tend to get stuck or break due to unsafe exploration. Therefore, in [76] a method of safe exploration is proposed that, based on an initial set of states that are known a priori to be safe, iteratively identifies which nearby states are safe to visit. This is done by making regularity assumptions on the safety feature, and by such modeling this safety feature by a GP prior with a Matérn 5/2 kernel. The goal of their `SAFEMDP` algorithm is to visit those states that expand the set of safe states (of which is known that a safe return route exists which is inferred from the MDP's transition dynamics) as quickly as possible, so to minimize the resources required to explore the MDP. Therefore, the acquisition is performed such that those states are selected that are known to be safe and comprise the highest uncertainty, so that the acquired knowledge with every sample is maximized.

Finally, multiple papers have shown that the BO framework can be employed to speed up online policy search for RL tasks. The proposed algorithms in these papers, have the BO framework select and evaluate a policy in each iteration and keep track of estimates of the expected value of all policies. One example is the `PILCO` framework [21], a model-based online policy search method, which models the dynamics of a system by a non-parametric GP so that policies are retrieved indirectly. Another example is seen in [81] in which Bayesian Optimization is used to optimize policies for RL by exploiting generated trajectory data. To take advantage of the obtained trajectory information in the Bayesian Optimization this information is intertwined with the kernel of the GP prior, in such way that the induced behavior (i.e., defined by sequences of selected actions) of pairs of policies are taken into account in a behavior-based kernel (BBK). Accordingly, given an evidence set $\mathcal{D}_{1:n} = \{(\pi_i, \eta(\pi_i), \xi_i)\}$ an (estimate) of the expected return $\eta(\pi_i)$ of policy π_i is computed using a set of trajectories ξ_i. This paper also proposes a model-based Bayesian Optimization method which learns transition probabilities and reward functions based on collected trajectory data to approximate the expected return through simulations on a probabilistic model. To account for inaccuracies of learned models, this model-based algorithm introduces a β term in the kernel that ensures that the model information is neglected when it turns out to be inaccurate based on the occurrence of systematic errors.

3

Related Work

A common approach in DTP, as was discussed in Chapter 1, is to formulate plans by making use of (compact) probabilistic models of the systems under consideration. In particular we focus our attention to devising MDP models, which need to accurately define what constitutes the state of the system, reflect uncertainty through transition probabilities, and specify goals through rewards. The problem we face is that handcrafting accurate, well-generalizing models is a difficult and time-costly process for a human designer. To overcome this problem the goal is to automatize the model development process by applying learning algorithms on data that describes the dynamics of the system under consideration. Therefore, this chapter aims to provide some insight into the existing approaches for automating the development of probabilistic models for planning under uncertainty and the existing alternatives for automated control of systems involving uncertainty.

First Section 3.1 gives an overview of the existing algorithms for learning models from execution traces of the system. Then, in Section 3.2 a number of approaches are examined, which combine offline methods for learning (initial) models with RL algorithms which update the model parameters online. Finally, Section 3.3 looks into approaches that extend the traditional MDP framework by accounting for uncertainty in the transition probabilities, with the aim to overcome the infeasibility of obtaining completely accurate probabilistic models.

3.1. Learning Probabilistic Models from Execution Traces

To facilitate the learning of system models, the first step is to obtain a dataset which describes the dynamics of the system under consideration. A common approach of obtaining this data is through the recording of observations made in an exploration phase in which the system aims to gather information about the effect of performing various actions in different situations. This exploration could be done selecting actions at random, executing a specific policy, or following another method of choosing actions. The next step is to learn a probabilistic model which most accurately explains the execution traces obtained in this exploration phase. In this section a number of the algorithms for this purpose are considered, where Section 3.1.1 evaluates the well-known *Baum-Welch* algorithm that iteratively updates transition and emission probabilities starting from a model with initial estimates of these probabilities, while Section 3.1.2 addresses a class of algorithms based on merging (time-)states for developing MDPs.

3.1.1. The Baum-Welch Algorithm

The most widely applied approaches for learning probabilistic models are based on maximizing the likelihood of observing the execution traces of the dataset. When the goal is to learn the parameters of a Markov Chain, the transition probabilities can be estimated as we have seen in Section 2.1.2, by the ratio of the times a particular transition is observed and the total number of transitions observed from the corresponding starting state. However, while Markov Chains are similar to MDPs in that they describe the evolution of a stochastic pro-

cess over time, they do not involve decisions on actions to perform. Due to the close nature of these probabilistic models the likelihood maximization can be applied almost directly to learn the transition probabilities of MDPs, although requires more data due to the addition of actions.

In practice, it is more common to only have a sequence of observations at one's disposal which do not directly map to the states of the system. The approach of likelihood maximization generalizes to learning partially observable Markov Models through the well-known *Baum-Welch algorithm* [79], which is a special instance of the Expectation Maximization (EM) algorithm generally used for learning the parameters of HMMs. Given a discrete or continuous observation space $\mathcal{V}$, a discrete state space $\mathcal{S}$, and a set $\mathcal{O}$ of i.i.d. observation sequences, the algorithm learns a model $\mathcal{M}^* = \arg\max_{\mathcal{M}} p(\mathcal{O}|\mathcal{M})$. Informally, this is done by first providing manual estimates of the model parameters and then iteratively computing the expectations of how frequently the transitions and emissions are used (i.e., the *E-step*) and updating these parameters based on those computed expectations (i.e., the *M-step*). For a more detailed explanation of the Baum-Welch algorithm one may refer to the tutorial by Bilmes in [7].

Although the approach turns out to work quite well for learning HMMs for recognition purposes, some problems emerge when applying the algorithm to learn POMDPs for planning under uncertainty. That is, the algorithm is well known to be sensitive to its initialization (particularly in the case of continuous observations) and depending on the choice of the initial model parameters it may result in local maxima. However, to some extent this problem can be overcome by segmenting the observation sequences using k-Means clustering to restrict the model to a discrete HMM or POMDP (e.g., see [13]). This discretization however, depending on the choice of the hyperparameter(s) of the clustering algorithm, might neglect some valuable distinctions between observations or states of the system.

A particularly relevant work is that of Shatkay and Kaelbling [69] in which the algorithm is applied to learn POMDP models in the context of mobile robot navigation. In their work they make the assumption of a finite set of states whose size is known, and associate each state with a point in some metric space derived from the odometric ability of the robot. Based on a set of gathered odometric readings an initial *topological* model is learned by applying k-Means clustering, taking the clusters as the states in which the observations were made, and based on state and observation counts make initial estimates of the model parameters. Then, an adapted version of Baum-Welch is applied which takes into account odometric information to iteratively update the model. The obtained results are compared to those that emerge from the traditional Baum-Welch algorithm, which is used to demonstrate that exploiting odometric information can reduce the number of iterations and improve the final model.

Similarly, an adaptation of the Baum-Welch algorithm is applied in [44] to learn POMDPs which exploits prior knowledge of map symmetry and does not adjust probabilities that are assumed to be approximately correct since the sensor and actuator models are expected to be similar in different environments. In this way the amount of required training data is restricted and updating the model happens more selectively and efficiently, although it demands the need for handcrafting an initial topological model and defining constraints on the model structure.

As opposed to the related work described in this section, in our approach we aim to avoid any assumptions or constraints on the MDP model parameters. For instance, in the experiments presented in Chapter 5, we obtain the state space in exactly the same way as in [69], although avoid the assumption that the size is known. Instead, we choose to learn the most appropriate choice for this hyperparameter with which the best performance in the execution of the system's tasks can be obtained.

3.1.2. State Merging Algorithms

A completely different class of algorithms is based on the merging of states to acquire probabilistic models. The type of algorithms that are considered here are based on a method called *Best-First Model Merging* which was first proposed by Stolcke and Omohundro in [73] to learn HMMs for the purpose of speech recognition.

Algorithm 6 State Merging by Trajectory Clustering

Require: Training set $\mathcal{O} = \{O_1, \ldots, O_k\}$, Clustering algorithm, hyperparameter(s) θ, actions A
Ensure: MDP $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$ ▷ s_0 and R are here irrelevant
1: Compute similarities of time-states based on trajectories
2: Cluster most similar time-states to obtain the state space $\mathcal{S}$
3: Compute the transition probabilities to obtain δ
4: **return** MDP $\mathcal{M}$

Given a training set $\mathcal{O}$ of i.i.d. observation sequences, this algorithm starts from an initial HMM whose state space consists of separate states for each time-step in the sequences. The entries of the transition and emission matrix are thus initialized with probability one for each of the transitions between subsequent states and each percept in the observation sequence respectively. Although this HMM explains the data perfectly it is usually not efficient and therefore the model is iteratively updated by merging those pairs of states that decrease the likelihood the least. This repeats itself until a stop criterion is reached, such as the likelihood falling below a certain value or reaching a predefined number of states.

This approach can be adapted for learning POMDPs as described in [53] such that the transition function is initially incomplete as it will only be defined for the actions corresponding to the transitions between subsequent states. A particular advantage of this approach is that one can choose to adapt the merging criterion so that, for instance, states are only merged if they both map to the same goal percept. However, a major disadvantage of the approach is that alternative merges are never considered by the algorithm, although in the end they may result in better models. This means that the approach is prone to local maxima, although this can be avoided to some extent by considering a set of most promising merges in each iteration, although it would make the algorithm even more cost-expensive than it already is.

A variation of the algorithm is a method seen in [53] that goes under the name of *State Merging by Trajectory Clustering*, which makes the state merging approach more resilient to *perceptual aliasing*. In this approach the merging criterion is changed such that in each iteration those pairs of states are merged whose prior and posterior trajectories in the training set are most similar. The pseudocode of this method is shown in Algorithm 6 where the similarity of states is thus defined by the length of the common trajectories from and towards the states. In case the observations have an underlying metric space, one can choose to apply algorithms like k-Means clustering to cluster the most similar time-states, or alternatively turn to clustering methods that work on similarity matrices.

Although a simply likelihood maximization approach for MDPs has been followed in our experiments presented in Chapter 5, the algorithms described here may prove useful for potential future work. Especially when we want to learn state spaces for POMDPs for our application domain, the latter algorithm seen in [53] appears better capable of dealing with perceptual aliasing that is often encountered in real-world systems. However, when the state space of POMDPs is not learned from the available data, employing the Baum-Welch algorithm is probably a more attractive alternative, considering the state merging algorithms are computationally quite expensive.

3.2. Active Learning

Devising a completely accurate specification of the MDP's transition probabilities typically turns out infeasible for real-world DTP problems. However, it should be noted that the learning algorithms that have been addressed earlier, usually recover enough to let an MDP reflect the actual environment in which an agent operates. The plans that are derived accordingly from these models yield better performance than a random choice of actions in the execution of the system would. A class of approaches that we consider in this section are those that combine offline model learning and planning methods with online (reinforcement) learning methods which incrementally update the model parameters.

3.2.1. Model-Based Reinforcement Learning

In RL a distinction exists between model-free and model-based methods, where in the former the choice of action is made based on previously realized value, while the latter evaluates candidate actions based on their expected future rewards. Model-free or *direct* RL can therefore be viewed as corresponding to habitual behavior, while model-based RL corresponds to goal-directed behavior. When the system under consideration encompasses considerable structure and its dynamics can be modeled accurately, model-based approaches tend to use experiential data more efficiently, be more resilient to changing goals and are better capable of obtaining near-optimal policies [1]. The SDM problems involving these type of systems are those that qualify for applying model-based RL in which an initial MDP could be obtained by offline model learning. However, one should note that model-free RL approaches are better suited for learning from scratch (i.e., SDM problems with very limited prior knowledge and data), as model-based RL approaches tend to suffer from model bias [21]. That is, model-based RL makes the inherent assumption that the underlying model sufficiently accurately reflects the real-world environment.

To improve the estimations of the model parameters in model-based RL, the algorithms demand for efficient exploration, for which various approaches exist that are optimistic in face of uncertainty. An example is the E^3 algorithm [43] which explicitly chooses between exploiting the known part of the MDP and optimally reaching the part that needs to be explored. Its exploration is based on *balanced wandering*, which means that the agent will prefer the action that has been executed the least from its current state (breaking ties randomly) until the state has been visited a sufficient number of times. Another algorithm is the R-Max algorithm [9], which has a built-in mechanism for switching between exploitation and exploration and differs in its exploration by letting the agent make the assumption that a maximum possible reward can be obtained from unexplored state-action pairs.

Another issue that should be taken care of in model-based RL is when and how to update the MDP and its plan based on acquired experience. A simple approach would be to update the plan on each model update applying a well-known algorithm like VI, although this is computationally expensive. To gain efficiency, frameworks like RTMBA [38] or DYNA and its various extensions [70], aim to update their plans more intelligently (e.g., by updating value functions only for a subset of state-action pairs). A different approach could be to limit exhaustive exploration of states in the first place, exemplified by the RL-DT algorithm proposed in [37], which generalizes the effects of actions across states. This, however, is particularly useful for these domains that require sample-efficient learning and for which the available time for exploration is limited.

3.2.2. Active Reinforcement Learning

An appealing approach to overcome inaccuracies of offline learned probabilistic models is to use them as prior models for the model-based RL algorithms shown in Section 3.2.1. In [26] an approach referred to as *Active Reinforcement Learning* is presented which uses a provided MDP specification as prior knowledge for exploration through model-based RL. Rather than using this MDP for planning, its model parameters are used as a blueprint for exploration in the actual environment. The algorithm guides the exploration by selecting these actions for which is known that the 'prior' MDP is most sensitive to changes in the corresponding transition probabilities and rewards.

3.2.3. Bayesian Reinforcement Learning

Considerable work has been conducted on investigating the application of Bayesian methods to RL motivated by their close relation to decision theory for making decisions under uncertainty. In *Bayesian Reinforcement Learning (BRL)* [30] the idea is to learn a model of the environment by considering the unknown parameters of the model as random variables. Over these random variables a prior distribution $p(\theta)$ is defined corresponding to the prior beliefs over the unknown parameters θ of the model. A common choice for discrete-state and action MDPs is a multinomial *Dirichlet* prior, which is parameterized by a count vector $\Phi = (\phi_1, \dots, \phi_k)$ which defines the number of observations of all possible transitions. Then, as evidence is gathered, the posterior on θ (which quantifies the uncertainty over the parameters) is updated

Algorithm 7 Model-Based Bayesian Reinforcement Learning

Require: Initial MDP $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$
1: Initialize prior distribution(s) over unknown parameter(s)
2: **repeat**
3: Select action $a \in A$ based on the posterior distribution(s)
4: Execute action a
5: Record observed reward and state
6: Update posterior of unknown parameters based on observations
7: **until** Agent terminates

based on Bayes' rule from which an estimation of the model parameters can be obtained. A framework that is accordingly used in model-based BRL is the *Bayes-Adaptive MDP* [34], which augments the original state space $\mathcal{S}$, transition function δ and reward function R by jointly combining them with the posterior's hyperparameters Φ. A general formulation of the steps of model-based BRL is shown in Algorithm 7 [60].

A major drawback of model-based BRL is that obtaining exact solutions appears computationally intractable. Nonetheless various approximate BRL methods have been devised to provide feasible solutions (such as Bayesian DP, BOSS and Monte-Carlo tree search approaches), of which a majority is summarized in [30] for the interested reader. Still, a major challenge is scaling BRL methods to large-scale MDPs and models with continuous spaces.

3.3. MDP Models with Transition Probability Uncertainty

Another method to overcome the infeasibility of obtaining completely accurate specifications of transition probabilities is to extend the models and planning algorithms to take imprecise probabilities into account. Over the years various frameworks have been introduced that utilize this idea, such as the *MDP with Imprecise Probabilities (MDP-IP)* first introduced in [66] which describes the transition probabilities as a set of linear inequalities to represent incomplete, ambiguous or conflicting beliefs over these probabilities. That is, in this framework the probability of moving from a state s_i to s_j with action a could, for instance, be expressed by a variable p_{ij} s.t. a constraint $0 \leq p_{ij} \leq 0.5$. An illustrative example of a possible application of such a framework would be that of MDPs for traffic light control, for which the estimation of lane-turning probabilities for all traffic lanes is problematic, especially as they may be different over the day or throughout the year [23]. For these type of domains in particular, the MDP-IP framework was devised to account for strict uncertainty in the transition function. The framework also allows one to find plans under either optimistic or pessimistic assumptions about the true transition function. However, while the traditional MDP-IP framework demands solution techniques that are notably time-expensive, in practice factored MDP-IP representations are used for efficient planning, for instance, by synchronous dynamic programming based on parameterized algebraic decision diagrams (PADDs) [23] or RTDP solutions [22].

An alternative framework is the *Bounded-parameter Markov Decision Process (BMDP)* where a range of the transition probabilities and rewards are known, such that it can be viewed as a set of possible MDPs. In fact, BMDPs are a special case of MDP-IPs where the attainable probabilities can be defined by a discrete set rather than linear constraints, but where ranges can also be defined on the rewards. When it is possible to restrict the parameters of the MDP at intervals as is done in the BMDP framework, the algorithms that exist for BMDPs exploit the restricted structure of the parameters more efficiently than MDP-IPs, because the MDP-IP solutions demand LP approaches in each iteration. There exist solution techniques for BMDPs that can efficiently learn optimal policies, such as *interval VI* [31] or robust versions of RTDP [11].

4

Model Optimization Framework

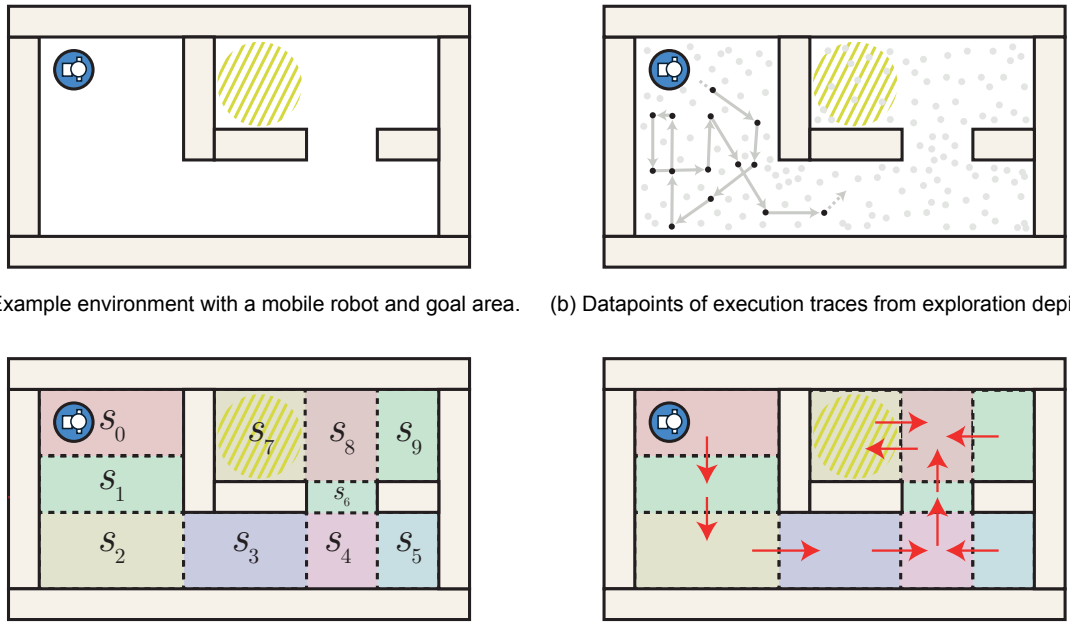
Our interest in this thesis lies in the development of probabilistic models for systems whose dynamics are stochastic. The objective is to use a dataset of execution traces of a system to attain an accurate system representation, which can be used to obtain plans for an agent's decision making in a system that involves uncertainty. In this chapter we describe our proposed framework that applies learning algorithms to obtain probabilistic models in the form of MDPs and optimizes for a performance-maximizing model by sampling multiple hyperparameter settings for these learning algorithms.

First off, in Section 4.1 we define the problem at hand that the proposed framework aims to deal with. Then, in Section 4.2, the application of mobile robot navigation is introduced, which is one of the possible applications of the framework and is used as a running example in this chapter. This application is also used for evaluating the proposed framework through the experiments that were carried out and are documented in Chapter 5. Next, Section 4.3 describes the prerequisites for the dataset serving as the input of the proposed model optimization framework. Subsequently, in Section 4.4, a detailed description of the learning and optimization routine is provided which forms our *base framework* for learning optimal MDPs. Finally, in Section 4.5, we define our extension of this base framework in the form of a *multi-phase framework* consisting of three phases whose aim is to further improve on the effectiveness of the framework.

4.1. Problem Statement

As mentioned in Chapter 1, the problem this thesis is concerned with is the development of probabilistic models in the form of MDPs to be used for obtaining plans for automated control of systems that involve uncertainty. The development of such probabilistic models tends to be a difficult and time-costly process for a human designer. Therefore, an appealing idea is to automate the model development process by applying learning algorithms on a dataset of execution traces of the system under consideration. However, to achieve effective planning in a real-world environment, the models that are learned should accurately reflect the dynamics of the system, such that derived plans account for the uncertainty that is present in the execution of actions, observations and exogenous events. Although algorithms for learning MDPs from a dataset do already exist, the state of the art lacks a method of setting the hyperparameters θ of these algorithms such that a model is retrieved that best reflects the underlying system and maximizes the performance in its execution of derived plans. Besides, although Reinforcement Learning (RL) allows us to bypass the development of a model, it often turns out to be quite slow for finding optimal policies in complex environments. More formally, summarizing the above, the problem can be formulated as follows:

Problem Statement. Given a set E of execution traces describing the dynamics of a system that involves uncertainty, an MDP $\mathcal{M}$ needs to be developed, utilizing learning algorithms parameterized by a set of hyperparameters θ configured in such way to maximize the performance yielded by following the policies computed from this model $\mathcal{M}$ in a real-world setting.



(a) Example environment with a mobile robot and goal area.

(b) Datapoints of execution traces from exploration depicted.

(c) State space of a learned MDP depicted.

(d) Corresponding optimal policy for the MDP.

Figure 4.1: Planning for the navigation of a mobile robot by learning MDPs from data about the environment.

All in all, the problem we deal with is that of automating the task of learning a performance-maximizing MDP that generalizes well over multiple tasks for a system involving uncertainty. This involves the need for a proper performance measure to express the value of a learned MDP and allow for a fair comparison of different MDPs. That is, depending on the parameter settings of the learning algorithm used, an MDP may yield low performance in its execution for two major reasons. First of all, the learned model may be too complicated for the amount of available data, which leads to model overfitting. On the other hand, one could acquire a model that is too simple to explain the available data and so does not accurately reflect the underlying system in that situation. Therefore, the parameters of the learning algorithm should be properly adjusted in order to obtain accurate probabilistic models. Finally and most importantly, a solution to this problem should take into account the costs of learning and evaluating these system models.

4.2. Application to Mobile Robot Navigation

In this chapter a model optimization framework is proposed that could be employed for various planning problems involving uncertainty. To illustrate the type of problems this framework is aimed at and what comprises a probabilistic model for such problems, we use its application to path planning in mobile robot navigation as a running example throughout this chapter. The problem statement for this application is as follows: a mobile robot needs to be autonomously controlled by an agent in such way to navigate from one location to another in some world, say, an office environment, as fast as possible. This problem domain is particularly suited as the robots tend to operate under significant uncertainty in their actions (e.g., slipping) and observations (e.g., sensor noise). Although there are other potential applications as described in Chapter 1, this particular application can be used to illustrate the procedure of learning MDPs from data and obtaining plans accordingly quite well.

For example, let us consider the scenario in which the robot needs to operate in an environment as depicted in Figure 4.1a. For now let us assume, the action space A is defined as a finite set of movements in the cardinal and inter-cardinal directions, such that:

$$A = \{\mathtt{S}, \mathtt{SE}, \mathtt{E}, \mathtt{NE}, \mathtt{N}, \mathtt{NW}, \mathtt{W}, \mathtt{SW}\}$$

Before our model optimization framework is applied to learn MDPs for the system, the data about the environment could have been obtained in the form of execution traces that consist of logs of the robot's pose and the (next) action it is about to take. This could then give a set of datapoints, as depicted in Figure 4.1b, which can be used to learn the states and transition probabilities of an MDP. Quantizing this dataset of observed robot poses into 10 symbols (e.g., using k-Means clustering), might give us the state space $\mathcal{S} = \{s_0, \dots s_9\}$ depicted in Figure 4.1c. Accordingly, based on such a quantization into a state space $\mathcal{S}$, one can learn a transition function $\delta : \mathcal{S} \times A \times \mathcal{S} \mapsto [0, 1]$ for an MDP applying the maximum-likelihood estimation as described in Section 2.1.3. Then, say that, for instance, we define a task for the robot as moving from an initial position in state $s_0 \in \mathcal{S}$ to a goal area corresponding to the state $s_7 \in \mathcal{S}$ as depicted. This would make us define a reward function $R : \mathcal{S} \times A \times \mathcal{S} \mapsto \mathbb{R}$ for our MDP such that only a reward can be obtained from the 'goal' state s_7, i.e.:

$$R(s, a, s') = \begin{cases} 1 & \text{if } s' = s_7 \wedge s \neq s' \\ 0 & \text{otherwise} \end{cases}$$

Putting the earlier defined components together, an MDP $\mathcal{M}$ for this particular task is $\mathcal{M} = (\mathcal{S}, s_0, A, \theta, R)$. Accordingly, an optimal policy π^*, defining the most appropriate action from any state, is quite straightforward for this simple MDP and is depicted in Figure 4.1d.

In this application the value of a learned MDP needs to depend on how fast the robot would reach its goal location for the various tasks it is expected to perform. It should be noted that a proper assessment of model value should be made based on the performance in simulations and the real world as there may exist discrepancies between states in the model and the actual state of the system in its execution. In the following sections we discuss how the proposed framework takes these aspects into account so that on convergence an MDP is obtained that reflects the real world and generalizes over multiple tasks quite well.

4.3. Dataset Prerequisites

Prior to describing the proposed framework for the problem as specified in Section 4.1, we are obliged to specify the expected format of the datasets to learn MDPs from. A self-evident approach is making use of a dataset generated from execution traces, in which the system under consideration logged a set of readings/observations describing the system state accompanied by the actions selected in between at a fixed time interval. One should note that this means that the action space A and time-step t, that will be used for any MDP for the system, should already have been established at this point. When adding the auxiliary condition of needing to learn fully observable MDPs, the need arises for the observations in these execution traces to be usable for distinguishing between the different system states.

For our running example of mobile robot navigation such a dataset may be obtained by executing the robot in its real-world environment following a random action policy (or a policy that exploits any further prior knowledge). That is, one could choose to let the robot record its location and orientation in terms of its sensor readings which are logged at a fixed time interval. In the exploration, the robot would then start from some position in the environment, choose an action (e.g., driving forward in some direction) at random and execute it according to the fixed time-step. Then, what will be logged are the robot's sensor readings and selected action at each stage. Recording the action at each stage is appropriate as it allows us to observe the effect of the actions and identify actions that do not affect the robot's state (e.g., being unable to move because of a wall or other obstacle close to the robot).

Further, one should ensure that the dataset obtained from the exploration at least holds observation-action pairs for the area of the observation/state space of interest for the tasks the system is expected to perform. For our running example, this would be covering most of the reachable parts of the environment and trying out actions in each area multiple times. Clearly, one could decide to exclude or limit exploration for those areas that are deemed not of interests for the tasks the system is expected to perform. All in all, the learning algorithm requires one to fix an action space A and time-step t and gather a dataset consisting of a sufficient number of observation-action pairs to describe the dynamics of the system under consideration.

Algorithm 8 MDP-Optimization Base Framework

Require: Set of execution traces E, Parameter space Θ, Set of tasks T, Action space A, Time-step t, Discount factor $\gamma \in [0, 1)$, Weight factor $\beta \in [0, 1]$, Acquisition function $a : \Theta \mapsto \mathbb{R}$
Ensure: MDP $\mathcal{M}_{\text{max}}$, Evidence set $\mathcal{D}$

$\mathcal{M}_{\text{max}} \leftarrow$ NULL
$V_{\mathcal{M}_{\text{max}}} \leftarrow 0$
$\mathcal{D} \leftarrow \emptyset$ ▷ The evidence set storing samples and observations
$gp \leftarrow GP(0, K)$ ▷ Zero-mean GP with covariance K
Sample $\theta \in \Theta$ at random
repeat
  $\mathcal{M}, V_{\mathcal{M}} \leftarrow$ LearningStep(θ)
  if $V_{\mathcal{M}} > V_{\mathcal{M}_{\text{max}}}$ **then**
    $V_{\mathcal{M}_{\text{max}}} \leftarrow V_{\mathcal{M}}$
    $\mathcal{M}_{\text{max}} \leftarrow \mathcal{M}$
  end if
  $\theta \leftarrow$ OptimizationStep($\theta, V_{\mathcal{M}}$)
until convergence
return $\mathcal{M}_{\text{max}}, \mathcal{D}$

function LearningStep(θ)
  $\mathcal{S} \leftarrow$ Quantization(E, θ)
  $\delta \leftarrow$ LearnTransitionProbs($E, \mathcal{S}, A, \theta$) ▷ For instance, a likelihood maximization approach
  $V_{DTP} \leftarrow 0$
  $V_{SIM} \leftarrow 0$
  for all $(start, goal) \in T$ **do**
    $\{s_0, s_g\} \leftarrow$ ToStates($\{start, goal\}$)
    $R \leftarrow$ ToRewardFunction(s_g)
    $\mathcal{M} \leftarrow (\mathcal{S}, s_0, A, \delta, R)$
    $V, \pi \leftarrow$ DTPPlanning($\mathcal{M}$) ▷ Solve for a policy π using a DTP algorithm, such as VI
    $V_{DTP} \leftarrow V_{DTP} + V[s_0]$
    if $\beta < 1$ **then**
      task_status, $steps \leftarrow$ RunSim($\mathcal{M}, \pi$)
      if task_status = success **then**
        $V_{SIM} \leftarrow V_{SIM} + \gamma^{steps} \cdot R[s_g]$
      end if
    end if
  end for
  $V_{\mathcal{M}} \leftarrow \beta \cdot \frac{V_{DTP}}{|T|} + (1 - \beta) \cdot \frac{V_{SIM}}{|T|}$
  return $\mathcal{M}, V_{\mathcal{M}}$
end function

function OptimizationStep($\theta, V_{\mathcal{M}}$)
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\theta, V_{\mathcal{M}})\}$
  gp.fit($\mathcal{D}$) ▷ Fit a GP given $\mathcal{D}$, maximizing the log-marginal likelihood
  return $\arg\max_{x \in \Theta} a(x|\mathcal{D}, gp)$ ▷ Suggest a sample θ to evaluate next
end function

4.4. Base Framework

The learning and optimization routine described in this section forms the *base framework* for the problem dealt with. This routine, schematically depicted in Figure 4.2, aims to properly adjust the hyperparameters θ of the employed model learning algorithm by posing it as an optimization task. The objective is then to obtain an MDP that maximizes the performance of executing the plans that are derived from it. In the remainder of this section we explain the working of this routine of which the corresponding pseudocode is presented in Algorithm 8.

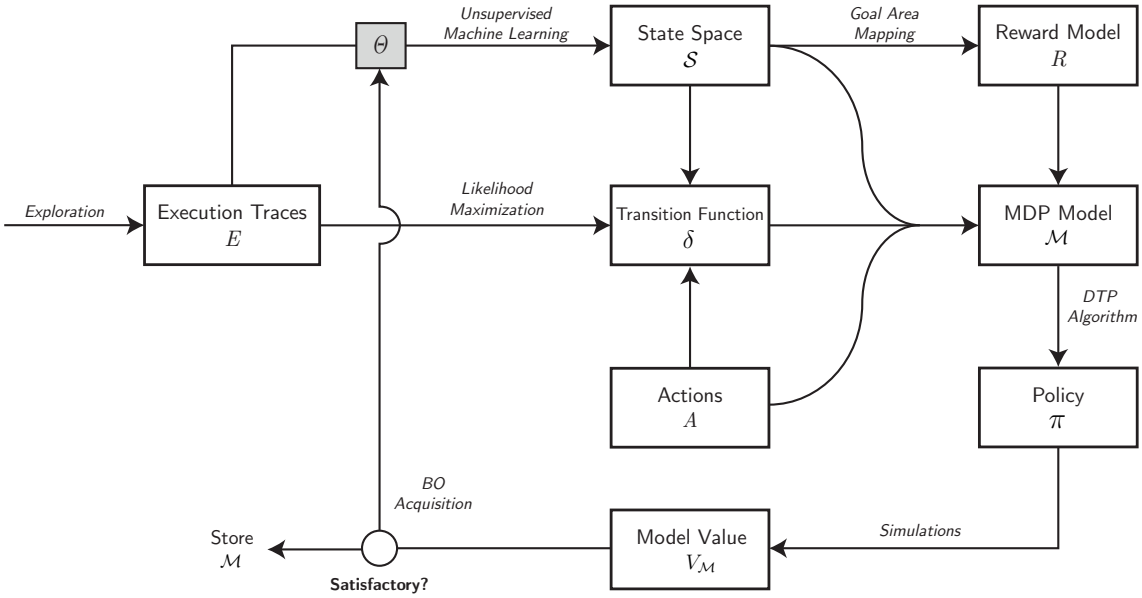


Figure 4.2: Schematic diagram of the model learning and optimization routine.

4.4.1. Learning Step

The routine can be viewed as consisting of two subsequent steps which are repeated until an MDP model that yields satisfactory performance is obtained. The first of these two main steps will be referred to as the *learning step*, in which an MDP model is learned and its value is assessed.

This learning step starts off with a dataset of execution traces E obtained beforehand as described in Section 4.3. As we can see in Algorithm 8, Line 15, the step is parameterized by θ, which represents the parameter setting to be used for the employed model learning algorithm. Given a parameter setting θ, a state space $\mathcal{S}$ is acquired by quantizing the execution traces E using an unsupervised machine learning algorithm. Subsequently, a transition function δ is acquired by applying a known model learning algorithm (such as one of those described in Section 3.1), given the dataset E, state space $\mathcal{S}$ and possible actions A. The tasks to perform are each accordingly mapped to a reward function R over the acquired state space $\mathcal{S}$ (or alternatively over state-action pairs or state-action-state triplets). The state space $\mathcal{S}$, transition function δ, action set A, reward function R and a configurable initial state s_0 are then combined into an MDP $\mathcal{M} = (\mathcal{S}, s_0, A, \delta, R)$.

To assess the performance yielded by $\mathcal{M}$, the MDP is solved by applying known DTP algorithms like VI or PI to obtain policies for a set of tasks the system is expected to perform. The value of the model $V_{\mathcal{M}}$ is then determined by checking how well the system is expected to perform by execution according to the derived policies (e.g., based on simulations and/or the value function obtained by a DTP algorithm). The sections below describes each of the components of this algorithm in more detail and most importantly defines the performance measure that is used for assessing the value of learned MDPs.

Action Space and Time-Step

The action space A of the MDP to be learned is one of the input parameters for the framework and should match the actions used in obtaining the set E of execution traces. Similarly, the time-step t is an input parameter as well, which specifies the time in between stages in which the agent selects an action, again matching the time-step used in obtaining the set E of execution traces. In our running example, one possibility is to define the action space as the set of movements in the cardinal and inter-cardinal directions as seen earlier in Section 4.2.

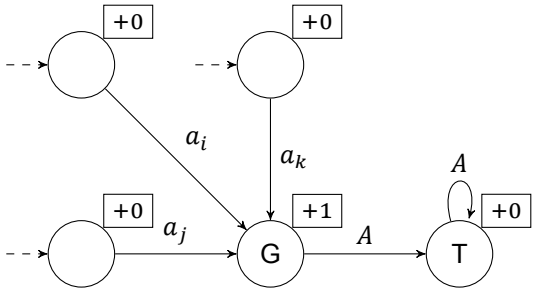


Figure 4.3: One-time reward in goal state G in an MDP realized by an absorbing state T.

The time-step t for the mobile robot would then correspond to the period of time it moves in one of these directions after it has selected an action $a \in A$.

State Space

From the provided set of execution traces E, a state space $\mathcal{S}$ is learned based on the current hyperparameter setting θ in the learning step. For instance, in our running example, a k-Means clustering can be applied on, for instance, coordinates in the execution traces E describing the reachable locations of the robot, where k is specified by the setting of θ.

Transition Function

In the learning step the transition function δ is obtained by applying a known model learning algorithm, for which a likelihood maximization approach may be selected for learning the transition probabilities of a fully observable MDP. The transition function is defined for the provided action space A and earlier acquired state space $\mathcal{S}$ and approximated from the transitions in the set E of execution traces.

Tasks and Reward Functions

The goal for the class of problems of interest is to obtain a well-generalizing model that performs well on multiple different tasks. Therefore, an estimate of the value of a model is assessed over a set T of tasks provided as input to the framework, where each task is presented as the combination of a starting and goal configuration.

The starting configurations are each mapped to corresponding states in the MDP, matching the quantization to state space $\mathcal{S}$, so that one can define $\mathcal{S}_0 \subseteq \mathcal{S}$ as a set of initial states. Similarly, the goals are each mapped to states, such that one can define $\mathcal{S}_G \subseteq \mathcal{S}$ as a set of goals. One problem that emerges, however, is that there might not be a one-to-one correspondence between learned and true (goal) states, which leads to the possibility of small state spaces yielding high value while the goal states might not map well to the true goal states. To take care of this, we introduce a discrepancy factor $\xi \in [0,1]$, which corresponds to the fraction of entries from the set E within the goal state that map to the true goal.

Let us then define R_G as the set of reward functions R_i for each $i \in \mathcal{S}_G$ in which a one-time reward (which is realized as depicted in Figure 4.3) is received only in goal state i. Note that as such, an equivalent formulation can be given as a *stochastic shortest path* problem [35] for the resulting MDPs. The reason the reward needs to be only obtainable once is to ensure that the expressions in Equation 4.2 and Equation 4.3, which are used to estimate the performance yielded by an MDP, lie in the same range and can be used together as discussed in the next section. The reward function $R_i : \mathcal{S} \times A \times \mathcal{S} \mapsto \mathbb{R}$ for some task with goal state $i \in \mathcal{S}_G$ is then defined as follows:

$$R_i(s,a,s') = \begin{cases} \xi & \text{if } s' = i \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Accordingly one can define $T_{\mathcal{M}} = \mathcal{S}_0 \times R_G$ as the set of tasks, in the form of all pairs of initial states and reward functions, over which the performance is assessed. The next section discusses how the value of a model $\mathcal{M}$ is assessed in the learning step over these tasks.

Performance Measure

Posing the learning of probabilistic models as an optimization task requires us to define a mapping of MDPs to a performance measure that can be used to fairly compare different models. As discussed earlier, one can express the performance of an MDP in terms of how well the agent performs tasks following the policies that are derived from the model. Therefore, one option is to make use of the value function V obtained by a DTP algorithm, such as VI or PI. Based on such a value function, an expression of how well the agent is expected to perform some task $t = (s_0, R) \in T_{\mathcal{M}}$ is:

$$V_{DTP,t=(s_0,R)} = V[s_0] \tag{4.2}$$

where $s_0 \in \mathcal{S}$ is the initial state and R the reward function of the MDP for the task.

However, although the DTP algorithm may yield high value for the task, it might be that the agent will not perform well following the policy in the real world. To better assess the performance, one could execute simulations in which the agent follows the policy derived from the MDP model. Although performing simulations is more cost-expensive, it yields a better approximation of how well the agent executes a task following a policy obtained from the model. Accordingly, let us express the performance on some task in simulations as:

$$V_{SIM,t=(s_0,R)} = \gamma^n \cdot R[i] \tag{4.3}$$

where $s_0 \in \mathcal{S}$ is the initial state, R the reward function, γ the discount factor, $i \in \mathcal{S}$ the goal state and n the number of steps taken to reach the goal.

Combining the expressions of Equation 4.2 and Equation 4.3 yields the following expression of the performance on some task $t \in T_{\mathcal{M}}$:

$$\beta \cdot V_{DTP,t=(s_0,R)} + (1-\beta) \cdot V_{SIM,t=(s_0,R)} \tag{4.4}$$

where $\beta \in [0,1]$ is a parameter which specifies the relative weight of $V_{DTP,t}$ against that of $V_{SIM,t}$ and where s_0 and R are the initial state and reward function for the task respectively.
Putting this all together, the performance of an MDP $\mathcal{M}$ can in this way be expressed as:

$$V_{\mathcal{M}} = \frac{\sum_{t \in T_{\mathcal{M}}} \beta \cdot V_{DTP,t} + (1-\beta) \cdot V_{SIM,t}}{|T_{\mathcal{M}}|} \tag{4.5}$$

where $T_{\mathcal{M}}$ is the set consisting of the pairs of initial states and reward functions of the tasks over which the performance is assessed.

4.4.2. Optimization Step

The second main step of the routine will be referred to as the *optimization step*, in which a parameter setting θ is selected for the next learning step in such way that yielded performance converges to a global maximum as quickly as possible. As initially there is no knowledge of the performance given the settings of the learning parameter(s) θ, in the first few routine-iterations parameter settings are selected at random from the designer-specified domain Θ (i.e., the parameter space). Together with the value $V_{\mathcal{M}}$ of the corresponding model obtained in the learning step for each of these parameters θ, they are stored in an evidence set $\mathcal{D}$. The objective function f in our case is the `LearningStep` function which maps parameter settings $\theta \in \Theta$ to real-valued performance yielded by MDPs.

As discussed earlier in Section 2.2.1, the objective functions for the class of SDM problems tend to be expensive to evaluate as proper estimations can only be made through time-costly simulations. Therefore, Bayesian Optimization emerges as an attractive method to find a global maximizer $\theta^* \in \Theta$ for our objective function. After each iteration the evidence set $\mathcal{D}$ is augmented with a new observation, based on which the posterior $p(f|\mathcal{D})$ of the objective function f is updated accordingly. A new parameter setting for the next learning step is then sampled at the point with maximum utility in the selected acquisition function $a : \Theta \mapsto \mathbb{R}$. This procedure is repeated until some criterium is reached, which can be a fixed number of iterations, a fixed time or a custom stopping criterion, e.g., observing no improvement for some number of iterations.

Algorithm 9 MDP-Optimization Multi-Phase Framework (Phase 1 and 2)

Require: Set of execution traces E, Parameter space Θ, Set of tasks T, Action space A, Time-step t, Discount factor $\gamma \in [0, 1)$, Acquisition function $a_1 : \Theta \mapsto \mathbb{R}$
Ensure: MDP $\mathcal{M}_{\text{max}}$

$\mathcal{D}_1, gp_1 \leftarrow$ Phase1()
$\mathcal{M}_{\text{max}} \leftarrow$ Phase2()

function Phase1
  $\beta \leftarrow 1.0$ ▷ Compute $V_{\mathcal{M}}$ based on the value function, i.e., $V_{\mathcal{M}} = V_{DTP}$
  $\mathcal{D} \leftarrow \emptyset$
  $gp \leftarrow GP(0, K)$
  repeat
    $\mathcal{M}, V_{\mathcal{M}} \leftarrow$ LearningStep(θ)
    $\theta \leftarrow$ OptimizationStep($\theta, V_{\mathcal{M}}$)
  until convergence
  return $\mathcal{D}, gp$
end function

function Phase2
  $\beta \leftarrow 0.0$ ▷ Compute $V_{\mathcal{M}}$ based on simulations, i.e., $V_{\mathcal{M}} = V_{SIM}$
  $\mathcal{D}_2 \leftarrow \emptyset$
  $gp_2 \leftarrow GP(0, K)$
  $\mathcal{M}_{\text{max}} \leftarrow$ NULL
  $V_{\mathcal{M}_{\text{max}}} \leftarrow 0.0$
  $\theta \leftarrow \arg\max_{x \in \Theta}$ EI$(x|\mathcal{D}_1, gp_1)$
  $\alpha \leftarrow 0.5$
  repeat
    $\mathcal{M}, V_{\mathcal{M}} \leftarrow$ LearningStep(θ)
    if $V_{\mathcal{M}} > V_{\mathcal{M}_{\text{max}}}$ **then**
      $V_{\mathcal{M}_{\text{max}}} \leftarrow V_{\mathcal{M}}$
      $\mathcal{M}_{\text{max}} \leftarrow \mathcal{M}$
    end if
    $\theta \leftarrow$ OptimizationStep2($\theta, V_{\mathcal{M}}, \alpha$)
    $\alpha \leftarrow \alpha/2$
  until convergence
  return $\mathcal{M}_2$
end function

function OptimizationStep2($\theta, V_{\mathcal{M}}, \alpha$) ▷ The α parameter weighs the EI of the first phase.
  $\mathcal{D}_2 \leftarrow \mathcal{D}_2 \cup \{(\theta, V_{\mathcal{M}})\}$
  gp_2.fit($\mathcal{D}_2$)
  return $\arg\max_{x \in \Theta} \alpha \cdot$ EI$(x|\mathcal{D}_1, gp_1) + (1 - \alpha) \cdot$ EI$(x|\mathcal{D}_2, gp_2)$
end function

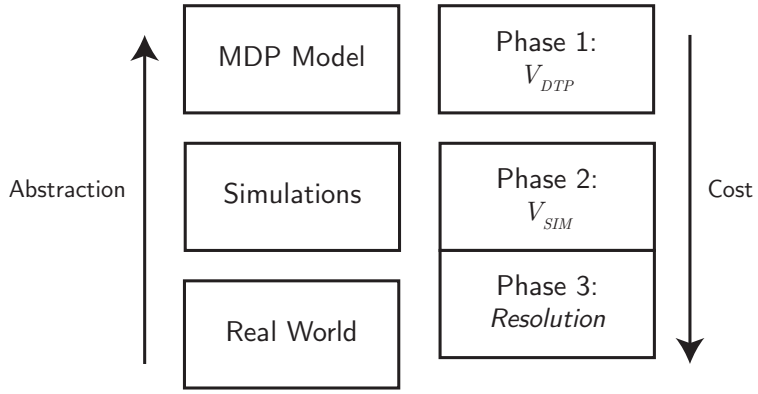


Figure 4.4: Correspondence of the phases of the model optimization framework to the different levels of abstraction.

4.5. Multi-Phase Framework

To make the solution more cost-effective, the framework is extended by defining three phases with the aim of exploiting the lower cost of computing the value function for MDPs in comparison to the cost of executing plans in simulations or the real world. That is, assessing performance solely based on learned MDP models is less cost-expensive, although the model might not accurately reflect the real world as its abstracts over low-level details. This section defines the phases of the extension, of which each subsequent phase better reflects the real world, but on the other hand are accompanied by higher costs as is depicted in Figure 4.4.

4.5.1. Phase 1: Value Function Pre-Processing

In the first phase the performance is assessed solely based on the value functions derived by the DTP algorithm used for planning. This means the β parameter is set to $\beta = 1.0$ such that the performance $V_{\mathcal{M}}$ of MDP $\mathcal{M}$ is expressed only in terms of V_{DTP} and so no simulations are performed in this phase. Therefore, this first phase is relatively cost-cheap, and although it abstracts from the real-world the most, it may be used to identify the interesting area of the parameter space. Hence, the goal of this first phase is to narrow down the search space to a subspace of learning parameters more likely to yield high performance.

The block diagram in Figure 4.5 depicts the main steps of this first phase, whose pseudocode is presented in Algorithm 9, starting from Line 3. In each iteration of this first phase the execution traces E from the exploration are used to obtain an MDP by applying the learning algorithms, given a parameter setting θ. The value of the learned model $\mathcal{M}$ is then assessed based only on the value function V for a set of tasks $T_{\mathcal{M}}$ the system is expected to perform. The value of model $\mathcal{M}$ is thus computed as:

$$V_{\mathcal{M}} = \frac{\sum_{t \in T_{\mathcal{M}}} V_{DTP,t}}{|T_{\mathcal{M}}|}$$

where each task $t \in T_{\mathcal{M}}$ is defined in terms of an initial state and reward function.

Based on the gathered evidence new parameter settings θ are sampled iteratively, which are those settings for which the utility is the highest in the acquisition function a_1 used for BO. This continues until a stopping condition has been met, where in this first phase stopping after a fixed number of iterations should be acceptable, as the goal is only to narrow down the search space. At the end of the first phase, the resulting posterior GP distribution gp_1 is taken and used for the acquisition in the next optimization phase.

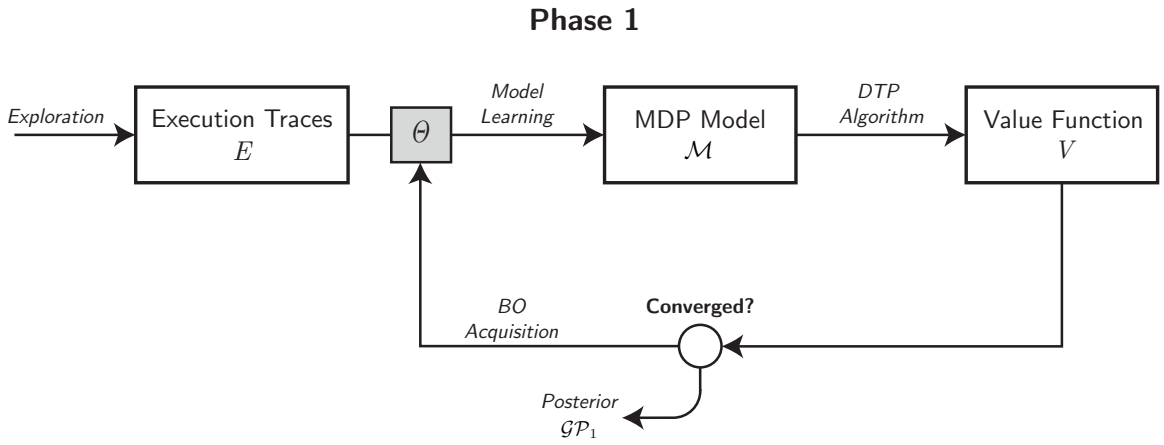


Figure 4.5: Block diagram showing the steps of the first phase of the multi-phase optimization framework. This phase develops a distribution which reflects the interesting area of the parameter space solely based on the value functions of MDPs.

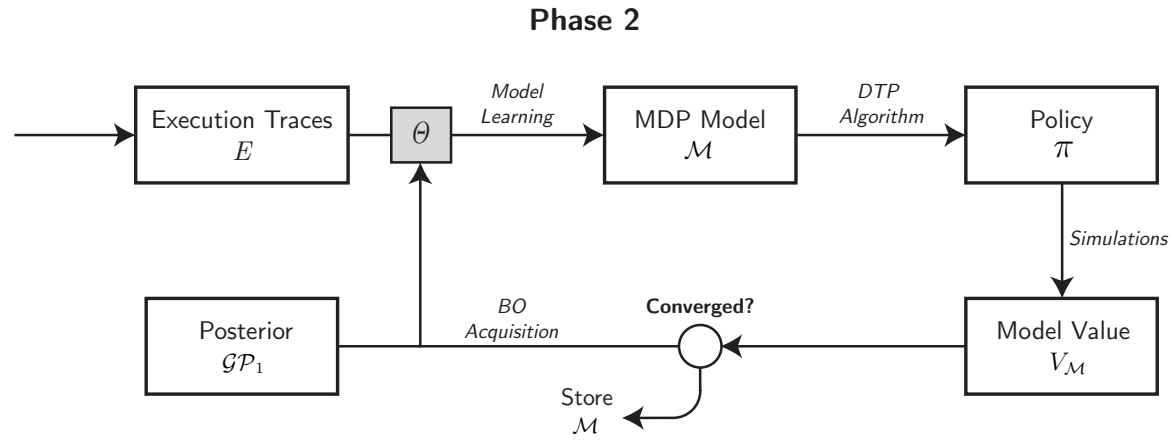


Figure 4.6: Block diagram showing the steps of the second phase of the multi-phase optimization framework. This phase uses the distribution $\mathcal{GP}_1$ learned in the first phase in the acquisition in the optimization for an MDP which maximizes performance in simulations.

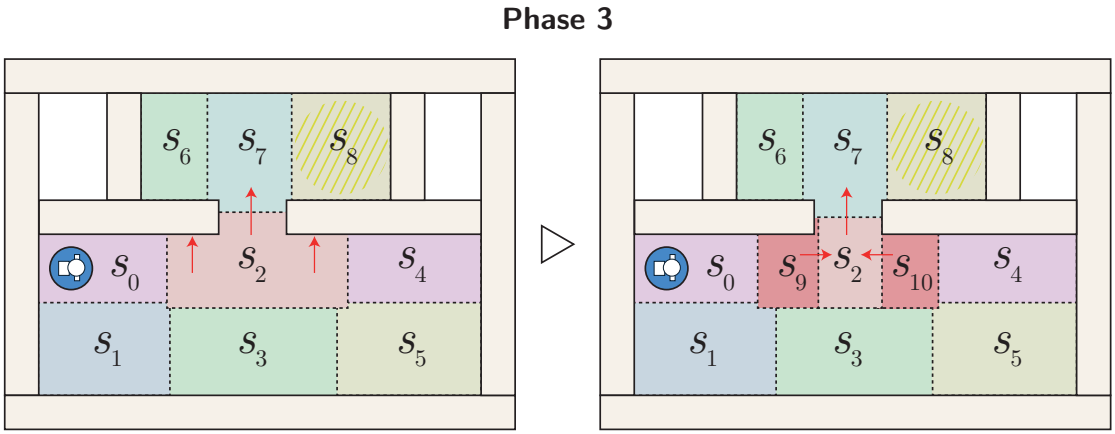


Figure 4.7: An illustration of the goal of the third phase in the context of mobile robot navigation. In the navigation towards the goal state s_8, the robot might get stuck in state s_2 following the optimal policy. Therefore, higher resolution is needed in this area of the state space.

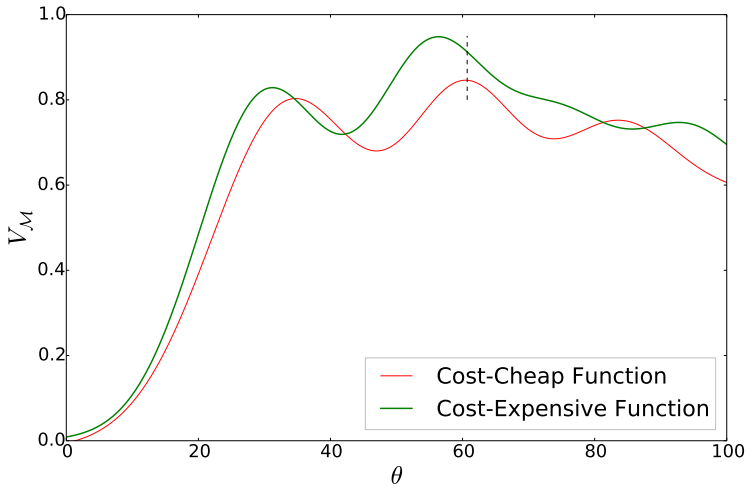


Figure 4.8: An illustration of the idea behind the second phase of the multi-phase framework. The plot shows two artificial functions, one being much cheaper to evaluate compared to the other. As the maxima are close, starting to sample around the maximizer of the cheap function may steer the optimization towards the maximum of the expensive function.

4.5.2. Phase 2: Simulation-Based Optimization

In the second phase the performance is assessed solely based on simulations of the system under consideration. This means that, instead, the β parameter is now fixed at $\beta = 0.0$ such that the performance $V_{\mathcal{M}}$ of MDP $\mathcal{M}$ is expressed only in terms of V_{SIM}. Therefore, the second phase is much more cost-expensive and so a desirable aim is to minimize the number of evaluations needed. In an attempt to do so, the second phase exploits the knowledge gathered in the first phase by utilizing the GP gp_1 in the acquisition of new samples in the optimization process. The idea is thus to first sample from those parts of the parameter space Θ expected to yield high value according to the observations in the first phase.

The block diagram in Figure 4.6 depicts the main steps of this second phase, whose pseudocode is presented in Algorithm 9, starting from Line 13. The first thing that differs from the previous phase is that, in each iteration, the value of the learned model $\mathcal{M}$ is assessed based on time-costly simulations. The value of model $\mathcal{M}$ is thus computed as:

$$V_{\mathcal{M}} = \frac{\sum_{t \in T_{\mathcal{M}}} V_{SIM,t}}{|T_{\mathcal{M}}|}$$

where each task $t \in T_{\mathcal{M}}$ is defined in terms of an initial state and reward function.

The main difference, however, lies in how new parameter settings θ are sampled in each iteration in this phase. The acquisition in this phase is based on the *integrated acquisition function*, defined in Equation 2.12, as it uses the GP from the prior phase and the GP posterior based on the observations of this phase. That is, the acquisition function is defined as:

$$\alpha \cdot \mathrm{EI}(x|\mathcal{D}_1, gp_1) + (1 - \alpha) \cdot \mathrm{EI}(x|\mathcal{D}_2, gp_2)$$

where $\mathcal{D}_1$ and $\mathcal{D}_2$ are the evidence sets, and gp_1 and gp_2 the GP approximations of the first and second phase respectively, with the expected improvement EI at $x \in \Theta$ weighed by α.

The knowledge obtained in the first phase is utilized by weighing the expected improvement more heavily for selecting the first few samples (i.e., setting α to a high value). Over time, the influence of the first phase decreases by lowering the value of α as more observations are made. Our hypothesis is that, if the global maximum in the first phase lies close to that of the second phase (as illustrated in Figure 4.8), a performance-maximizing MDP may be found faster, as the sampling is initially steered by the approximation of the first phase.

Algorithm 10 MDP Optimization Multi-Phase Framework (Phase 3)

Require: MDP $\mathcal{M}_{\text{in}} = (\mathcal{S}, s_0, A, \delta, R)$, Set of tasks T, Time-step t, Discount factor $\gamma \in [0, 1)$
Ensure: MDP $\mathcal{M}_{\text{out}}$

function Phase3
  for all $(start, goal) \in T$ **do**
    task_status, $steps \leftarrow$ RunTask$(start, goal)$
    $n \leftarrow 1$
    while task_status $=$ `stuck` **do**
      Let s_f be the state in which the system got stuck
      $E_{SIM} \leftarrow$ GatherEvidence(s_f)
      Compute n sub-clusters within s_f using E_{SIM} as training set
      Learn a new transition function δ_f for the new states using E_{SIM}
      task_status, $steps \leftarrow$ RunTask$(start, goal)$
      $n \leftarrow n + 1$ ▷ Increment n only if the agent got stuck in the same state
    end while
  end for
  return $\mathcal{M}_{\text{out}} \leftarrow (\mathcal{S}_f, s_0, A, \delta_f, R)$ ▷ $\mathcal{S}_f$ being the updated state space
end function

function RunTask$(start, goal)$
  $(s_0, s_g) \leftarrow$ ToStates$(\{start, goal\})$
  $R \leftarrow$ ToRewardFunction(s_g)
  $\mathcal{M} \leftarrow (\mathcal{S}, s_0, A, \delta, R)$
  $V, \pi_t \leftarrow$ DTPPlanning$(\mathcal{M})$
  return RunSim$(\mathcal{M}, \pi_t)$ ▷ Execute the policy in simulations or the real-world
end function

function GatherExperience(s_f)
  $R \leftarrow$ ToRewardFunction(s_f)
  $\mathcal{M} \leftarrow (\mathcal{S}, s_0, A, \delta, R)$
  $V, \pi_f \leftarrow$ DTPPlanning$(\mathcal{M})$
  Let E_{SIM} be a dictionary
  for all $a \in A, s' \in \mathcal{S}$ **do**
    $E_{SIM}[a][s'] \leftarrow \emptyset$
  end for
  for all $i \in \{1, \ldots, 100\}, a \in A$ **do** ▷ Perform trials of all actions from state s_f
    Let o be the current system state observations
    Execute a and let s' be the resulting state
    $E_{SIM}[a][s'] \leftarrow E_{SIM}[a][s'] \cup \{o\}$ ▷ Store the observed transition in E_{SIM}
    while $s' \neq s_f$ **do** ▷ Move back to state s_f
      Execute $\pi_f(s')$ and update s' to the resulting state
    end while
  end for
  return E_{SIM}
end function

4.5.3. Phase 3: Model Fine Tuning

The third phase starts after the optimization process has reached its stop condition and has retrieved a performance-maximizing MDP $\mathcal{M}_{\text{max}}$. In this phase, the goal is to further improve the learned model by checking whether the transition probabilities are likely to match the real world environment. The identification of discrepancies in these probabilities happens by executing actions in those areas of the state space that are visited most often and in which the agent tends to get stuck (according to the transition probabilities) and seeing if the observed transitions yield comparable probabilities. For those areas and their corresponding states for

which mismatches are identified, higher resolution is provided to better reflect the system's dynamics in the real world.

The illustration of Figure 4.7 aims to clarify the goal of this third phase by means of an example for the context of mobile robot navigation. Although the model obtained in the previous phase might yield high performance for most of the tasks used to assess model value, a situation as illustrated in this figure might occur. That is, the robot, depicted in blue, needs to move from its current position to the goal, indicated by the yellow shaded area. For the state space depicted on the left-side however, the robot might get stuck following the optimal policy from which the need for higher resolution in this area emerges.

In Algorithm 10, a proof-of-concept solution for this problem is shown which fine-tunes an MDP $\mathcal{M}$ resulting from the previous two phases. Given a set of tasks T (possibly different from the tasks used in the prior phases), a policy π is obtained based on this model, which is executed in a simulation or real-world environment. When the agent detects it is stuck in some state s_f, it starts to gather a set of new execution traces E_{SIM} of transitions from this state. This set is augmented by iteratively selecting an action (in a round-robin fashion) and executing it from the state s_f and recording the end state s'. For the example in Figure 4.7, one entry of this dataset E_{SIM} may express that starting from a location (x, y) in state $s_f = s_2$, choosing to execute the action `N` of moving north resulted in ending up in state s_7. Then, another entry may express that starting from a location (x', y') in state $s_f = s_2$, executing the same action resulted in ending up in state s_2.

Based on the set E_{SIM}, the state s_f in which the agent got stuck is split in a number of (sub-)states by clustering the observations. The transition function of the MDP is then updated accordingly for the resulting state space. The agent then attempts to perform the task again with the updated MDP. If the agent then succeeds in executing the task, it moves on to the next task, and otherwise repeats the earlier steps with higher resolution in the clustering.

Note that this third phase is a phase that is mostly independent of the prior phases of the framework. That is, in the first two phases the goal is to find the optimal parameters θ of an MDP model learning algorithm, while the third phase is concerned with tuning the resulting MDPs. As such, this last phase is rather an additional (and optional) step with the attempt of further improving the performance yielded by these system models.

5

Experimental Setup and Results

In Chapter 4 a framework was proposed for finding an optimal MDP for planning problems that involve uncertainty given a dataset describing the dynamics of the system under consideration. The framework aims to achieve this by posing the adjustment of the parameters of model learning algorithms as an optimization task in which the yielded performance is to be maximized. The domain of mobile robot navigation, where the problem statement is to navigate a robot between locations as fast as possible, was identified as a potentially suitable application. This chapter discusses the experiments that were conducted to evaluate the framework for this application and the results that were obtained accordingly. First of all, Section 5.1 elaborates upon the setup for the experiments, discussing the relevant details on the implementation of the framework for this application and the software and other resources used. Subsequently, Section 5.2 discusses the different configurations that are used to test the framework. This, for instance, involves various combinations of learning algorithms and different settings of the weight factor β on different environments. In, Section 5.3 the results obtained for these different configurations are presented, inspected and compared to one another, after which the most notable conclusions that can be drawn are discussed.

5.1. Setup

For our experiments we made an implementation of the framework in the form of a module that can be used to learn an optimal MDP for the navigation of a mobile robot. This implementation allows the control of a mobile robot in simulations by an MDP and a corresponding policy. The model values assessed from these simulations are used to find a globally maximizing parameter settings of the learning algorithm used. In this section we will describe the implementation in detail and how it is used in our experiments to find performance-maximizing MDPs for a mobile robot in an office environment.

5.1.1. Software

In this section we discuss the software that is used in the implementation of our optimization module. An overview of the main packages used for this implementation is presented in Table 5.1. In the remainder of this section, we briefly explain for what part of the implementation each of these packages are used, and support the choices made where necessary.

Python Libraries

The module has been implemented in Python 2.7, due to its easily usable libraries for machine learning and plotting and other widely available packages, but also because of its convenient capability of interacting with the simulation software used. An overview of the main software packages used for the implementation is shown in Table 5.1.

Table 5.1: Software packages used for the implementation of the model learning framework for mobile robot navigation.

Package Name	Version	Purpose
`bayesian-optimization`	0.4.0	Bayesian optimization
`matplotlib`	1.3.1	Plotting and visualizing robot actions
`pymdptoolbox`	4.0_b3	MDP planning algorithms
`ros-indigo-strands-desktop`	0.0.14	Simulation software and environments
`scikit-learn`	0.18.1	Machine learning algorithms

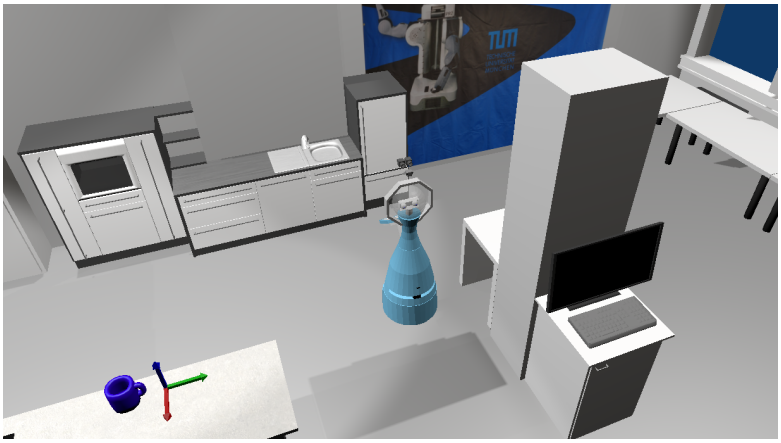


Figure 5.1: The SCITOS-A5 mobile service robot in a running Morse simulation of the `tum_kitchen` environment.

For MDP planning, the `pymdptoolbox` library [19], is used, which implements various planning algorithms for discrete MDPs. To solve for optimal policies in learned MDPs we rely on this library and apply the VI algorithm with a discount factor of $\gamma = 0.95$.

For the machine learning algorithms used, we employ the `scikit-learn` library. This library includes the implementations of k-Means clustering and the EM algorithm for fitting Gaussian Mixture Models (GMMs).

For Bayesian Optimization the `bayesian-optimization` library [56] has been employed. This library offers an implementation of the BO procedure based on a GP prior including the implementation of the most-used acquisition functions such as MPI, MEI and GP-UCB.

Simulator and Mobile Robot

For performing simulations the *MORSE* simulator [24], a generic simulator for academic robots, has been used in combination with the *ROS* middleware to control the robot in an environment. The simulations are performed with the Metralabs GmbH *SCITOS-A5* [50], an industry-standard mobile service robot designed specifically for interacting with humans and guiding them to products or exhibits. This robot is equipped with several sensors which can be used for navigation and Human Robot Interaction (HRI), such as an omni-directional camera, 24 ultrasonic sensors, a collision sensor and a SICK laser range finder [33]. As described in Section 5.1.2 we are particularly interested in the odometric capabilities of the robot for the implementation that has been used in our experiments. Figure 5.1 shows this robot in a MORSE simulation of one of the office environments used for our experiments. The `strands-desktop` meta-package (developed as part of the *STRANDS* project [36]) was used to obtain all required simulation software mentioned above with ease, in which the environments used in the simulations of our experiments are contained as well.

5.1.2. Dataset Acquisition

In order to be able to learn MDPs from data and establish the optimization, we should obtain a dataset that describes the environment the robot will operate in. This dataset should describe possible robot poses and to what other poses the execution of the possible actions may lead to, in order to properly describe the dynamics of the system.

Table 5.2: An excerpt of one of the execution traces datasets showing the format of the entries. Each entry stores the pose (`x`, `y` and `yaw`) of the robot based on odometry readings, and the action executed from this pose.

x	y	action_id	yaw
2.243550300598145	3.1634166240692	1	0.7840424207077832
2.890618801116943	3.8128550052643	7	-0.8159573629295891
3.541900634765625	3.1447956562042	6	-1.6159579833378268
3.544688224792481	2.1472022533417	0	-0.0326251734671011
...	...	...	...

Table 5.3: Details about the Morse simulation environments used and the size of the gathered datasets consisting of execution traces of the SCITOS-A5 robot.

Environment Name	Approximate Area (m^2)	Number of Entries in Dataset
`tum_kitchen`	100	4370
`uol_bl`	800–1000	17 245

For our implementation and the experiments that have been carried out, execution traces have been obtained by letting the robot follow a random action policy during which subsequent poses and actions are logged to a file. This exploration is performed inside the simulator both for a relatively small environment (i.e., `tum_kitchen`, based on a university kitchen of the Technical University of München) and large environment (i.e., `uol_bl`, based on a floor in a faculty of the University of Lincoln) obtained from a repository of the *STRANDS* project.

In the exploration a new entry is recorded in a file after each time-step of $t = 1.0\,\mathrm{s}$, which each contain the robot's pose based on odometric readings with its location as x and y position and its orientation described by the yaw. Apart from that, each entry also stores the action that is executed next from the current pose. As a result, the next entry tells us the robot's pose after the action in the previous entry has been executed.

In Table 5.2 an excerpt of one of the datasets is shown, which might give a clearer picture of the data that has been gathered and the format in which it has been stored. The possible actions of the robot correspond to a discrete set of robot movements in 8 different directions, being *south*, *south-east*, *east*, *north-east*, *north*, *north-west*, *west* and *south-west*. For example, the first two entries describes the transformation of the robot's pose after trying to make a *south-east* movement (n.b., positive difference in x corresponds to moving south, while positive difference in y corresponds to moving east). Table 5.3 presents additional information about the environments and the size of the datasets gathered accordingly.

5.1.3. Framework Implementation

To evaluate the framework proposed in Chapter 4, an implementation was made for the domain of mobile robot navigation. This implementation optimizes for an MDP that maximizes the yielded performance of following plans derived from it. That is, it aims to find a model that ensures a mobile robot moves from one location in an environment to another as fast as possible. In this section the relevant details of each part of the implementation are discussed.

Learning Step

For learning discrete-state MDPs from the acquired execution traces, a likelihood maximization approach is applied based on a state space obtained from clustering algorithms. That is, the state space is first obtained by applying a clustering algorithm (i.e., k-Means, GMM) with θ defining the number of components and the geometric positions of the execution traces as its training set. Then, a transition probability distribution is fitted on the execution traces, such that an MDP may be obtained that comprehends the transitions that are possible when actions are performed from any of its states.

After having learned an MDP, the learning step next assesses the corresponding model value. The model value is assessed based on how fast (i.e., expressed by the number of discrete time-steps) the agent would execute the tasks it is expected to perform when it is employed with the learned MDP.

Table 5.4: Settings for goal radius, parameter domain Θ, time-outs and discount factor used for each of the environments in the experiments.

Environment	Goal Radius (m)	Parameter Domain Θ	Goal T/O (s)	Stuck T/O (s)	Global T/O (s)	Discount Factor γ
tum_kitchen	0.5	$[2, 300]$	10	10	30	0.95
uol_bl	1.0	$[100, 1000]$	10	10	120	0.95

For our domain we define each task as navigating from a start location to a goal location in the environment, each presented as a pair of coordinates, as fast as possible. We translate these tasks into something that can be fed to the learned MDP, first of all, by setting the initial state of the MDP to the state predicted by the selected clustering algorithm for the start coordinates. Similarly, a goal state is obtained, and accordingly a reward function for the MDP is defined as prescribed in Section 4.4.1.

For each task, a value function and policy is computed for the MDP using the VI algorithm. Based on the computed value functions and simulations following the corresponding policies, the value $V_{\mathcal{M}}$ of the learned MDP $\mathcal{M}$ is computed as in Equation 4.5 of Section 4.4.1. In the simulations, for each task, the robot is first put at the start location defined by the task and then moves in the direction imposed by the computed policy at each discrete time-step. It repeats this until the goal state has been reached. At that point, however, the "true" goal might not have been reached by the robot. This is a problem, because if we would quit the simulation as soon as the goal state has been reached, then MDPs with state spaces that are too simple would yield high value. To take this into account, as soon as the robot reaches the goal state, it checks if it is inside a designer-specified range of the goal location (i.e., the *goal radius* in Table 5.4). If not, the robot is moved into the direction of the goal location and the same check is performed.

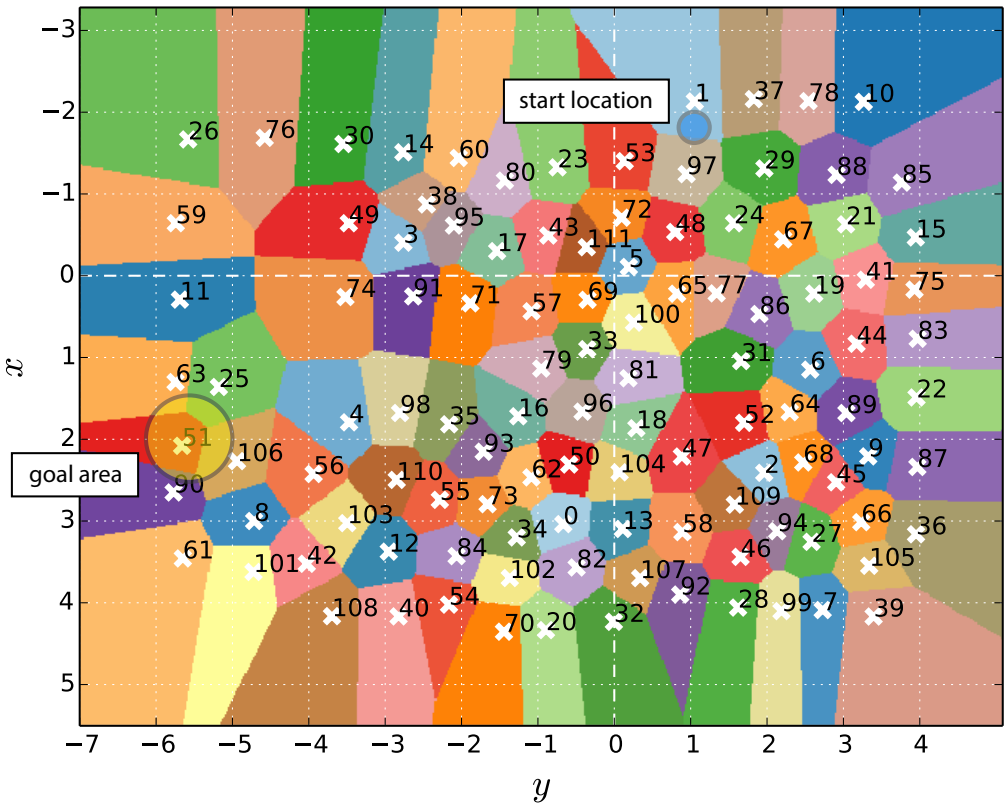
To avoid the simulations running endlessly on a certain task, time-outs are defined at which the simulation is quit. First of all, a (relatively short) *goal time-out* is defined for reaching the goal from the goal state, which clearly cannot take too long. Secondly, a *stuck time-out* is defined, which starts when the robot remains at the same location after performing an action. Finally, there is a *global time-out*, that defines the maximum time the robot is allowed to spend on performing a task, taking into account any extra time caused by slipping.

The output of each learning step is the value of the learned model $V_{\mathcal{M}}$ and the time spent on model learning and planning. The last being used when the MEIPS acquisition function is employed in the optimization. Further, additional data is logged in each iteration, which is the total iteration time, simulation time, model learning time, total planning time and the parameter setting θ used.

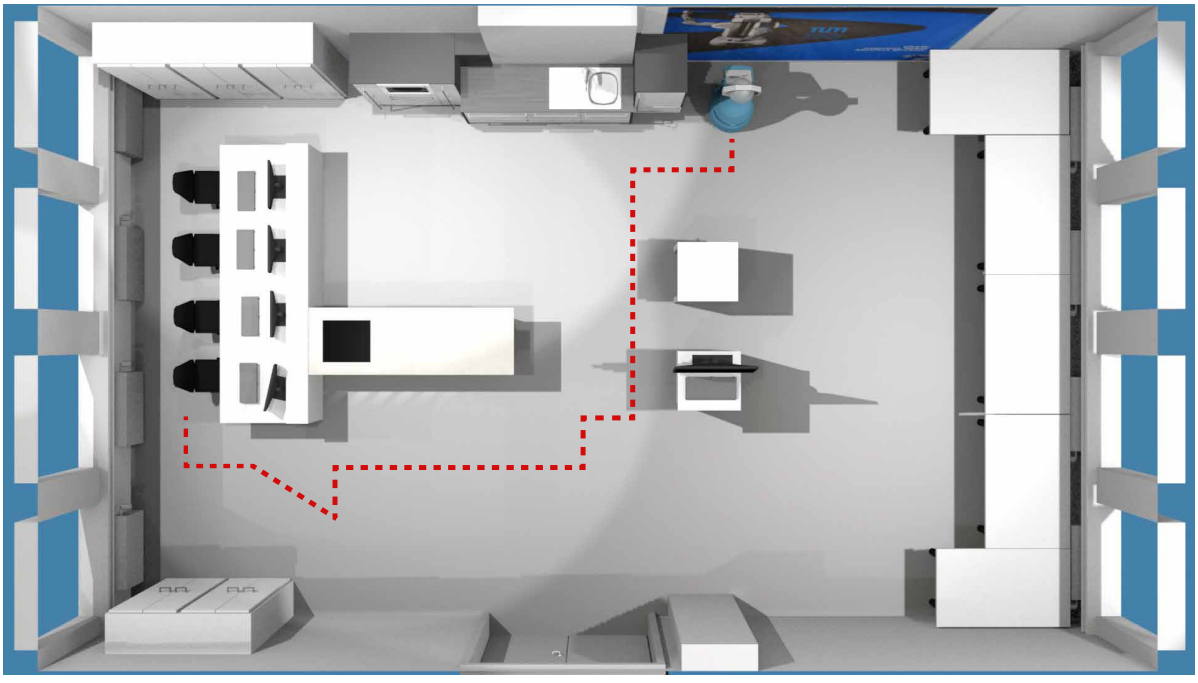
Optimization Step

For the implementation of the framework for this domain we make use of the BO framework for optimization based on a GP with a Matérn $3/2$ kernel where the noise in the simulations is approximated by adding Gaussian white noise through a White kernel with estimated noise level of 10^{-4}. As initially there is no knowledge about the model value given a parameter setting θ, first, a number of random settings are selected for which yielded performance is assessed and stored in an evidence set. In the following iterations new settings for θ are chosen with maximum utility in the selected acquisition function, which is repeated for a fixed number of iterations.

The first phase of the multi-phase framework described in Section 4.5 is executed as an individual optimization procedure, where the model values $V_{\mathcal{M}}$ are solely based on computed value functions and no simulations are performed. The resulting posterior of this first phase is then used in the acquisition of the first few samples in the second phase with the aim of finding a global maximizer faster.



(a) State space of an MDP with 112 states for the `tum_kitchen` environment learned by k-Means clustering. The plot depicts one of the tasks the mobile robot is expected to perform. The start location maps to a single state, labeled '1'. The goal center maps to the state labeled '51', while the goal area intersects with multiple states.



(b) A MORSE simulation of the `tum_kitchen` environment, showing the trajectory the SCITOS-A5 robot will follow to accomplish the task depicted in Figure 5.2a according to the policy computed from the MDP using the VI algorithm.

Figure 5.2: Demonstration of the implementation learning the state space of an MDP and using it for path planning for a task in the `tum_kitchen` environment.

Table 5.5: Details on the different configurations used for the experiments on the base (single phase) framework.

ID	Environment	Weight Factor β	Acquisition Function	Algorithm	Dataset Used
1	tum_kitchen	0.0	MEI	k-Means	100%
2	tum_kitchen	0.0	MEI	k-Means	75%
3	tum_kitchen	0.0	MEI	k-Means	50%
4	tum_kitchen	0.0	GP-UCB	k-Means	100%
5	tum_kitchen	0.0	MEIPS	k-Means	100%
6	tum_kitchen	0.0	MEI	GMM	100%
7	tum_kitchen	0.0	GP-UCB	GMM	100%
8	tum_kitchen	0.25	MEI	k-Means	100%
9	tum_kitchen	0.50	MEI	k-Means	100%
10	uol_bl	0.0	MEI	k-Means	100%
11	uol_bl	0.0	GP-UCB	k-Means	100%
12	uol_bl	0.0	MEIPS	k-Means	100%

Table 5.6: Details on the different configurations used for the experiments on the multi-phase framework.

ID	Environment	Weight Factor β	Acquisition Function	Algorithm	Dataset Used
13	tum_kitchen	0.0	MEI	k-Means	100%
14	tum_kitchen	0.0	GP-UCB	k-Means	100%
15	uol_bl	0.0	MEI	k-Means	100%
16	uol_bl	0.0	GP-UCB	k-Means	100%

5.1.4. Demonstration

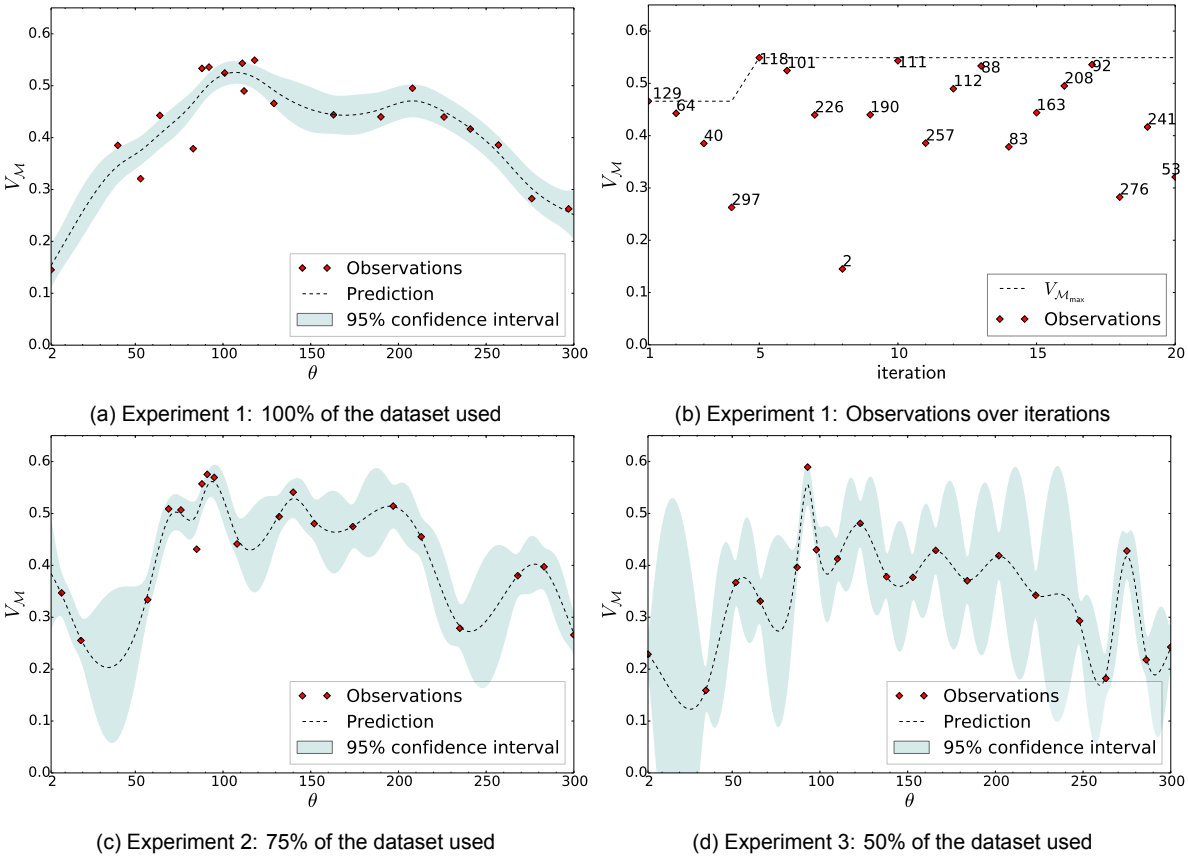
To provide the reader with a better insight as to how the implementation utilizes execution traces to learn MDPs and use these for path planning, a demonstration is given of a single iteration of the framework, supported by the illustrations in Figure 5.2. For this demonstration, let us assume that the optimization step provides us with a new parameter setting $\theta = 112$, specifying the number of states of the MDP, to evaluate.

First, given this parameter setting, a state space $\mathcal{S}$ is, in this case, learned by a k-Means clustering on the execution traces E, with the result shown in Figure 5.2a. Correspondingly, the transition function δ is fitted by maximum likelihood and a set of tasks T are mapped to this state space. To illustrate, let us take the task of moving from the position $(-1.9, 1.05)$ to $(2.0, -5.6)$, mapping to the states labeled with '1' and '51' respectively as shown in Figure 5.2a. Translating this into an initial state s_0 and reward function R, as described in Section 4.4.1, results in an MDP $(\mathcal{S}, s_0, A, \delta, R)$ that can be solved using the VI algorithm to obtain a policy (and value function). Then, to assess the performance in the simulator, this policy is employed, so that the robot will follow the trajectory as shown in Figure 5.2b (n.b., additionally, the robot might slip, so that it slightly deviates from this path). Then, with n being the recorded total number of time-steps, a performance measure for this task is obtained through Equation 4.3.

5.2. Experiment Configurations

In this section an overview is provided of the experiments that have been performed with the robot navigation implementation of the framework. Table 5.4 shows the auxiliary settings used in the experiments for aspects like time-outs, parameter domain and goal radius for each of the environments. Experiments are conducted for both the *base* framework and *multi-phase* framework with each of the environments. The configurations for these experiments are shown in Table 5.5 and Table 5.6 respectively.

The configurations were chosen with the aim of being able to identify how the results are affected based on changes in the used acquisition function, weight factor, clustering algorithm and the part of the dataset used. In each of these experiments data is stored that comprise the generated evidence sets, log-files and the MDPs that yield the best performance. In the next section the obtained results for each of these configurations are shown and discussed accordingly.



(a) Experiment 1: 100% of the dataset used

(b) Experiment 1: Observations over iterations

(c) Experiment 2: 75% of the dataset used

(d) Experiment 3: 50% of the dataset used

Figure 5.3: Plots showing a comparison of the resulting GP posterior for different dataset sizes. The plots correspond to 20 iterations of the base framework on the `tum_kitchen` environment with $\beta = 0.0$, k-Means clustering and MEI acquisition function used.

5.3. Results and Discussion

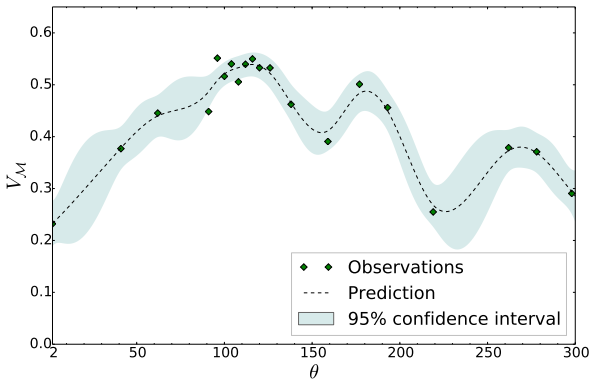
From the experiments that have been performed according to the configurations presented in Section 5.2 results have been obtained that are discussed in this section. First, we present and review the results obtained for the experiments that follow the base framework in Section 5.3.1. Then, in Section 5.3.2 the results for the multi-phase extension of the framework are presented and discussed. Each of the figures in these sections present plots for each of the experiments, showing the resulting posterior from the gathered evidence, and the value $V_{\mathcal{M}}$ of the learned model in each iteration of the BO optimization process.

5.3.1. Base Framework

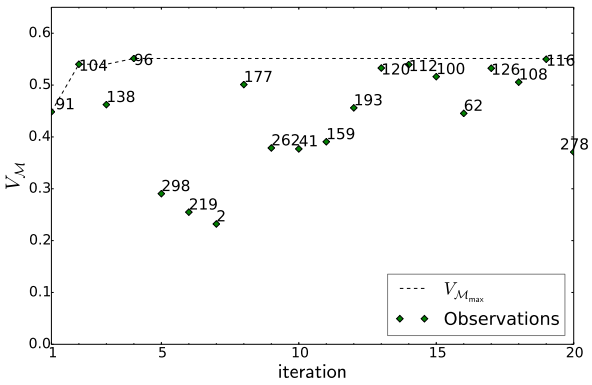
First off, we present and evaluate the results obtained from the experiments on the base framework. As shown in Table 5.5, experiments have been performed on two different simulation environments: the small `tum_kitchen` environment and the, in comparison, large `uol_bl` environment, for which the results are discussed in the following sections.

Tum Kitchen Environment

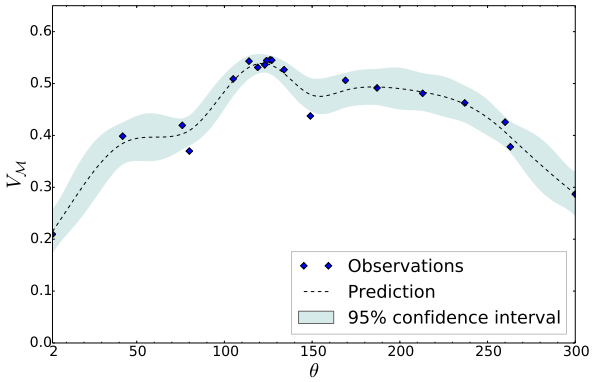
First of all, let us consider the results obtained for the small `tum_kitchen` environment. Overall, in Figures 5.4 to 5.6 one can see that the optimum $\theta_{\max}$ is mostly found within similar areas of the parameter space Θ. In Figure 5.3 we compare the effect of using different dataset sizes for learning MDPs. In Figure 5.3a, Figure 5.3c and Figure 5.3d the resulting GP is shown after respectively using 100%, 75% and 50% of the dataset in the corresponding experiments. One can see in Figure 5.3d that with 50% of the data employed, the observed model values $V_{\mathcal{M}}$ for different settings of θ involve quite some noise. In this figure one can also see that the GP is quite sensitive to this noise and accordingly over-fits on these observations, so that



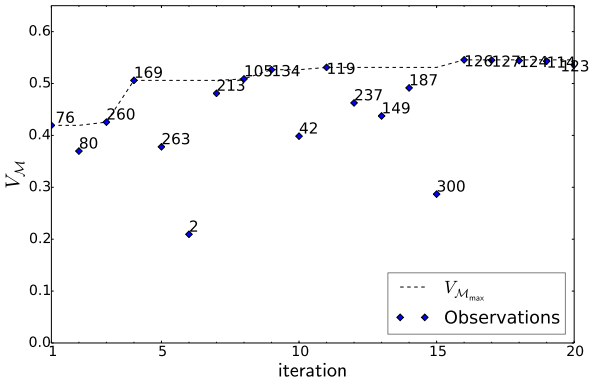
(a) Experiment 4: GP-UCB acquisition function used



(b) Experiment 4: Observations over iterations

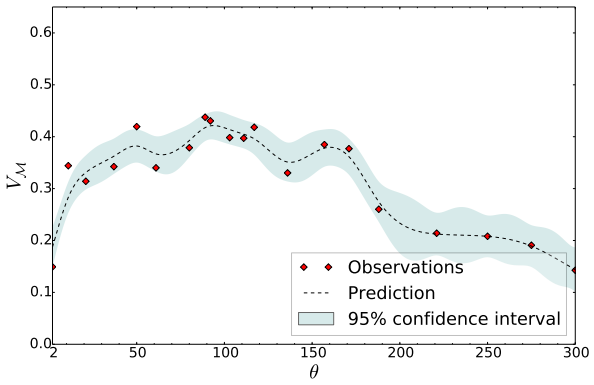


(c) Experiment 5: MEIPS acquisition function used

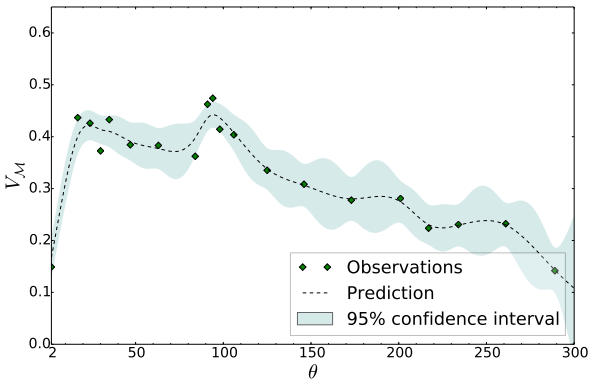


(d) Experiment 5: Observations over iterations

Figure 5.4: Plots showing a comparison of the resulting GP posterior for varying acquisition functions. The plots correspond to 20 iterations of the base framework on the `tum_kitchen` environment with $\beta = 0.0$ and k-Means clustering used.



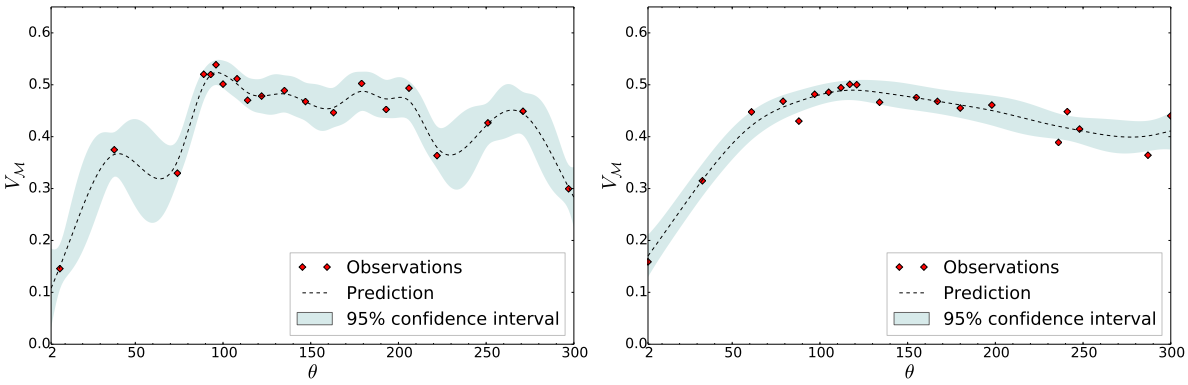
(a) Experiment 6: MEI acquisition function used



(b) Experiment 7: GP-UCB acquisition function used

Figure 5.5: Plots showing a comparison of the resulting GP posterior for varying acquisition functions. The plots correspond to 20 iterations of the base framework on the `tum_kitchen` environment with $\beta = 0.0$ and GMM used.

the uncertainty about the parameter space is quite large. At the other hand, Figure 5.3c shows that with 75% of the data the distinction becomes much more clear, even though the higher variance is still present, particularly for larger settings of θ. This clearly shows that sufficient data is needed to avoid overfitting and obtaining more stable results with discernible performance measures.



(a) Experiment 8: Weight factor $\beta = 0.25$

(b) Experiment 9: Weight factor $\beta = 0.5$

Figure 5.6: Plots showing a comparison of the resulting GP posterior for varying settings of the β factor. The plots correspond to 20 iterations of the base framework on the `tum_kitchen` environment with k-Means clustering and MEI acquisition used.

Next, let us compare the results depicted in Figure 5.3a and Figure 5.4 where different acquisition functions are used in the BO framework (i.e., MEI, GP-UCB and MEIPS respectively). In addition, we present some plots that show how the different acquisition function samples new observations in Figure 5.3b, Figure 5.4b and Figure 5.4d.

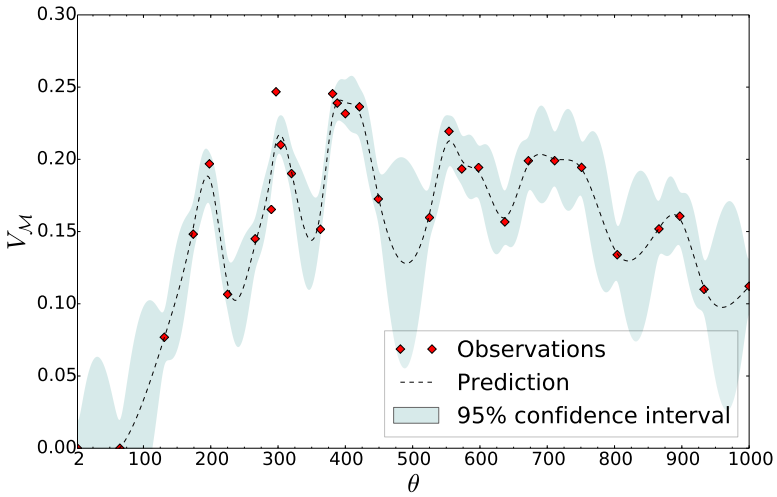
First of all, we see that in each of these experiments a global maximum is found in the same area of the parameter space. For this environment the GP-UCB and MEIPS lead to a more clear convergence to the area of the parameter space with the global optimum. The benefits of the timing GP in the MEIPS is only limited for this environment as the time for model learning and planning are closer together than for the `uol_bl` environment.

Then, in Figure 5.5a and Figure 5.5b we see the plots for the experiments in which instead GMMs are used for learning state spaces for our MDPs. We observe that again for this algorithm, the optima are found within the same areas of the parameter space for both experiments with different acquisition functions. The main thing we observe is that, the observed model values $V_{\mathcal{M}}$ is overall lower with the GMM employed in comparison to the values observed with k-Means employed. Therefore, employing k-Means appears more suitable to learn the state space for MDPs in our application of mobile robot navigation.

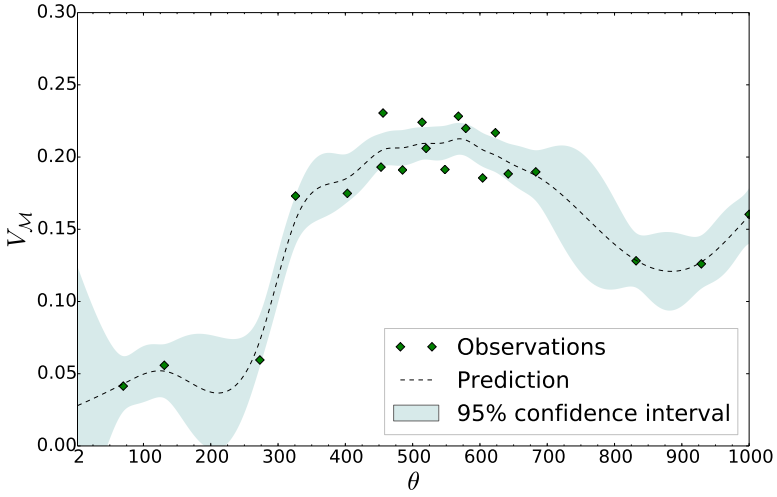
Finally, we look at the influence of the weight factor β for the small environment considering the plots of Figure 5.3a, Figure 5.6a and Figure 5.6b. In this particular situation, one might benefit from weighing V_{DTP} in the model value, as it could make a clearer distinction between values for different settings of θ, because it cancels out part of the uncertainty from the simulations. The intuition of why this works well for this particular environment can be elucidated by Figure 5.8, which shows that the maxima of the V_{DTP} and V_{SIM} measures lie close together. However, setting $\beta > 0$ might also lead to a bias in other environments, which is that, although V_{DTP} may have high value, computed policies may not work well in the simulations or the real world.

UOL BL Environment

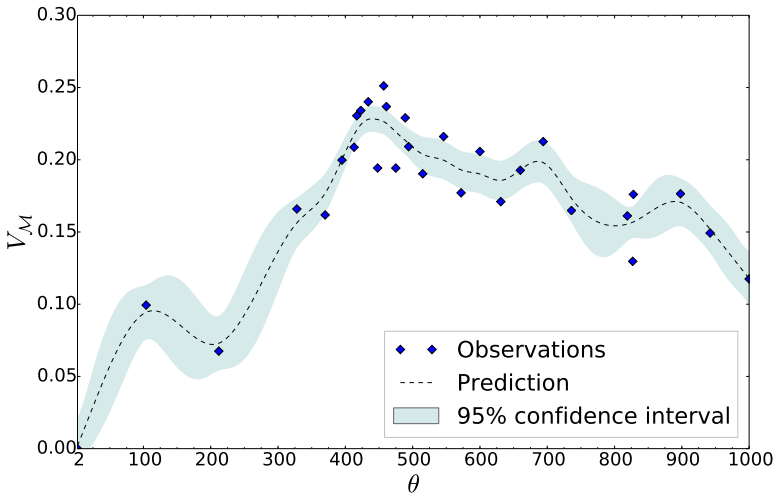
Let us next consider the results for the large `uol_bl` environment. In Figure 5.7 plots are shown for the experiments in which different acquisitions were used in the BO. One can observe the optima found in each of these experiments lie close together. The experiment that employs the MEI function, however, appears ineffective to find the same global maximum in multiple repetitions. All in all, the GP-UCB and especially the MEIPS function appear most effective in sampling parameter settings that result in MDPs yielding high performance.



(a) Experiment 10: MEI acquisition function used

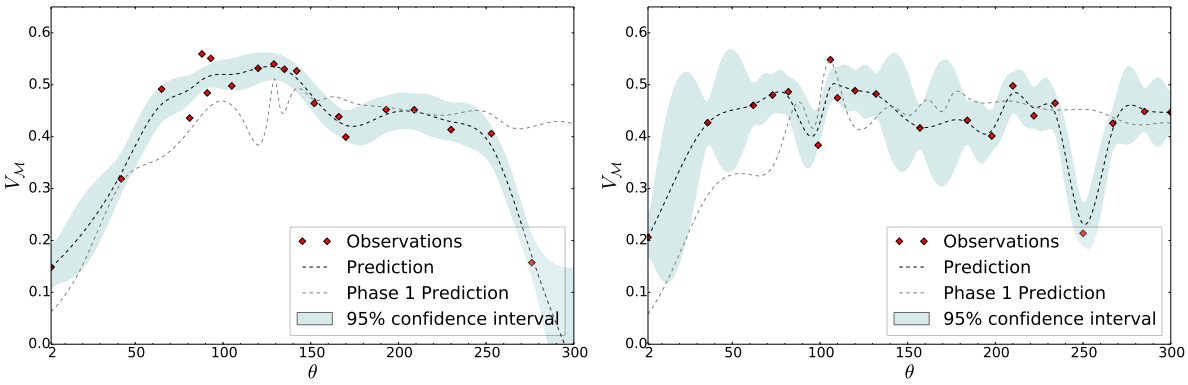


(b) Experiment 11: GP-UCB acquisition function used



(c) Experiment 12: MEIPS acquisition function used

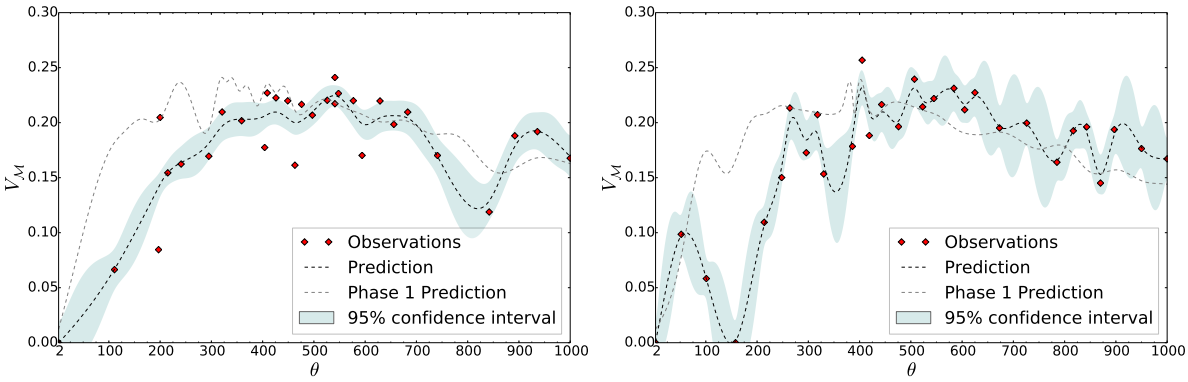
Figure 5.7: Plots showing a comparison of the resulting GP posterior for varying acquisition functions. The plots correspond to the `uol_bl` environment with $\beta = 0.0$ and k-Means clustering used.



(a) Experiment 13: MEI acquisition function in phase 1

(b) Experiment 14: GP-UCB acquisition function in phase 1

Figure 5.8: Plots showing a comparison of the resulting GP posterior for varying acquisition functions. The plots correspond to 20 iterations of the multi-phase framework on the `tum_kitchen` environment with $\beta = 0.0$ and k-Means clustering used.



(a) Experiment 15: MEI acquisition function in phase 1

(b) Experiment 16: GP-UCB acquisition function in phase 1

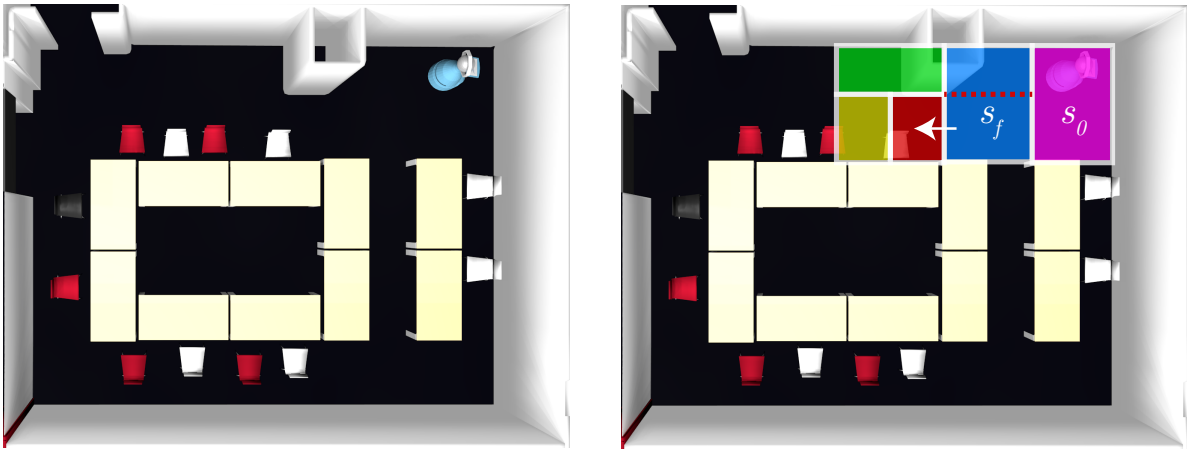
Figure 5.9: Plots showing a comparison of the resulting GP posterior for varying acquisition functions. The plots correspond to 30 iterations of the multi-phase framework on the `uol_bl` environment with $\beta = 0.0$ and k-Means clustering used.

Table 5.7: Overview of the time expenses of a single iteration for model learning, planning and simulation with the algorithms used in the experiments. The time expenses are presented as ranges for the smallest and largest MDPs learned for the environment.

Environment	Model Learning Time (s)	Planning Time (s)	Simulation Time (s)
`tum_kitchen`	0.2–15.0	0.01–0.5	110–200
`uol_bl`	1.0–60.0	0.05–5.0	100–700

5.3.2. Multi-Phase Framework

Next, we present and evaluate our results obtained in the experiments on the multi-phase framework, for which the configurations are presented in Table 5.6. In these experiments, first an optimization takes place of the model value computed solely from the value functions for the tasks the system is expected perform. Only computing the value functions for MDPs as is done in this first phase is relatively time-cheap (i.e., a matter of minutes) in comparison to performing time-costly simulations (i.e., taking multiple hours) as is done in the second phase. In Table 5.7 we present an overview to give an idea of the difference in time expenses between learning MDP, running VI to compute plans, and running simulations in the Morse simulator. In the plots in Figures 5.8a to 5.9b, the prediction mean resulting from this first phase is depicted by a dotted line. In the second phase, the posterior from the first phase is utilized to steer the acquisition of the first few samples as described in Section 4.5. The BO in both of these phases is preceded by collecting a set of 5 random samples. The third phase of



(a) One of the rooms in the (simulated) `uol_bl` environment. The mobile robot, seen in the top-right, is presented the task of moving from its current location to a location in the main hall, outside the room.

(b) Illustration of the area of the earlier-learned state space of interest. The resolution for the part of the state space as shown is too low, particularly, considering the state s_f, which might cause the robot to get stuck. The third phase of the multi-phase framework therefore splits the state, as depicted by the dotted red line, based on new centroids.

Figure 5.10: Illustration of the third phase of the multi-phase framework in a simulation of the `uol_bl` environment. The phase starts off with the MDP model learned in the previous phase, and further fine-tunes this model after the robot gets stuck executing a new task.

the multi-phase framework will be covered separately, as it is mostly independent of the two prior phases, being concerned with fine-tuning a specific MDP model rather than searching for one.

Tum Kitchen Environment

First, we consider the experiments that apply the multi-phase framework to find an MDP for path planning in the `tum_kitchen` environment. In Figure 5.8a and Figure 5.8b one can see that there appears to be a correspondence between the maxima of the posterior of the first and second phase. Particularly, in Figure 5.8b we see a direct correspondence of the global maximums, so that a proper model is found in the first few iterations of the optimization process.

UOL BL Environment

Next, we consider the experiments that apply the multi-phase framework to the `uol_bl` environment, for which the results are shown in Figure 5.9a and Figure 5.9b. As opposed to the experiments on the `tum_kitchen` environment, the prediction mean from the first phase has a less evident correspondence with the maxima of the posterior of the second phase. However, this does not prevent the BO of eventually finding a maximum for both of these experiments in the same area of the parameter space. One thing that should be noted is that for this environment there seems to be significantly more noise in the observations than for the observations made with the smaller `tum_kitchen` environment. We identified that one aspect that influences the amount of noise is the size of the dataset for model learning, having observed that with only half of our data significantly more noise could be observed. Certainly we cannot exclude other causes for these highly varying observations, as part of it is naturally caused by the uncertain dynamics being simulated, such as the robot slipping. It could also be that assessing the performance based on a larger set of tasks or implementing a mechanism for detecting outliers may result in more stable observations.

Phase 3: MDP Tuning

Finally, we get back at the third phase of our multi-phase framework, in which the MDP $\mathcal{M}$ obtained in the previous phase is further fine-tuned, given a set of new tasks. To investigate the expediency of this third phase, an implementation of the solution proposed in Algorithm 10 has been made for the mobile robot navigation application.

Given a task to execute, the solution solves MDP $\mathcal{M}$ (with new initial state and reward function for this task) for a policy, which it uses to perform the task. Then, if the controlled robot gets stuck in a state s_f in its attempt to complete the task, it starts to gather new experience about the transitions possible from this state in a collection E_{SIM}. Based on this new experience, the solution splits the state into a number of new (sub-)states through a clustering of the observations in E_{SIM}, and accordingly updates the transition probabilities based on the new data. The updated MDP is then used to compute a new policy for the task, after which the agent tries to perform the task again. If the agent gets stuck in the same state s_f a clustering with higher resolution is applied to split the state.

For our implementation of this proof-of-concept solution, the mobile robot executes and gathers new experience in a simulation environment. In gathering new experience, the mobile robot stores the observations (i.e., odometric readings) it makes and the state s' it ends up in after executing an action a from these observations.

To evaluate the approach, we take the MDP $\mathcal{M}$ learned in the second phase of our last experiment (i.e., see Figure 5.9b) for the `uol_bl` environment. Then, a new task is presented, which requires the robot to move from the corner of a room in the environment, as shown in Figure 5.10a, to a location in the main hall outside the room. The problem, is that the resolution in the learned state space $\mathcal{S}$ of $\mathcal{M}$ for this area is too low, as shown in Figure 5.10b, caused by the limited execution traces gathered for this area in the set E used for learning the model. As a result, there is a considerable chance the robot gets stuck in the state labeled $s_f \in \mathcal{S}$ from its starting location.

To take care of this, after the observation is made that he robot got stuck, the algorithm makes the robot gather new data about the transitions possible from the different locations within the state s_f in which it got stuck. Based on this data, the algorithm clusters the data to effectively split the state in two, well-nigh as shown in Figure 5.10b. The resulting MDP with updated state space and transition probabilities then allows the mobile robot to successfully accomplish the task (i.e., by the computed policy suggesting to move south and west subsequently).

Although the algorithm works well for fixing small discrepancies like these in the model, caused by limited data about a certain area of the environment, it is not well suited for learning major parts of the model from the ground up. For such scenarios, one is better off using existing RL solutions, such as continuous Q-learning or active learning approaches which use the learned MDP as a prior model (see Section 3.2).

6

Conclusions

The goal of this thesis is to provide a foundation for an algorithmic technique for learning Markov Decision Processes (MDPs) for planning under uncertainty which maximize yielded performance given a dataset describing the dynamics of a system that involves uncertainty. This chapter presents a summary of the presented work and the contributions made in Section 6.1 and revisits the identified research questions in Section 6.2. Finally, this chapter concludes this thesis with recommendations and suggestions for future work in Section 6.3.

6.1. Summary of Contributions

Previous work in the field of probabilistic model learning for planning under uncertainty has already presented us with various algorithms for learning Markov Decision Processes (MDPs) offline from a dataset describing the dynamics of the system under consideration. The state of the art however lacks an automated method of setting the hyperparameters of these learning algorithms so to best reflect the underlying system and maximize its performance in the execution of the tasks it is expected to perform. To address this issue, we pose the adjustment of the hyperparameters of such learning algorithms as an optimization task. In this optimization task, the goal is to find the setting of the hyperparameters θ which maximizes the performance yielded by executing the plans or *policies* derived from the learned MDP.

In this thesis, we present a solution (which we refer to as the *MDP optimization framework*) that employs the Bayesian Optimization (BO) framework for this optimization task, defining a probability distribution over functions which maps parameter settings θ to an assessment of the value $V_{\mathcal{M}}$ of a learned MDP. Although algorithms that employ the BO framework for SDM problems do already exist, all of them are online RL approaches that do not utilize an available dataset prior to interacting with the real-world environment. The model value $V_{\mathcal{M}}$ of an MDP, used as a relative performance measure in the optimization, is assessed based on computed value functions and simulations of the tasks the system is expected to perform.

Additionally, we extended our proposed solution by exploiting the lower cost of computing a value function in comparison to performing time-expensive simulations, to define our *multi-phase optimization framework*. That is, we define a first phase in which BO is performed to maximize the average expected value for the MDP for a set of tasks to perform. The posterior resulting from this phase is then used to steer the acquisition in a second optimization phase, in which the performance in simulations over a set of tasks is being optimized.

Then, finally, we present a solution to further fine-tune the parameters of the MDP resulting from this optimization. This is done by increasing the resolution of the state space and updating the transition probabilities where necessary, e.g., when the agent gets stuck in a certain state.

An implementation of the aforementioned framework has been made for path planning in a mobile robot navigation domain, in which a robot needs to move from one location to another in an office environment. A dataset of execution traces with odometric readings has been gathered based on which our implementation learns MDPs by clustering the data into

a state space, with the number of states defined by the parameter setting θ. The implementation thus optimizes for an MDP that can produce policies for a mobile robot offline, which can be followed to move from one location to another as fast as possible.

The results of the experiments show that our framework can effectively be used to obtain an MDP for the path planning of a mobile robot. The first phase in the multi-phase framework shows out to be able to steer the acquisition in the second phase towards a global maximizer in some scenarios. And finally, the proof-of-concept implementation of the last phase, can successfully be used to further fine-tune the parameters of a learned MDP when the mobile robot gets stuck in some state when presented with a new task. However, it should be noted that the last phase only has been tested on a single task (considering it is just an extra step on top to potentially further improve MDPs), which means we cannot simply guarantee it can be used to recover from all discrepancies that may be present.

6.2. Revisiting the Research Questions

To answer the main research question of this thesis, the four research questions presented in Section 1.3 are revisited in this section. First off, the following research question is mostly concerned with getting a good overview of the state of the art for learning MDPs from a dataset:

Research Question 1. Which learning algorithms exist that can be employed for learning MDPs from data for systems involving uncertainty that require plans for automated control?

As seen in Section 2.1.3, one of the most straightforward methods can be deducted from methods for fitting Markov Chains, i.e. by maximum likelihood or Bayesian inference. The difference is in the addition of actions, so that a transition probability matrix needs to be learned for each possible action, given some user-defined state space. For various domains, such as that of mobile robot navigation, defining the state space may not be a trivial task, although there are approaches for this, such as k-Means clustering or (time-)state merging approaches, which are parameterized by the number of states. Then, when one needs to learn partially observable models one needs to consider other approaches, as seen in Section 3.1, to account for emission probabilities as well.

Research Question 2. How should a performance measure be defined which can be used to fairly compare the value of different MDPs?

The value $V_{\mathcal{M}}$ of an MDP $\mathcal{M}$ should be defined in terms of how well the agent performs tasks based on the policies computed from the model. Therefore, first of all, the performance can be estimated using the expected value in the initial state from the value function for multiple tasks. However, as a model abstracts from the real world, using the value function to express model value is not always sufficient. Therefore, a more accurate estimation can be made through simulations of the tasks the system is expected to perform, discounting the obtained reward based on the number of steps made. Combining these estimations results in the expression shown in Equation 4.5 in Section 4.4.1, which can be used as a relative performance measure of different MDPs for an environment.

Research Question 3. How can the parameter space of model learning algorithms cost-effectively be explored towards a global maximizer with only limited knowledge about the system under consideration?

We have seen that making an accurate assessment of the value of an MDP requires time-expensive simulations to be performed. Therefore, it is important to limit the number of evaluations of a parameter setting for the used model learning algorithm, with BO emerging as an attractive framework for optimization. As we have seen in our experiments, BO could effectively be employed to explore the parameter space with a limited number of evaluations.

Research Question 4. How can the hierarchy of different abstraction levels be exploited to find a performance-maximizing MDP in a more cost-effective way?

This research question has been addressed by the multi-phase framework proposed in Section 4.5. In the first phase BO for an MDP with the model value based solely on the value functions for the set of tasks over which to evaluate the performance. The resulting posterior is then used in the second phase to steer the acquisition of the first few samples. In our experiments we have seen that it is possible to use the first phase to successfully steer the optimization in the second phase towards a global maximizer.

Putting our findings for our sub-questions together, we provide an answer to our main research question reiterated below.

Main Research Question. How can the task of obtaining a (discrete) MDP that maximizes the yielded performance of executing plans that are derived from it, given a dataset about the system under consideration, be automated?

As handcrafting MDPs for systems with uncertain dynamics is a difficult task, an appealing approach is to employ learning algorithms to automate this task. To learn an MDP which maximizes performance yielded from following the policies computed from it, one needs to properly adjust the hyperparameters of these algorithms. As evaluating all possible hyperparameter settings may be a cost-expensive endeavor, the parameter space of model learning algorithms may be explored more effectively using the sequential model-based optimization framework known as Bayesian Optimization (BO). To compare MDPs, assessments of their value can be made at different levels of abstraction, i.e., based solely on the MDP and its value function, based on performance in simulations, or based on performance in a real-world environment. Additionally, it was found that assessments on a higher level of abstraction (with lower costs) can be employed to steer the optimization on a lower level of abstraction towards the area of the parameter space that maximizes the performance. However, we should note that to give a conclusive answer to our main research question, we need to investigate how well our methodology works for different application domains.

6.3. Recommendations and Future Work

As the framework proposed in this thesis is targeted at offline planning, the policies computed from resulting MDPs can only to a certain extent account for the exogenous events in the environment. Suppose an agent for a mobile robot, like in our example application, is employed in the real world and at some point the planned path is obstructed. In this scenario the agent will be unable to get to its target location based *solely* on the policy that was computed offline. To account for these scenarios, first off, one could choose to periodically explore the environment (using the current model as a blueprint, as in [26]) and update the model parameters accordingly instead of having just a single exploration phase. An alternative is to employ the learned MDP in one of the model-based RL methods seen in Chapter 3, which automatically balance exploration and exploitation.

Another issue that might be interesting to address in future work, is to identify whether or not sufficient data is available for learning an appropriate MDP model. Although in our experiments we were able to observe a high level of noise and overfitting on smaller datasets, still the problem exists of how to assess what dataset size is appropriate and whether or not the dataset covers the complete environment.

Considering the optimization aspect of the presented framework, one of the issues that were identified, is that selecting the acquisition function and the hyperparameters of the GP prior for BO is not a trivial task. For future work, we recommend the use of the portfolio allocation [39, 68] and integrated acquisition [72] functions, discussed in Chapter 2, which ease the above task as they reduce the number of choices that need to be made.

Probably one of the most interesting directions for future work is to examine and experiment with model learning through our framework for different domains. The main question then is whether or not the application domain under consideration can provide a dataset that describes the dynamics of the system and allows for learning probabilistic models. Considering that MDPs and especially POMDPs are data intensive, one should carefully contemplate the contents of the dataset and whether they allow for learning transition probabilities, emission probabilities, state spaces and/or observation spaces which accurately reflect the dynamics of the underlying system. One recommendation is to look into learning POMDPs for spoken dialogue systems [16, 60, 82]. Recently, there has been much interest in modeling the dialogue manager of these systems this way, although estimating their dynamics is quite a difficult task, and so it is desirable to automate this process. As an example, in [16], the POMDP's components (i.e., state space, observation space, transition and emission probabilities) are learned based on a dataset of human-to-human dialogues. For such domains, our framework may prove useful in learning a probabilistic model that maximizes the performance of the system. For other potential application domains, one may want to consult [14] which presents a variety of applications for POMDPs which may potentially be learned from data.

Another topic we briefly touched upon in Chapter 2 is *automated model checking*, which encompasses techniques for formal model verification through temporal logic formulas. Examples can be found in [6, 45] where LTL formulas are used to formally express temporal goals and produce a 'product-MDP' to derive plans which (partially) satisfy these formulas. An interesting direction for future work would be to investigate the possibilities of combining such model checking techniques with our framework. That is, combining the methods from automated model checking and model learning through our framework, one may be able to develop a framework that optimizes for an MDP which is best capable of satisfying a collection of LTL formulas.

Acronyms

BMDP Bounded-parameter Markov Decision Process. 27

BO Bayesian Optimization. ix, 4, 5, 7, 17–19, 21, 22, 35, 37, 44, 46, 49, 51, 53, 54, 57–59

BRL Bayesian Reinforcement Learning. 26, 27

DTP Decision-Theoretic Planning. 1, 5, 7, 8, 10–12, 21, 23, 25, 32, 33, 35, 37

EM Expectation Maximization. 24, 44

GMM Gaussian Mixture Model. x, 44, 45, 48, 50, 51

GP Gaussian Process. ix, x, 18–22, 32, 37, 39, 44, 46, 49–53, 59, 61

GP-UCB GP Upper Confidence Bound. ix, 18–20, 44, 48, 50–53

HMM Hidden Markov Model. ix, 8, 10–12, 24, 25

HRI Human Robot Interaction. 44

LP Linear Program. 15, 27

LTL Linear Temporal Logic. 11, 60

MDP Markov Decision Process. ix, x, 1, 3–5, 7, 8, 10–16, 21, 23–27, 29–41, 43–49, 51, 53–55, 57–61

MDP-IP MDP with Imprecise Probabilities. 27

MEI Maximum Expected Improvement. x, 19–21, 44, 48–53

MEIPS Maximum Expected Improvement Per Second. 20, 46, 48, 50–52

MPI Maximum Probability of Improvement. 19, 20, 44

PI Policy Iteration. 14, 15, 33, 35

POMDP Partially Observable MDP. 8, 11, 12, 21, 24, 25, 60

RL Reinforcement Learning. 1, 2, 5, 8, 12, 16, 22, 23, 26, 29, 55, 57, 59, 61

RTDP Real-Time Dynamic Programming. 14, 27

SDM Sequential Decision Making. 5, 7, 8, 12, 17, 26, 35, 57

VI Value Iteration. 13–15, 26, 27, 32, 33, 35, 44, 46–48, 53

Bibliography

[1] C. G. Atkeson and J. C. Santamaria. A Comparison of Direct and Model-Based Reinforcement Learning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4 of *ICRA'97*, pages 3557–3564, April 1997.

[2] D. Bacciu, P. J. G. Lisboa, A. Sperduti, and T. Villmann. Probabilistic Modeling in Machine Learning. pages 545–575. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-43505-2.

[3] J. Baker, J. Baker, P. Bamberg, K. Bishop, L. Gillick, V. Helman, Z. Huang, Y. Ito, S. Lowe, B. Peskin, R. Roth, and F. Scattone. Large Vocabulary Recognition of Wall Street Journal Sentences at Dragon Systems. In *Proceedings of the Workshop on Speech and Natural Language*, pages 387–392. Association for Computational Linguistics, 1992. ISBN 1-55860-272-0.

[4] D. Barber. *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[5] A. G. Barto, S. J. Bradtke, and S. P. Singh. Learning to Act using Real-Time Dynamic Programming. *Artificial intelligence*, 72(1-2):81–138, 1995.

[6] A. Bhatia, L. E. Kavraki, and M. Y. Vardi. Sampling-Based Motion Planning with Temporal Goals. In *2010 IEEE International Conference on Robotics and Automation*, pages 2689–2696, 2010.

[7] J. A. Bilmes. A Gentle Tutorial of the EM Algorithm and its Application to Parameter Estimation for Gaussian Mixture and Hidden Markov Models. *International Computer Science Institute*, 4(510):126, 1998.

[8] C. Boutilier, T. Dean, and S. Hanks. Decision-Theoretic Planning: Structural Assumptions and Computational Leverage. *Journal of Artificial Intelligence Research*, 11:1–94, 1999. ISSN 10769757.

[9] R. I. Brafman and M. Tennenholtz. R-MAX – A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. *Journal of Machine Learning Research*, 3:213–231, 2002. ISSN 15324435.

[10] E. Brochu, V. M. Cora, and N. de Freitas. A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning. eprint arXiv:1012.2599, December 2010.

[11] O. Buffet and D. Aberdeen. Robust Planning with (L)RTDP. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, IJCAI'05, pages 1214–1219, San Francisco, CA, USA, Morgan Kaufmann Publishers Inc., 2005.

[12] T. Caelli, A. McCabe, and G. Briscoe. Shape Tracking and Production using Hidden Markov Models. *International Journal of Pattern Recognition and Artificial Intelligence*, 15(01):197–221, 2001.

[13] S. Calinon, F. Guenter, and A. Billard. On Learning, Representing, and Generalizing a Task in a Humanoid Robot. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 37(2):286–298, 2007.

[14] A. R. Cassandra. A Survey of POMDP Applications. In *Working Notes of AAAI 1998 Fall Symposium on Planning with Partially Observable Markov Decision Processes*, pages 17–24, 1998.

[15] M. E. Chamie and B. Açıkmeşe. Finite-Horizon Markov Decision Processes with State Constraints. *arXiv preprint arXiv:1507.01585*, 2015.

[16] H. R. Chinaei and B. Chaib-draa. *Learning Dialogue POMDP Models from Data*, pages 86–91. CAI'11. Springer Berlin Heidelberg, Berlin, Heidelberg, May 2011.

[17] J. K. Cochran, A. Murugan, and V. Krishnamurthy. Generic Markov models for Availability Estimation and Failure Characterization in Petroleum Refineries. *Computers & Operations Research*, 28(1):1–12, 2001.

[18] E. Contal, V. Perchet, and N. Vayatis. Gaussian Process Optimization with Mutual Information. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *ICML'14*, pages 253–261, 2014.

[19] S. A. W. Cordwell. Python Markov Decision Process Toolbox, 2015. URL `https://github.com/sawcordwell/pymdptoolbox`.

[20] O. Cronvall and I. Männistö. Combining Discrete-Time Markov Processes and Probabilistic Fracture Mechanics in RI-ISI Risk Estimates. *International Journal of Pressure Vessels and Piping*, 86(11):732–737, 2009.

[21] M. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the 28th International Conference on Machine Learning*, ICML'11, pages 465–472, January 2011.

[22] K. V. Delgado, L. N. de Barros, D. B. Dias, and S. Sanner. Real-time dynamic programming for Markov decision processes with imprecise probabilities. *Artificial Intelligence*, 230:192–223, 2016.

[23] K. V. Delgado, S. Sanner, and L. N. De Barros. Efficient Solutions to Factored MDPs with Imprecise Transition Probabilities. *Artificial Intelligence*, 175(9-10):1498–1527, 2011.

[24] G. Echeverria, S. Lemaignan, A. Degroote, S. Lacroix, M. Karg, P. Koch, C. Lesire, and S. Stinckwich. Simulating Complex Robotic Scenarios with MORSE. *Simulation, Modeling, and Programming for Autonomous Robots*, pages 197–208, 2012.

[25] A. M. El-Nashar. Optimal Design of a Cogeneration Plant for Power and Desalination Taking Equipment Reliability into Consideration. *Desalination*, 229(1):21–32, 2008.

[26] A. Epshteyn, A. Vogel, and G. DeJong. Active Reinforcement Learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, pages 296–303, New York, NY, USA, ACM, 2008. ISBN 978-1-60558-205-4.

[27] M. Gales and S. Young. The Application of Hidden Markov Models in Speech Recognition. *Foundations and trends in signal processing*, 1(3):195–304, 2008.

[28] J. Garcıa and F. Fernández. A Comprehensive Survey on Safe Reinforcement Learning. *Journal of Machine Learning Research*, 16(1):1437–1480, 2015.

[29] Z. Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521 (7553):452–459, 2015. ISSN 0028-0836.

[30] M. Ghavamzadeh, S. Mannor, J. Pineau, A. Tamar, et al. Bayesian Reinforcement Learning: A Survey. *Foundations and Trends® in Machine Learning*, 8(5-6):359–483, 2015.

[31] R. Givan, S. Leach, and T. Dean. Bounded-parameter Markov Decision Processes. *Artificial Intelligence*, 122(1-2):71–109, 2000.

[32] S. D. Grimshaw and W. P. Alexander. Markov Chain Models for Delinquency: Transition Matrix Estimation and Forecasting. *Applied Stochastic Models in Business and Industry*, 27(3):267–279, 2011.

[33] H. M. Gross, H. J. Boehme, C. Schröter, S. Müller, A. König, C. Martin, M. Merten, and A. Bley. ShopBot: Progress in Developing an Interactive Mobile Shopping Assistant for Everyday Use. In *2008 IEEE International Conference on Systems, Man and Cybernetics*, SMC'08, pages 3471–3478. IEEE, 2008.

[34] A. Guez, D. Silver, and P. Dayan. Efficient Bayes-Adaptive Reinforcement Learning using Sample-Based Search. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pages 1025–1033. Curran Associates Inc., 2012.

[35] M. Guillot and G. Stauffer. The Stochastic Shortest Path Problem: A polyhedral combinatorics perspective. *CoRR*, 2017.

[36] N. Hawes, C. Burbridge, F. Jovan, L. Kunze, B. Lacerda, L. Mudrová, J. Young, J. Wyatt, D. Hebesberger, T. Körtner, et al. The STRANDS Project: Long-Term Autonomy in Everyday Environments. *IEEE Robotics Automation Magazine*, 24(3):146–156, September 2017. ISSN 1070-9932.

[37] T. Hester, M. Quinlan, and P. Stone. Generalized Model Learning for Reinforcement Learning on a Humanoid Robot. In *2010 IEEE International Conference on Robotics and Automation*, ICRA'10, pages 2369–2374, May 2010.

[38] T. Hester, M. Quinlan, and P. Stone. RTMBA: A Real-Time Model-Based Reinforcement Learning Architecture for Robot Control. In *2012 IEEE International Conference on Robotics and Automation*, ICRA'12, pages 85–90, May 2012.

[39] M. D. Hoffman, E. Brochu, and N. de Freitas. Portfolio Allocation for Bayesian Optimization. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence*, UAI'11, pages 327–336, Arlington, VA, USA, AUAI Press, 2011. ISBN 978-0-9749039-7-2.

[40] R. A. Howard. *Dynamic Programming and Markov Processes*. The MIT Press, 1st edition, June 1960. ISBN 0262080095.

[41] L. P. Kaelbling, M. L. Littman, and A. W. Moore. Reinforcement Learning: A Survey. *Journal of Artificial Intelligence Research*, 4:237–285, May 1996. ISSN 1076-9757.

[42] K. Kawaguchi, L. P. Kaelbling, and T. Lozano-Pérez. Bayesian Optimization with Exponential Convergence. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, NIPS'15, pages 2809–2817, Cambridge, MA, USA, MIT Press, 2015.

[43] M. Kearns and S. Singh. Near-Optimal Reinforcement Learning in Polynomial Time. *Machine Learning*, 49(2-3):209–232, 2002.

[44] S. Koenig and R. G. Simmons. Unsupervised Learning of Probabilistic Models for Robot Navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 3 of *ICRA'96*, pages 2301–2308, April 1996.

[45] B. Lacerda, D. Parker, and N. Hawes. Optimal Policy Generation for Partially Satisfiable Co-Safe LTL Specifications. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, pages 1587–1593, 2015.

[46] T. C. Lee, G. G. Judge, and A. Zellner. Maximum Likelihood and Bayesian Estimation of Transition Probabilities. *Journal of the American Statistical Association*, 63(324):1162–1179, December 1968.

[47] M. L. Littman, T. L. Dean, and L. P. Kaelbling. On the Complexity of Solving Markov Decision Problems. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*, UAI'95, pages 394–402. Morgan Kaufmann Publishers Inc., 1995. ISBN 1-55860-385-9.

[48] D. J. Lizotte. *Practical Bayesian Optimization*. PhD thesis, University of Alberta, 2008.

[49] R. Martinez-Cantin, N. de Freitas, E. Brochu, J. Castellanos, and A. Doucet. A Bayesian Exploration-Exploitation Approach for Optimal Online Sensing and Planning with a Visually Guided Mobile Robot. *Autonomous Robots*, 27(2):93–103, 2009. ISSN 09295593.

[50] MetraLabs, 2017. URL `http://www.metralabs.com/en/shopping-rfid-robot`.

[51] T. P. Minka. Bayesian Inference, Entropy, and the Multinomial Distribution. 2003.

[52] T. M. Moldovan and P. Abbeel. Safe Exploration in Markov Decision Processes. In *Proceedings of the 29th International Conference on Machine Learning*, ICML'12, pages 1451–1458. Omnipress, 2012. ISBN 978-1-4503-1285-1.

[53] D. N. Nikovski and I. R. Nourbakhsh. Learning Discrete Bayesian Models for Autonomous Agent Navigation. In *Proceedings of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, CIRA'99, pages 137–143, 1999.

[54] D. N. Nikovski and I. R. Nourbakhsh. Learning Probabilistic Models for Decision-Theoretic Navigation of Mobile Robots. In *Proceedings of the 17th International Conference on Machine Learning*, ICML'00, pages 671–678. Morgan Kaufmann Publishers Inc., 2000. ISBN 1-55860-707-2.

[55] D. N. Nikovski and I. R. Nourbakhsh. *State-Aggregation Algorithms for Learning Probabilistic Models for Robot Control*. PhD thesis, Carnegie Mellon University, The Robotics Institute, 2002.

[56] F. M. F. Nogueira. BayesianOptimization: A Python Implementation of Global Optimization with Gaussian Processes, 2014. URL `https://github.com/fmfn/BayesianOptimization`.

[57] E. Parent, F. Lebdi, and P. Hurand. Stochastic Modeling of a Water Resource System: Analytical Techniques versus Synthetic Approaches. In *Water Resources Engineering Risk Assessment*, pages 415–434. Springer, 1991. ISBN 978-3-642-76971-9.

[58] A. Pasanisi, S. Fu, and N. Bousquet. Estimating Discrete Markov Models from Various Incomplete Data Schemes. *Computational Statistics & Data Analysis*, 56(9):2609–2625, September 2012. ISSN 0167-9473.

[59] J. Pazis. *Non-Parametric Approximate Linear Programming for MDPs*. PhD thesis, Duke University, 2012.

[60] S. Png and J. Pineau. Bayesian Reinforcement Learning for POMDP-Based Dialogue Systems. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP'11, pages 2156–2159. IEEE, May 2011.

[61] D. L. Poole and A. K. Mackworth. *Artificial Intelligence: Foundations of Computational Agents*. Cambridge University Press, 2010. ISBN 0521519004, 9780521519007.

[62] P. Poupart. *Bayesian Reinforcement Learning*, pages 90–93. Springer US, Boston, MA, USA, 2010. ISBN 978-0-387-30164-8.

[63] M. L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 2014.

[64] L. R. Rabiner. A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition. *Proceedings of the IEEE*, 77(2):257–286, February 1989. ISSN 0018-9219.

[65] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*, volume 1. MIT press Cambridge, 2006.

[66] J. K. Satia and R. E. Lave Jr. Markovian Decision Processes with Uncertain Transition Probabilities. *Operations Research*, 21(3):728–740, June 1973. ISSN 0030-364X.

[67] A. J. Schaefer, M. D. Bailey, S. M. Shechter, and M. S. Roberts. Modeling Medical Treatment Using Markov Decision Processes. In *Operations Research and Health Care: A Handbook of Methods and Applications*, pages 593–612. Springer US, Boston, MA, 2005. ISBN 978-1-4020-8066-1.

[68] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1): 148–175, January 2016.

[69] H. Shatkay and L. P. Kaelbling. Learning Hidden Markov Models with Geometric Information. Technical report, Brown University, 1997.

[70] D. Silver, R. S. Sutton, and M. Müller. Sample-based Learning and Search with Permanent and Transient Memories. In *Proceedings of the 25th International Conference on Machine Learning*, ICML'08, pages 968–975, New York, NY, USA, ACM, 2008. ISBN 978-1-60558-205-4.

[71] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári. Convergence Results for Single-Step On-Policy Reinforcement-Learning Algorithms. *Machine learning*, 38(3):287–308, March 2000. ISSN 0885-6125.

[72] J. Snoek, H. Larochelle, and R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms. In *Proceedings of the 25th International Conference on Neural Information Processing Systems*, NIPS'12, pages 2951–2959. Curran Associates Inc., 2012.

[73] A. Stolcke and S. M. Omohundro. Best-First Model Merging for Hidden Markov Model Induction. Technical report, International Computer Science Institute, Berkeley, CA, USA, 1994.

[74] R. S. Sutton and A. G. Barto. *Reinforcement learning: An Introduction*, volume 1. MIT press Cambridge, 1998.

[75] I. Teodorescu. Maximum Likelihood Estimation for Markov Chains. *arXiv preprint arXiv:0905.4131*, May 2009.

[76] M. Turchetta, F. Berkenkamp, and A. Krause. Safe Exploration in Finite Markov Decision Processes with Gaussian Processes. In *Proceedings of the 29th International Conference on Neural Information Processing Systems*, NIPS'16, pages 4305–4313, 2016.

[77] Z. Wang, B. Zhou, and S. Jegelka. Optimization as Estimation with Gaussian Processes in Bandit Settings. In *Artificial Intelligence and Statistics*, AISTATS'16, pages 1022–1031, 2016.

[78] C. J. C. H. Watkins and P. Dayan. Q-Learning. *Machine learning*, 8(3-4):279–292, 1992.

[79] L. R. Welch. Hidden Markov Models and the Baum-Welch Algorithm. *IEEE Information Theory Society Newsletter*, 53(4):10–13, 2003.

[80] M. A. Wiering, J. van Veenen, J. Vreeken, and A. Koopman. Intelligent Traffic Light Control, 2004.

[81] A. Wilson, A. Fern, and P. Tadepalli. Using Trajectory Data to Improve Bayesian Optimization for Reinforcement Learning. *Journal of Machine Learning Research*, 15(1): 253–282, 2014.

[82] S. Young, M. Gašić, B. Thomson, and J. D. Williams. POMDP-Based Statistical Spoken Dialog Systems: A Review. *Proceedings of the IEEE*, 101(5):1160–1179, May 2013. ISSN 0018-9219.