Institut für Parallele und Verteilte Systeme

Abteilung Maschinelles Lernen und Robotik

Universität Stuttgart
Universitätsstraße 38
D - 70569 Stuttgart

Master Thesis

Complexity Analysis of Meta Decision Processes

Li Yang Wu

Studiengang:	M.Sc. Informatik
Prüfer:	Prof. Dr. rer. nat. Marc Toussaint
Betreuer:	Hung Ngo
begonnen am:	05.04.2017
beendet am:	05.10.2017
CR-Klassifikation:	I.2.1, I.2.4, I.2.9, I.2.11, F.2.2

Contents

Abstract

This thesis investigates human-robot interactions focusing on theoretical reasoning about a general formalism that is suitable to describe most HRI systems. We belief that the question whether to ask the human for help or not is a decision that should be made at the meta level. This work summarizes the milestones in the literature that contribute to HRI, compares different MDP concepts to choose from and proposes a new formalism along with several analysis' on its properties.

1. Introduction

Technological advancements has always been a characteristic during the ongoing process of human civilization. One of the main goals thereby is always automation. An early example from ancient Rome is the aqueduct that transports drinking water from natural sources directly to the cities. In the recent past of human history, industrialization had an huge impact on humanity and redefined the way how human live. Today, we look forward to technical systems that do not only replace most manual labour, but assist the human with intellectual work as well. *Industry 4.0* [8] include cooperative tasks solving between human and robot, where decision making and learning is a central task of the robot. The human takes the role of an expert or teacher and provide the agent(s) with useful data, which can be used to retrieve the objectives of a task. We assume that the true objectives cannot simply be hard coded because for many task fields it is impossible to do that in terms of problem complexity. Often, objectives involve human needs, and this can already be as complex as the human psychology, which itself is by far not fully understood.

The problem of retrieving the reward function (in Reinforcement Learning terminology) is well studied by the name of *Inverse Reinforcement Learning*. While the Sections 2 and 3 can be skipped by the readers that are familiar with Reinforcement Learning, Section 4 provide a summary of some milestones in the research on Human-Robot Interaction which originates from the research on Inverse Reinforcement Learning. Sections 6 and 7 compare an formalism called RAP that is capable of describing multi-agent real time systems to others and show that it is most general and suitable for our purposes. In Section 8 we propose our own formalism which is a two layered meta process that decides to act or to collect meta data (e.g. from human). The execution layer is a semi Markov process for which we want to find the unknown components like the reward function. The meta layer is a special RAP, where the set of decisions are predefined. A one-step-look-ahead algorithm is presented and analysed. A summary and discussion finishes this work in Section 9.

2. Notations

- $\mathbb{P}$, probability distribution.

- $\mathbb{E}$, expectation over distributions.
- $\mathcal{O}$, O-notation according to [Landau, 1924].
- $\mathbb{F}, \mathbb{N}, \mathbb{Z}, \mathbb{R}$, binary, natural, integer and real numbers. $0 \in \mathbb{N}$.

3. Background

This section presents some mathematical concepts used throughout the field of study of Reinforcement Learning.

3.1. Bayes' Rule

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space, $A \in \mathcal{F}$ with $\mathbb{P} > 0$ and $(B_i)_{i \in I} \subset \mathcal{F}$ a Partition of Ω. Then for all $i \in I$ it holds

$$\mathbb{P}(B_i \mid A) = \frac{\mathbb{P}(A \mid B_i) \cdot \mathbb{P}(B_i)}{\mathbb{P}(A)} \tag{3.1}$$

$$= \frac{\mathbb{P}(A \mid B_i) \cdot \mathbb{P}(B_i)}{\sum_{k \in I} \mathbb{P}(A \mid B_k) \cdot \mathbb{P}(B_k)} \tag{3.2}$$

We yield the second line using the rule of total probabilities [**?**].

3.2. Markov Decision Process

General decision making take into account the full history of observations. The Markov property assumes that all information that is needed is encoded in the current state, reducing the problem complexity drastically. It turns out that many environments can be modelled well under this assumption. Note that states can be arbitrarily composed with Cartesian products of sets, making a Markov state to be able to represent more than merely a scalar value. Formally, a *Markov Decision Process* (MDP) [14] can be described as a tuple

$$M = (S, A, T, \gamma, S_0, R) \tag{3.3}$$

with $S = \{s_i\}_{i=1}^n$, the set of states, $A = \{a_i\}_{i=1}^m$, the set of actions, $T : S \times A \to S$, the state transition function, $\gamma \in [0, 1]$, the discount factor, $S_0 \subseteq S$, the set of initial states, $R : S \to \mathbb{R}$, the reward function. We can easily express stochasticity of the environment by turning T and/or R into stochastic functions; for that, define distributions $\mathbb{P}(s' \mid s, a)$ and $\mathbb{P}(r \mid s)$ respectively. The discount factor γ penalizes reward collected in the far future during the planning phase. For many infinite horizon problems, it is even necessary for converging to a fixed plan (e.g. Q-function).

3.2.1. Partial Observability

In many environments the state is not fully observable for the agent. In order to model this situation, we can extend the MDP formulation by the notion of partial observability [14]. After completion of an action, the agent receives an observation signal o instead of the next state s' with probability $\mathbb{P}(o \mid s')$. The agent can use this information to update its *belief* $b(s')$ over states, which is a probability distribution over all states, by

$$b'(s') = \alpha \mathbb{P}(o \mid s') \sum_{s'} \mathbb{P}(s' \mid s, a) b(s) \,, \tag{3.4}$$

using its prior $b(s)$. α is a normalization constant to normalize the distribution to 1. For prediction, we need to sum over all possible weighted observations, i.e.

$$\begin{array}{rcl} \mathbb{P}(b' \mid b, a) & = & \sum_o \mathbb{P}(b' \mid o, a, b) \mathbb{P}(o \mid a, b) \\ & = & \sum_o \mathbb{P}(b' \mid o, a, b) \sum_{s'} \mathbb{P}(o \mid s') \sum_s \mathbb{P}(s' \mid s, a) b(s) \,. \end{array} \tag{3.5}$$

Formally, a *Partial Observable Markov Decision Process* (POMDP) is a tuple

$$M = (S, A, T, \Omega, O, \gamma, S_0, R) \tag{3.6}$$

where Ω is the set of all possible observations and $O : S \rightarrow \Omega$ the observation function.

3.2.2. Distributed Multi-Agents

[Bernstein et al., 2000] formulated a generalization to POMDPs for multi-agent environments that do not share information, so that exchanging information is associated with some cost. This formalism is called *Decentralized Partial Observable Markov Decision Process* (Dec-POMDP). Given m agents each with their own observations, the reward function, the state transition probabilities and the observations are as follows:

$$\begin{array}{l} R(s, a^1, ..., a^m), \\ \mathbb{P}(s' \mid s, a^1, ..., a^m), \\ O(s) \sim \mathbb{P}(o^1, ..., o^m \mid s, a^1, ..., a^m) \,. \end{array} \tag{3.7}$$

They showed that finite horizon Dec-POMDP problems are NEXP-hard, where NEXP $=$ $\mathrm{NTIME}(2^{n^c})$. Since P $\neq$ NEXP, there exists no polynomial algorithm to solve Dec-POMDP, probably not even exponential algorithms as most people believe P $\neq$ NP.

3.2.3. Continuous Time

Furthermore we can extend any MDP formalism with continuous time were each action a performed in a certain state s takes a certain amount of time τ with probability $\mathbb{P}(\tau \mid s, a)$. This is called a *semi-Markov decision process* (sMDP). Things get more interesting when sMDPs are used for multi-agent tasks. In this case, it is not clear what a "Markov step" exactly is and thus needs to be defined in a way that is consistent with the Markov assumption. One of these models is described in Section 4.8.

3.3. Bellman Equation

The optimality principle of Bellman, first presented in [Bellman, 1956], says that solutions from certain optimization problems can be constructed by partial solutions. In the case of Reinforcement Learning, where we try to maximize the expected return, solutions are sequences of actions. Each sequence $\{(s_i, a_i, r_i)\}_{i=0}^{M}$ generated by a policy π can be constructed by their sub-sequences, showed by

$$\begin{aligned} V^\pi(s) &= \mathbb{E}_\pi\left[r_0 + \gamma r_1 + \gamma^2 r_2, ... \mid s_0 = s\right] \\ &= \mathbb{E}_\pi\left[r_0 \mid s_0 = s\right] + \gamma \mathbb{E}_\pi\left[r_1 + \gamma r_2 +, ... \mid s_0 = s\right] \\ &= R(s,a) + \gamma \sum_{s'} \mathbb{P}(s' \mid \pi(s), s) \mathbb{E}_\pi\left[r_1 + \gamma r_2 +, ... \mid s_1 = s'\right] \\ &= R(s,a) + \gamma \sum_{s'} \mathbb{P}(s' \mid \pi(s), s) V^\pi(s'). \end{aligned} \tag{3.8}$$

Note that if $\pi = \pi^*$ is the optimal policy, then $V^\pi = V^{\pi^*}$ is optimal as well.

3.4. First-Order Logic

First-Order Logic (FOL) [15] is a very expressive model to describe problems based on mathematical logic principles. It extends *Propositional Logic* (or Boolean Logic) by setting relations between objects and make general statements *for all* values of some variable x_i or stating the *existence* of some objects to fulfil a statement. It is the best to define syntax and semantics separately.

3.4.1. Syntax of FOL

A *variable* has the form x_i, where $i = 1, 2, 3, ...$. A *predicate* (or *relation*) is denoted as P_i^k and a *function* as f_i^k with $i = 1, 2, 3, ...$ as the *distinctive index* and $k = 0, 1, 2, ...$ as the *adicity*. Adicities can be neglected if it follows from the context. First we define a *term* in an inductive manner:

1. Every variable is a term.
2. If f is a function symbol with adicity of k and $t_1, ..., t_k$ are terms, then $f(t_1, ..., t_k)$ is a term.

Here we also include functions with adicity of 0. In such a case, the brackets can be omitted. We call those functions *constants*.

Now we can define what a (FOL) *formula* is, also inductive, as follows:

1. If P is a predicate with adicity of k and $t_1, ..., t_k$ are terms, then $P(t_1, ..., t_k)$ is a formula.
2. For each formula F, $\neg F$ is a formula.
3. For all formulas F and G, $(F \wedge G)$ and $(F \vee G)$ are formulas.
4. If x is a variable and F is a formula, then $\forall x F$ and $\exists x F$ are formulas.

Atomic formulas are those that are defined according to 1. If F is a formula and it is part of another formula G, then F is a *sub-formula* of G.

For variables used in formulas, we distinguish between *bound* and *free* variables. A Variable x in F is bound if F contains a sub-formula G that has the form $\forall x G'$ or $\exists x G'$; they are called free otherwise. Note that by this definition, one single variable can have both free and bound occurrences in a formula. A formula without free variables is called a *sentence*. The symbols $\forall$ and $\exists$ are called *universal quantifier* and *existential quantifier* respectively.

EXAMPLE One example of an FOL formula is a criteria for the Markov property:

For all $A \in S$ and all $s, t \in I$ with $s < t$, $\mathbb{P}(X_t \in A \mid \mathcal{F}_s) = \mathbb{P}(X_t \in A \mid X_s)$.

The set of predicates used here is $\{\in, <, =\}$, where each of them can be written in our standard notation as binary predicates $\{P_1^2, P_2^2, P_3^2\}$. The same can be done for the two probability distribution functions, which results in two binary functions $\{f_1^2, f_2^2\}$. The full formula written in the standard notation is then:

$$\forall x_1 \forall x_2 \forall x_3 (((((P_1^2(x_1, x_4) \land P_1^2(x_2, x_5)) \land P_1^2(x_2, x_5)) \land P_2^2(x_2, x_3)) \land P_1^2(x_6, x_1)) \land \\ P_3^2(f_1^2(x_6, x_7), f_2^2(x_6, x_8))) \ .$$

As we can see, the standard notation can be quite lengthy and hard to read (but makes it machine-parsable). Therefore it makes sense to introduce own symbols to write down FOL formulas. Typically, for variables we use $x, y, z, ...$, for predicates $P, Q, R, ...$, for functions $f, g, h, ...$ and for constants $a, b, c, ...$.

Until now, formulas are only some strings of symbols without any meaning. With the semantics definition, we interpret function symbols as functions and predicate symbols as predicates based on their underlying sets each. Additionally, some free variables needs to be interpreted as an element of such sets. The semantics is defined next.

3.4.2. SEMANTICS OF FOL

A *structure* is a tuple $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$, where $U_\mathcal{A}$ is an arbitrary but non-empty set, denoted as the *universe* of $\mathcal{A}$. $I_\mathcal{A}$ is a mapping that maps predicate and function symbols and variables to actual useful mathematical objects. The domain of $I_\mathcal{A}$ is therefore the set of all these symbols, i.e. $\{P_i^k, f_i^K, x_i \mid i = 1, 2, 3, ... \text{ and } k = 0, 1, 2, ...\}$. More precisely, $I_\mathcal{A}$ maps

- each k-ary predicate symbol (that lies in the domain of $I_\mathcal{A}$) to a k-ary predicate over $U_\mathcal{A}$,
- each k-ary function symbol (that lies in the domain of $I_\mathcal{A}$) to a k-ary function over $U_\mathcal{A}$,
- each variable x (if $I_\mathcal{A}$ is defined over x) to an element in $U_\mathcal{A}$.

Instead of writing $I_\mathcal{A}(P)$, $I_\mathcal{A}(f)$ and $I_\mathcal{A}(x)$, we abbreviate with $P^\mathcal{A}$, $f^\mathcal{A}$ and $x^\mathcal{A}$. Let F be a formula and $\mathcal{A}$ a structure, then $\mathcal{A}$ is *fit* for F if the domain of $\mathcal{A}$ covers all predicate and function symbols and all free variables in F.

EXAMPLE For a formula $F = \exists x P(f(x), y)$ we choose the following fitting structure $\mathcal{A} = (U_{\mathcal{A}}, I_{\mathcal{A}})$ with

$$\begin{array}{lcl} U_{\mathcal{A}} & = & \mathbb{N}, \\ P^{\mathcal{A}} & = & \{(m,n) \mid m,n \in U_{\mathcal{A}} \text{ and } m > n\}, \\ f^{\mathcal{A}} & = & \text{smallest prime factor over } U_{\mathcal{A}}, \\ y^{\mathcal{A}} & = & 2. \end{array}$$

Under this structure, F evaluates to true (for instance $x = 21$, then $f(21) = 3$ and $3 > 2$), but it might not be the case for other structures.

We continue with defining the value of a term. Let F be a formula and $\mathcal{A}$ a fitting structure. Then each term t appearing in F has a value $\mathcal{A}(t)$ defined inductively:

1. If t is a variable (say $t = x$), then $\mathcal{A}(t) = x^{\mathcal{A}}$.
2. If t has the form $t = f(t_1, ..., t_k)$, where $t_1, ..., t_k$ are terms and f is a k-ary function symbol, then $\mathcal{A}(t) = f^{\mathcal{A}}(\mathcal{A}(t_1), ..., \mathcal{A}(t_k))$.

In the same way we define the truth value of a formula F under structure $\mathcal{A}$. We again use the notation $\mathcal{A}(F)$ to denote the value.

1. If F has the form $F = P(t_1, ..., t_k)$, with the terms $t_1, ..., t_k$ and the k-ary predicate P, then
$$\mathcal{A}(F) = \begin{cases} 1, & \text{if } (\mathcal{A}(t_1), ..., \mathcal{A}(t_k)) \in P^{\mathcal{A}} \\ 0, & \text{else} \end{cases} \tag{3.9}$$
2. If F has the form $F = \neg G$, then
$$\mathcal{A}(F) = \begin{cases} 1, & \text{if } \mathcal{A}(G) = 0 \\ 0, & \text{else} \end{cases} \tag{3.10}$$
3. If F has the form $F = G \wedge H$, then
$$\mathcal{A}(F) = \begin{cases} 1, & \text{if } \mathcal{A}(G) = 1 \text{ and } \mathcal{A}(H) = 1 \\ 0, & \text{else} \end{cases} \tag{3.11}$$
4. If F has the form $F = G \vee H$, then
$$\mathcal{A}(F) = \begin{cases} 1, & \text{if } \mathcal{A}(G) = 1 \text{ or } \mathcal{A}(H) = 1 \\ 0, & \text{else} \end{cases} \tag{3.12}$$
5. If F has the form $F = \forall x G$, then
$$\mathcal{A}(F) = \begin{cases} 1, & \text{if for all } d \in U_{\mathcal{A}} \text{ it holds: } \mathcal{A}_{[x/d]}(G) = 1 \\ 0, & \text{else} \end{cases} \tag{3.13}$$

6. If F has the form $F = \exists x G$, then

$$\mathcal{A}(F) = \begin{cases} 1, & \text{if there exists at least one } d \in U_{\mathcal{A}} \text{ such that: } \mathcal{A}_{[x/d]}(G) = 1 \\ 0, & \text{else} \end{cases} \tag{3.14}$$

The notation $\mathcal{A}_{[x/d]}$ is defined as a new structure $\mathcal{A}'$ that is everywhere identical with $\mathcal{A}$, except for the definition of $x^{\mathcal{A}'}$, which is defined as $x^{\mathcal{A}'} = d$, where $d \in U_{\mathcal{A}'} = U_{\mathcal{A}}$. It does not matter whether the original interpretation $I_{\mathcal{A}}$ is defined over x or not.

3.4.3. FOL Complexity

More expressive models (like Second-Order Logic) are not needed because many problems, even the Satisfiability problem, are undecidable in FOL already. In fact, subclasses of FOL are used more often in practical computer science, for example logic programming in PROLOG.

3.5. Relational MDP Domains

The easiest way to describe components of a MDP is using propositional objects, where everything is attribute-valued. The consequence is that any solution based on Q-learning needs to build a huge lookup table storing values for the entire state-action pair space. Additionally, if altering the problem instance slightly, the whole table must be relearned. [Džeroski et al., 2001] combines *Inductive Logic Programming* and *Reinforcement Learning* to achieve a more flexible and general representation, where MDP components are described in FOL. Typically, a restricted form of FOL is used because the satisfiability problem for FOL is already semi-decidable. (For comparison: It is NP-complete for propositional logic.) Now the Q-values can be encoded sparsely in *Decision Trees*, that has furthermore the property of abstracting from concrete objects and using variables to encode knowledge about classes of objects, which is more general and reusable.

Since the state consists of all constants and truth values of predicates, it can be formally defined as the structure $\mathcal{A}$ (Section 3.4). A state transition in our reinforcement learning setting means to change the structure, transforming it into another structure $\mathcal{A}'$, where in fact only the mapping $I_{\mathcal{A}}$ changes; the universe $U_{\mathcal{A}}$ is typically fix, e.g. $\mathbb{N}^n$ for discrete domains and $\mathbb{R}^n$ for continuous domains. The only mappings that actually change are those for variables x_i. For example one of the variables could be the position of the robot's endeffector. Functions (like calculating Euclidean distances) do typically not change. Predicates also do not change, as the uncertainty of its truth value is already encoded in the uncertainty of variable values. For example, a predicate that tells, whether an object is above another object should be designed correctly from begin with. The uncertainty of its truth value comes from the uncertainty of the exact position of the objects.

In order to test FOL for reinforcement learning, two kinds of experiments were done in [Džeroski et al., 2001]. In the first experiment, a decision tree was learned from one problem instance and used for different problem instances. In the second experiment, a decision tree was learned while the problem size increased multiple times during the learning phase. A near optimal policy could always be learned which shows the generality of using relational domains. But there are differences compared to propositional domains. One advantage is that problem instances of large size can be learning more efficiently given a solution for smaller problems. The disadvantage, however, is that the knowledge base needs to be tuned well when the task to be solved is very complex, otherwise the results are not guaranteed to converge to the desired outcome regardless of how much computation time is used. I personally expect that relational domains will be used for all reinforcement learning problems as soon as "predicate crafting" is well studied.

4. Literature Review

This section gives an overview of the development and milestones of *Inverse Reinforcement Learning* (IRL) methods. A novel class of problem was then formulated by combining IRL with cooperative learning and task solving. Cooperation was also studied in the field of *Reinforcement Learning* (RL) in terms multi-agents that perform actions in a concurrent manner. For each of these methods I summarize the problem formulation, the solutions and the results respectively. Although the basic idea about the problem of IRL are all the same, the *assumptions* made about the system might differ.

4.1. Inverse Reinforcement Learning

Reinforcement Learning (RL) is a large class of problems where the tasks is to find an optimal behaviour of an agent acting in an environment by maximizing the return, or, if stochasticity is involved, the expected return. Subclasses of RL typically involve modifications of the system components, for example adding stochasticity or partial observability of state transitions. The common assumption thereby is that the *reward* can always be observed. This information is crucial since there is no other way to decide which behaviour is better or worse. The problem around this situation is that the reward function is not naturally given but *hard coded* by the system designer. In practice it can be difficult to craft the reward function that is precise and robust enough to suit our needs. In fact the agent can even behave noxiously if the reward function is wrongly specified. Imagine we want to turn off a robot because it starts to cause some damage. But the robot prevents us from doing so because it learned that turning it off prevents it from collecting additional reward. *Value Alignment* is the keyword nowadays to refer to this issue.

Therefore [Russell, 1998] first formulated a problem where the goal is to recover the reward function of the system of interest after observing the *behaviour* of an expert. Under the assumption of optimization it is well defined and can be written as the dual problem of unsupervised Reinforcement Learning, thus naming it Inverse Reinforcement Learning. In the following I present the first algorithms proposed by [Ng et al., 2000] to solve IRL.

4.1.1. Problem Formulation

Given:

- a MDP with unknown reward function $R(s)$.
- $\{(s_i,\ \pi^*(s_i),\ s_{i+1})_{i=1}^d \mid d \in D,\ D \subset \mathbb{N},\ D$ finite$\}$, a set of demonstrations where π^* denotes the experts behaviour.
- optionally: $\{(m(s_i))_{i=1}^d \mid d \in D,\ D \subset \mathbb{N},\ D$ finite$\}$, any measurements from some sensors.
- if available: $\mathbb{P}(s' \mid s, a)$, the model of the environment.

Determine:

- $R(s)$, the reward function being optimized.

There will be many degenerate solutions found. Use heuristics to find an unique solution.

4.1.2. Solutions

Simple Case The first algorithm solves the problem where the transition model $\mathbb{P}(s, a, s')$ and the experts policy $\pi^*(s)$ is fully known. Also, the state space is finite. The following optimality criteria is used to find a solution space, where all solutions within that space is consistent with the given behaviour.

Theorem 4.1. *Let* $|S| < \infty$*,* $A = \{a_1, ..., a_k\}$*,* $\boldsymbol{P}_a$ *probability transition matrices and* $\gamma \in (0, 1)$ *a discount factor. Then*

$$(\boldsymbol{P}_{\pi(s)} - \boldsymbol{P}_a)(\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi(s)})^{-1} R \succeq 0 \tag{4.1}$$

In order to find a solution that is not degenerated, they prefer solutions for which the best action differentiates the most from the second best action, thus adding the objective term:

$$\sum_{s \in S} \left(Q^\pi(s, \pi(s)) \ - \max_{a \in A \setminus \pi(s)} Q^\pi(s, a) \right) \tag{4.2}$$

They also prefer smaller simpler rewards, thus adding the following penalty:

$$-\lambda \|R\|_1 \tag{4.3}$$

While λ can be hand-tuned, one can also compute a certain λ_0 as large as possible such that it is just small enough to prevent $R(s) = 0,\ \forall s \in S$.

The resulting ILP problems is then as follows:

$$\begin{array}{rll} \max & \sum_{i=1}^N \min_{a \in A \setminus \pi(s)} \{(\boldsymbol{P}_{\pi(s)}(i) - \boldsymbol{P}_a(i))(\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi(s)})^{-1} \boldsymbol{R}\} - \lambda ||R||_1 & \\ \text{s.t.} & (\boldsymbol{P}_{\pi(s)}(i) - \boldsymbol{P}_a(i))(\boldsymbol{I} - \gamma \boldsymbol{P}_{\pi(s)})^{-1} \boldsymbol{R} \succeq 0 & \forall a \in A \setminus \pi(s) \\ & |R_i| \leq R_{\max}, & i = 1, ..., N \end{array} \tag{4.4}$$

INFINITE STATE SPACE Now they model the state space as $S = \mathbb{R}^n$ for sake of correctness. In order to optimize efficiently, they use linear function approximations to describe the reward function:

$$R(s) = \alpha_1\phi_1(s) + ... + \alpha_d\phi_d(s) \tag{4.5}$$

Linear programming can be used here since linear parameters are optimized. Also, because each $\phi_1(s)$ induces a value function $V_i^\pi(s)$ and because of the linearity of expectations, the resulting value function is:

$$V^\pi(s) = \alpha_1 V_1^\pi(s) + ... + \alpha_d V_d^\pi(s) \tag{4.6}$$

The new optimality criteria is now:

$$\mathbb{E}_{s'\sim P(s'|s,\pi(s))}[V^\pi(s')] \geq \mathbb{E}_{s'\sim P(s'|s,a)}[V^\pi(s')], \quad \forall a \in A \setminus \pi(s) \tag{4.7}$$

This results in an infinite amount of constraints to check, but in practice only a finite subset $S_0 \subset S$ is chosen for optimization. A second problem arises as they use linear function approximation. The true reward function can in general not be expressed. They solve this by relaxing the constraints and add a penalty if a constraint is not met. The resulting linear program is:

$$\begin{array}{ll} \max & \sum_{s\in S_0} \min_{a\in A\setminus\pi(s)} p(\mathbb{E}_{s'\sim P(s'|s,\pi(s))}[V^\pi(s')] - \mathbb{E}_{s'\sim P(s'|s,a)}[V^\pi(s')]) \\ s.t. & |\alpha_i| \leq 1, \quad i = 1, ..., d \end{array} \tag{4.8}$$

where $p(x) = x$ if $x \geq 0$ and $p(x) = 2x$ if $x \leq 0$ is the penalty mechanism with a weight of 2. This value is heuristic as experiments have shown that the result does not change notably by increasing the penalty.

UNKNOWN REWARD FUNCTION In this more realistic assumption, only trajectories from the policy can be observed while the policy itself is not given. There is no need to explicitly model the MDP. The solution does not find an optimal policy for a given reward function, only an approximation. Let S_0 be the set of initial states. The goal is to find R such that the policy maximizes $\mathbb{E}_{s_0\sim S_0}[V^\pi(s_0)]$. W.l.o.g. they assume that $s_0 \in S$ is the only start state. As in the previous algorithm, R and V are described by a linear function approximation. They use Monte Carlo simulations to estimate V as follows. For each trajectory with state sequence $(s0, s1, ...)$, a single component is estimated by

$$\hat{V}_i^\pi(s_0) = \phi_i(s_0) + \gamma\phi_i(s_1) + \gamma^2\phi_i(s_2) + ... \tag{4.9}$$

and then averaged over all trajectories. And the resulting value functions is

$$\hat{V}^\pi(s) = \alpha_1\hat{V}_1^\pi(s) + ... + \alpha_d\hat{V}_d^\pi(s). \tag{4.10}$$

The algorithm is an iterative algorithm that finds a new reward function R_i along with a new policy π_i optimizing R_i. It starts with computing value estimates as described above for the experts policy π^* and for a random policy π_1. In each iteration, given a set of policies $\{\pi_1, ..., \pi_k\}$ they optimize over the following criteria

$$\hat{V}^{\pi^*}(s) \geq \hat{V}^{\pi_i}(s), \quad i = 1, ..., k, \tag{4.11}$$

using the fact that the observed behaviour is optimal, with the following linear program

$$\begin{aligned} \max \quad & \textstyle\sum_{i=1}^{k} p\big(\hat{V}^{\pi^*}(s) - \hat{V}^{\pi_i}(s)\big) \\ s.t. \quad & |\alpha_i| \leq 1, \quad i = 1, ..., d \end{aligned} \tag{4.12}$$

and obtain a new reward function $R = \alpha_1\phi_1 + ... + \alpha_d\phi_d$ and a new policy π_{k+1} that maximizes $V^{\pi}(s_0)$ under R. Finally, add π_{k+1} to the set of policies to finish the current iteration.

At each iteration a better policy can be found. Iterate until the solution satisfies your needs.

4.1.3. Results

The methods were successfully applied on small discrete and continuous problems. For the iterative algorithm, it could be shown that it converges to the true reward function.

4.2. Linear Function Approximation

Like in the previous solutions, [Abbeel and Ng, 2004] model the reward function by a linear function approximation, reducing the problem complexity. The authors provide novel algorithms to improve convergence speed. Although this approach can only lead to local optima in general, the results show that the RL agent trained with the resulting reward function performs sufficiently well.

4.2.1. Solution Schema

Approach:

- Given MDP$\backslash R = (S, A, T, \gamma, S_0)$, a MDP without the reward function which is hidden and only known by the expert π_E, which need not to be optimal.
- Rewrite value of a given policy: $\mathbb{E}_{s_0 \sim S_0}[V^{\pi}(s_0)] = w \cdot \mu(\pi)$, where $\mu(\pi) = \mathbb{E}[\sum_{t=0}^{\infty} \gamma^t \phi(s_t) \mid \pi] \in \mathbb{R}^k$.
- Use Monte Carlo simulation to obtain m trajectories $\{s_0^{(i)}, s_1^{(i)}, \dots\}_{i=1}^{m}$.
- Estimate policy of the expert by $\hat{\mu}_E = \frac{1}{m}\sum_{i=1}^{m}\sum_{t=0}^{\infty}\gamma^t\phi(s_t^{(i)})$

Determine:

- $R(s) = w \cdot \phi(s)$, the linear function approximation of reward functions with weights $w \in \mathbb{R}^k$ to be learn and features $\phi : S \to [0,1]^k$ given.

- $\tilde{\pi}$ such that $\|\mu(\tilde{\pi}) - \mu_E\|_2 \leq \epsilon$.
- Then, for any normalized w with $\|w\|_1 \leq 1$ it holds that $V^{\tilde{\pi}}(s) - V^{\pi_E}(s) \leq \epsilon, \forall s \in S$

Thus, they try to find reward functions for which the optimized policy behaves as similar as possible to the expert's policy. This has the advantage that it is not necessary to specify the true reward function in order to measure performance.

4.2.2. Algorithm

Initialize by sampling from a random policy and calculate its value over the trajectory. At each iteration i find a new weight $w^{(i)}$ that maximizes the margin $t^{(i)}$ between the experts value μ_E and the values of all policies generated so far $(\mu(\pi_0), \ldots, \mu(\pi^{(i-1)}))$ by solving it as a quadratic programming problem (QP). If $t^{(i)} \leq \epsilon$ terminate, otherwise compute $\mu(\pi_i)$ as a RL problem with $w^{(i)}$ and add it to the policy set, then iterate again.

Selecting the best policy out of the returned set can be done manually by the human agent designer or computationally by yet another QP which minimizes the distance between μ_E and the $\mu(\pi^{(i)})$'s.

Instead of maximizing the margin to find the next weight, the authors provide a projection method as an alternative that avoids solving an QP. Both methods were tested against each other.

4.2.3. Results

Complexity Analysis The authors provide the runtime complexity of the algorithm as

$$\mathcal{O}\left(\frac{k}{(1-\gamma)^2\epsilon^2} \log \frac{k}{(1-\gamma)\epsilon}\right) \tag{4.13}$$

and the sampling complexity for estimating μ_E to achieve the quality $V^{\tilde{\pi}}(s) - V^{\pi_E}(s) \leq \epsilon, \forall s \in S$ for the resulting policy $\tilde{\pi}$ within the given runtime constraints with a probability of at least $1 - \delta$ as

$$m \geq \frac{2k}{(\epsilon(1-\gamma))^2} \log \frac{2k}{\delta} \tag{4.14}$$

and the error bound

$$\mathcal{O}(\|\epsilon\|_\infty) \tag{4.15}$$

if $R^*(s) = w^* \cdot \phi(s) + \epsilon(s)$ for some residual error term $\epsilon(s)$.

First Experiment Use a 128×128 gird world with non-overlapping 64 "macrocells" to show that

- the projection method converges slightly faster than the max-margin method.
- the proposed methods outperform any kind of mimic learning, in a sense that they need far less amount of trajectories of the expert.

SECOND EXPERIMENT In a simple car drive simulation, it could be shown that the algorithms could learn different driving styles given a demonstration of 2 minutes each. In this experiment, a true reward function was never designed but the recovered reward function by the algorithm is very intuitive.

It could also be shown that the performance can be increased if only the non-zero weight features are used.

4.3. BAYESIAN INVERSE REINFORCEMENT LEARNING

The Bayesian approach is motivated by two problems: First, how to avoid over-fitting the data? Second, how to choose non-degenerate solutions? Explicitly expressing the uncertainty over observations and estimating the mean of the distribution over possible rewards can solve these problems directly. [Ramachandran and Amir, 2007] investigated this matter and give solutions to solve IRL in the Bayesian setting.

4.3.1. PROBLEM FORMULATION

Given:

- MDP = (S, A, T, γ, R), the Markov Decision Problem.
- MDP = (S, A, T, γ), the Markov Decision Process. This definition is meant by default if mentioning MDP here.
- $\boldsymbol{O}_X = \{O_X^{(i)}\}_{i=1}^m = \{(s_1^{(i)}, a_1^{(i)}), \ldots, (s_k^{(i)}, a_k^{(i)})\}_{i=1}^m$ the set of *evidences* performed by the expert X.
- Expert X is greedy and stationary.
- $\mathbb{P}_X(O_X \mid \boldsymbol{R}) = \mathbb{P}_X((s_1, a_1) \mid \boldsymbol{R}) \ldots \mathbb{P}_X((s_k, a_k) \mid \boldsymbol{R})$, with $\boldsymbol{R}$ being the unknown true reward function for $O_X \in \boldsymbol{O}_X$.
- $\mathbb{P}_X((s_i, a_i) \mid \boldsymbol{R}) = \frac{1}{Z_i} e^{\alpha_i Q^*(s_i, a_i, \boldsymbol{R})}$ to express the confidence α_i that X chooses a good action.
- $\mathbb{P}_X(O_X \mid \boldsymbol{R}) = \frac{1}{Z} e^{\alpha_X \mathbb{E}(O_X, \boldsymbol{R})}$ with $\mathbb{E}(O_X, \boldsymbol{R}) = \sum_i Q^*(s_i, a_i, \boldsymbol{R})$

Determine:

- Recover R as *reward learning*.
- Solve MDP given $\boldsymbol{O}_X$ as *apprenticeship learning*.
- Solve $\mathbb{P}_X(\boldsymbol{R} \mid O_X) = \frac{\mathbb{P}_X(O_X \mid \boldsymbol{R}) \mathbb{P}_R(\boldsymbol{R})}{Pr(O_X)} = \frac{1}{Z} e^{\alpha_X \mathbb{E}(O_X, \boldsymbol{R})} \mathbb{P}_R(\boldsymbol{R})$ according to Bayes' theorem.

Choose prior i.i.d. if no domain knowledge is available. Many problems have only positive reward at very few states. Therefore a Gaussian, Laplace or Beta prior is appropriated if such an assumption can be made.

4.3.2. Solutions

Reward Learning Solve the estimation task with the following loss functions from which one can choose:

$$L_{\text{linear}}(\boldsymbol{R}, \hat{\boldsymbol{R}}) = \|\boldsymbol{R} - \hat{\boldsymbol{R}}\|_1 \tag{4.16}$$
$$L_{\text{SE}}(\boldsymbol{R}, \hat{\boldsymbol{R}}) = \|\boldsymbol{R} - \hat{\boldsymbol{R}}\|_2. \tag{4.17}$$

Set $\hat{\boldsymbol{R}}$ to the mean of the posterior for L_{SE} or to the median of the distribution for L_{linear}. Both can be acquired by the Policy Walk Sampling Algorithm described below.

Apprenticeship Learning Solve with the model shown above by minimizing a policy loss function from the following class:

$$L^p_{\text{policy}}(\boldsymbol{R}, \pi) = \|V^*(\boldsymbol{R}) - V^\pi(\boldsymbol{R})\|_p \tag{4.18}$$

The authors have shown that instead of minimizing the expected policy loss directly which is a difficult task, one can also find optimal policy for the mean reward function which produces the same result. This again can be acquired by the Policy Walk Sampling Algorithm.

Policy Walk Sampling Algorithm The algorithm works roughly as follows:

- Initialize by choosing a random reward and apply policy iteration accordingly.
- Iterate by picking a new reward $\tilde{\boldsymbol{R}}$ close to the old reward $\boldsymbol{R}$. Compute the new Q-function. If $\tilde{\boldsymbol{R}}$ performs better for at least one state-action pair replace the policy with a new one, calculated by policy search with $\tilde{\boldsymbol{R}}$ as the new reward; do this step with a probability of $\min\{1, \frac{P(\tilde{\boldsymbol{R}},\pi)}{P(\boldsymbol{R},\pi)}\}$, keep the old policy otherwise. Then replace $\boldsymbol{R}$ by $\tilde{\boldsymbol{R}}$ with the same probability.

Computing the Q-function can be done efficiently if we keep track of the optimal policy for the current reward while moving along the Markov chain. Then the Q-function can simply be expressed by the linear function of the reward variables.

The authors showed that the runtime complexity is

$$\mathcal{O}\left(N^2 \log \frac{1}{\epsilon}\right) \tag{4.19}$$

with $N = |S|$ and ϵ as the error.

4.3.3. Results

The authors compared there Bayesian IRL algorithm against those proposed by [Ng et al., 2000] and showed that BIRL outperforms IRL clearly w.r.t. the two critical loss functions: the reward loss and the policy loss. They also showed how domain knowledge can be encoded as the prior to achieve even better results.

4.4. Maximum A Posteriori Estimate

[Choi and Kim, 2011] have shown that using the posterior mean to estimate the reward in the Bayesian setting can lead to generalizations that are inconsistent with the observed behaviour; remember that one of the goal for BIRL was to avoid over-fitting. It would then need a lot of expert demonstrations to eventually be able to converge. Using Maximum A Posteriori for estimation avoids this problem. They propose an efficient gradient descent method to solve BIRL using MAP estimate.

4.4.1. Problem Formulation

How to model the problem:

- MDP $= (S, A, T, \gamma, R, \alpha)$, a Markov Decision Process with α as the initial state distribution.
- $R(s,a) = \sum_{i=1}^{d} w_i \phi_i(s,a)$, the linear function approximation of the reward function. (Note that R depends also on the action a which is more general. Since all RL methods using $R(s)$ can be trivially extended to the general case, the simpler one is typically used. This section uses $R(s,a)$ to comply with the formulation of the authors in the original work.)
- $\{(s_1^m, a_1^m), \ldots, (s_H^m, a_H^m)\}_{m=1}^{M}$, the set of observations of the experts behaviour.
- $\hat{V}^E = \frac{1}{M} \sum_{m=1}^{M} \sum_{h=1}^{H} \gamma^{h-1} R(s_h^m, a_h^m)$, the estimated value of the expert based on the observations using the approximated reward function.
- $\hat{\pi}_E(s,a) = \frac{\sum_{m=1}^{M} \sum_{h=1}^{H} 1_{(s_h^m = s \wedge a_h^m = a)}}{\sum_{m=1}^{M} \sum_{h=1}^{H} 1_{(s_h^m = s)}}$, the estimated policy of the expert.
- $\hat{\mu}_E(s) = \frac{1}{MH} \sum_{m=1}^{M} \sum_{h=1}^{H} 1_{(s_h^m = s)}$, the visitation frequency of state s given the observation.
- $[\boldsymbol{I} - (\boldsymbol{I}^A - \gamma \boldsymbol{T})(\boldsymbol{I} - \gamma \boldsymbol{T}^{\pi})^{-1} \boldsymbol{E}^{\pi}]\boldsymbol{R} \leq \boldsymbol{0}$, the optimality criteria w.r.t. policy π where $\boldsymbol{E} = \mathbb{F}^{|S| \times |S||A|}$, $(s, (s', a')) = 1 \Leftrightarrow s = s' \wedge \pi(s') = a$ and $\boldsymbol{I}^A$ as an stacked vector with $|A|$ identity matrices $I^{|S| \times |S|}$. The region bounded by this inequality is denotes as the reward optimal region.
- $\mathbb{P}(X \mid \boldsymbol{R}) = \prod_{m=1}^{M} \prod_{h=1}^{H} P(a_h^m \mid s_h^m, \boldsymbol{R}) = \prod_{m=1}^{M} \prod_{h=1}^{H} \frac{e^{\beta Q^*(s_h^m, a_h^m; \boldsymbol{R})}}{\sum_{a \in A} e^{\beta Q^*(s_h^m, a; \boldsymbol{R})}}$, the likelihood expressed as an independent exponential distribution.

Determine:

- $\mathbb{P}(\boldsymbol{R} \mid X) \propto \mathbb{P}(X \mid \boldsymbol{R}) \mathbb{P}(\boldsymbol{R})$, the MAP problem formulation.

The authors have shown that using the posterior mean to estimate the reward function can lead to a result that is inconsistent with the data. Therefore they investigate the

MAP approach as it does not exhibit this behaviour. They propose a general MAP framework by formulating the posterior as

$$\mathbb{P}(\boldsymbol{R} \mid X) \propto e^{\beta f(X;R)} \tag{4.20}$$

where β is a scaling constant for the likelihood and $f(X;R)$ is a specific measure of error between the estimate and the observed data. It can be shown that most non-Bayesian IRL algorithms are special cases of this model by choosing $f(X;R)$ accordingly and solving MAP afterwards.

4.4.2. Solutions

It can be shown that $\boldsymbol{V}^*(\boldsymbol{R})$ and $\boldsymbol{Q}^*(\boldsymbol{R})$ are convex and differentiable almost everywhere. Combing it with the fact that the probability $\mathbb{P}(X \mid \boldsymbol{R})$ can be expressed as a differentiable function of $\boldsymbol{V}^*$ or $\boldsymbol{Q}^*$, the optimization problem

$$\boldsymbol{R}_{\text{MAP}} = \arg\max_{\boldsymbol{R}} \mathbb{P}(\boldsymbol{R} \mid X) = \arg\max_{\boldsymbol{R}} [\log \mathbb{P}(X \mid \boldsymbol{R}) + \log \mathbb{P}(\boldsymbol{R})] \tag{4.21}$$

can be solved by a gradient method. To increase performance, they cache old gradients for reuse, which is possible if the new reward $\boldsymbol{R}_{\text{new}}$ lies in a reward optimality region of an already computed reward $\boldsymbol{R}_{\text{old}}$. It effectively prevents from computing the optimal policy for the current reward function and a matrix inversion.

If the posterior is not convex, it is not guaranteed to find the global optimum. But experiments have shown that a local optimum still outperforms the posterior mean approach as the MAP approach is guaranteed to find solutions inside the reward optimal region.

4.4.3. Results

On a grid world example the authors have shown that the MAP approach converges much faster than the posterior mean approach. The difference gets bigger as the dimension of the reward function grows. In a second experiment, which is a simple car race setting, they showed that if the expert is not optimal, MAP robustly converges to a more correct behaviour, while posterior mean tend to learn the bad behaviour.

4.5. Cooperative Inverse Reinforcement Learning

Classic IRL problem formulations expect expert demonstrations as given to recover the underlying reward function. An expert knows the reward function and tries to maximize the expected return. However, [Hadfield-Menell et al., 2016] have shown that this kind of demonstration is not optimal w.r.t. convergence speed for the learner to determine the reward function. Therefore the authors give a criteria for an optimal teaching behaviour so that teacher and learner work together to maximize the expected return in a cooperative environment. Since this work is motivated by human-robot collaboration, we refer the teacher as the human H and the learner as the robot R.

4.5.1. Problem Formulation

Given:

- CIRL game $M = (S, \{A^{\mathrm{H}}, A^{\mathrm{R}}\}, T, \Theta, R, P_0, \gamma)$, a modified MDP with two actors teacher and learner with common reward. Only the teacher observes reward.
- $T : S \times A^{\mathrm{H}} \times A^{\mathrm{R}} \to S$, with $s_{t+1} \sim \mathbb{P}_T(s' \mid s_t, a_t^{\mathrm{H}}, a_t^{\mathrm{R}})$
- $R : S \times A^{\mathrm{H}} \times A^{\mathrm{R}} \times \Theta \to \mathbb{R}$
- $R(s) = \phi(s)^T\theta$, assume a linear reward function.

Determine:

- $\pi^{\mathrm{R}} = \mathrm{IRL}(\tau^{\mathrm{H}})$, the robot simply applies a state-of-the-art IRL algorithm given the demonstration τ^{H} commonly *assuming* that it was performed by an expert π^{E}.
- π^{H} the behaviour of the human that tries to teach the robot the exact reward parameter θ by demonstrating τ^{H} which is *not necessarily* the expert behaviour π^{E}.

4.5.2. Solutions

A CIRL problem can be reduced to a POMDP problem with only a single set of actions $A = A^{\mathrm{H}} \times A^{\mathrm{R}}$.

The authors have shown that it is actually not optimal for the human to maximize the reward during the demonstration, i.e. to execute the expert's policy π^{E}, even if the robot assumes π^{E} performing the demonstration. Instead, the human should also try to minimize the similarity between the expected feature count calculated by the robot and the actual feature values obtained from the trajectory τ^{H}. Therefore, the human should demonstrate

$$\tau^{\mathrm{H}} = \arg\max_{\tau} \ \phi(\tau)^{\mathrm{T}}\theta - \eta\|\phi_\theta - \phi(s)\|^2, \tag{4.22}$$

where we add a penalty for feature count deviation with a constant weight of η.

4.5.3. Results

The authors experimentally confirmed that the best-response behaviour of the human clearly leads to a better performance of the robot than maximizing the reward. But as the feature dimension rises the performance gap between both gets smaller. This is due the fact that the collaboration problem is in general more difficult to solve in higher dimensions, i.e. the curse of dimensionality outweighs the benefits of the improved teaching behaviour.

By using the Maximum Entropy algorithm for IRL, we can vary the assumption of the robot about the optimality of the humans behaviour with a parameter λ. $\lambda = 0$ means that the robot assumes that the human behaves purely random and with $\lambda = \infty$ it assumes a pure greedy behaviour. They have shown that the least regret is obtained by an intermediate value of λ.

4.6. Active Learning for POMDPs

Aside from CIRL, IRL problems typically describe processes that have two phases: a demonstration phase and a learning phase. Active Learning combines these two phases into one single phase. This means, the agent starts to act even if it cannot gain knowledge by acting on its own. However, it has always access to an *oracle* (typically an human) that can tell what the optimal action for the current state is. Of course such a *policy query* has high cost, otherwise the available oracle could simply do the job. [Doshi-Velez et al., 2012] show how an agent selects actions and updates its belief over the true model within model-uncertain POMDPs and how it eventually becomes self-contained.

4.6.1. Problem Formulation

How to model the problem:

- $(S, A, O, T, \Omega, R, \gamma)$, a model-uncertain POMDP with uncertainty about T, S, R (instead only about S in normal POMDPs).
- $M = T \times \Omega \times R$, the parameter space for expressing the belief over the model of the world.
- $S' = S \times M$, the joint state space.
- $p_M(m) \approx \sum_i w_i \delta(m_i, m)$, we approximate the belief over models using samples. w_i and m_i are updated according to weight importance sampling.
- $A' = A \cup \{\psi\}$, the augmented action space where $\{\psi\}$ denote the set of oracle queries. Such a query has a cost of $-\xi$. Whenever the risk of acting is too high, i.e. the expected loss is lower than $-\xi$, then the agent will make a query. This behaviour reduces the problem complexity as we do not plan generally over A', but either over A or over $\{\psi\}$.

Determine:

- an action selection strategy.
- a model update mechanism.

4.6.2. Solutions

Action Selection Applying the Bellman optimality equation to the uncertainty-model POMDP, we get

$$\begin{aligned} Q^*(p_M, \{b_m\}, a) \quad = \quad & \int_M p_M(m) R(b_m(s), a) \\ + \quad & \int_M p_M(m) \gamma \sum_{o \in O} \mathbb{P}(o \mid b_m(s), a, m) V^*(p_M^{a,o}, \{b_m^{a,o}\}). \end{aligned} \tag{4.23}$$

Using the loss function

$$L_m(a, a^*; b_m) = Q^*(b_m, a) - Q^*(b_m, a^*), \tag{4.24}$$

that is the loss under model m if a is chosen instead of the optimal action a^*, we can write the expected loss of the value as the Bayes risk

$$\begin{aligned} \mathbb{E}_M[L] &= \mathrm{BR}(a; \{b_m\}) \\ &= \int_M L_m(a, a^*_m; b_m) p_M(m) \\ &= \int_M Q^*(b_m, a) - Q^*(b_m, a^*_m) p_M(m) \\ &= \int_M Q^*(b_m, a) p_M(m) - \int_M Q^*(b_m, a^*_m) p_M(m). \end{aligned} \tag{4.25}$$

The last equation shows that only the first term depends on a. Therefore we can neglect the second term when optimizing the value, which results in

$$V_{\mathrm{BR}} = \max_a \int_M Q(b_m, a) p_M(m). \tag{4.26}$$

Model Update As mentioned above, the model belief is estimated by $p_M(m) \approx \sum_i w_i \delta(m_i, m)$. We update the belief by updating the weights according to importance weight sampling in the following manner:

$$w(m) = \frac{p_{M|h,\Psi}}{K(m)}. \tag{4.27}$$

After each time step, the weight for a model m can be updated by

$$w_{t+1}(m) = w_t(m) \frac{p_{M,t+1}(m)}{p_{M,t}(m)}. \tag{4.28}$$

Furthermore, it is important to understand the impact of the constraints caused by policy queries. Intuitively we want to enforce our model to fit expert behaviour, thus setting

$$p(\psi \mid m) = \begin{cases} 1 & a^*_m = a^*_{m^*} \\ 0 & \text{otherwise} \end{cases}, \tag{4.29}$$

where a^*_m is the action that would be selected under the estimated model m and $a^*_{m^*}$ is the expert demonstration under the true model m^*. This, however, makes the search space not smooth and thus inhibits gradient ascent. Instead, the authors model the ratio of the number of consistent responses to the number of inconsistent ones by the binomial parameter p_e, leading to

$$p(\psi \mid m) = \begin{cases} p_e & a^*_m = a^*_{m^*} \\ 1 - p_e & \text{otherwise} \end{cases}. \tag{4.30}$$

Now we have two option for updating the model: First, resample new models according to equation 4.27, which is computational costly. Second, recompute the weights, which might get stuck in bad regions. Therefore it is best to do both. Resample models on regular basis between larger time intervals. Meanwhile only recompute the weights. The authors use forward-filtering backward-sampling (FFBS) to sample state sequences $\{s_t\}_{t=1}^T$ and derive models $\{m_t\}_{t=1}^T$ from those states. To recompute the weights, multiply the old weight with $p(\psi \mid m)$ defined by equation 4.30.

4.6.3. Results

Theoretical Results The authors have shown that using the action selection strategy and model update mechanism above, the number of required samples is

$$n_m = \frac{(R_{\max} - \min(R_{\min}, \xi))^2}{2(1-\gamma)^2(\xi - \frac{2(R_{\max}-R_{\min})\delta_B}{(1-\gamma)^2})^2} \log \frac{1}{\delta} \tag{4.31}$$

to achieve an risk of at most ξ with probability $1-\delta$, where δ_B denotes the density of the belief points, defined to be the maximum distance between any reachable belief to the nearest sampled belief point, and γ the discount factor. Now we can assume that the action chosen by the agent at every step has a risk less than ξ. Then the lower bound of the expected value is

$$V' > \eta(V^* - \frac{\xi}{1-\gamma}) + (1-\eta)\frac{R_{\min}}{1-\gamma}\,, \tag{4.32}$$

where

$$\eta = \frac{(1-\delta)(1-\gamma)}{1-\gamma(1-\delta)}\,. \tag{4.33}$$

The number of samples derived from equation 4.31 is very pessimistic as experiments have shown that much less is actually needed. Furthermore, note that this lower bound for V' assumes the worst case in which the agent gets stuck in an absorbing state where it receives $R_{\min}$ forever with probability δ at each step. Such an absorbing state does often not exist.

The last theoretic result shows that the number of policy queries is bounded in infinite learning trials. Even if the reward space is infinite, the process will eventually come to a point where the model is close enough to the true one such that the expected risk is always below the threshold ξ. This holds due to the convergence guarantee of the value iteration for the Bellman equation and the assumption that the oracle behaves correctly. Based on the sample complexity (equation 4.31), the authors additionally give a proof for an upper bound p_q which is the probability of making a policy query with some confidence δ_q after performing k additional Markov steps.

Experiments First, the authors solved a domain with only few discrete parameters with standard POMDP solvers and their Bayes risk method using equation 4.26. Comparing both makes clear that the proposed Bayes risk method does not decrease the total collected reward.

In the second experiment they have shown that this inference approach performs better than previous ones. This hold for the active as well as for the passive learner.

4.7. Concurrent Action Model

The *Concurrent Action Model* (CAM) is a model that combines continuous time for action durations and concurrent multi-actions, developed by [Rohanimanesh and Mahadevan, 2003]. This general and yet simple model can describe many real systems and inspired other researchers to investigate further in this direction, for example [Toussaint et al., 2016].

4.7.1. Model Definition

The notations used here deviates from that in the original work to comply with the rest of this thesis. I define CAM as a tuple (S, A, T, γ, R), where S and A are the sets of states and actions respectively and $R(s)$ the reward like in standard MDPs. At each Markov step, multiple actions in A can be performed, formally $\mathcal{A} \in \mathfrak{P}(A)$. Therefore the transition probabilities is $\mathbb{P}(s' \mid s, \mathcal{A}, n)$, where $n \in N$ is the number of micro time steps needed for this semi-Markov step to terminate. I define $\tau_a \in \mathbb{N}$ as the number of micro time steps to finish action a.

Since actions have different execution times in general, we need to define when a semi-Markov step terminates. The authors describe three termination schemes T_{any}, T_{all} and T_{continue}. The first one terminates as soon as possible such that $n = \min_{a \in \mathcal{A}} \tau_a$. All other actions are interrupted and need to be included in the next multi-action in order to continue, of course with reduced number of time steps consistent with real physical time. T_{all} terminates as late as possible, i.e. $n = \max_{a \in \mathcal{A}} \tau_a$. The T_{continue} scheme is sort of a combination of the previous ones. It terminates like T_{any}, but includes actions that has yet to finish automatically to the next multi-action, which is called the *continue set*. This adds some overhead to the MDP, because the continue set changes from step to step depending on Markov decisions, behaving like a kind of a "state". Therefore the value function naturally depends on both, the state $s \in S$ and the continue set.

4.7.2. Theoretical Results

The authors have shown the following relation between optimal policies, when different termination schemes are used:

$$\pi^{*\text{seq}} \leq \pi^{*\text{all}} \leq \pi_{\text{cont}} \leq \pi^{*\text{any}} , \tag{4.34}$$

where $\pi^{*\text{seq}}$ is the policy for a sequential version of the problem. This result shows that the concurrent model covers the policy space of sequential problems and that the T_{any} termination scheme performs the best.

4.7.3. Experiments

A grid world navigation task is used compare these different termination schemes. The agent needs to find keys for locked doors and to open them and navigate to the exit. Holding keys and navigation are two different types of actions. Only one action of the same kind can be include in a single multi-action. All actions suffer from some kind of noise, which lead to some failures, i.e. moving to the wrong direction or dropping the key.

The optimality relations from the theoretical results can be confirmed as $\pi^{*_{\text{any}}}$ produced the best plan. With increasing episodes, it converged to the fastest policy in terms of navigation steps. But it needed a higher number of Markov steps to complete the task. This measure gives the opposite ordering, which can be explained that the T_{any} scheme terminates earlier, for example when the key is unintentionally dropped, and therefore make decisions more often.

4.8. Relational Activity Process

RAP is a new formalism for describing relational learning and planning domains with concurrent actions, proposed by [Toussaint et al., 2016]. The advantage over previous formalisms is that it elegantly combines simplicity and expressiveness allowing us to easily define concurrent RL systems of any kind, whether it is single agent with multiple manipulators, multi-agent or human-robot collaboration.

4.8.1. System Components

Environment The following lists the components of the environment the agent acts upon.

- $(\mathcal{P}, \mathcal{C})$, the relational domain with a set of first order predicates $\mathcal{P}$ and a set of constants $\mathcal{C}$.
- $\mathcal{P}$ is used to encode the dynamics of the system.
- $\mathcal{C}$ is used to name all entities of interest, from abstract software objects to real world concrete objects. For example, the boolean fact that an activity is currently running or the position of the robot's arm.
- $(\text{KB}, S) \subset (\mathcal{P}, \mathfrak{P}(\mathcal{C}))$ is a specific environment, with KB being the knowledge base and S the state space which is a subset of the power set of all possible constants $\mathfrak{P}(\mathcal{C})$.
- $s_i \in S$, the state at semi-Markov step i.

Activities and Decisions An *action* in MDPs denotes an atomic unit of work that is done during a discrete artificial time unit called a Markov step. The authors use the term *activity* to denote units of work that take real time to finish. For activities, it therefore makes sense to have the option to terminate them early. The following describes the mechanism of the decision making of an agent when running a RAP and its related components.

- $a \in \mathcal{A} \subset \mathcal{C}$, an continuous activity from a set of activities which is part of the constants.

- $o_i(a, \bar{X}) : \texttt{pre}_i(a, \bar{X}) \to \texttt{go}(a, \bar{X}) = \tau_a, \texttt{post}_i(a, \bar{X})$, one or multiple initiation operators. If the constants for the preconditions are `true`, add τ_a to the constants and set other post conditions to `true`. Activating this operator means that the agent wishes to start activity a.
- $o_t(a, \bar{X}) : \texttt{pre}_t(a, \bar{X}) \to \neg\texttt{go}(a, \bar{X}), \texttt{post}_t(a, \bar{X})$, one or multiple termination operators. It works analogue to an initiation.
- $\mathcal{D}(s) \subset \{o_i(a, \bar{X}) \mid a \in \mathcal{A}\} \cup \{o_t(a, \bar{X}) \mid a \in \mathcal{A}\} \cup \{\texttt{wait}\}$, the set of decisions. At each Markov step we can decide to initiate or terminate an activity for (only) one actor or to `wait`. Note that decision making is sequential. The concurrency of the system will be explained next, which is heavily related to the `wait` decision.

STATE TRANSITION AND REAL TIME Neither an initiation nor a termination decision progresses real time (in theory), but a `wait` decision does. Upon initiations or terminations the change to state is typically not big, like inverting a few truth values in the KB. If the central decision maker decides to `wait`, the system performs all active activities until the first activity finishes, i.e. the waiting time is

$$\tau_{\min} = \min_a \tau_a.$$

During this time, no decision can be made. When $\tau_{\min}$ has elapsed, the duration of all active activities are reduced by $\tau_{\min}$. Now we are in a new semi-Markov step. These changes to the state s result in an intermediate state $\hat{s}$. We get the next state s' of the sMDP by forward chaining the facts in $\hat{s}$ with the rules from KB, thus

$$s' = \hat{s} \wedge \text{KB}$$

The duration and the reward are sampled from some distribution

$$\begin{aligned} \tau_a &\sim \mathbb{P}(\tau_a \mid s, a, s'), \\ r &\sim \mathbb{P}(r \mid s, a, s', \tau_a). \end{aligned}$$

4.8.2. PLANNING

Even if this formalism describes concurrent actions, the decision making is done in a sequential manner. Therefore any planner for standard MDP can be applied here. The authors use UCT for their experiments, which is a Monte-Carlo Tree Search method that uses the UCB1 measure for the tree policy. For backups, plain MC is used as it is quite robust and domain independent.

4.8.3. LEARNING FROM DEMONSTRATION

As with planning, existent methods from direct policy learning and IRL can be applied on apprenticeship learning problems described in RAP. Here, TBRIL for direct policy leaning and RCSI for IRL are used for the experiments.

4.8.4. Experimental Results

Two different domains were used to formulate planning and apprenticeship learning problems. In the first domain, the robot should help the human to build a box consisting of five single pieces. In particular, it should supply the human with box pieces that are needed next. The process always starts in a unique state where the human has no pieces. The second domain is a blocksworld domain where two activities can be realized at the same time by using two robot endeffectors. The task is to build one single tower from a set of blocks. The initial state space is randomly distributed over the number towers and their sizes (number of blocks it consists of). While the first domain is hard for planning because it takes a lot of steps to receive any reward, the second domain is hard for learning because of the large state space and initial state distribution.

In the simulation, the convergence to the optimum can be verified. In the experiments, the tasks were done successfully.

5. Propositional Formulation of RAP

Many test and real world environments are not described in relational form. Here I propose a non-relational version of RAP, that can be used if RAP is a suitable formalism for a task in an already implemented environment in non-relational form. Many of these transformations are similar to those when translating from RAP to CAM, since CAM is propositional and describes concurrent sMDPs as well. If a FOL formula is expressible in propositional logic, then we typically do a "rollout". That mean instead of using variables to address a wide range of objects, we list every object explicitly.

The key property of RAP is that it describes concurrent processes despite the underlying MDP itself being sequential. This has an interesting effect that the planning tree grows more into the depth compared to other formalisms like CAM. We can call the translated model *Sequential Initiation Model* (SIM).

We start with translating activities to actions. In RAP, a decision can be either an initiation of an activity, a termination of an activity or a `wait`. The set of actions in SIM is

$$A = A_{\text{Initiation}} \cup A_{\text{Termination}} \cup \{\texttt{wait}\}, \tag{5.1}$$

where $A_{\text{Initiation}}$ contains all actions that correspond to all initiation operators o_i's and $A_{\text{Termination}}$ contains all actions that correspond to termination operators o_t's. In order to address the corresponding activity given an initiation or termination action, define a mapping

$$\alpha : A_{\text{Initiation}} \cup A_{\text{Termination}} \to \mathcal{A} \tag{5.2}$$

that does the desired mapping.

The crucial part of this translation is to simulate the effects of pre- and postconditions. In RAP, the state consists of all constants in the knowledge base. For SIM, we need to separate "normal" state information from those that are model specific. The states S in SIM should be a composite state (S_E, M). S_{E} contain information that are independent from the choice of model. M is the set of active activities. The management of activity initiations can be mapped to a set of mutual exclusion rules

$$\mathcal{M} \subset \mathfrak{P}(\mathcal{A}), \quad \mathcal{A} \in \text{RAP} . \tag{5.3}$$

This subset of the power set of the original activities tells, which activities cannot run in parallel. The set of valid initiation actions in S_E can be determined by checking

$$M \cup \{a\} \in \mathcal{M}, \quad \forall\, a \in A_\text{Initiation} . \tag{5.4}$$

If evaluation for action a results to `true`, a is not valid. After an action is chosen for the current step, we modify

$$M = \begin{cases} M \cup \{a\} & \text{if } a \in A_\text{Initiation}, \\ M \setminus \{a\} & \text{if } a \in A_\text{Termination}, \\ M & \text{otherwise} \end{cases} . \tag{5.5}$$

The last thing we need to discuss is continuous time. As in RAP, every activity has a cost-to-go value τ_a but initiating this activity does not progress real time. When `wait` is chosen, time progresses until the first activity has finished. The cost-to-go value of all other activities need to be adjusted. In order to do that, we define a tuple c of real valued elements where each element is the remaining cost-to-go value for an activity. If an activity is not active, $c(a)$ is zero. If an activity is newly initiated, then $c(a)$ is τ_a. After every `wait`, the values in c reduces as follows

$$c(a) = c(a) - \tau_\text{min}, \quad \forall\, a \in \mathcal{A} \tag{5.6}$$

with

$$\tau_\text{min} = \min_{a \in M} \tau_a . \tag{5.7}$$

Putting everything together, SIM is formally the tuple $(S, A, \mathcal{A}, T, R, \gamma, M, \mathcal{M}, \tau, c)$, where each component is described above or in Section 4.8.

Note, however, that domain specific predicates can utilize the full potential of FOL expressiveness such that it is in general not possible to convert any problem described in RAP to SIM.

6. Comparison CIRL and RAP

In RAP, the human is part of the environment for which the planner cannot apply an policy to. Concurrency is handled by sequencing the decision making, so that most planning methods are ready to use. The CIRL formalism describes the human's policy as part of the MDP and defines a criteria for optimal teaching behaviour. The problem with CIRL is that a human at best knows the optimal policy and is able to execute accordingly but, as I believe, in general cannot solve the optimization problem in his head to take the next action that is better for teaching. In case the human is merely the executor of $a = \pi^H(s)$ and not the decision maker, he or she can be modelled as a noisy actuator.

Now let us consider the case where the teacher H is a software agent as well. For this case we can show that CIRL is a subclass of RAP stated as in the following theorem.

Theorem 6.1. *For any problem P in CIRL with optimal solution π^* there exist an equivalent problem $\tilde{P}$ in RAP with optimal solution $\tilde{\pi}^*$ such that $\pi^* = \tilde{\pi}^*$.*

Proof. Our goal is to derive a special subclass of RAP and show that its value function is identical to that of CIRL. The proposition would then directly follow because the same value functions leads to the same Bellman optimality equations with the same optimal solutions. Applying the Bellman equation for CIRL we have:

$$V^{\pi^H,\pi^R}(s) = R(s, \pi^H(s), \pi^R(s); \theta) + \gamma \sum_{s'} \mathbb{P}(s' \mid s, \pi^H(s), \pi^R(s)) V^{\pi^H,\pi^R}(s') \,. \tag{6.1}$$

The Bellman equation for a general RAP is:

$$V^{\pi}(s) = \sum_{s',\tau} \mathbb{P}(s', \tau \mid s, \pi(s)) \left[R(s, \pi(s), s', \tau) + \gamma^{\tau} V^{\pi}(s')\right] \,. \tag{6.2}$$

When reducing the class of RAP problems with two agents to the class of CIRL problems step by step, we can apply these constraints on the value function in equation 6.2 simultaneously which results in the value function for the new MDP which is denoted as M. First, we remove s' as a parameter for R to put it on the same level of generality as in CIRL. Furthermore, restrict R to be linear in θ just as in the CIRL formulation. Next, we can reduce a sMDP to a MDP by setting the cost-to-go value to a constant value. This value should represent the duration of a Markov step such that all activities can be completed within that period of time, therefore we set $\tau_a = \max_{a \in A} \tau_a = \tau_{\max},\ \forall a \in A$. As τ is deterministic now, we do not need to take the expectation over τ in the value function. Note that initiations and terminations still do not progress real time, only a `wait` decision does. The intermediate result is:

$$V^{\pi}(s) = R(s, \pi(s); \theta) + \gamma^{\tau} \sum_{s'} \mathbb{P}(s' \mid s, \pi(s)) V^{\pi}(s') \,. \tag{6.3}$$

In a single Markov step in RAP we can initiate or terminate an activity or decide to `wait`. Since the action pair $(a^H(s), a^R(s))$ in CIRL structurally corresponds to the following specific *initiation batch* $(d_1(s), d_2(s), \texttt{wait})$ in RAP, we modify M to a process over initiation batches instead of single decisions (If you don't like the idea to change the process, encode this behaviour by modifying the KB). The policy generating an initiation batch is denoted as $\tilde{\pi}$. Because any initiation batch has exactly one `wait` and all activities take $\tau_{\max}$ time, the new discount $\tilde{\gamma} = \gamma^{\tau_{\max}} < 1$ is now static. In order to enforce $\tilde{\pi}$ to initiate exactly one activity for H and exactly one for R, we can add some first-order logic rules to our knowledge base KB and modify the pre- and postconditions for initiations accordingly. For example, at the beginning of the process, those rules tell that only an activity for H can be initiated. Each such initiation has postconditions that disable initiations for H and enable initiations for R and so on. The resulting value function with $\tilde{\pi}$ and $\tilde{\gamma}$ is:

$$V^{\tilde{\pi}}(s) = R(s, \tilde{\pi}(s); \theta) + \tilde{\gamma} \sum_{s'} \mathbb{P}(s' \mid s, \tilde{\pi}(s)) V^{\tilde{\pi}}(s') \,. \tag{6.4}$$

The remaining question is how to simulate the lack of knowledge π^R has regarding θ in M. We can simply define $\tilde{\pi}$ as a composite policy $(\tilde{\pi}^H, \tilde{\pi}^R)$ that generates initiation batches in form of $(\tilde{\pi}^H(s), \tilde{\pi}^R(s), \texttt{wait})$ and declare that any planning algorithm generating $\tilde{\pi}^R$ does not have access to θ, effectively only allowing IRL planning methods, making it equivalent to the definition of π^R in CIRL. The final value function for M is:

$$V^{\tilde{\pi}^H, \tilde{\pi}^R}(s) = R(s, \tilde{\pi}^H(s), \tilde{\pi}^R(s); \theta) + \tilde{\gamma} \sum_{s'} \mathbb{P}(s' \mid s, \tilde{\pi}^H(s), \tilde{\pi}^R(s)) V^{\tilde{\pi}^H, \tilde{\pi}^R}(s') \,. \tag{6.5}$$

We can see that it is indeed equivalent to the value function for CIRL in equation 6.1, i.e.

$$\text{RAP} \supsetneq M = \text{CIRL} \,. \tag{6.6}$$

Given a concrete CIRL problem, we can encode its states, actions, rewards and state transitions in first-order logic, making it a problem of M. Due to the equivalence of the value functions, the optimal policies are the same. $\square$

7. Comparison Dec-POMDP and RAP

Both formalisms each describe a set of problems where multiple agents perform tasks concurrently to achieve a common goal, that is maximizing the expected return using a single shared reward function. It is clear that both models cannot be compared directly, as RAP does not describe partial observability and Dec-POMDP does not support continuous time. But if we extend both models with the respectively missing properties, we can see that both extended versions have the same expressiveness.

7.1. Continuous Time for Dec-POMDP

The formalism of Dec-POMDP does not consider continuous time. However, we could extend it using the idea from the concurrent action model developed in [Rohanimanesh and Mahadevan, 2003] summarized in Section 4.7, where each action takes a certain number of micro time steps to complete. Use the T_{any} terminations scheme to define Markov steps because it performs the best.

The new system component is action duration. For each action a define a duration distribution where actual durations during runtime are sampled from:

$$\tau_a \sim \mathbb{P}(\tau_a \mid a), \quad \tau_a \in \mathbb{N} \,. \tag{7.1}$$

I define durations to be natural numbers, that are interpreted as micro time steps, to comply with the original definition of CAM. (Note that this is not less general than using real number in practice because the resolution of micro time steps can be as high as the machine float representation.) As in CAM, the state transition needs to depend on the number of micro time steps n:

$$s' \sim \mathbb{P}(s' \mid s, \boldsymbol{a}, n) , \tag{7.2}$$

where n is determined according to the T_{any} termination scheme and $\boldsymbol{a}$ is a multi-action. The set of possible multi-actions $\{\boldsymbol{a}\} \subsetneq \mathfrak{P}(A)$ is restricted by the Dec-POMDP model itself. It is ordered, so that we know which agent performs what, and has fixed length, i.e.

$$\boldsymbol{a} = (a_1, ..., a_m) \in A^m, \quad \forall i \neq j : a_i \neq a_j , \tag{7.3}$$

where m is the number of agents. All other parts of the model are untouched; the resulting POMDP stays structurally the same. I call the this new model *Decentralized Partial Observable semi-Markov decision process* (Dec-POsMDP).

VALUE FUNCTION Based on the Bellman optimality equation, the value function for Dec-POsMDP is presented next. Let $\boldsymbol{\pi}$ be a composite policy and $\boldsymbol{b}$ a tuple of beliefs for a given set of m agents, then

$$V^{\boldsymbol{\pi}}(s) = R(s) + \gamma \sum_{s'} \mathbb{P}(s' \mid s, \boldsymbol{\pi}(\boldsymbol{b}), n) V^{\boldsymbol{\pi}}(s') , \tag{7.4}$$

where the beliefs $\boldsymbol{b}$ are updated according to Equation 3.4 after each Markov step.

7.2. Distributed Partial Observation for RAP

RAP describes a centralized decision maker that has access to the information about the state to make its decisions, where agents and humans are modelled in the knowledge base. In order to decentralize decision making, we can proceed similar as in the reduction of RAP to CIRL in Section 6. We adjust the knowledge base by replicating activities as needed. Let m be the number of agents we want to simulate, then each activity needs to be replicated at most $m - 1$ times. We add preconditions to each of these activities to ensure that only the matching agent is allowed to initiate it. Here as well, we can modify the MDP over decision to an MDP over initiation batches, where $(\pi_1, ..., \pi_m)$ are the agents we want to simulate and $(\pi_1(b_1), ..., \pi_m(b_m), \texttt{wait})$ the corresponding initiation batch. The set of allowed initiation batches can be derived by pre- and postcondition predicates of initiation and termination operators. They implicitly provide mutual exclusion rules.

Notice that the policies should depend on the beliefs derived from observation histories of each agent, not on states. For that, we need to extend the RAP model with partial observability, which is straight forward because the state transition and reward function differs from a standard MDP only by additionally depending on the cost to go τ_a. Beliefs and observations are constants in the knowledge base. Pre- and postcondition predicates can be modified to depending on observations or beliefs instead of the states directly if needed. Lets call this new model *Decentralized Partial Observable Relational Activity Process* (Dec-PORAP).

VALUE FUNCTION Let $\boldsymbol{\pi}$ be a composite policy and $\boldsymbol{b}$ a tuple of beliefs for a given set of m agents, then the value function for Dec-PORAP is

$$V^{\boldsymbol{\pi}}(s) = \sum_{s',\tau} R(s, \boldsymbol{\pi}(\boldsymbol{b}), s', \tau) + \gamma^{\tau}\mathbb{P}(s' \mid s, \boldsymbol{\pi}(\boldsymbol{b}), \tau)V^{\boldsymbol{\pi}}(s') \,, \tag{7.5}$$

where the beliefs $\boldsymbol{b}$ are updated according to Equation 3.4 after each Markov step.

7.3. CONCLUSION

Both Dec-POMDP and RAP are very general and have a strong expressiveness for concurrent MDPs. While Dec-POMDP focuses on partial observability, RAP stresses out continuous time and tires to generalize over actors in the systems by encoding them as FOL constants. The centralized decision making seems to be a restriction, but in fact, decentralization is implicitly given by the freedom we have to set up our knowledge base, in particular the preconditions for initiating activities.

Theorem 7.1. *Every decision problem formulated in Dec-POsMDP can be translated into a problem of Dec-PORAP and vice versa.*

Proof. We have extended Dec-POMDP with continuous time and RAP with partial observability. Both extensions have a common notion of partial observability and decentralization. The only difference is the way how they handle continuous time. Since we extended Dec-POMDP with the CAM model, it is no surprise that the difference between these extended model are the same as the difference between CAM and RAP, if we look at the respective value functions. [Toussaint et al., 2016] showed that these models are actually equivalent w.r.t. expressiveness. Thus, Dec-POsMDP and Dec-PORAP are equivalent as well. □

8. META DECISION PROCESS

The field of study of *Human-Robot-Interaction* involve IRL (learning from human expert), concurrency (human-robot cooperation, multi-agent systems) and active learning (online planning with human help). In the literature review (Section 4) I have summarized some milestones regarding the research in these fields. This thesis explores the notion of cooperation in a general way, using state-of-the-art solutions from the literature to solve certain sub-problems, in particular RL and IRL. The main reference for this work is [Doshi-Velez et al., 2012], because or formalism involve active queries as well. Active learning is a very general formalism that fully covers the traditional setting of IRL, but on top of that the agent can take over work much faster to increase the overall efficiency of the system.

8.1. MOTIVATION

The supervisor of this thesis had the idea of "learning at the meta level". We think of learning as to intelligently collect data (and in RL we additionally take the reward into account). This directly implies that at the meta level, we need to decide how to collect data, or, which reinforcement-enabling method to choose. Making a good choice at this needs to be meta learned. One of the real world motivations for this formalism is as follows: Under model uncertainty, exploring often has security issues as the robot typically has high capabilities to cause damage at different degrees of severity. This problem is tackled by learning from demonstration. But demonstrating has high costs and the agent is basically idle. The question is: How much and often do we need demonstrations? We would like the robot to act on its own if the risk is low, and at the same time, it is not efficient to wait for long trails of demonstrations every time the agents feels unsure. Instead, the it can ask the expert for the next optimal action or whether it should perform action a_i or not and act accordingly; this is a good choice if the expected improvement to the belief is large enough. Furthermore, we want to generalize the notion of learning from demonstrations, where potential demonstrators are not limited to the human expert only but could be potentially any actor in the cooperative environment. Putting all together, this leads to three classes of meta actions:

$$A_\mu = \{\texttt{execute}, \texttt{query}, \texttt{observe}\} . \tag{8.1}$$

Use standard RL to `execute` which means to chose the next action for the current state deeming the current meta knowledge as the true model. This has no meta level cost, i.e. $c = 0$. Meta action `query` asks for the optimal action for the current state or whether it should perform a_i or not with some cost $c < 0$ (Note that `yes/no` answers are less costly). And finally the meta action `observe` means to wait and reason about what is happening in the world with some cost $c < 0$; and of course this meta action can include a demonstration request to the oracle. After observing, use standard IRL to update the model.

8.2. Definition

For the underlying MDP M we assume it to be a RAP because it is very general as we have seen in Section 6 and 7. Thereby we explicitly neglect partial observability since we want to deal with model uncertainty at meta level; For the meta level, use a special PORAP with tree predefined decisions: `execute` should take the current knowledge about the model as input and should greedily maximize reward. Remember, the agent should only `execute` if the risk is low. In order to simplify analysis later on, we assume that the reward function has the following form: All goal states have a common positive reward and all other states have a common non-positive reward. Real world tasks mostly have this kind of reward function. For example, while assembling a furniture it is hard to tell how good an intermediate result is, but at the end, one can decide if this furniture was correctly assembled or not. The same goes for games like chess: win or lose. The quality of non-goal states should be learned and represented by the value function.

In order to uncover the unknown model M, we present the system components in terms of a POMDP for simplicity, in particular, we define a *Meta Decision Process* (metaDP) as a tuple

$$M_\mu = (S_\mu, A_\mu, T_\mu, R_\mu, \Omega_\mu, O_\mu, \gamma)\ . \tag{8.2}$$

I have already presented the meta actions since it is the most significant part of this formulation and our main motivation. Other components are chosen to fit the purpose, mostly serving as connections between M and M_μ to make sure that the optimality at the meta level implies optimality for problem M. This can simply be achieved by applying any profound machine learning method, mapping from data to model. If the model is correct, any RL planner can derive or approximate the optimal policy. The system components are listed next.

Meta State As we want to reduce uncertainty, we need to keep track of it and include it as part of the state, therefore $S_\mu = S \times \Theta$, where Θ is the space of model parameters for uncertain components in M; that could be S, T and/or R. Note that $s \in S \in M$ is not the true state of the world, the planner for M deems it to be and *beliefs* that s is the true state. At the meta level we know that this is generally not the case.

Meta Transition The transition function $T_\mu : S_\mu \times A_\mu \times A \to S_\mu$ does not only depend on the meta action but also on the concrete action for M. The meta action tells how confidently the action was chosen, in order to update the model parameter θ, whereas the concrete action is needed for state transition $T \in M$.

Meta Reward The reward function $R_\mu : S_\mu \times A_\mu \times A \times S'_\mu \to \mathrm{im}(R) \times \mathrm{im}(C)$, with $C : A_\mu \to \mathbb{R}$, outputs both the reward received from the environment and the cost for taking meta action a_μ. Note that $S \in S_\mu$, therefore the arguments for R can be found in the arguments for R_μ.

Meta Observations The set of observations $O = S \times F$ consists of the set of states $S \in M$ and the set of actions augmented with the empty set $F := A \cup \emptyset$. The observation function $O_\mu : S_\mu \times A_\mu \times A \to S' \times F$ tells, what the next state is and which optimal action was chosen. If the meta action was `execute`, then $f = \emptyset$.

Note that the underlying MDP planners and IRL learners are black boxes. They can be arbitrarily substituted and the properties like optimality or convergence of the Meta Decision Process would stay the same as long as the algorithms used are guaranteed to converge to the optimum given their respective problem formulation. Of course the overall runtime or space consumption would change, but if we analyse time and space complexity independent from black boxes, then even those properties won't change.

8.3. Value Function

As an optimality criteria, we want to maximize the expected return analogous to MDPs. The difference here is that we have additional meta cost that reduces the return. Thus we have

$$V_\mu^*(s_\mu) = \max_{\pi_\mu} \mathbb{E}_\pi \left[\sum_{t=0}^{\infty} \gamma^t (c_t + r_t) \,\middle|\, s_{\mu,0} = s_\mu \right] . \tag{8.3}$$

Let $R'_\mu := \|R_\mu\|_1$ (the sum over all tuple elements). Using the transformations in Equations 3.8, we get

$$V_\mu^{\pi_\mu}(s_\mu) = \sum_{s'_\mu, o_\mu} \mathbb{P}(s'_\mu, o_\mu \mid s, \pi_\mu(\theta), a) \left[R'_\mu(s_\mu, \pi_\mu(\theta), a, s'_\mu) + \gamma V_\mu^{\pi_\mu}(s'_\mu) \right] . \tag{8.4}$$

8.4. PORAP Formulation for metaDP

With the standard POMDP formulation we can describe the relation between model and meta model. For continuous time and concurrency, we need more and choose PORAP (Section 7.2) because it has two advantages: First, it enables concurrency with sequential decision making such that all available MDP planners are ready to use. Second, the relational encoding of objects as plain constants abstracts away from concrete systems components like actors or end-effectors to describe any kind of human involved multi-agent systems in a simple and natural way (similar to the advantage of "object oriented programming" in software engineering). For example, we have two agents a and b. If it is important for a to keep track of b's actions, we can add a predicate `canObserve(x,y)` and evaluate at `x=a, y=b`, whose truth value can change after each step. This value then can be used for other calculations, like preconditions for initiations, risk evaluation functions or belief updates.

For the rest of this Section, we will assume the PORAP formulation for metaDPs; that means A_μ refers to the set of activities and at each Markov step we make decisions $d \in D(s_\mu)$ where $D(s_\mu)$ is the set of valid decisions in meta state s_μ.

8.5. Generality

Our metaDP heavily inherits idea from active learning as making queries is hard coded in the model definition. In CIRL, the human is an explicit actor in the MDP, which is not the case for metaDP. But as we have seen in Section 6, a RAP formulation can simulate CIRL by applying certain restrictions on the set of actors and its behaviours. If model parameters are known, the metaDP will always `execute`, degenerating into a plain RL planner. If we set the cost to query very high and the cost to observe very low, the process will exhibit a two phase behaviour: Observing and updating, then planning; which is a traditional IRL process.

8.6. Meta Policy

As explained in the motivation, we want to explicitly state criteria when to choose which meta action. For the algorithm we propose, our goal is as follows:

- `query`, if the expected improvement of our model is large enough, counted against its meta cost; else

- `execute`, if the risk is too large compared to the cost for observing; else
- `observe`.

This fixed selection strategy is a one-step-look-ahead concept. The reason we do not want to plan in the full MDP at the meta level is that it is computationally too expensive: It involves a RL planning to convergence for all meta states and a new plan for any change to the belief.

The quantities that are needed in order to make this decision are estimated in the next couple sections.

8.6.1. Value Improvement of Queries

The first quantity we need to determine is the value improvement that is gained if we make a query. One simple estimation is the *Information Gain* using the Kullback-Leibler divergence (KLD)

$$\mathrm{IG}_Q = \sum_{a \in A} \mathbb{P}(a = \pi^*_M(s_t)) \mathrm{KL}(b^\theta_t \| \tilde{b}^\theta_t(a)) \,, \tag{8.5}$$

where

$$\tilde{b}^\theta_t(a) = \mathbb{P}(\theta \mid a = \pi^*_M(s_t)) \,. \tag{8.6}$$

Alternatively, use *Expected Value Gain* (more expensive but better)

$$\mathrm{VG}_Q = \sum_{a \in A} \mathbb{P}(a = \pi^*_M(s_t)) \left[Q^*_{\tilde{b}}((s_t, \tilde{b}^\theta_t), a^*_{\tilde{b}}) - Q^*_{\tilde{b}}((s_t, \tilde{b}^\theta_t), a^*_b) \right] \,. \tag{8.7}$$

Both are bootstrapping estimates as $\mathbb{P}(a = \pi^*_M(s_t))$ is calculated from the own belief. The most expensive operation for KLD computation is the integration over a distribution (if the distributions for the beliefs b^θ_t and $\tilde{b}^\theta_t$ are not very fancy). For VG_Q however, we need to calculate the optimal value function given the knowledge θ. That basically means that we need to do value iteration until convergence!

8.6.2. Risk to Execute

In order to estimate the risk (or expected loss), we need to know what we are expected to achieve in case we execute, but also what is approximately the best achievable result. We choose the latter Q-value as the upper bound of the former Q-value:

$$\begin{aligned} Q^*_b((s_t, b^\theta_t), a^*_b) &\leq Q^*((s_t, b^\theta_t), a^*_b) && (8.8)\\ &\leq Q^*((s_t, b^\theta_t), a^*_\theta) && (8.9)\\ &= \int_\theta Q^*_\theta(s_t, a^*_\theta) b^\theta_t \mathrm{d}\theta =: Q_{\max} && (8.10) \end{aligned}$$

and then, we approximate the value to execute $Q^*_b((s_t, b^\theta_t), a^*_b)$ with its one-step lookahead

$$\int_\theta Q^*_\theta(s_t, \hat{a}^*_M) b^\theta_t \mathrm{d}\theta =: \hat{Q} \,, \tag{8.11}$$

where

$$\hat{a}^*_M = \arg\max_{a \in A} \int_\theta Q^*_\theta(s_t, a) b^\theta_t \mathrm{d}\theta \,. \tag{8.12}$$

Finally, the *Bayes Risk* for `execute` is

$$\begin{aligned} \mathrm{BR_e} &= \int_\theta Q^*_\theta(s_t, a^*_\theta) b^\theta_t \mathrm{d}\theta - \int_\theta Q^*_\theta(s_t, \hat{a}^*_M) b^\theta_t \mathrm{d}\theta & (8.13) \\ &= Q_{\max} - \hat{Q} \,. & (8.14) \end{aligned}$$

8.6.3. Resulting Policy

Using the information gain IG_Q for estimating the value improvement and the Bayes risk BR_Q as described above for risk evaluation, our meta policy can be precisely described as

$$\pi^*_\mu(s_\mu) = \begin{cases} \texttt{query} & \text{if } \mathrm{IG}_Q + c(s_\mu, \texttt{query}) \geq 0 \\ \texttt{execute} & \text{if } (\mathrm{IG}_Q + c(s_\mu, \texttt{query}) < 0) \wedge (\mathrm{BR_e} + c(s_\mu, \texttt{observe}) < 0) \\ \texttt{observe} & \text{otherwise} \end{cases} \,. \tag{8.15}$$

The cost function need to be hand tuned (or via cross validation) to get a reasonable policy. Of course the meta cost also is a feedback received from the environment in real world settings, but even then we need to set a constant scaling by hand, otherwise the meta costs could always overweight or get overweighted by IG_Q and/or $\mathrm{BR_e}$.

8.7. Belief Update using Particle Filters

We approximate the model θ using a set of K particles $b^\theta_t = \{(w_i, \theta_i)\}_{i=1}^K$ for computational efficiency. Whenever we receive information, for instance form a query, we update the weights as follows: Let

$$p^*_d = \mathbb{P}(d = \pi^*_M(s_t)) \approx \sum_{i=1}^K w_i \mathbb{P}(d = \pi^*_{\theta_i}(s_t)) = \hat{p}^*_d \tag{8.16}$$

be the probability for d being the optimal decision and $\hat{p}^*_d$ its approximation using the current model knowledge. Then

$$\tilde{b}^\theta_t(d) = \{(w'_i(d), \theta_i)\}_{i=1}^K \,, \tag{8.17}$$

with

$$w'_i(d) = w_i \frac{\mathbb{P}(d = \pi^*_{\theta_i}(s_t))}{\hat{p}^*_d} \,. \tag{8.18}$$

It remains to define $\mathbb{P}(d = \pi^*_{\theta_i}(s_t))$. Making hard constraints by setting these probabilities to 1 or 0, i.e. exactly matching the observation, only works out for a certain special cases, where first, the space of these K particles covers the space of the true model and second, the planning is optimal for each model. This would lead to a very fast and effective model update, however, if this special case does not hold, the parameters degenerate, especially when there is some noise in the system as well. Since this special case is not realistic, we suggest a soft update

$$\mathbb{P}(d = \pi^*_{\theta_i}(s_t)) \propto \eta\delta_{dd^*} + (1-\eta)(1-\delta_{dd^*}) \,, \tag{8.19}$$

where $\eta \in (0,1)$ and $\delta_{dd^*} = [d = d^*]$ is the Kronecker delta. Using this particle filter, our estimates for Bayes risk and information gain can now be calculated more simple using discrete sums:

$$\mathrm{IG}_Q = \sum_d \hat{p}^*_d \mathrm{KL}(\{w_i\} \| \{w'_i(d)\}) \tag{8.20}$$

$$\mathrm{BR}_\mathrm{e} = \sum_{i=1}^{K} w_i \max_d \hat{Q}^*_{\theta_i}(s_t, d) - \max_d \sum_{i=1}^{K} w_i \hat{Q}^*_{\theta_i}(s_t, d) \,. \tag{8.21}$$

8.8. Algorithm

Using the policy and the estimates derived above, we can apply a Monte-Carlo tree search algorithm to solve metaDP:

Algorithm 1 Meta Decision Process for Concurrent Cooperations

```
1: procedure METAMCTS($s_t, b^\theta_t$)
2:     if $D(s_t)$ involves oracle only then
3:         return free-query
4:     for $i = 1 : K$ do
5:         $\hat{Q}^*_{\theta_i}(s_t, \cdot) \leftarrow \mathrm{RapMCTS}(s_t, \theta_i)$
6:     return meta action $d^*_\mu$
```

Here, we also have a little optimization to make free queries with no cost, if the current action space is limited to queries (which means that all actors/agents/endeffectors are busy). Otherwise, in the normal case, it bootstraps form its own current belief to rollout simulations in order to update the Q-function. Based on the new updated Q-function, a meta action is returned. K is the number of dimensions that is used to estimate the true model.

The free query that can happen at any time has the positive effect of making convergence faster. But it is very problem specific whether we will have free queries at all. Therefore we ignore this property in our theoretical analysis.

Convergence We are interested in the convergence of the model parameter θ and assume that Θ covers the space of the true model. If the true model is recovered (or estimated well enough), the optimal policy for the underlying MDP M can be computed or estimated using any arbitrary planning method.

Looking at the meta actions, we can see that `query` and `observe` give data derived from the true model. Statistics tell us that the true model can be retrieved given enough data. Particle filters we use here is one method to do that. The question remains: When do we stop to collect these data? We stop at a point where $\pi^*_\mu(s_\mu) = \texttt{execute},\ \forall s_\mu \in S_\mu$ from equation 8.15. This means, we converge to a point where the difference between the estimated model and the true model $\xi = |\theta^* - \theta|$ implies $\mathrm{IG}_Q < -c(s_\mu, \texttt{query})$ for all states, using the information gain as estimation. In the discrete case, we can argue that all states will eventually be visited and each query will cancel out some models. For continuous state spaces, it is not appropriate to assume that important weight sampling with K samples can cover the true model. We need to explicitly show that the information gain measure steadily decreases and converge to a point where no queries are made, which we will see in the following.

The more data we receive (via queries, observations), the closer the models weights become. And thus, for any $\epsilon > 0$ there exist a point in time $t < \infty$ where

$$\|\{w_i\} - \{w'_i\}\|_\infty < \epsilon \tag{8.22}$$

for all sampled models. Furthermore, the information gain measure from equation 8.20 is upper bounded by

$$\mathrm{IG}_Q \leq \max_d \mathrm{KL}(\{w_i\} \| \{w'_i(d)\}) \leq g(\epsilon)\,, \tag{8.23}$$

where $g(\epsilon)$ is the following constraint optimization:

$$\begin{aligned} \max \quad & \mathrm{KL}(\{w_i\} \| \{w_j\}) \\ \text{s.t.} \quad & \|\{w_i\} - \{w_j\}\|_\infty < \epsilon \quad \forall \text{ models } m_i, m_j \end{aligned} \tag{8.24}$$

Note that the set of constraints consider every possible pair of weights whose max-component distance is smaller ϵ. There is an infinite amount of these pairs, making it impossible to analyse each individually. But the existence of a maximum is enough for us. We only need to make a statement that for the maximum value it holds: $g(\epsilon) \to 0$ if $\epsilon \to 0$. It would follow directly that eventually (with enough data), the information gain will be smaller than the query cost for all states. In order to show that, we consider the following limit:

$$\lim_{\epsilon \to 0} \mathrm{KL}(\{w_i\} \| \{w_j\})\,. \tag{8.25}$$

In the limit, the distributions represented by the weights are the same and thus, the KLD is zero. We conclude that for any $-c(s_\mu, \texttt{query}) > 0$, the number of queries is finite under an infinite meta execution trail.

Note that the belief can still improve, because if the Bayes risk is still to high, the agent collects more data by observing. Also, the concrete value of this upper bound depends on the estimation method used for value improvement as well. For example, the expected value gain estimate as described above would give better upper bounds than the information gain estimate.

TERMINATION Assume that the given task has a goal and is intended to terminate. Then there exist some goal state(s) that are reachable and each optimal policy eventually reaches the goal.

The fact that the model parameter θ converges to a point with error ξ has some implications on termination. If the meta cost to make a query is too high, with the effect that even the largest information gain is overweighted, the resulting policy is in worst case no better than a policy that is u.a.r. In infinite horizon problems, such a random policy would still eventually reach some termination state in theory. The situation gets worse, if the expert/oracle is noisy and give wrong information and right after that, the meta cost overweights the information gain, then the agent could get stuck in certain states and it never terminates.

We conclude that the proposed algorithm for metaDP only terminates if the meta cost $-c(s_\mu, \texttt{query})$ and thus the error ξ is small enough to allow a Q-function to be computed, such that greedy selections of actions form paths from every state to a terminating state.

8.8.1. ERROR BOUND OF BAYES RISK

Before we analyse the complexity of the algorithm above, we apply the analysis on the error bound of the Bayes risk according to [Doshi-Velez et al., 2012] described in Section 4.6. We show that the properties of both algorithms that are needed to do this analysis are the same. First, both algorithms use weight importance sampling to estimate the model. Therefore we can bound the difference between the estimated risk and true risk for every state in the same way:

$$\epsilon_{\mathrm{PB}} = \frac{2(R_{\max} - R_{\min})\delta_{\mathrm{B}}}{(1-\gamma)^2}\,, \tag{8.26}$$

where δ_{B} is the density of the belief points and $R_{\max}$ and $R_{\min}$ are the maximum and minimum receivable rewards respectively.

The second issue is the policy. The Bayes-risk-action-selection method only takes the estimated Bayes risk into account to make a (actually meta) decision whether to do a query or not. The meta policy we described uses an estimation on the value gain to decide whether to query or not; if it decides not to query, the Bayes risk is checked whether to passively observe or not. In the active learning setting, there is no notion of passive observation. But as we see later, we can express the cost of observations in a certain number of queries and use this approximation to make statements only over the number of queries. The decision making, whether to query or not, is different in our meta policy, but it does not change the fact that each query has an impact on the Bayes risk estimation. Therefore, the analysis on how many queries we need in order to achieve a bounded Bayes risk error ϵ with probability $1-\delta$ is the same: see equation 4.31.

This analysis has several drawbacks. It depends on $R_{\max}$ and $R_{\min}$, becoming less expressive, because for most problems there exists an infinite amount of possible reward functions that all lead to the same optimal behaviour. Intuitively, the number of queries needed should be independent of specific reward values. Furthermore, the measure is overestimating about several factors and it is imprecise, because the fact that model samples are not independent is simply ignored. The same holds for the lower bound of the value.

8.8.2. Complexity

For complexity analysis of this algorithm we first prepare by defining an error measure of the Q-function. Since many different Q-functions lead to the same policy (e.g. constant scaling or shifting), it is not appropriate to compare function values directly. We rather propose an error measure that compares the minimum number of Markov steps that is needed for a greedy agent to reach the goal.

Definition 8.1. *Let Q be any Q-function for an MDP M. Then*

$$\text{path}(s_i, Q) = (s_i, ..., s_n) \tag{8.27}$$

is the ***greedy path*** *from s_i to a goal state s_n, where for each $s_{j+1} \in \text{path}(s_i, Q),\ j \geq i$:*

$$s_{j+1} := \arg\max_{s} \left[\max_{a} \mathbb{P}(s \mid s_j, a) Q(s_j, a) \right] , \tag{8.28}$$

and $\text{dist}(s, Q) := |\text{path}(s, Q)|$ *is the* ***greedy path distance****.*

Using this distance measure, we can define an error measure of an estimated Q-function.

Definition 8.2. *Given any arbitrary optimal Q-function Q^*, the* ***Q-error*** *is a function $\delta_{Q^*} : \mathcal{Q} \to \mathbb{N}$ that maps a given Q-function in a function space $\mathcal{Q}$ to a natural number. For the discrete case, it is defined as follows:*

$$\delta_{Q^*}(Q) := \sum_{s \in S} \left[\text{dist}(s, Q) - \text{dist}(s, Q^*) \right] . \tag{8.29}$$

For continuous domains, the Q-error can be defined analogously by integrating instead of summing over the state space. For stochastic environments, this definition is appropriate because the optimality of Q-functions already encodes the stochastic behaviour of the system. Observe that $\text{dist}(s, Q)$ can be infinite if Q leads the greedy policy to some dead ends starting from state s. We can now give a precise optimality criteria for any Q-function:

Theorem 8.1. *Let Q^* be an arbitrary optimal Q-function for an MDP M as defined in Section 8.2, then any Q-function Q is optimal for M as well if and only if $\delta_{Q^*}(Q) = 0$.*

Proof. In case the greedy paths are the same for the same distances, we have

$$\begin{aligned} Q \text{ is optimal} \quad &\Leftrightarrow \quad \forall s \in S : \arg\max_{a \in A} Q(s, a) = \arg\max_{a \in A} Q^*(s, a) \\ &\Leftrightarrow \quad \forall s \in S : \text{dist}(s, Q) = \text{dist}(s, Q^*) \\ &\Leftrightarrow \quad \delta_{Q^*}(Q) = 0 . \end{aligned}$$

In case the greedy paths can differ although the distances are the same, we have to show

$$\forall s \in S : \mathrm{dist}(s, Q) = \mathrm{dist}(s, Q^*) \Rightarrow Q \text{ is optimal.}$$

We initially assumed that in our environment only the goal states give reward, where all other states have a constant non-positive return. Under this assumption, the total return on both greedy paths are the same. Since this is true for all states, Q is optimal. □

Now we want to investigate the complexity of metaDP using the algorithm above in case Θ covers the true model and the meta costs are small enough to allow a sufficient precision (e.g. $\delta_{Q^*}(Q) < \infty$). We want to estimate the number of queries that are needed as the complexity measure. The reason is that we assume that queries and observations are the most costly operations in this kind of RL setting. The cost for observations can be replaced by the cost for queries as follows: If an observation trail takes n Markov steps, starting from state s, to finish, the cost can be estimated with n such queries pretending the agent to be in state s initially.

In order to avoid to impose too many assumptions on the process, we analyse a soft definition of optimality. A soft variant of the Q-error we are interested in is

$$\delta'_{Q^*}(Q) := \sum_{s_0 \in S_0} \left[\mathrm{dist}(s, Q) - \mathrm{dist}(s, Q^*)\right] , \tag{8.30}$$

summing (or integrating) only over the set of initial states S_0. Lets call it the soft Q-error and denote it by $\delta'_{Q^*}(Q)$. Under the assumption that there is no knowledge about the model given initially, we need to query or observe the optimal paths from all initial states to its respectively nearest goal states to achieve $\delta'_{Q^*}(Q) = 0$. The states on these paths are

$$I = \{s \in S \mid s \in \mathrm{path}(s_0, Q^*), \forall s_0 \in S_0\} . \tag{8.31}$$

And therefore the expected number of queries needed is

$$\mathbb{E}[n] = |I| + C , \tag{8.32}$$

where $C \geq 1$ is a constant that gets greater the noisier the state transitions and query answers are. Our goal is now to upper bound exactly this C to yield an upper bound of the overall complexity to achieve soft optimality.

First, let us have a quick look at the situation if the system is soft optimal. This has an impact on the Q-values for all other states for which the optimal actions are not known (after performing RL to convergence). These states now are pulled towards states in I as illustrated in Figure 8.1. Depending on the actual values of the reward function, the "direction" can be different. Choosing any arbitrary state $s \in S \setminus I$, the best case is that s is pulled towards the nearest goal state. In the worst case, it is pulled towards the state in I that has the largest distance to a goal state. With pulling towards a state s_i we mean that for the resulting Q-function, the greedy selection $a = \arg\max_{a \in A} \mathbb{P}(s \mid s_j, a) Q(s, a)$ leads most probably to a state $s' = \arg\max_{s' \in S} \mathbb{P}(s' \mid s, a)$ that is on the shortest path from s to s_i. Note that if state transitions are deterministic, we only need I many queries

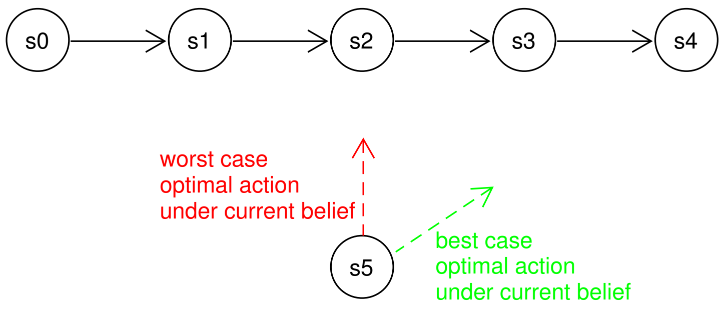

Figure 8.1: s0 is an initial state, s4 is a goal state, (s0,...,s4) is a greedy path. Imagine that the states are placed in an Euclidean space.

and the agent will always behave optimally.

Now we start to upper bound C. Let the failure rate in average be p_{fail}. Failing means, taking an action a in state s does not lead to the desired state $s' = \arg\max_{s' \in S} \mathbb{P}(s' \mid s, a)$. Let s_0 be the starting state and $d = \text{dist}(s_0, Q^*)$ be the greedy path distance starting from s_0 given optimal $Q*$ (see above). We define a random variable n_1 that is the number of queries that is needed for the agent, starting in s_0, to reach the nearest goal state s^* excluding queries made in I, and want to determine the probability for n_1 to have a certain value x given the values above, i.e. $\mathbb{P}(n_1 = x \mid s_0, s^*, d, p_{\text{fail}})$.

The last thing we need to consider before diving into the analysis is: What happens in case the agent actually fails? The question is asking about the *failure model*. If it could potentially end up in any arbitrary state in S, then it is hard to make reasonable statements about the number of queries that is further needed to reach the goal. Therefore we need to make assumptions here, but they should match most real world problems. There are three different assumptions we want to present first that apply for many problems, for which $C = 0$. After that, two similar generic failure models are analysed in detail.

SIMPLE FAILURE MODELS The first assumption is *restart*: Whenever the agent fails, the next state is a starting state $s' \in S_0$. For example, the agent should stack cubes to build a tower and at some point the tower falls apart. This leads to an initial state where the following predicate is true: $\forall c_1, c_2 \in \texttt{cubes} : \neg\texttt{onTop}(c_1, c_2)$.

The second assumption is *replan*: Whenever the agent fails, the next state is the previous state right before the failure. This is very typical in many robotics settings. For example, during an assembling task, the robot makes a wrong object placement. The human sees that, reverses the action and tells the robot to do it correctly.

And the last simple failure model is *backtrack*: Whenever the agent fails in s, the agent will undo the failed action to get back to s; it knows how to do that and will eventually be successful. For example, in a "Pick and Place" problem, the agent picks an object o_1 with its endeffector e_1 and is therefore in an state where $\texttt{picked}(e_1, o_1) = \texttt{true}$ and `onGround`(o_1) = `false`. Then, the object somehow slips away form the endeffector and drops, before the agent could place it correctly. The resulting state has inverted truth values for these predicates. The agent now tries to pick the object again, in order to continue where it was before, that is, trying to place o_1 correctly.

Under these models the only queries that are needed are those for states in I.

FAILURE MODEL 1 The next assumption we want to investigate is as follows: Whenever the agent fails at state s, it is put l Markov steps away from s and the oracle guides the agent towards s again, in case it is queried against. On the way back to s, the agent can fail again, which puts it an additional l steps away form s. It follows that each time the agent fails, it produces a need for l queries (in worst case). This assumption might not be realistic for many applications, but it can be seen as a pessimistic averaging over a more general assumption. Imagine that after failing, the agent ends up anywhere in a state space around s. If it was lucky, it gets closer to the goal, if it was unlucky it is put further away form the goal. The average is that it is put some distance l away from s. Furthermore, this assumption is pessimistic in two ways. First, it could happen that the agent visits some states that is already known from previous failures before, but we assume that it always ends up in unknown regions. Second, in the average case, the oracle would normally not guide the agent to s again, but to some state in I that is closer to the goal. Therefore this assumption is not quite the average but assumes that the agent moves further afar from the goal after a failure, which is more realistic.

In order to measure the expected number of queries needed under this assumption, we observe that the probability that x queries are made in one episode is the same as the probability to fail $m = \frac{x}{l}$ times. By failing m times, the total number of Markov steps that this episode has is $d + ml = d + x$. Now we can see that this actually is the following Binomial distribution:

$$\begin{aligned} \mathbb{P}(n_1 = x \mid s_0, s^*, d, p_{\text{fail}}) &= \mathrm{B}(m \mid p_{\text{fail}}, d + x) && (8.33) \\ &= \binom{d+x}{m} {p_{\text{fail}}}^m (1 - p_{\text{fail}})^{d+x-m} && (8.34) \end{aligned}$$

with expectations

$$\begin{aligned} \mathbb{E}[m] &= (d + x)p_{\text{fail}} \, , && (8.35) \\ \mathbb{E}[n_1] &= \frac{dp_{\text{fail}}}{\frac{1}{l} - p_{\text{fail}}} && (8.36) \end{aligned}$$

for the number of failures and queries respectively. Equation 8.35 is simply the expectation from the Binomial distribution above. Equation 8.36 is calculated by placing $\mathbb{E}[m]$ into $m = \frac{x}{l}$ and solving it for x.

At first, the result looks strange, as $\mathbb{E}[n_1]$ can become negative, which is nonsense. But at a second glance, it totally makes sense. First, we observe that the term in the numerator is very intuitive: If d or p_{fail} gets larger, we need more queries in average. Now, in order to understand the controversial denominator, we first investigate the case it gets zero, which happens if and only if $lp_{\text{fail}} = 1$. For illustration, we choose $l = 2$ and $p_{\text{fail}} = 0.5$. Each time the agent fails, it is put $l = 2$ Markov steps away from s, which in fact means that we have to play an additional Binomial experiment $\mathrm{B}(k \mid 0.5, 2)$. In expectation, we fail in this experiment exactly once ($0.5 \cdot 2 = 1$), but this however induces yet another Binomial experiment $\mathrm{B}(k \mid 0.5, 2)$, and so forth. Therefore $\mathbb{E}[n_1]$ should indeed be infinity. If l or p_{fail} increases, the situation gets worse, but since nothing is greater than infinity, a negative value implies an infinite expected amount of queries as well.

For a specific starting state s_0, we now can conclude that the expected number of queries n' we need in total to solve metaDP with a flat belief at the beginning and assuming Failure Model 1 is upper bounded by

$$E[n'] \leq \sum_{s_0 \in S_0} \mathbb{P}(s_0) \left(|I| + \mathrm{E}[n_1 \mid s_0] \right) , \tag{8.37}$$

because we analysed the failure behaviour of the system for all states in I and overestimated the query count by assuming that the agent will always encounter new states if it is not in I, which leads, in worst case, always to queries. In order to achieve the soft optimality criteria we need to sum over all initial states:

$$E[n] \leq |I| + \sum_{s_0 \in S_0} \left[\mathrm{E}[n_1 \mid s_0] \right] , \tag{8.38}$$

In all future runs, we expect that the number of queries will be reduced drastically.

FAILURE MODEL 2 The fact that Failure Model 1 has an infinite expected query count very quick is due to the combination of different worst case assumptions and a hard punishment mechanism, where the agent moves further and further away form the goal with each failure. This is actually too much; and systems that really behave like that are probably not deployable, except l or p_{fail} is really small.

With the second failure model, we want to keep the worst case assumptions but reduce failure rate drastically. Failure Model 2 is the same as Failure Model 1, except that the agent now only can fail in states in I. This especially means, that it cannot fail multiple times in a row (unless $l = 0$). The parameter l now has a different meaning. It can be interpreted as a black box expressing the expected number of additional queries needed, with further failures already considered, to get back to state s. Now, when a failure happens in state s, we will get back to s without incident. Coming back to s, however, the agent can still fail without making progress. Therefore, in terms of probability measures, this failure model is actually a special case of Failure Model 1 with parameter $l = 1$. This leads directly to an expected query count of

$$\mathbb{E}[n_1] = \frac{dp_{\text{fail}}}{1 - p_{\text{fail}}} . \tag{8.39}$$

At worst, we have $p_{\text{fail}} = 1$ which makes the value infinity, matching our anticipation. Furthermore, the value cannot become negative, making the result more intuitive.

8.9. Query and Observation Policy

We already mentioned that in case of `execute`, a plain RL solver can be used. What about `query` and `observe`? In its simplest form, we only have one option each. In case of a query the agent asks: "What should I do?" In case of an observation, the agent sends a demonstration request to the oracle and observes for a fixed time Δt.

For queries, we can add the possibility to asks `yes/no` questions whether it should perform a certain action a or not as less costly options; these queries are denoted as $\{a_i\}_{i=1}^n = A$ corresponding to the possible actions each and the query about what to do as a_0. The decision making can be based on the information gain estimates and the costs. Let the cost for a_0 and $\{a_i\}_{i=1}^n$ be $-c_0$ and $-c_1$ respectively. Then, the query policy is

$$\operatorname*{arg\,max}_{a_0, a \in A} \left[\{\mathrm{IG}_Q + c_0\} \cup \{\mathrm{KL}(\{w_i\} \| \{w_i'(a)\}) + c_1 \mid a \in A\} \right] . \tag{8.40}$$

For observing, we can apply similar measures to options about who to send a demonstration request (there might be multiple oracle and/or expert agents in the system that can demonstrate as well) and how long to observe.

More sophisticated decision making might be a research topic for future work.

8.10. Feedback to Human Oracle

As [Hadfield-Menell et al., 2016] (Section 4.5) have shown, optimal behaviour w.r.t. the task is not the same as behaving optimally to teach. However, the optimal teaching behaviour cannot be derived by the human, because he or she would need to calculate an optimization over complex feature spaces in his or her head, which no human can do. One way to tackle this problem is by using a feedback system that helps the human to develop a sense of how to teach optimally over a longer period of time (multiple execution episodes).

Formally, we define a *Feedback System* (FS) as a mapping

$$\tau_{\pi_\mu} \to \{(t_i, a_i)\}_{i=1}^n , \tag{8.41}$$

which works as follows: After an episode τ_{π_μ}, we use the current belief b_t^θ as the true reward to derive actions for optimal teaching. For every Markov step t where a query answer or demonstration move d^* was provided by the human, we compute the optimal teaching decision d^{H} according to equation 4.22. If the set of decisions is a metric space with metric M, we calculate $\|d^* - d^{\mathrm{H}}\|_M^2$, and if it exceeds some threshold ϵ, the tuple (t, d^{H}) is added to the output set. In case the set of decisions is not a metric space, we check $d^* \overset{?}{=} d^{\mathrm{H}}$ instead. After we finished processing the whole episode, the output $\{(t_i, a_i)\}_{i=1}^n$ is visualized for presentation to the human in order to give the following feedbacks: At which steps, which decisions would have been better for teaching.

Note that calculating the optimal teaching behaviour does not require the true reward to work correctly. In the original work, the authors used linear function approximations from begin with. In order to have less noise, we could restrain to give feedback in the first couple of episodes and start giving feedback when the model samples do not vary too much.

Furthermore, it is up to the human whether he or she is willing to adapt his or her behaviour to the feedback; and this will strongly depend on whether the human is able to recognize a pattern behind the difference of optimal task solving and optimal teaching in the problem domain he or she is working on.

8.11. Modelling Cooperation Partners

In decentralized settings, we could use predefined FOL statements that help to model other actors in the system in order to predict state changes better. Other than in classic RL settings, cooperation implies that, from a perspective of a single agent a_0, the environment changes on itself without active manipulation (because other actors will probably manipulate it). In our FOL setting, the all variables and constants in the knowledge base represent the state; denote them as $C = \{c_i\}_{i=1}^{n}$. Furthermore we have a set of actors excluding the agent itself $A = \{a_i\}_{i=1}^{m}$. This set can be extended dynamically. What we would like to do is

1. find a mapping $f : A \to \mathfrak{P}(C)$ that tells which objects in the world are relevant for which agents,
2. find a probability distribution $\mathbb{P}(\mathcal{A}(f(a))' \mid s, a, \mathcal{A}(f(a)))$ that give information about how agent a would probably manipulate its control space $f(a)$,

where $\mathcal{A}(\{c_1, ..., c_l\})$ are the value assignments for $\{c_1, ..., c_l\}$. Note that $f(a)$ is not necessarily the real control space of a but include only those objects that a_0 can observe. Our assumption of the model is that if objects in a subset $C_i \in C$ are likely to be changed simultaneously, then we identified a agent with its control space. These subsets can overlap. In order to solve this task, we can use deep learning techniques, that often involve unsupervised learning steps, to extract these regions of single actor manipulation and at the same time yield a predictive model of a very simplified state transition for other actors.

The information about the number of cooperative actors and its behaviours can help to solve tasks more efficiently. For example we have a task in the form of a pipelined architecture. Different actors have different capabilities to work on their respective stages in the pipeline. The dynamic identification of the number of actors working on which stage enables to detect congestion and/or overstuffing. Agents that are able to perform at stages that suffer from congestion will tend to help out at these stages. This results in an automatized load balancing, which also works when agents are leaving and joining dynamically.

9. Discussion

The first thing I want to stress out is that relational representation should be the standard for the field of study around Reinforcement Learning. It enables formulation of complex problems where it would be hard or impossible to do in propositional representations. That is why I gave a very detailed definition in the background section for a better understanding. Taking advantage of that, it was fairly easy to show the generality and expressiveness of RAP, if fact, is swallows CIRL and Dec-POMDP while at the same time being simple and even sequential in the Markov process, making any analysis on metaDP very simple.

The final failure model for the one-step-look-ahead algorithm for metaDP, for which it was argued to be representative for real problems, induces an upper bound of linear query complexity in the task duration, i.e. $\mathcal{O}(d)$. A complexity measure as simple and explicit as that was never provided in former research on HRI before. However, this thesis lacks experimental results. Future work will focus on describing many HRI problems in metaDP and testing it against other formulations.

Appendices

A. Zusammenfassung

In dieser Arbeit wird die Mensch-Roboter-Interaktion auf Grundlage von Inverse Reinforcement Learning und, etwas spezieller, Active Learning untersucht. Dabei werden ausschließlich theoretische Überlegungen angestellt und mathematische Eigenschaften hergeleitet; auf Experimente wird vollständig verzichtet.

Zunächst wird Hintergrundwissen über das Themenfeld Reinforcement Learning knapp dargestellt, mit dem Schwerpunkt, eine detaillierte Definition von Prädikatenlogik anzugeben. Der Grund ist, dass später argumentiert wird, dass die Problembeschreibung mit Hilfe von Prädikatenlogik wesentliche Vorteile mit sich zieht; auch das ist der Grund, warum RAP als Grundlage für unseren neuen Formalismus verwendet wird. Zudem zeigen wir die Allgemeinheit, die RAP bezüglich der Beschreibung nebenläufiger Multi-Agenten-Systeme hat, obwohl es von den Grundzügen einfach und der eigentliche Prozess sequenziell ist.

Als Vorbereitung werden einige der wichtigsten Meilensteine in diesem Forschungsfeld zusammengefasst dargestellt. Für jeder dieser Publikationen wird das Problem genannt, anschließend die Lösungen erklärt und schließlich die Ergebnisse präsentiert.

Als letzter Beitrag dieser Arbeit wird ein neuer Formalismus definiert, das einen Meta-Prozess beschreibt, bei dem der Agent zunächst die Entscheidung treffen muss, ob er selber handelt, um Hilfe fragt oder die Umgebung nur beobachtet. Die Allgemeinheit des Formalismus wird kurz dargestellt, bevor ein Algorithmus präsentiert wird, der, statt dem vollen Markov-Prozess, nur einen Schritt vorausschaut aufgrund von Effizienz. Im Zusammenhang des Meta-Lernens, interessieren wir uns für die Untersuchung des Algorithmus fast ausschließlich für die Anzahl der Fragen, die ein Agent stellt, weil das in realen Systemen die Operationen mit den höchsten Kosten sind. Diesbezüglich werden Konvergenz und Komplexität gezeigt.

References

Abbeel, P. and Ng, A. Y. (2004). Apprenticeship learning via inverse reinforcement learning. In *Proceedings of the twenty-first international conference on Machine learning*, page 1. ACM.

Bellman, R. (1956). Dynamic programming and lagrange multipliers. *Proceedings of the National Academy of Sciences*, 42(10):767–769.

Bernstein, D. S., Zilberstein, S., and Immerman, N. (2000). The complexity of decentralized control of markov decision processes. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, pages 32–37. Morgan Kaufmann Publishers Inc.

Choi, J. and Kim, K.-E. (2011). Map inference for bayesian inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1989–1997.

Doshi-Velez, F., Pineau, J., and Roy, N. (2012). Reinforcement learning with limited reinforcement: Using bayes risk for active learning in pomdps. *Artificial Intelligence*, 187:115–132.

Džeroski, S., De Raedt, L., and Driessens, K. (2001). Relational reinforcement learning. *Machine learning*, 43(1):7–52.

Hadfield-Menell, D., Russell, S. J., Abbeel, P., and Dragan, A. (2016). Cooperative inverse reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 3909–3917.

Hermann, M., Pentek, T., and Otto, B. (2016). Design principles for industrie 4.0 scenarios. In *System Sciences (HICSS), 2016 49th Hawaii International Conference on*, pages 3928–3937. IEEE.

Landau, E. (1924). Über die anzahl der gitterpunkte in gewissen bereichen. *Nachrichten von der Gesellschaft der Wissenschaften zu Göttingen, Mathematisch-Physikalische Klasse*, 1924:137–150.

Ng, A. Y., Russell, S. J., et al. (2000). Algorithms for inverse reinforcement learning. In *Icml*, pages 663–670.

Ramachandran, D. and Amir, E. (2007). Bayesian inverse reinforcement learning. *Urbana*, 51(61801):1–4.

Rohanimanesh, K. and Mahadevan, S. (2003). Learning to take concurrent actions. *Advances in neural information processing systems*, pages 1651–1658.

Russell, S. (1998). Learning agents for uncertain environments. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 101–103. ACM.

Russell, S. and Norvig, P. (2009). *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition.

Schöning, U. (1987). *Logik für Informatiker*. BI Wissenschaftsverlag Mannheim.

Toussaint, M., Munzer, T., Mollard, Y., Wu, L. Y., Vien, N. A., and Lopes, M. (2016). Relational activity processes for modeling concurrent cooperation. In *Robotics and Automation (ICRA), 2016 IEEE International Conference on*, pages 5505–5511. IEEE.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben.
Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet.
Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens.
Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.
Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Unterschrift:

Stuttgart, 05.10.2017

Declaration

I hereby declare that the work presented in this thesis is entirely my own.
I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations.
Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before.
The electronic copy is consistent with all submitted copies.

Signature:

Stuttgart, 5th of October, 2017