Représentations relationnelles et apprentissage interactif pour l'apprentissage efficace du comportement coopératif

Thibaut Munzer

THÈSE

PRÉSENTÉE À

L'UNIVERSITÉ DE BORDEAUX

ÉCOLE DOCTORALE DE MATHÉMATIQUES ET D'INFORMATIQUE

par **Thibaut Munzer**

POUR OBTENIR LE GRADE DE

DOCTEUR

SPÉCIALITÉ : INFORMATIQUE

Représentations Relationnelles et Apprentissage Interactif pour l'Apprentissage Efficace de Comportement Cooperatif

---

Date de soutenance : 21 avril 2017

Devant la commission d'examen composée de :

– 2017 –

Résumé Cette thèse présente de nouvelles approches permettant l'apprentissage efficace et intuitif de plans de haut niveau pour les robots collaboratifs. Plus précisément, nous étudions l'application d'algorithmes d'apprentissage par démonstration dans des domaines relationnels. L'utilisation de domaines relationnels pour représenter le monde permet de simplifier la représentation de comportements concurrentss et collaboratifs.

Nous avons commencé par développer et étudier le premier algorithme d'apprentissage par renforcement inverse pour domaines relationnels. Nous avons ensuite présenté comment utiliser le formalisme RAP pour représenter des tâches collaboratives comprenant un robot et un opérateur humain. RAP est une extension des MDP relationnels qui permet de modéliser des activités concurrentes. Utiliser RAP nous a permis de représenter à la fois l'humain et le robot dans le même processus, mais également de modéliser des activités concurrentes du robot. Sous ce formalisme, nous avons montré qu'il était possible d'apprendre le comportement d'une équipe, à la fois comme une politique et une récompense. Si des connaissances a priori sur la tâche à réaliser sont disponibles, il est possible d'utiliser le même algorithme pour apprendre uniquement les préférences de l'operateur. Cela permet de s'adapter à l'utilisateur.

Nous avons montré que l'utilisation des représentations relationnelles permet d'apprendre des comportements collaboratifs à partir de peu de démonstrations. Ces comportements sont à la fois robustes au bruit, généralisables à de nouveaux états, et transférables à de nouveaux domaines (par exemple en ajoutant des objets). Nous avons également introduit une architecture d'apprentissage interactive qui permet au système de faire moins d'erreurs tout en demandant moins d'efforts à l'opérateur humain. Le robot, en estimant sa confiance dans ses décisions, est capable de demander des instructions quand il est incertain de l'activité à réaliser. Enfin, nous avons implémenté ces approches sur un robot et montré leurs impacts potentiels dans un scenario réaliste.

Mots-clés Cooperatif, Robotique, Apprentissage par Imitation, Representations Relationnelles, Apprentissage Interactif

Laboratoire d'accueil Equipe Flowers, Inria Bordeaux - Sud-Ouest, 200 Avenue de la Vieille Tour, 33405 Talence, France

Title Relational Representations and Interactive Learning for Efficient Cooperative Behavior Learning

Abstract This thesis presents new approaches toward efficient and intuitive high-level plan learning for cooperative robots. More specifically this work study Learning from Demonstration algorithm for relational domains. Using relational representation to model the world, simplify representing concurrent and cooperative behavior.

We have first developed and studied the first algorithm for Inverse Reinforcement Learning in relational domains. We have then presented how one can use the RAP formalism to represent Cooperative Tasks involving a robot and a human operator. RAP is an extension of the Relational MDP framework that allows modeling concurrent activities. Using RAP allow us to represent both the human and the robot in the same process but also to model concurrent robot activities. Under this formalism, we have demonstrated that it is possible to learn behavior, as policy and as reward, of a cooperative team. Prior knowledge about the task can also be used to only learn preferences of the operator.

We have shown that, using relational representation, it is possible to learn cooperative behaviors from a small number of demonstration. That these behaviors are robust to noise, can generalize to new states and can transfer to different domain (for example adding objects). We have also introduced an interactive training architecture that allows the system to make fewer mistakes while requiring less effort from the human operator. By estimating its confidence the robot is able to ask for instructions when the correct activity to do is unsure. Lastly, we have implemented these approaches on a real robot and showed their potential impact on an ecological scenario.

Keywords Cooperative, Robotics, Imitation Learning, Relational Representations, Interactive Learning

Acknowledgments

I would like, first, to thank Manuel Lopes, for giving me the opportunity to do this work and for his supervision during the last three years. While letting me a lot of autonomy he was always available for discussions and directions. He took the time to debate the implications of different ideas that some times required me a long time to understand while they were oblivious for him.

I would also like to thank Yoan Mollard, the second person without which this work would have not be possible. Yoan is a talented engineer who headed the implementation of the robotic setup and conducted the user study. It was always a pleasure to work with him and the tools he created.

I am grateful to all the persons with whom I collaborated extensively, by alphabetic order: Bilal Piot, Marc Toussaint, Matthieu Geist and Olivier Pietquin.

I owe a lot to Olivier Sigaud who directed my research internship before starting my PhD. He presented me the world of research and taught me a lot, I would not be the same today without him.

All the people composing the 3rdHand project have been great coworkers. It was always a good time going to the meetings and collaborating with them.

Being part of the Flowers team was a great experience. All the team members, past and present, were always open to discuss interesting subjects from science to politics.

Lastly, I would like to thank Inria and Nicolas Jahier. The first for being a great place to work and the second for being a very good team assistant.

This work was funded by European Union under the grant agreement FP7-ICT-2013-10-610878.

Contents

Chapter 1

Introduction

1.1 Context

Robots have been used in an industrial context for a long time. They allowed to automate tedious and repetitive tasks, for example in the automotive industry, liberating time for humans to do other things. Because industrial robots are fast and powerful machines, they have mainly been used in separations of humans as to avoid arming them. However, new technological developments, in software and hardware, makes them safer and introduce a future where human and robot could work together.

1.1.1 A need for collaborative robots

This evolution would be profitable for both the employees and companies level. For the company, it makes it possible to tackle new tasks that neither human or robot alone can accomplish today. At the employee level, it changes his role to rely less on his strength and more on his versatility. A team composed of a human worker and a robot combines the strength and precision of the robot and the intelligence and flexibility of the human worker.

This trend already started in the form of the *cobots*. Derived from the words *collaborative* and *robot*, a cobot is a neologism that design robotic systems that work closely with a human in order to improve its capacities. Most cobots today are passive systems not showing initiative but only exposing a reactive behavior. A simple example of cobot, as presented in Figure 1.1, is a compliant robotic arm equipped with a power tool where the motion is restricted to predetermined path letting the operator achieve his task while improving strain on the user by carrying most of the load and security by restraining the possible position of the power tool.

Figure 1.1: Illustration of a cobot.

1.1.2 A need for easily programmable robots

The main lever that pushed robots from research laboratory to industry was economical. Companies realized it was more economical to pay one time experts to design a robotic system to realize a task than to pay for every task realization. However, this equation is only true for tasks that are done a lot of time in order to amortize the investment. For short-lived tasks, it stayed economically better to pay for each task realization than to pay a big upfront cost to an expert to design a robotic system.

By making robots easier to program, and thus reducing the upfront cost of the robotic system, robots become also economically viable for short-lived task. However, to really change the economics and thus the viability to program a robot for short-lived tasks, it should be doable by a non expert. By making a factory worker able to reprogram the robot at will, the cost of such an operation becomes trivial. In order to do so, the programming must be intuitive and require no advanced mathematical, robotics or computer science concepts.

In order to make robot viable for short-lived tasks, this learning process must be as effective as possible, optimally making the robot useful from the first task realization.

1.1.3 A collaborative and easily programmable robot

In this thesis, we explore the combination of these two aspects: how to make a collaborative robot more easy to program. The term program is used here loosely, it refers to the process to make a robot behave a certain way. In practice, we propose, for the robot, to use machine learning to acquire a behavior from experience. We will also rely on relational representations of the world as it allows encoding the concurrency needed to learn a cooperative behavior as well as transferring well in a variety of scenarios.

1.2 Objectives

The main goal of this thesis is to work toward an easily programmable collaborative robots, the objectives are threefold:

- First, we want to improve the state of the art in Programming by Demonstration in relational domains. By developing an IRL algorithm for relational domains, we hope to endow robots with a higher capability to transfer to new domains and tasks. Indeed, both IRL and relational domains allow transferring, we think that a combination of these two tools would allows transferring more robustly to new domains and tasks.

- Second, we aim to apply Programming by Demonstration algorithms to cooperatives and concurrent domains. Most research in Programming by Demonstration has been focused on sequential task. However to learn the behavior of an support robot we propose to learn the behavior at the team level which require us to work on concurrent domains so both the robot and the human can act simultaneously. This also allows the robot to realize concurrent actions and be more useful to the human.

- Lastly, we want to build an efficient learning procedure. We plan to use the interactive learning paradigm to this end as it makes the robot helpful as soon as possible.

1.3 State of the art

As robotics research evolved, different sub-field emerged. This work is based on different concepts and approaches proposed by the robotics and machine learning communities.

Some notions used in this section are defined or explained in Chapter 2. We advise the reader not familiar with Markov Decision Process (MDP) and Relational Representation to read Chapter 2 before pursuing.

1.3.1 Robot Learning

An important sub-field of robotics is Robot Learning, the use of Machine Learning to endows robots new possibilities. In some cases, machine learning is used to create control models that outperform human designed ones. One famous example is the work of Kober and Peters [2009] where the authors make a robot plays the Ball-in-a-Cup and Ball Padding games using Reinforcement Learning (RL). These kind of highly dynamic tasks are hard to solve using an analytic controller because some forces involved are hard to model. However it is also possible to use machine learning algorithms and tools to create robots that can learn to act from human teaching, usually called Learning from Demonstration (LfD).

Research on LfD has included: i) low-level learning, where the goal is to learn a mapping from sensor output to motor command, e.g. learning motor policies [Chernova and Veloso, 2009] or navigation [Knox *et al.*, 2013; Jain *et al.*, 2013]; ii) symbolic learning, where the goal being is to learn a policy, a mapping from a symbolic space state to the space of actions [Natarajan *et al.*, 2011] or learning rewards [Abbeel and Ng, 2004] from demonstration. Learning the reward, called Inverse Reinforcement Learning (IRL) usually allow to generalize better. Some works also consider learning both motor policies and symbolic learning at the same time using a hierarchical approach [Niekum *et al.*, 2013].

1.3.2 Interactive Learning

In most examples of Robot Learning, there is a clear separation between the learning and the execution phase. This has several drawbacks as the number of demonstrations might be larger than needed, and might not even cover critical aspects of the task. To address such problems interactive scenarios where both phases are merged have been proposed. Chernova and Veloso [2009] propose a system where the robot only makes queries when, in a given state, the confidence on the actions passes a given threshold. Alternatively, in other works [Lopes *et al.*, 2009b], the system request information about relevant states to learn a good reward representing the task. Other approaches provide a smooth transition between the phases, a first phase of teleoperation where the policy [Grollman and Jenkins, 2007], or the preference [Mason and Lopes, 2011], is learned and, at any time, the user can resume teleoperating to provide corrections.

A new trend of interactive learning systems is to rely on weak feedback to handle situations where optimal feedback is impossible or costly to produce by the human teacher. In particular, Shivaswamy and Joachims [2012]; Jain *et al.* [2013] relies on local improvements of the current policy while Akrour *et al.* [2011] asks for ranking between two or more policies.

The concept of cross-training is explored in the work of Nikolaidis and Shah [2013] where the robot and the user simultaneously adapt to each other by switching roles. The robot learns directly a policy that better adapts to the user preferences. This improves team efficiency and acceptance metrics.

1.3.3 Collaborative Behavior Representation

Another line of research considers not just learning individual tasks but also how to learn collaborative tasks. Several works have shown that learning the expected behavior from the human teammate has a positive impact on both cooperation and engagement of the user [Lee *et al.*, 2012; Mitsunaga *et al.*, 2008].

Different formalisms can be used to model such cooperation. Most research relying on concurrent formalisms focus on planning. An example is the Concurrent Action Model (CAM) framework of Rohanimanesh and Mahadevan [2005] where the action space is defined as the scalar product of the actions. Another core reference for concurrent action planning is Mausam and Weld [2008], which discusses reductions of Concurrent MDPs (roughly, MDPs with multi-actions) and Concurrent Probabilistic Temporal Planning (CPTP) (planning with durative multi-actions) to planning in MDPs with extended state spaces. The relation or difference to CAM, especially between the *aligned* and *interleaved epoch* reductions of CPTPs and the *any*, *all*, and *continue schemes* of CAMs is not explicitly clarified in Mausam and Weld [2008], but they are very similar.

Concerning planning algorithms, Smith and Weld [1999] propose Temporal GraphPlan (TGP), Younes and Simmons [2004] present a Generate, Test and Debug (GTD) algorithm, and Mausam and Weld [2008] evaluate Real Time Dynamic Programming (RTDP) on the various MDP-reductions. Aberdeen and Buffet [2007] improves on this based on policy gradient methods. We are not aware of previous work using Monte Carlo methods, despite the recent success of Monte Carlo (MC) methods in very large domains and games [Browne *et al.*, 2012].

1.3.4 Relational Representations

Some works have considered using Relational Representation to model the world. In a relational domain the state is described in terms of properties of, and relations between "objects". Models of the environment (the transition dynamics and reward function) as well as policies generalize over objects and naturally generalize to domains with different and varying number of objects. Therefore, statistical relational learning (regression & classification learning in relational domains [Natarajan *et al.*, 2012]) as well as relational RL (including model-free relational RL [Džeroski *et al.*, 2001], model-based relational RL [Lang *et al.*, 2012]) showed great success in scaling to much larger domains than what would be possible with propositional representations.

Learning to act from demonstrations in relational domains have been a research problem for a long time [Segre and DeJong, 1985; Shavlik and DeJong, 1987; Khardon, 1999; Yoon *et al.*, 2002]. The use of relational representations is attracting even more attention due to new algorithmic developments, new problems that are inherently relational and the possibility of learning the representations from real-world data, including robotic domains [Lang and Toussaint, 2010; Lang *et al.*, 2012]. Natarajan *et al.* [2011] propose to use of gradient-tree boosting [Friedman, 2001] to achieve Imitation Learning (IL) in relational domains. However no IRL algorithm has been proposed for relational domains.

Relational representation also allow representing more naturally concurrency. While concurrency using Relational Representations has been extensively considered in a planning context, it has received less attention in a RL context. Relational RL [Džeroski *et al.*, 2001; Lang *et al.*, 2012] has demonstrated the great benefit of exploiting relational representations in respective domains. But it has not yet been leveraged to learn in concurrent cooperative domains.

Of course, this effort is aimed toward building tools that can be used in an industrial context. As such, proposed methods should be validated on a real setup and a user study should be conducted.

1.4 Contributions

We now list the main contributions of this thesis:

- Inverse Reinforcement Learning in Relational Domains: We developed and evaluated the first IRL algorithm for Relational Domains. See Chapter 3.
- Supervised Behavior Learning in Relational Activity Process: We showed that it is possible to learn behaviors in RAPs either using Imitation Learning or Inverse Reinforcement Learning. See sections 4.5 and 4.6.
- Multiple choices confidence estimation: We proposed a novel method for potential error estimation in tasks with multiple optimal actions (see section 5.3.5)
- Interactive Learning in Relational domains: We created an architecture allowing to interactively learn behavior in Relational Domains. See section 5.3.1.
- Interactive Learning for collaborative robots: Using RAP this architecture can be used in concurrent decision processes and in particular for collaborative decision processes. See section 5.3.2.
- Interactive Preferences Learning: We showed how one can learn user preferences interactively by treating the task as prior knowledge and using a gradient method to learn a policy reflecting the preferences. See section 5.3.3.
- A user study on the impact of robot initiative: We conducted a user study comparing a robot that progressively takes initiative and a robot that wait for an order to act. See 6.

1.5 How to read this thesis

This thesis is composed of six chapters. The first chapter is this one and serves to introduce the general context of this thesis. The second chapter presents the main theoretical tools used through this thesis. Then the next four chapters are the four most important papers produce during the thesis. The last chapter concludes this document.

The first core chapter of this thesis details an IRL algorithm for relational domains. In propositional domains, IRL has been shown to generalize better than IL to unseen state as well as transfer better to different domains. In this chapter, we develop and evaluate the first IRL algorithm for relational domain.

The second core chapter presents the Relational Activity Process (RAP) framework. This framework based on semi-Relational Markov Decision Process (semi-RMDP) allows representing concurrent actions in relational domain. A necessary step toward cooperative behavior learning.

The third core chapter introduces interactive cooperative behavior learning and is the final product of this thesis. It presents an efficient architecture for behavior learning in teams built on RAP.

The last core chapter presents a small scale user-study where we studied the impact of robot initiative on user acceptance. It compares the architecture developed in the previous chapter with a passive behavior where the user has to instruct the robot to do each activity.

Chapter 2

Background

2.1 Markov Decision Process

Markov Decision Process (MDP) are a formal model of a sequential decision process of an agent in a dynamic environment. It is dynamic in the sense that the agent actions modify stochastically the state of the environment. MDPs have been traditionally used to solve the Reinforcement Learning (RL) problem. In RL after each action taken, the agent receives a reward depending on the state and the action. The goal is to predict the correct action in each state in order to maximize the cumulated reward.

Formally, a MDP is a tuple $M_R = \{S, A, R, P, \gamma\}$ where $S = \{s_i\}_{1 \le i \le N_S}$ is the state space, $A = \{a_i\}_{1 \le i \le N_A}$ is the action space, $R \in \mathbb{R}^{S \times A}$ is the reward function, $\gamma \in]0, 1[$ is a discount factor and $P \in \Delta_S^{S \times A}$ (Δ_S is the set of distributions over S) is the Markovian dynamics which gives the probability, $P(s'|s, a)$, to reach s' by choosing action a in state s. A deterministic policy $\pi \in A^S$ maps each state to an action and defines the behavior of the agent.

In some MDP, different actions are available to the agent depending on the state. In this case, we define $\mathcal{A}$ as a function from the space of states to the space of sets of actions: $\mathcal{A} : S \to P(A)$ ($P(A)$ is the powerset of A).

The quality function $Q_R^\pi \in \mathbb{R}^{S \times A}$ for a given policy is a measure of the performance of this policy and is defined for each state-action couple (s, a) as the expected cumulative discounted reward when starting in state s, performing the action a and following the policy π afterwards. More formally, $Q_R^\pi(s, a) = \mathbb{E}_{s,a}^\pi[\sum_{t=0}^{+\infty} \gamma^t R(s_t, a_t)]$, where $\mathbb{E}_{s,a}^\pi$ is the expectation over the distribution of the admissible trajectories $(s_0, a_0, s_1, \pi(s_1), \dots)$ obtained by executing the policy π starting from $s_0 = s$ and $a_0 = a$. A policy π is said optimal when :

$$\forall \pi' \in A^S, \forall s \in S, Q_R^\pi(s, \pi(s)) > Q_R^{\pi'}(s, \pi'(s)). \tag{2.1}$$

Moreover, the function called the optimal quality function, noted $Q_R^* \in \mathbb{R}^{S \times A}$ and defined as $Q_R^* = \max_{\pi \in A^S} Q_R^\pi$ is important as each optimal policy π^* is greedy with respect to it [Puterman, 1994]:

$$\forall s \in S, \pi^*(s) \in \operatorname*{argmax}_{a \in A} Q^*_R(s,a). \tag{2.2}$$

In addition, it is well known [Puterman, 1994] that the optimal quality function satisfies the optimal Bellman equation:

$$Q^*_R(s,a) = R(s,a) + \gamma \sum_{s' \in S} P(s'|s,a) \max_{b \in A} Q^*_R(s',b). \tag{2.3}$$

Equation (2.3) expresses a one-to-one relation between optimal quality functions and their reward functions [Puterman, 1994].

2.2 Learning from Demonstration

A second task apart from RL, for which MDP have been successfully used is the Learning from Demonstration (LfD) problem, sometimes called apprenticeship learning problem. In this problem, the agent observes an expert realizing a task. After some time, it must mimic the expert behavior.

It can be useful in different scenarios. One important case is to learn a policy when the reward is not trivial. For a user, defining a reward is in most cases a difficult problem. The agent might find ways to maximize the reward not foreseen by the user and that don't satisfy him. This is especially true for naive users. In that case, it can be simpler and more practical to demonstrate a behavior.

The observation of the expert behavior is a dataset of expert decisions, $D_E = [(s_i, a_i)]_{i=0}^N$. The dataset is composed of N couples (s,a) with $s \in S$ and $a \in A$. The problem is to find a policy $\hat{\pi}$ such that $\forall(s,a) \in D, \hat{\pi}(s) = a$.

Of course, without other constraints, defining such a policy is trivial. In practice, it is expected that the policy can generalize to unseen states.

Another difficulty arises when the expert realizes different actions for a given state. Indeed, often the expert can realize different actions that are optimal. While encountering the same state more than once, the expert might choose different actions each time.

There are two main approaches to the LfD: Imitation Learning (IL) and Inverse Reinforcement Learning (IRL).

2.2.1 IL

In IL the problem of LfD is cast as a classification problem. The states are treated as the inputs of the classifier while the actions are treated as the labels.

One advantage of IL is that the agent doesn't need to have access to the dynamics of the environment to replicate the expert behavior. However, as it generalizes directly based on the state representation, it is only useful when the representation allows generalizing the expert policy.

2.2.2 IRL

In RL one tries to find a policy that maximize a given reward, in IRL, one tries to find a reward that explain an observed policy. More formally, given a dataset of expert decisions, $D_E = [(s_i, a_i)]_{i=0}^{N}$, the goal is to find a reward, $\hat{r}$ such that $\forall(s,a) \in D, \forall b \in A, Q^*_{\hat{r}}(s,a) \geq Q^*_{\hat{r}}(s,b)$

The IRL problem is ill-posed as they are an infinite number of reward function for any policy. As for IL, the problem here is to find a reward such that the learned policy generalizes to unseen state. In order to do so, most IRL algorithms make assumptions on the reward shape.

2.3 Relational MDP

Relational MDPs generalize MDPs for high-level representations [Džeroski *et al.*, 2001]. This representation makes possible to model generic objects instead of specific instances of objects, allowing to generalize over objects, contexts, tasks and policies. Solutions to the planning and learning problems can be found in the literature [Kersting *et al.*, 2004; Lang and Toussaint, 2010].

2.3.1 Model

Relational MDPs use a subset of first-order logic. We now define the main concepts in a simplified way:

- Constant: a constant is a symbol that refers to an object of the domain, usually a real world object but it can also be a number, a color, ...
- Variable: a variable is a symbol that can be substituted by a constant. We follow the classic prolog notation where identifier starting with a capital refer to variable and lowercase refers to constant
- Term: a term is either a constant or a variable.
- Predicate: a predicate is a symbol p/n where n is the arity of the predicate. It represents a relation between n constants or variables.
- Atom: an atom is an expression of the form $p(a_1, .., a_n)$ where p/n is a predicate and a_i are terms.
- Formula: a formula is a set of atoms and negated atoms.
- Substitution: a substitution is a mapping from variables to terms, noted $\sigma = \{X_1 \mapsto t_1, ..., X_n \mapsto t_n\}$. Applying a substitution to a formula f, noted $f\sigma$ consist in replacing all occurrences of $X_1, ..., X_n$ in t by, respectively, $t_1, ..., t_n$.

- Grounding: a grounding is a substitution such that there are no variables in the mapped terms.

Under this representation, the state of the environment, given a set of predicates and a set of constants, is the set of all true grounded atoms.

For example, we can model, using a relational representation, an environment where three blocks are on the top of a table and can be stacked. There are four objects represented by the constants: b1, b2, b3 and table and we use three predicates to represent the different states: on/2, clear/1 and block/1. The state where b1 is on top of b2 and both b2 and b3 lie on the table will then be:

```
{on(b1, b2), on(b2, table), on(b3, table), clear(b1),
 clear(b3), block(b1), block(b2), block(b3)}
```

The state of the environment changes when relational actions are applied. In the blocksworld example, we can model a move(A, B, C) action that represents moving object A from the top of B to the top of C. The action is abstract in the sense that it can be grounded to different objects. This action can represent moving b1 from b2 to table, move(b1, b2, table) as well as moving b2 from table to b1, move(b2, table, b1).

Traditionally, in such domains, the transition function is represented as a set of rules that contain 3 parts: the action, the context, and the outcomes. There are different ways to represents it, we use the following one:

- the action and its argument (actions are atoms),
- the context, a formula that represents a precondition to executing the action,
- the outcome, a formula that represents the effect of executing the action,

and we represent them like this:

```
action:
  { context }
  { outcome }
```

The set of feasible grounded actions in a given state, $\mathcal{D}(s_t)$, is composed of all grounded abstract actions for which a rule, $a : \{c\}\{o\}$, exists and there is a grounding σ such that the context is true in this state, $c\sigma \subset s_t$. An abstract action can be present more than once in the decision set with different groundings.

If the agent decides to execute the action $a\sigma$, the resulting state is obtained by applying the list of effects.

In our running example the transition function for the action move(A, B, C) will be represented with three rules:

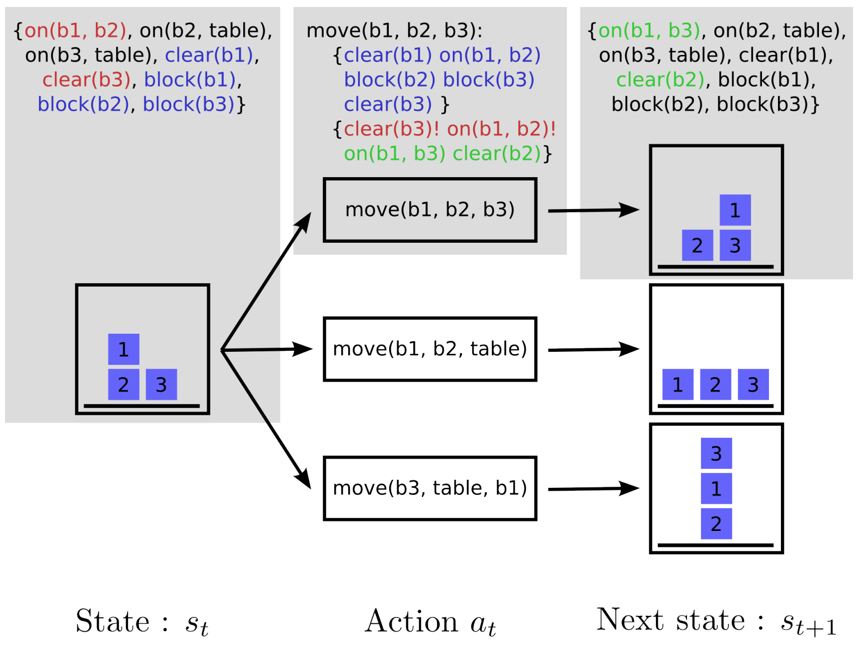

{on(b1, b2), on(b2, table), on(b3, table), clear(b1), clear(b3), block(b1), block(b2), block(b3)}
move(b1, b2, b3): {clear(b1) on(b1, b2) block(b2) block(b3) clear(b3) } {clear(b3)! on(b1, b2)! on(b1, b3) clear(b2)}
{on(b1, b3), on(b2, table), on(b3, table), clear(b1), clear(b2), block(b1), block(b2), block(b3)}
move(b1, b2, b3)
move(b1, b2, table)
move(b3, table, b1)
1
2
3
State : s_t
Action a_t
Next state : s_{t+1}

Figure 2.1: Sketch of an example of transition in the blocksworld domain. The context is highlighted in blue, the positives outcomes are highlighted in green and the negatives ones in red. We favor red over blue. The rule used in the first action is the first one with the substitution $\{A \mapsto b1, B \mapsto b2, C \mapsto b3\}$.

```
move(A, B, C):
  { clear(A) on(A, B) block(B) block(C) clear(C) }
  { ¬clear(C) ¬on(A, B) on(A, C) clear(B) }
move(A, B, C):
  { clear(A) on(A, B) ¬block(B) block(C) clear(C) }
  { ¬clear(C) ¬on(A, B) on(A, C) }
move(A, B, C):
  { clear(A) on(A, B) block(B) ¬block(C) }
  { ¬on(A, B) on(A, C) clear(B) }
```

We use the ¬ symbol to represent the negation.

The figure 2.1 illustrates a transition in the blocksworld example.

2.3.2 Relational Regression

TILDE [Blockeel and De Raedt, 1998] is an algorithm designed to do classification and regression over relational data. It is a decision tree learner similar to C4.5 [Quinlan, 1993]. It follows the principle of top down induction of decision trees where the dataset is recursively and greedily split by building a tree according to a criterion until all data points in one subset share the same label. The change made to handle relational data is to have first order logic tests in each node. These tests are logical formulas of one atom with free variables. TILDE can be used for regression if one allows leaves to contain real numbers.

2.3.3 IL for RMDP

One of the main difficulties working with relational representation, as opposed to vectorized ones, is the little number of learning methods available. In this thesis, we will use the TBRIL algorithm for policy learning. TBRIL is an algorithm that works by finding a quality function such that seen actions are optimal for their associated state.

By first defining a smooth mapping between quality function and policy:

$$\pi(s,a) = \frac{e^{-\beta q(s,a)}}{\sum_{b \in D(s)} e^{-\beta q(s,b)}},$$

the problem becomes finding the function q^* that minimizes the negative log-likelihood of the dataset D. Formally:

$$q^* = \text{argmin}(-log(L(D|q))$$

$$q^* = \text{argmin}(\sum_{(s,a) \in D} -log(\pi(s,a)))$$

TBRIL solves this problem using the Gradient Boosting (GB) algorithm [Friedman, 2001]. GB it the transposition of the gradient descent algorithm in function space. Given a function space $C \to \mathbb{R}$ and an objective functor L to minimize, to apply gradient descent in function space, is to iteratively define $f_{i+1} = f_i + \frac{\delta L(f_i)}{\delta f_i}$ with f_0 an initial solution (can be the null vector). More precisely, for any value of the input space, $c \in C$, $f_{i+1}(c) = f_i(c) + \frac{\delta L(f_i)}{\delta f_i(c)}$.

Gradient Boosting comes into play when it is impossible or not sensible to compute $\frac{\delta L(f_i)}{\delta f_i(c)}$ for every $c \in C$, either because $|C|$ is infinite or because there are $c \in C$ for which values of $f_i(c)$ as no influence on $L(f_i)$. A weak regressor R_i is learned to predict/approximate the values of $\frac{\delta L(f_i)}{\delta f_i(c)}$ for any $c \in C$. From a subset $B \subset C$, a dataset is build $[(c, \frac{\delta L(f_i)}{\delta f_i(c)})]_{c \in B}$ from which R_i is learned. f_{i+1} is then defined as $f_{i+1} = f_i + R_i$.

In our case, the function space is $S \times A \to \mathbb{R}$, the functor is $L(Q) = \sum_{(s,a) \in D} -log(\pi(s,a))$ and B is defined as $B = \cup_{(s,a) \in D}\{(s,a')|a' \in \mathcal{D}(s)\}$.

Because we are working with relational representations, we use as weak learner relational regression trees [Blockeel and De Raedt, 1998].

As a gradient algorithm, TBRIL needs a starting point. This starting point is a quality-function, Q^*_{prior}, and can be set to 0. However, it can also be set to any quality function closer to the goal if prior knowledge is available, allowing to speed up learning. Alg. 1 sums up the algorithm.

Algorithm 1 TBRIL

```
1: procedure TBRIL($Q^*_{prior}$, $D$, $nb\_iter$)
2:     $Q^*_{target} \leftarrow Q^*_{prior}$
3:     for i $\in [0, ..., nb\_iter]$ do
4:         $D_g \leftarrow compute\_gradient(Q^*_{target}, D)$
5:         $f_i \leftarrow learn\_regression\_tree(D_g)$
6:         $Q^*_{target} \leftarrow Q^*_{target} + f_i$
7:     return $Q^*_{target}$
```

Chapter 3

Relational Inverse Reinforcement Learning

This chapter describes an Inverse Reinforcement Learning (IRL) algorithm for relational MDP. It has been published previously as a conference paper under the name "Inverse Reinforcement Learning in Relational Domain" in Ijcai conference [Munzer *et al.*, 2015]. Co-authors are Bilal Piot, Matthieu Geist, Olivier Pietquin and Manuel Lopes. It has been edited to improve coherence throughout the thesis.

This chapter main goal was to develop an IRL algorithm that we could use later to learn policies that generalize and transfer better than IL. It was also a good entry point to familiarize ourselves with relational representation, relational MDP and relational policy learning.

3.1 Introduction

Learning control strategies or behaviors from observations is an intuitive way to learn complex skills [Schaal, 1999; Argall *et al.*, 2009; Khardon, 1999]. When learning from observing another agent, one can aim at learning directly the behavior or, instead, the criteria behind such behavior. The former approach is usually called Imitation Learning (IL) while the latter is called IRL.

The main advantage of IRL is the robustness of the behavior induced. It can handle different initial states and changes in world dynamics [Ng and Russell, 2000; Neu and Szepesvári, 2009]. With IRL, an explanation of the behavior is found, so the system can continue to learn online to fulfill this explanation via an online Reinforcement Learning (RL) algorithm. IRL also improves performance over learning the policy if a change in the world dynamics occurs. Indeed, in that case, the system will learn to adapt in order to achieve its goal while continuing to follow the same behavior (via IL) would not. Being robust to dynamics modifications is an important property as, for instance, when a system ages its dynamics changes (*e.g.*, the brake of a car will get worn).

IRL can also offer a more compact representation of the behavior, modeled as a reward function. For instance, in a blocks world domain, the task of building a tower with all objects requires a non-trivial policy but can be described with a simple reward [Džeroski *et al.*, 2001]. This is useful when the user has to be aware of the internal state of the system, as a more compact representation is easier to understand for a human operator. With active learning, the user has to correct/advise the system. So, he has to quickly be able to determine what it should do. Human-machine collaboration is another setting where the user has to take into account the system and so, be informed of the internal state.

In this chapter, the main contribution is to introduce the first IRL algorithm for relational domains. For this we generalize a previous approach for IRL, namely Cascaded Supervised IRL (CSI) [Klein *et al.*, 2013], to handle relational representations. Another contribution is to augment CSI with a reward shaping step to boost performance. A third contribution is to show that using data from different domain sizes can improve transfer to unseen domain sizes.

3.2 IRL in Relational Domains

IRL is a method that tries to find a reward function $\hat{R}$ that could explain the expert policy π_E [Ng and Russell, 2000; Russell, 1998]. More formally, an IRL algorithm receives as inputs a set D_E of expert sampled transitions $D_E = (s_k, a_k = \pi_E(s_k), s'_k)_{1 \leq k \leq N_E}$ where $s_k \in S$ and $s'_k \sim P(.|s_k, a_k)$ and some information about the world dynamics, for instance a set of non-expert sampled transitions $D_{NE} = (s_l, a_l, s'_l)_{1 \leq l \leq N_{NE}}$ where $s_l \in S$, $a_l \in A$, and $s'_l \sim P(.|s_l, a_l)$. Then the algorithm outputs a reward $\hat{R}$ for which the observed expert actions are optimal.

Most of IRL algorithms can be encompassed in the unifying trajectory matching framework defined by Neu and Szepesvári [2009]. These algorithms find a reward function such that trajectories following the optimal policy with respect to this reward function become *close* to the observed expert trajectories. Each step of the minimization thus requires an Markov Decision Process (MDP) to be solved so as to generate trajectories. These algorithms are for instance Policy Matching [Neu and Szepesvári, 2007] minimizing directly the *distance* between the obtained policy and the expert policy or Maximum Entropy IRL [Ziebart *et al.*, 2008] minimizing the Kullback-Leibler (KL) divergence between the distribution of trajectories. Those algorithms are incremental by nature and have to solve several MDPs. On another hand, other IRL algorithms such as Structured Classification for IRL (SCIRL) [Klein *et al.*, 2012] and Cascaded Supervised IRL (CSI) [Klein *et al.*, 2013] avoid resolving recursively MDPs.

In addition, most IRL algorithms are parametric and assume a linear parameterization of the reward. However this hypothesis do not hold for relational domains where states are logical formulas. Yet, CSI can be made non-parametric as we will demonstrate later. This, added to the fact that CSI doesn't need several MDPs to be solved, makes him a good candidate for an IRL algorithm adapted to relational domains. After presenting the original CSI algorithm, we show how it can be improved by introducing an intermediate step. Then, a relational version is developed.

3.2.1 CSI

The idea behind CSI is that it is hard to define an operator that goes from demonstrations to a reward function. On the other hand, one can define operators to go from demonstrations to an optimal quality function and from an optimal quality function to the corresponding reward function. The second option can be made computationally efficient because of the link between the score function of a multi-class classifier and an optimal quality function.

Indeed, given the data set $D_{CE} = (s_k, a_k)_{1\leq k\leq N_E}$ extracted from D_E, a classification algorithm outputs a decision rule $\pi_C \in A^S$ using s_k as inputs and a_k as labels. In the case of a Score Based Classification (SBC) algorithm, the output is a score function $q_C \in \mathbb{R}^{S\times A}$ from which the decision rule π_C can be inferred :

$$\forall s \in S, \pi_C(s) \in \operatorname*{argmax}_{a\in A} q_C(s,a). \tag{3.1}$$

A good classifier provides a policy π_C which often chooses the same action as π_E, $\pi_C \approx \pi_E$. Equation (3.1) is very similar to equation (2.2) thus by rewriting the optimal Bellman equation (2.3), q_C can be directly interpreted as an optimal quality function $Q^*_{R_C}$ for the reward R_C defined as follows:

$$R_C = q_C(s,a) - \gamma \sum_{s'\in S} P(s'|s,a) \max_{b\in A} q_C(s',b). \tag{3.2}$$

Indeed, as q_C verifies $q_C(s,a) = R_C(s,a) + \gamma \sum_{s'\in S} P(s'|s,a) \max_{b\in A} q_C(s',b)$ and by the one to one relation between optimal quality functions and rewards this means that $q_C = Q^*_{R_C}$. In addition, as π_C is greedy with respect to q_C, π_C is then optimal with respect to R_C. Thus, the expert policy π_E is quasi-optimal for the reward R_C as $\pi_C \approx \pi_E$.

However, R_C can be computed exactly only if the dynamics P is provided. If not, we can still estimate R_C by regression. For this, we assume that we have a set D_{NE} of non-expert samples. So, we can easily construct a regression data set $D_R = \{(s_i,a_i),\hat{r}_i\}_{1\leq i\leq N_{RL}}$ from $D_E \cup D_{NE} = (s_i,a_i,s'_i)_{1\leq i\leq N_E+N_{NE}}$ where $\hat{r}_i = q_C(s_i,a_i) - \gamma \max_{a\in A} q_C(s'_i,a)$ is an unbiased estimate of $R_C(s_i,a_i)$. The output of the regression algorithm is an estimate $\hat{R}$ of the target reward R_C.

Reward Shaping augmented CSI

Regression implies projecting the data onto an hypothesis space. Given this hypothesis space some reward functions are easy to represent while other are hard or impossible due to the so-called inductive bias. There is a one-to-many relation between the space of demonstrations and the space of quality functions and a one-to-one relation between the space of quality functions and the one of reward functions. This means that there are many optimal candidates for the score based classification step among which one could choose the one that will be projected with the smallest error in the hypothesis space during regression.

To improve the quality of its regression step, we propose to introduce an intermediate Reward Shaping (RS) step to CSI. Reward shaping is a technique aiming at modifying the reward shape while keeping the same optimal policy. Ng *et al.* [1999] proved that $\forall R \in \mathbb{R}^{S\times A}, \forall t \in \mathbb{R}^{S}, q_t(s,a) = Q^*_R(s,a) + t(s)$ and $Q^*_R(s,a)$ share the same optimal policies and one can interchangeably use q_t or Q^*_R (or their corresponding reward function) to represent a behavior or a task. Although RS is traditionally used to guide RL, here we propose to use it to shape the reward so that it can be more efficiently projected onto the chosen hypothesis space.

In this context the purpose of RS is to find a function $t^* \in \mathbb{R}^S$ such that the expected error of the regression step is minimal. If one can define a criterion c over the reward values that represent the expected regression error (even heuristically), an optimization problem can be written:

$$t^* = \operatorname*{argmin}_t J(t),$$

$$J(t) = c([q'_C(s_i,a_i) - \gamma \max_{a\in A} q'_C(s'_i,a)]_{(s_i,a_i,s'_i)\in D_E\cup D_{NE}}).$$

where $q'_C(s,a) = q_C(s,a) + t(s)$. The criterion c can vary substantially. For example, if the regression model is sparse the ℓ_1-norm can be used. On the other hand, if the hypothesis space is composed of regression trees the entropy of the set of reward values is a candidate: a low entropy implies a few numbers of values which heuristically leads to a better representation with a decision tree that can only represent a finite number of values.

As only a finite number of values of t change the value of c (those for which the state is present in D_E or D_{NE}), t can be treated as a vector and standard black-box optimization tools can be used. We propose to use a simplified version of CMA-ES [Hansen *et al.*, 2003].

Figure 3.1 summarizes the CSI method augmented with RS. It consists in a first step of score-based classification ① with the set D_E as input which outputs a score function q_C. RS ② is then used to produce $q'_C(s,a) = q_C(s,a) + t^*(s)$ which corresponds to an easier reward function to learn. From the set D_R constructed from D_E, D_{NE} and q'_C, it is possible to compute an estimate $\hat{R}$ of the reward function R_C via regression ③. Theoretical guarantees on the

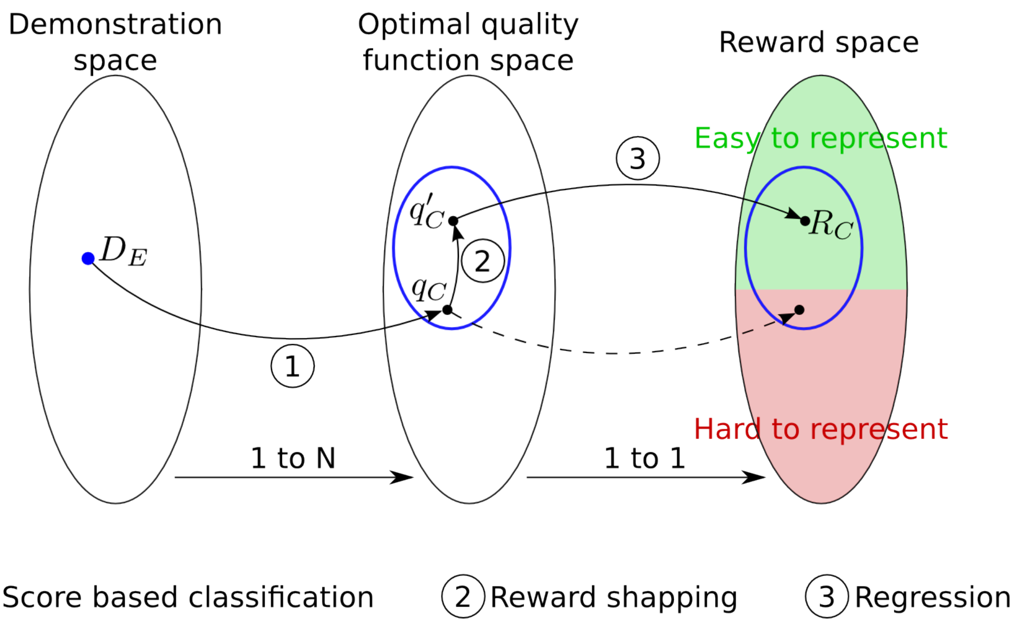


Figure 3.1: Sketch of the proposed method : CSI with reward shaping. See text for explanations.

quality of the reward function learned via CSI are provided by Klein *et al.* [2013] with respect to the errors of the classification step and the regression step.

3.2.2 Lifting to the relational setting

As stated earlier most algorithms for IRL in the literature rely on a parametric (propositional) representation of the MDP state. However, an IRL algorithm have to be non-parametric in order to be used in relational domains. We show here that CSI can be made non-parametric by using different supervised learners (step ① and ③) than Klein *et al.* [2013]

Relational SBC

Natarajan *et al.* [2011] have developed an algorithm to realize SBC in relational domains, Tree Boosted Relational Imitation Learning (TBRIL). Their algorithm is an adaptation of the gradient boosting method [Friedman, 2001] where standard decision trees have been replaced with TILDE, a relational decision trees learner (see next section). They use their method to learn a policy on relational domains from expert demonstrations but TBRIL can be more broadly used for any SBC problem in relational domains. We use TBRIL

[1] as a relational SBC ①.

Relational regression

We use the TILDE algorithm as a relational regressor ③. To be able to represent complex functions we allows TILDE to use the *count* aggregator as often done in the relational learning literature [De Raedt, 2008].

Relational CSI

We propose the Relational CSI algorithm (RCSI). We make three modifications to CSI : (i) use TBRIL as the SBC step ①, (ii) add an intermediate RS step ② to improve the performance and (iii) use TILDE for regression step ③.

TILDE is a decision tree based regressor and therefore we choose the criteria c of the RS step to be the entropy.

3.3 Experiments

To validate the proposed approach, experiments have been run to (i) confirm RCSI can learn a relational reward from demonstrations, (ii) study the influence of the different parameters and (iii) show that IRL outperforms classification based IL when dealing with transfer and changes in dynamics.

3.3.1 Experimental setup

To test RCSI quantitatively we use the following setup. From a target reward R^*, we compute an optimal policy π^*. The algorithm is given, as expert demonstrations, N_{expert} trajectories starting from a random state and ending when the (first) *wait* action is selected. As random demonstrations, the algorithm is given N_{random} one-step trajectories starting from random states. The optimal policy $\hat{\pi}$ corresponding to the learned reward $\hat{R}$ is then computed. As proposed by Klein *et al.* [2013], the expert dataset is added to the random one to ensure that it contains important (state, action, next-state) triplets such as (goal-state, wait, goal-state). Each experiment is repeated 100 times and results are averaged.

To sample the random dataset we use the following distribution, P_{state} : we first draw uniformly from the different relational spatial configurations and then, for each one, uniformly from the possible groundings.

[1] It should be noted that we did not use their implementation, so there are differences. In particular we do not learn a list of trees for each relational action but one list of trees for all relational actions.

The main parameters are set as follows: 10 trees of maximum depth 4 are learned by TBRIL during the SBC step ① and the reward is learned with a tree of depth 4, which acts as a regularization parameter.

Performance measure

To evaluate the proposed solution we define a performance measure, the Mean Value Ratio (MVR), that measures the ratio between the expected cumulative discounted reward obtain following the learned policy (optimal policy derived from the learned reward) and following the expert one.

$$\mathrm{MVR}(\hat{R}) = \frac{1}{1000} \sum_{i=0}^{1000} \frac{Q^{\hat{\pi}}_{R^*}(s_i, \hat{\pi}(s_i))}{Q^{\pi^*}_{R^*}(s_i, \pi^*(s_i))}, s_i \sim P_{state}$$

Comparison to TBRIL

TBRIL and RCSI have very different goals, different assumptions, and thus should not be compared directly. However, as we propose the first algorithm for IRL in relational domains, we have no baseline to compare to. TBRIL is an algorithm that has been developed to do IL in relational domains and so it can inform us on what to expect from imitation in relational domains and act as a baseline. Latter we will show the advantages of estimating the reward using IRL.

3.3.2 Sensibility to dataset sizes

Figure 3.2 shows the results of using RCSI to learn the *stack* and *unstack* reward of the blocks world domain. RCSI is able of learning the reward with enough expert and random training points. This graph also shows that the RS step ② always increases the performance of the algorithm.

The setting $N_{random} = 300$ and $N_{expert} = 15$ gives good results and so we will use it in the following experiments.

3.3.3 Transfer performance

The main claim of relational learning is the ability of transferring among domain sizes. Fig. 3.3 shows the performance while varying the number of blocks between training and testing. For the *stack* reward, the graphs show almost no loss of performances due to a changing number of blocks. On the other hand, for the *unstack* reward, results are clearly worse when using 4, 5 or 6 blocks for training. By looking at the learned rewards, we have observed that, in most cases, one of the two following rewards is learned : one where the value is high when the number of *clear* is 4, 5, or 6 (depending on the number of blocks in the training set) and a second where high rewards are given when the pattern

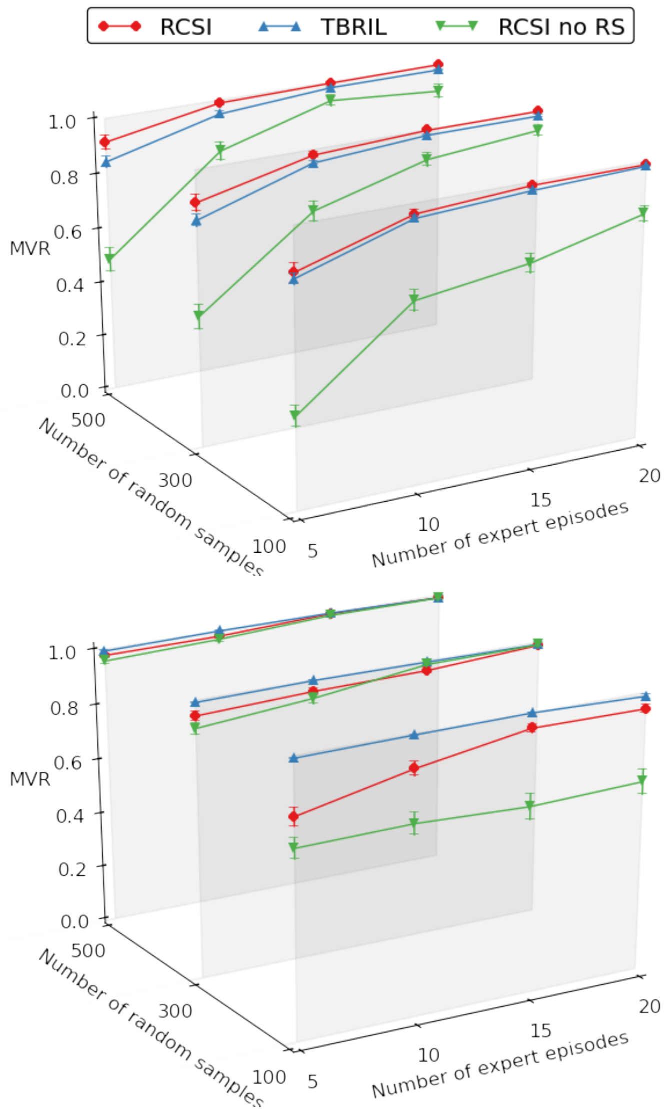


Figure 3.2: Performance for different amounts of training data on the *stack* (Top) and *unstack* (Bottom) task. Error bars represent standard errors.

$on(X, Y) \wedge block(Y)$ cannot be matched. Both solutions are correct for a given domain size, as long as the number of blocks is the same. However, if we change the number of blocks, an ambiguity appears and only the second one stays correct. Yet, there is no reason to prefer one over the other. Moreover, for both possible rewards the expert demonstrations would be similar so the system has to choose between them based on hidden hypotheses.

One way to counter this phenomenon is to use a varying number of blocks during learning. These results are shown in the last column of Fig. 3.3 where a reward learned with a dataset mixing demonstrations with 4 and 5 blocks successfully transfers over to 6-block problems. One can also observe that learning the reward does not perform better than directly learning the policy. This results would be surprising in a propositional domain and shows how relational representations allow to easily transfer among tasks. The advantage of IRL is shown in the following experiments.

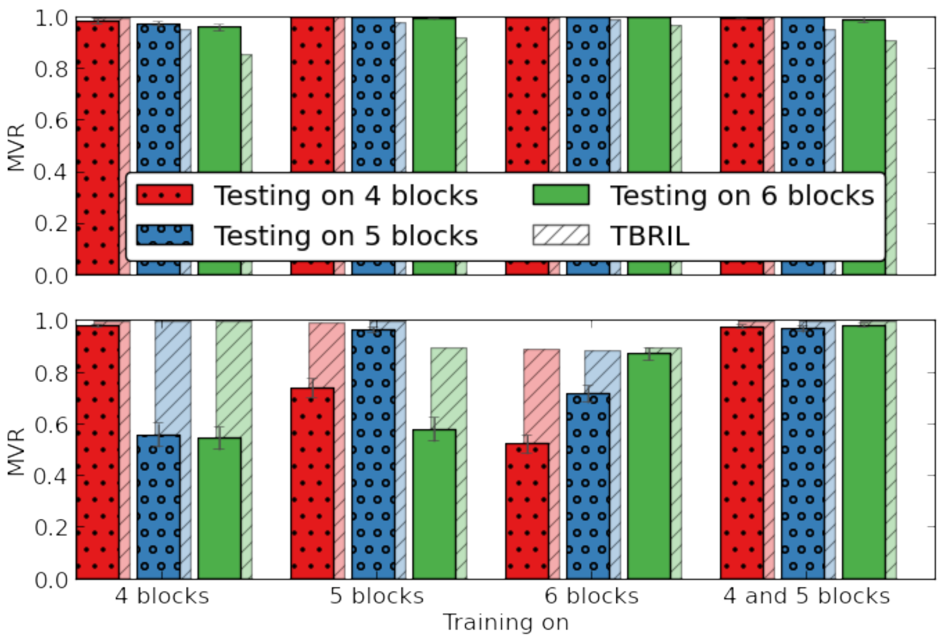


Figure 3.3: Performance of RCSI (and TBRIL) when different number of blocks are used for training and testing the *stack* (Top) and *unstack* (Bottom) task.

3.3.4 Online learning

One interesting feature of IRL over IL is to endow the system with online learning abilities. To showcase this feature, the optimal policy of the learned reward is no more computed exactly (with value iteration) but learned online by interacting with the system using the Q-Learning algorithm [Watkins, 1989]. We define an epoch as 1000 interactions. The RCSI algorithm first learns the corresponding quality function, so we can use the results of the SBC step ① to bootstrap the Q-Learning. Results are shown in Fig. 3.4. There are more efficient RL algorithms in the literature that we could have used. However, this is the proof of concept, not the better way to do online learning from the learned reward.

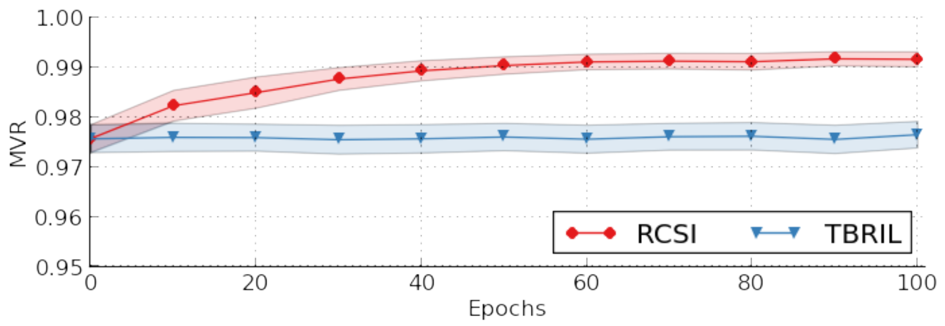


Figure 3.4: Performance of RCSI (and TBRIL) when the optimal policy is learned online.

3.3.5 Dynamics change

Learning the reward rather than directly the policy of the expert leads to a more robust behavior in particular when large modifications of the dynamics of the environment occur. To evaluate this ability, the dynamics of the blocks world is modified after 50 epochs. One of the blocks is made unmovable so in order to build a tower one have to stack them on top of this fixed block. The results are shown in Fig. 3.5. As expected, in this setup, learning the reward allows recovering a satisfying policy even after changes in the dynamics. TBRIL output, on the other hand, cannot learn from interaction and performance stays low.

Moreover, we display results of the algorithm when setting the maximum depth of the reward tree to be 2 in order to increase the regularization factor. It is done as a naive way to obtain a more general representation of the reward. It leads to better performance when the dynamics change; the learned reward transfers better to the new setting.

3.3.6 Big scale

It is important for an IRL algorithm to scale with the number of states. Most tasks of object manipulation are highly combinatorial, and the blocks world domain is no exception: for 5 blocks there are 501 states whereas for 15 blocks the number of states increases to 6.6×10^{13}. We evaluate RCSI in a 15 blocks

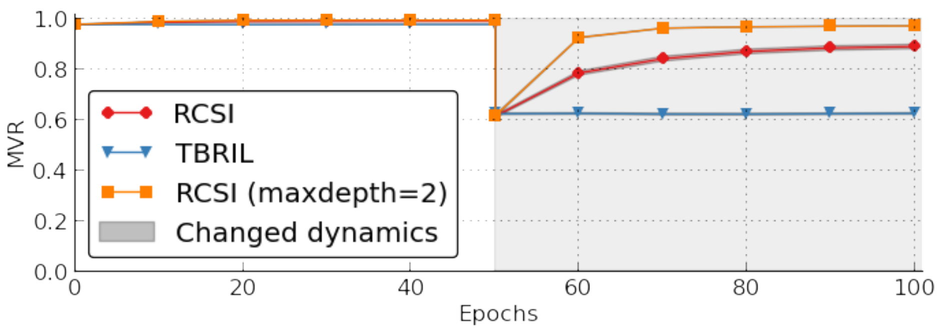


Figure 3.5: Performance of RCSI (and TBRIL) when dynamics is changed.

world (using $N_{random} = 1000$). We use an alternative performance measure that is computationally more efficient, the Optimal Actions Count (OAC):

$$\text{OAC}(\hat{R}) = \frac{1}{1000} \sum_{i=0}^{1000} \mathbf{1}_{\forall \hat{a} \in \text{argmax}_a Q_{\hat{R}}^{\pi_E}(s_i, a), \hat{a} \in \pi_E(s_i)},$$

which is the percentage of states for which all the optimal actions for the learned reward are optimal actions for the expert.

The results are shown in Fig. 3.6. On a 15 blocks world learning a quality function for the *stack* reward requires representing a very complex function, this explains that the performance for TBRIL is around 0.8; as a consequence RCSI performance is no more than 0.85. When the performance is not good enough, setting the maximum depth to 2 is too naive and prunes important features of the reward leading to low performance. For the reward *unstack*, optimal quality functions are trivial so TBRIL performs perfectly and RCSI performance is around 0.85.

However, the performance can be improved by learning in a small domain and rely on the transfer ability of relational representations to scale to 15 blocks world. As shown in Fig. 3.6, the reward learned from a mixed 4 and 5 blocks world, scales very well to a 15 block world. If maximum depth is set to 2 OAC is more than 0.98 for *stack* and *unstack* rewards.

Even if our goal is to learn a reward, it is also important to be able to find the corresponding optimal policy. We can do this using the prost[2] planner as described by Keller and Helmert [2013], we search a plan in a 15 block world for the rewards learned in a mixed 4 and 5 blocks world with RCSI (maximum depth set to 2). We consider a plan to be successful if a goal state is reached and stayed in for at least 4 time steps during the first 40 time steps. 10 plans starting from random states are computed to evaluate a learned reward function and results are averaged over 10 reward learned. For both *stack* and *unstack*, the plans are successful in more than 95% of the cases.

[2]http://prost.informatik.uni-freiburg.de

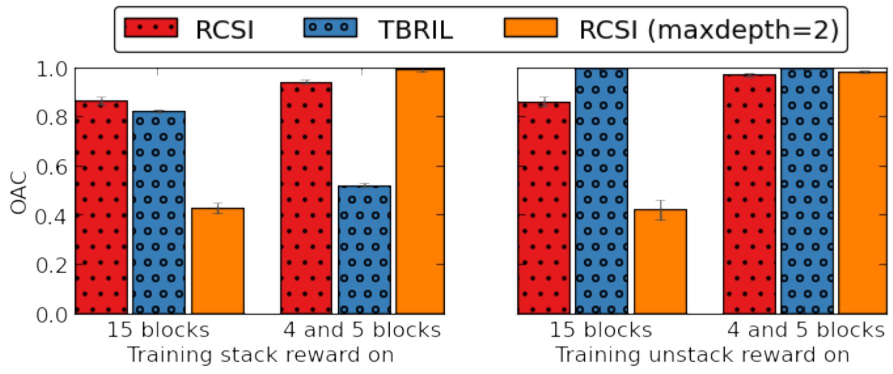


Figure 3.6: OAC of RCSI (and TBRIL) when testing in a 15 blocks world.

We emphasize the strong transfer properties of mixing relational representations and IRL that allowed us to learn a reward in a small world and having such reward valid at an extremely high-dimension problem. The policy can then be found using approximated search methods [Keller and Helmert, 2013; Lang and Toussaint, 2010].

3.3.7 Reward learned

Learning the reward often offers a better interpretation of the behavior of the expert. Due to its compactness it is possible to visualize the reasons behind the behavior of the expert. In Fig. 3.7 we display the reward learned on a *stack* task as a tree using the prolog language. We can see that the best thing to do is to get to a state where all the blocks are stacked (there is only one *clear* predicate true) and wait. In any case, it it is better not to put blocks on the floor and especially when all the blocks are stacked.

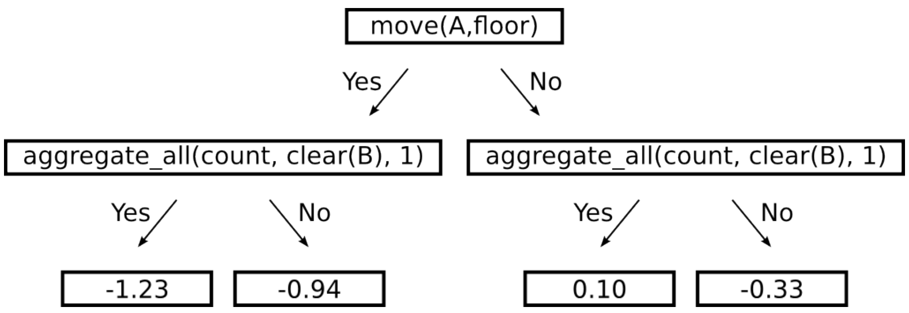

move(A,floor)
Yes
No
aggregate_all(count, clear(B), 1)
aggregate_all(count, clear(B), 1)
Yes
No
Yes
No
-1.23
-0.94
0.10
-0.33

Figure 3.7: Reward learned with RCSI on 5 blocks world.

3.4 Conclusions

In this chapter, we have presented the first approach to IRL for relational domains. We have shown how the IRL algorithm CSI can be generalized to the relational domain. The results indicate that it is possible to learn a relational reward that explains the expert behavior. From it, a policy that matches the expert behavior can be computed. Besides generalizing the classification and regression steps in CSI, we have introduced a reward shaping step so as to reduce the regression error. Finally, we have proposed a new trans-dimensional perspective on data collection where we increase robustness to transfer over domain size by including in the training set demonstrations with different number of objects.

IRL has the advantage of more compact explanations of behaviors and increased robustness to changes in the environment dynamics. The use of relational representations allows learning policies and rewards for changing number of objects in a given domain. This shows one great strength of relational representations, and such results would not be possible in propositional or factored domains even with special feature design. Moreover, when the dynamics changes, we can see the interest of inferring the reward that allows the system to re-evaluate the expected behavior in the new conditions.

In the following chapter, we will apply this algorithm in the context of concurrent domains, where multiple actions can be running at the same time.

Chapter 4

Relational Activity Process

This chapter describes the Relational Activity Process (RAP) framework. It has been published previously as a conference paper under the name "Relational Activity Processes for Modeling Concurrent Cooperation" in Icra conference [Toussaint *et al.*, 2016]. Co-authors are Marc Toussaint, Yoan Mollard, Li Yang Wu and Manuel Lopes. The RAP framework and Monte Carlo (MC) planning code have been mainly developed by Marc Toussaint while I adapted the Learning from Demonstration (LfD) algorithms, TBRIL presented in section 2.3.3 and RCSI presented in Chapter 3, and ran the experiments. It has been edited to improve coherence throughout the thesis.

This chapter introduces the RAP framework that we will use in the following chapters to models teams composed of a human operator and a robot assistant. We propose the use of two policy learning algorithm in this framework: i) TBRIL, an imitation learning algorithm and ii) RCSI, an Inverse Reinforcement Learning Algorithm that was introduced in the previous chapter.

4.1 Introduction

This chapter is primarily motivated by a human-robot collaborative domain where the robot has two manipulators to assist a human in assembling a piece of furniture. In this domain, the two hands of the robot as well as those of the human can be conceived as agents that execute multiple actions in concurrency. Fluently assisting the human requires to plan ahead, ensuring that pieces are fetched in time, and actions are initiated and terminated at the right times. However, beyond this concrete application, concurrency is a very natural aspect of robotic domains in general. Any realistic robot system (say, implemented with ROS) will involve concurrently running activities. One activity might execute a motor primitive controller, another activity might control the camera pose, and also perceptual processes such as tracking an object or actively reducing uncertainty about an object pose can be conceived

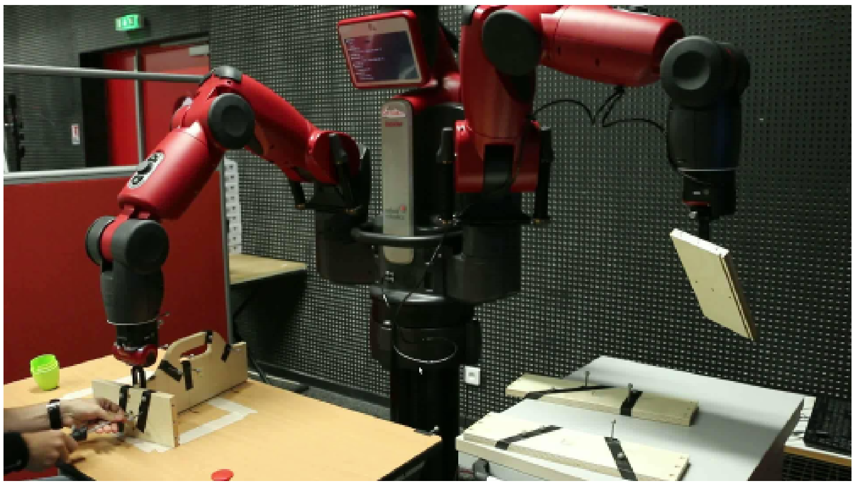


Figure 4.1: A robot concurrently uses its two arms to assist a human by feeding parts and by holding objects while the operator screws them together. See video at https://vimeo.com/139342248).

as a concurrently running activities. While the coordination of such processes could be thought of as a software engineering issue, we think that Reinforcement Learning (RL) and probabilistic planning methods should be scalable to become applicable to such concurrent activity domains in general.

In this chapter we propose a novel formalization of relational concurrent activity processes. While previous formulations such as coarticulation, Concurrent Action Model (CAM) [Rohanimanesh and Mahadevan, 2005] and Concurrent MDPs [Mausam and Weld, 2008] describe policies as choosing multi-actions, we describe a decision process in which different agents may initiate or terminate activities at different times, and which exploits a relational representation of the current activity state. We will review previous formulations in detail in the following section, none of which have addressed relational domains.

In our case of concurrent cooperation, the “objects” we want to generalize over are not only real objects, but also the agents: The two hands of a robot both can potentially execute the same action if the preconditions are met. It is therefore particularly natural to express multi-agent cooperation domains (as well as policies and learned reward functions) in relational terms. Formally, objects as well as agents are equally represented as constants in the underlying first-order logic.

The main contributions of this chapter are, *first*, to propose a novel formalization of relational concurrent activity processes that is well-suited to model concurrent cooperation. To our knowledge, this is the first formulation that considers initiation/termination decisions to describe the Markov process and exploits the relational state representation for this. When constraining the formalism to the propositional setting we show that it is at least as general as the existing CAM [Rohanimanesh and Mahadevan, 2005]. *Second*, to transfer existing MC planning as well as LfD methods [Munzer *et al.*, 2015] to the concurrent relational domain using this formalism. The MC methods not only allow us to estimate optimal decisions, they also provide a reward-weighted expectation over future decisions, for instance allowing the system to anticipate future activities of other agents, especially the human. And *third*, to demonstrate the proposed methods on a real-world human-robot assembly task, where the system uses either the MC planner or a relational policy learned from demonstrations to initiate concurrent activities that assist the human.

In the following section we first briefly sketch the approach of the formalization and explain our distinct use of the words *action*, *activity*, and *decision* throughout the rest of the thesis, to avoid confusion. In Section 4.3 we present our model in detail and discuss the relation to CAM and STRIPS. Due to this formalization, MC planning (Section 4.4) and LfD (Section 4.5) can efficiently be transferred to the concurrent setting. In Section 4.6 we report on the real-world human-robot collaboration experiments.

4.2 Overview

In a standard Markov Decision Process (MDP), the decision variables are called actions, which have no duration (beyond one step) or concurrency. One of the standard generalizations to the concurrent setting, the CAM [Rohanimanesh and Mahadevan, 2005], introduces multi-actions (tuples M of concurrent actions). CAMs are a semi-MDP where the state variable is as before while the decision variable is a multi-action for each phase, phases last for multiple steps, and the termination time of a phase is stochastic.

In our formalism we instead speak of concurrent *activities*, and the set M of currently running activities is part of the relational state. A decision is the initiation or termination of an activity by only one or few agents. Every such decision corresponds to a step of the underlying semi-MDP on the relational state, but only certain decisions lead to a real time duration. While in terms of the semi-MDP, all decisions are sequential, in terms of real time, the initiation and termination of activities may be synchronous or asynchronous and activities arbitrarily concurrent. Policies are single- or multi-agent initiation/termination rules instead of multi-action decision rules.

To avoid confusion, we therefore use the word *action* only when referring to the existing CAM model; in our framework we only speak of *activities* and initiation/termination *decisions*.

4.3 RAPs for modeling concurrent cooperation

A RAP is a way to model concurrent cooperation of multiple agents as a relational semi-MDP [Toussaint *et al.*, 2016]. The formalism allows for multiple actions taken simultaneously and asynchronously.

In order to do so, it decomposes the *action* concept into two different concepts: *decision* and *activity*. Indeed, in the classical MDP formalism action encompass at the same the time the decision to start and the time to perform an action. By splitting it in two, and including the action being performed (called activity) in the Markovian state, RAP can represent a concurrent process. Roughly, RAPs define a sequential, semi-Markovian decision process where decisions are about the initiation of activities, and activities run concurrently with random durations.

For brevity we first describe RAPs in the deterministic case. However, the reader familiar with non-deterministic decision rules [Zettlemoyer *et al.*, 2005] will anticipate how we represent stochasticity by probabilistic effects of all rules. The stochasticity of activity durations is less trivial and needs to be discussed in more detail later.

We will continue to use the blocksworld example from section 2.3. However, to showcase the concurrent aspect, we introduce the presence of two agents and two activities will be used to realize the displacement of a block: pick(Agent, Block) and place(Agent, Block1, Block2). We also add two predicates, hand_free/1 and in_hand/2.

As for relational MDPs, the transition function is represented using rules. Given a set $\mathcal{A}$ of activity constants, for each activity $a \in \mathcal{A}$ there exist one or multiple *initiation operators*, encoded as rules of a Relational MDP. These initiations operators have a real value predicate $\mathsf{go}(a) = \tau_a$ in the outcome set that represent the running activity. For instance:

```
start(pick, X, A):
  { hand_free(X) clear(A) }
  { go(pick, X Y)=1 ¬hand_free(X) ¬clear(A) }

start(put, X, A, B):
  { in_hand(X, A) block(B) clear(B) }
  { go(put, X, A)=0.7 ¬clear(B) ¬in_hand(X, A) }
start(put, X, A, B):
  { in_hand(X, A) ¬block(B) }
  { go(put, X, A)=0.7 ¬in_hand(X, A) }
```

are the initiation operators of activities pick and put. In state s, the decision set $\mathcal{D}(s)$ includes all initiation operators which context can be grounded (potentially more than once). In addition, the decision set includes a single special decision, the `wait` decision.

Transition model: The transition model for activities is equivalent as for

Relational MDPs except for the addition of a knowledge base `KB` that includes a set of first-order rules. When the state s_t is modified by a rule it produces an intermediate state s'_t. The new state, s_{t+1}, is obtained by the stable model under this `KB` (cf. answer set programming), the result of forward chaining all rules on s'_t until convergence.

When the decision is **`wait`**, the semantics is that all agents decide not to initiate anything further and that real time progresses until the relational state changes and activities might terminate. We concretely define the state transition for a **`wait`** as the following procedure:

1. Find the `go`-predicate with the minimal time-to-go value $\tau_{\min}$.
2. Decrement all `go`-predicate-values by $\tau_{\min}$.
3. All zero-valued `go`-literals, `go`$(a) = 0$, are deleted from s_t and a corresponding **`terminate`**(a) is added to s_t.

This defines the intermediate relational state s'_t. Again, the new state s_{t+1} is defined as the stable model under the `KB`. The `KB` is assumed to include the rules that express the effects of termination.

For the blocks world example, the termination rules are:

```
r1(X, A, B):
  { terminate(pick, X, A) on(A, B) }
  { ¬terminate(pick, X, A) in_hand(X, A) ¬on(A, B) clear(B) }
r2(X, A, B):
  { terminate(pick, X, A) ¬on(A, B) }
  { ¬terminate(pick, X, A) in_hand(X, A) }

r3(X, A, B):
  { terminate(put, X, A, B) block(B) }
  { ¬terminate(put, X, A, B) hand_free(A) on(A, B) }
r4(X, A, floor):
  { terminate(put, X, A, B) }
  { ¬terminate(put, X, A, B) hand_free(A) on(A, table) }
```

Duration model: In the context of hierarchical RL and the standard CAM [Rohanimanesh and Mahadevan, 2005], steps of the sMDP correspond to the execution of an option, and the duration of the sMDP step is integer-valued, counting the steps of the underlying MDP. However, in general sMDPs the duration of one Markov step is real-valued, arbitrarily depending on (s, d, s'). In the concrete case of RAPs, we assume that initiation and termination decisions themselves have zero duration, while τ is equal to $\tau_{\min}$ for the wait decision and therefore implicitly given by the `go` predicates in initiation operators.

Reward model: Rewards in RAPs are generally given as a relational mapping $(s, d, \tau, s') \mapsto r$. In our applications, we encode this mapping as a relational tree as it is a compact and easily readable way to represent such mappings.

Optimality: Unrolling a policy generates an episode $(s_0, d_0, \tau_0, r_0, s_1, ..)$. We define the discounted return for an episode as

$$R = \sum_{i=0}^{\infty} \gamma^{\bar{\tau}_i} \beta(\tau_i) r_i \ , \quad \bar{\tau}_i = \sum_{j=1}^{i} \tau_j \ , \beta(\tau) = \frac{1-\gamma^{\tau}}{1-\gamma} \tag{4.1}$$

The β term weight each reward by the time taken by the transition while taking into account the impact of the discount factor.

4.3.1 Stochasticity

State transition stochasticity in RAPs is represented by probabilistic effects of initiation operators as well as `KB` rules, exactly as done in NDRs [Zettlemoyer *et al.*, 2005]. We propose to generally define duration stochasticity via $P(\tau_{a,\bar{x}} \mid s, a, \bar{x})$, that is, the probability over the time-to-go of an activity $(a, \bar{x})$ depending on the current state. Let $M(s) = \{(a, \bar{x}) \mid s \models \texttt{go}(a, \bar{x})\}$ be the set of current activities in state s (that is, the multi-action in the conventional formalisms). Then we define the effect of the `wait` decision by the stochastic procedure:

1. Sample a $\tau_{a,\bar{x}} \sim P(\tau_{a,\bar{x}} \mid s, a, \bar{x})$ for each $(a, \bar{x}) \in M$
2. Select the minimal τ_{min} of these.
3. For all $\tau_{a,\bar{x}} = \tau_{\text{min}}$, delete the $\texttt{go}(a, \bar{x})$ literal and add the $\texttt{terminate}(a, \bar{x})$ literal to $\hat{s}$
4. For all $\tau_{a,\bar{x}} > \tau_{\text{min}}$, modify the value of the $\texttt{go}(a, \bar{x})$ literal such that $\mathbb{E}\{\tau_{a,\bar{x}} \mid s, a, \bar{x}\}$ reduces by τ_{min}.

In practice, we use Gaussians with the mean defined by the $\texttt{go}(a, \bar{x})$-value, which makes the last step simple to realize.

4.3.2 Generalization and Comparison to STRIPS

The above formulation differs from standard STRIPS (or its stochastic version, NDRs [Zettlemoyer *et al.*, 2005]) essentially in the `wait` operator and respective treatment of the `go`-predicate, as well as that the new state is the stable model under an additional knowledge base `KB`. We introduced the latter for representational convenience, allowing for a significantly more flexible declaration of environments. The `wait` operator, however, seems essential for the description of concurrent processes as it defines the relation between Markov steps and real time. This is out of the scope of what could be represented in plain STRIPS frameworks.

The special semantics of the initiation and termination operators, and that they necessarily need to set/delete a `go`-predicate, can be relaxed. An alternative formulation is the following: The rules in the `KB` are labeled as either "decision rules" or "auto rules"; decision rules are applied when the respective decision is made, auto rules are forward-chained as above to find the next stable model. The decision set is then the set of grounded decision rules s.t. s models $\texttt{pre}_r$ plus the `wait` decision. What we introduced as initiation and termination operators become special case decision rules. However, the explicit definition of initiation and termination operators clarifies the semantics in concurrent activity processes.

4.3.3 Comparison to CAM

The above formulation is a reduction of concurrent action planning in relational domains to sequential decision making in a relational sMDP. A plan (or unrolling of a reactive policy) will give a sequence of decisions, each referring to a different agent (or set of agents), that can be interpreted by a robot either as own decision or as anticipation of the other agent's decision. This is in contrast to multi-action policies, where it remains somewhat unclear how a single agent should actually react (do his own, single-agent decision) given that the another agent is observed to initiate his own activity.

We want to compare RAP in more detail to CAM as presented in Rohanimanesh and Mahadevan [2002]. This comparison becomes most explicit by reducing a CAM model to a RAP model:

Proposition 1. Every CAM process can be represented as a RAP; an optimal policy of this RAP can be translated back to an optimal CAM-policy.

We sketch a prove of this proposition by construction, making the reduction explicit. To this end we limit our RAP model to the case of a propositional state s. We first consider the **decision set**. In CAM, the decision space is the set of multi-actions $M \in \mathcal{M}(s) \subseteq \mathcal{A}^*$ in a subset $\mathcal{M}(s)$ of the power set of the action set $\mathcal{A}$. The subset $\mathcal{M}(s)$ depends on the state and is defined via mutex conditions, expressing that certain actions cannot be chosen concurrently (potentially depending on s). In RAP, let us define a decision episode as a sub-sequence of decisions $D = \langle d_1, .., d_m, \texttt{wait}\rangle$ where all decisions d_i are initiation or termination decisions, except for the last, which is a wait. We need to show that the preconditions of initiation operators can be chosen such that every feasible multi-action decision M can be reproduced by a decision episode D, and that feasible D exist that create an infeasible M. In other words, can the preconditions express constraints that are equivalent to the mutex conditions. If the preconditions are general propositional logic expressions this is clearly the case. But in the concrete case of mutex conditions that are literally mutexes of pairs or tuples of actions, this can very naturally be encoded as negative literals in initiation preconditions. Therefore we can construct an equivalence between $\mathcal{M}(s)$ in CAM and the set $\mathcal{D}(s)$ of feasible decision episodes in RAP. If the constructed initiation precondition allow for all possible permutations of initiation decisions, the $\mathcal{D}$ is larger than $\mathcal{M}$. This could be excluded by construction. However, even then a specific policy in RAP can always generate decision episodes equivalent to any multi-action decision.

In CAM, the **transition and duration model** is jointly and very generally given in terms of $P(\tau, s' \mid M, s)$ for $\tau \in \mathbb{N}$. We can reproduce this in RAP by creating rules in the `KB` that reproduce this transition whenever the last `wait` decision of a decision episode generates a `terminate` predicate; as the state $\bar{s}$ in RAP includes all information of (s, M) in CAM, any (probabilistic) mapping from (s, M) to s' can be realized by the `KB`. An interesting aspect of the CAM model are the three alternative termination schemes *any*, *all*, and *continue* (see Rohanimanesh and Mahadevan [2002] for details). The *any* scheme can be reproduced in RAP when the `KB` "deletes" all `go` literals (empties M) on a `wait`; the *continue* is reproduced simply by not deleting all `go` literals; and the *all* by introducing a `blocked` predicate that renders all initiations infeasible, becomes true after `wait` if there are still activities, and false if there are none left. Note that all three schemes are reproducible only by initiation decisions; allowing also for termination decisions generalizes these schemes.

In terms of the reward and notion of optimality, CAM and RAP do not differ. In conclusion, every CAM can be expressed as a RAP. Assuming that a RAP planner computes and optimal RAP-policy, it is straight-forward to construct a CAM-policy that chooses the multi-action M to be the activity state $M(s)$ after an decision episode D of the RAP-policy.

4.4 MC Planning in RAPs

A standard approach to planning in single-agent non-concurrent relational domains is UCT (UCB1 applied to Monte-Carlo Tree Search) [Kocsis and Szepesvári, 2006]. As our formulation sequentializes the decision process we can readily apply UCT or other MCTS variants also for planning in concurrent relational domains. For the purpose of the experiments in this chapter we utilized the simplest option, namely plain MC estimates of the Q-function over the decision set $\mathcal{D}(s)$ in every step. We empirically found in our specific domains that plain MC behaves more robust than other MCTS variants (UCT using MC backups and plain UCB1, UCT using Bellman backups and UCB1 [Keller and Helmert, 2013]) in terms of not getting stuck in sub-optimal tree branches. We believe this to be a rather special effect of the domains we consider, where "success" is rare. Further, plain MC has the advantage of allowing us to also compute an unbiased reward-weighted distribution over future states and decisions, for instance allowing one agent to anticipate the future decisions of another.

For generating finite rollouts we need to decide on a termination condition. In many domains, including our example domains, there is a natural termination condition reflecting success or failure. In addition, we always terminate a rollout in a *dead-end* state which we define as a state without `go`-predicates (no current activities) and $\mathcal{D}(s) = \{\texttt{wait}\}$. If no natural termination conditions are given, one typically constrains rollouts to a maximal horizon H. If rewards are bounded (as they need to be for UCB1) and $\gamma < 1$, one can choose H to ensure a small upper bound on return that could be collected beyond H.

Finally, when comparing to CAM we mentioned that the outcome of decision episodes may be invariant under permutation of initiation or termination decisions. This seems to introduce large redundancy in the decision tree in comparison to a tree spanned by CAM-multi-actions. While in plain MC this redundancy has no effect, in general MCTS multiple nodes are created for the same state (the effect of a decision episode) which compromises the efficient collection of statistics for this node. This can be corrected for by modifying MCTS to become somewhat like graph search: after a `wait` decision we uniquely sort the decision episode and hash if the same has been sampled before.

4.5 LfD in RAPs

As for planning, the RAP formulation allows us to transfer existing learning methods to the relational cooperation scenario. We consider both, Imitation Learning (IL) and Inverse Reinforcement Learning (IRL).

4.5.1 IL

For IL we consider a data set $D = \{(s_i, d_i)\}_{i=0}^{N}$ of state-decision pairs from expert demonstrations. Recall that in our case these are multi-agent concurrent activity demonstrations where d_i encodes which agent is (or agents are) initiating or terminating which activity. From this data we learn a policy $\pi : s \mapsto d$ that predicts expert decisions for a novel state.

We propose to use TBRIL [Natarajan *et al.*, 2012] within the RAP framework to learn a relational cooperation policy. The TBRIL algorithm represents the policy as a relational regression tree and uses gradient tree boosting [Blockeel and De Raedt, 1998; Friedman, 2001] to train the model. The training objective is the likelihood under the probabilistic policy model

$$\pi(d|s) = \frac{e^{\beta\psi(s,d)}}{\sum_{d' \in \mathcal{D}(s)} e^{\beta\psi(s,d')}} ,$$

where $\psi(s, d)$ are the features implicitly defined as the leaves of the regression trees.

4.5.2 IRL

In IRL we are given expert demonstrations and assume a generative model that considers these demonstrations as optimal w.r.t. some unknown reward function. The goal is to uncover this reward function; in our case a relational regression tree that represents such a reward function. Note that without further constraints this is an ill-posed problem as many solutions exist, for example, the reward function equal to zero for any input will always be a solution to the problem.

We propose to use RCSI [Munzer *et al.*, 2015] within the RAP framework to uncover a relational reward function from expert demonstrations of cooperation. This algorithm decomposes the problem in two steps: First, find a Q-function $Q(s, d)$ such that expert decisions are maxima of the Q-function (i.e. a discriminative function describing the expert policy). This first step is equivalent to IL and we use the TBRIL algorithm as above. Second, compute a reward consistent with the Q-function by inverting the bellman equation. There is a one to one correspondence between the Q-function and the reward function.

Given the learned reward function we can use planning methods to compute optimal decisions in novel states, with the potential to generalize much better than IL.

4.6 Experiments

We will use two domains to showcase the capability of the proposed model.

4.6.1 Example domain: Concurrent assembly assistance robot

In this domain a robot has to assist a human in an assembly task. We have two agents, the two end-effectors of the robot, and a human, which we model using the `KB` as part of the environment. The aim is that the robot fluently assists the human in assembling a box composed of 5 parts.

In order to assemble the box the different pieces have to be 1) put in the human workspace 2) positioned and 3) attached. To put a piece in the human workspace the robot can initiate two activities, *pick(hand, piece)* and *give(hand, piece)*. A third activity, *wait_for_human*, waits for the human to position the next piece. The activity *hold(hand, piece, identifier)* will hold a piece at a specific point allowing the human to screw them together. Two

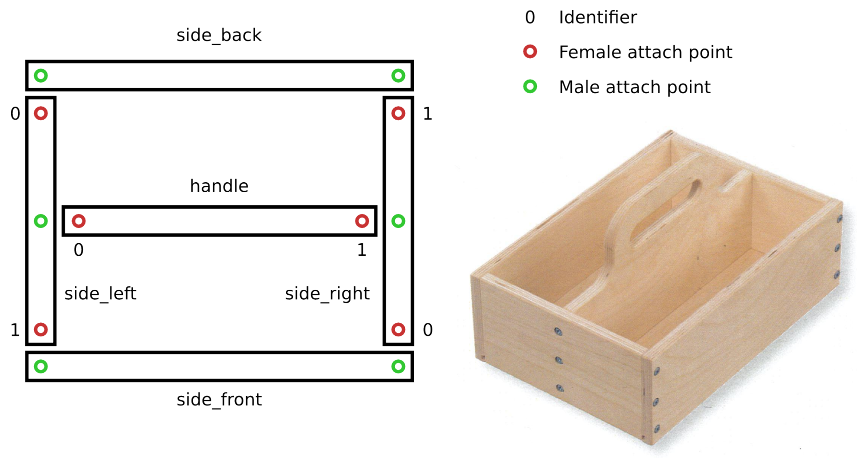


Figure 4.2: Schematics representation and photo of the toolbox.

additional activities, *go_home_left* and *go_home_right*, put the robot's arms back in homing position. All activities last one unit of time except for hold which lasts two. Each arm can only be involved in one activity at any time.

Fig. 4.2 illustrates the five pieces *handle*, *side_right*, *side_left*, *side_front* and *side_back*, and their attach points. Since we do not consider impossible builds, positioning or attaching activities need 3 arguments to avoid ambiguity (and are used in this order by convention): the object with the female attach point, the object with the male attach point and the identifier of the female attach point used.

The state of the domain is represented with the following predicates: *attached/3*, *positioned/3*, *in_human_ws/1*, *picked/1*, *at_home/1*, *busy/1*, *free/1*, *occupied_slot/2*, *human_can_do*.

The starting state is always the state where nothing is on the human workspace. This domain is challenging from the planning point of view because a high number (41) of decisions are necessary before reaching the goal state. For learning, on the other hand, the fact that the start state is unique makes it easier.

4.6.2 Example domain: Concurrent blocksworld

This domain is an adaptation of the standard blocksworld domain where two activities can be realized at the same time. Five blocks can be put on top of each other or on a surface (called floor) by two robotic arms.

The domain is represented with the following predicates: *on/2*, *clear/1*, *busy/1* and *in_hand/2*. The activities are *pick(arm, block)* and *put(arm, block, block)*, both of them last one unit of time and both of them can be realized by either arm.

The goal of the task is to stack all blocks in one tower. We generate a random starting state by first sampling the number of initial towers and then uniformly selecting one state that respects the total number of blocks. In the start state no activities are active.

This domain presents different difficulties than the assistance robot one. The task is shorter in terms of the number of decisions, which simplifies the planning problem. On the other hand, there are activities with a negative impact that put the agent further from the goal, so random walk is not an effective strategy. Another challenge in this domain, more for the learning method, is that there are many different paths to reach the goal depending on the start state. Thus it is important to generalize well from the demonstrations as the policy will often reach states not observed in demonstrations.

4.6.3 Simulation Results

Before showcasing the RAP model in a real-world robot assistance domain we evaluate the transferred planning, IL and IRL methods in simulation.

MC Planning For the assistance robot domain, results are presented in Fig. 4.3. We compare the performance of the planner in terms of real task execution time and success rate, to the optimal policy and to a random policy. There are 50 trials for every parameter value and results are averaged. After 1000 decisions, if the goal state is not reached we stop the trial and consider it to be a failure. The success rate is computed as the percentage of successes. Two rewards function are used for planner. The first one only rewards the final state while the second one tries to guide the exploration by giving intermediate reward depending on the number of *attached/3* predicates. With few rollouts, the MC planner is noisier than the optimal policy, resulting in longer task execution times. With 200 rollouts and the guided reward, the planner finds a policy to reach the goal in 24 robot time units against 21 for the optimal policy.

As stated earlier, the planning problem for the blocksworld domain is easier. Fig. 4.4 shows that with 20 rollouts the performance is almost optimal.

IL For the assistance robot domain we show in Table 4.1 the efficiency of policy learning from demonstration. With only few demonstrations the policy learned achieves near optimal behavior.

For the blocksworld, results are presented in Fig. 4.5. This domain is more challenging in terms of LfD because of the number of possible start states. However, the TBRIL learns the correct behavior with 20 demonstrations.

IRL The results for IRL are presented in Table 4.1 and Fig. 4.5 and are close to the ones of policy learning. This is to be expected as we use TBRIL for

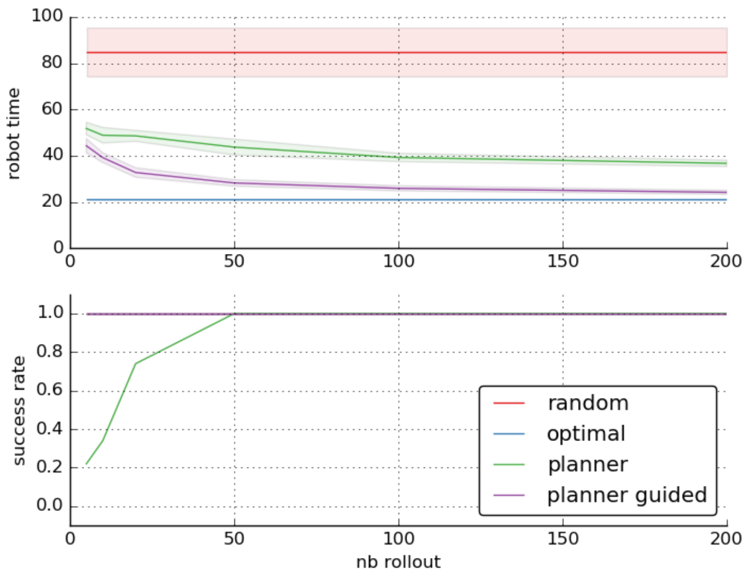


Figure 4.3: Performance of the planner on the assistance robot domain. Shaded areas represent standard error.

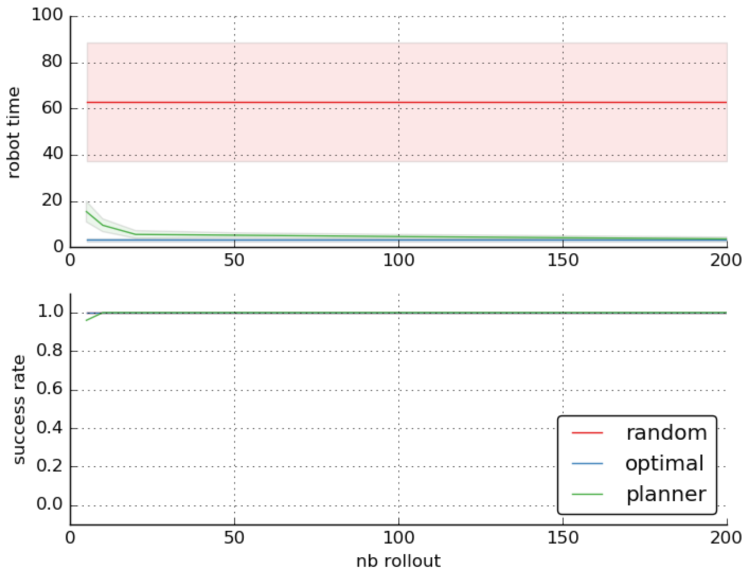


Figure 4.4: Performance of the planner on the blocksworld domain. Shaded areas represent standard error.

Table 4.1: Performance of IL algorithms on the robot assistance domain.

	robot time	success rate
optimal policy	21.0	1.0
random policy	84.96	1.0
policy learned (1 demos)	21.3	1.0
policy learned (2 demos)	22.25	0.96
reward learned (1 demos)	22.0	0.8
reward learned (2 demos)	21.6	1.0

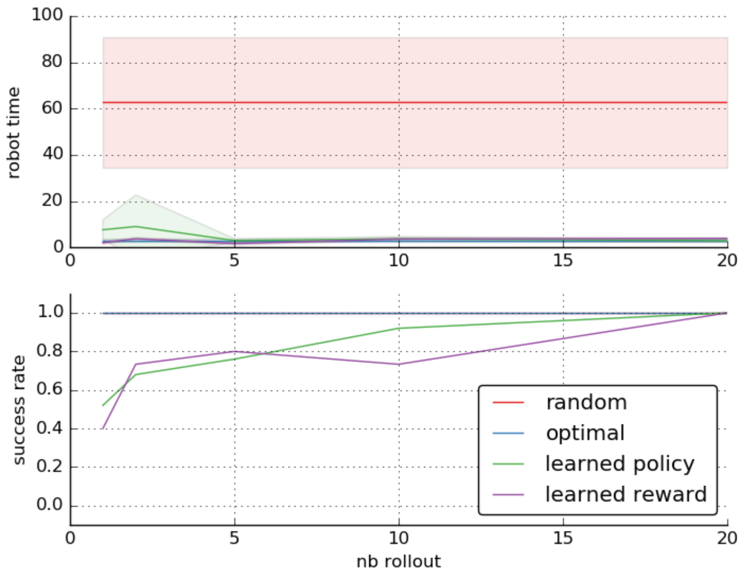


Figure 4.5: Performance of the learning algorithms on the blocksworld domain. Shaded area represent standard error.

the first step of RCSI and use the full knowledge of the dynamic model of the world.

4.6.4 Robot application

We realized the robot assembly domain on a Baxter robot. To this end we additionally had to implement 1) a sensing module that allows us to detect the truth values of the different predicates and 2) an activity module with a routine for each activity.

Note that now, in real-world, the `wait` decision really waits until the first sensing or activity module reports a termination of an activity or change of state. This replaces the model assumption made in the definition of the RAP model, namely the effect of a `wait` decision in simulation.

For each predicate, a detector has been coded, e.g.:

- *positioned/3*: The 6D pose of the objects are retrieved with an Optitrack system and compared to the ground truth which is provided to the system beforehand.
- *attached/3*: This is true if it was true in the previous state or if objects are positioned and the human is operating a screwdriver (tracked with Optitrack) nearby for 7 seconds.
- *in_human_ws/1*: We check if the object is in a cube around the human.

MC Planning Planning on the real robot is achieved with a simple loop that retrieves the current relational state of the scene and re-plans to find the next decision using the MC planner. The previous plan is not reused, the planner starts from scratch each time.

The results are presented in the video supplement at https://vimeo.com/139342248. The decisions are computed online and result in a complete assembly of the box. Twice the robot holds at a place not needed, otherwise the decisions are optimal. We assume that allowing the planner more rollouts would solve this problem. We used the guided reward in this setup.

IL In order to learn the policy, we first recorded some expert demonstrations. A specific command line interface has been developed that allows to command the robot to execute any activity. When a command is given, and before it is executed, the state of the scene and the decision are logged and used later to learn the policy. Once the demonstrations are recorded, the policy is learned. It is then used to control the robot with a simple loop similar to the planning case.

We have successfully learned and played a policy from two demonstrations on the Baxter robot. This learned policy is presented in the video https://vimeo.com/139342248.

4.7 Conclusions

This chapter proposes a model of concurrent cooperation that allows for efficient transfer of existing planning and RL methods to such domains. While in other formalisms policies map to multi-actions for all agents, RAP describes a process of sequential initiation, termination and wait decisions that each involve only one or few agents and exploits the underlying relational state representation. The knowledge base and generality of activity duration distributions offer great representational flexibility. We compared the generality of

RAP to previous concurrent action models. Using RAP we transferred MC planning, IL and IRL to relational concurrent cooperation domains, which previously has not been demonstrated. We also illustrated the approach on a real-world robot assistance scenario, where the robot concurrently uses both end-effectors to assist a human in an assembly task.

The RAP model will be used in the next chapter. We will present how it can be used to model a team as well as an interactive learning architecture that will allow the robot to efficiently learn tasks.

Chapter 5

Cooperative Behavior Learning in RAPs

This chapter present a framework for learning tasks in a cooperative setting. It has been submitted as a journal paper. Co-authors are Marc Toussaint and Manuel Lopes. It has been edited to improve coherence throughout the thesis. Part of this work has also been published as a conference paper under the name "Preference Learning on the Execution of Collaborative Human-Robot Tasks" in Icra conference [Munzer *et al.*, 2017b]

This chapter describes building on the Relational Activity Process (RAP) framework presented in Chapter 4 to handle cooperation between two agents, a human operator and a robot helper. It also describes an Interactive approach for cooperative Learning from Demonstration (LfD).

5.1 Introduction

Robots are still restricted to very controlled environments and mostly separated from humans. New technological developments have improved the safety of robots, making it possible to have robots and humans sharing the same space. For high volume production, it will still be more efficient to fully automatize the task. For low volume production, on the other hand, having a team composed of a human expert and an easily reconfigurable robot might be more efficient than full automation as the setup cost will be greatly reduced allowing for fast task switching. As a consequence, to be useful, robots should be able to help as soon as possible. In this context, robots should be able to learn how to assist a human operator. This process should be intuitive for the operator, requiring as little as possible knowledge about how the robot learns and acts. It is also important for this process to be time efficient, so that the learning phase is no longer than the actual task execution.

In this chapter we propose an interactive learning system allowing a robot to learn to assist a human operator. We mean by interactive learning, fusing the training (creating a dataset of correct behavior) and execution (using the behavior learned from the dataset to solve the task) phases. It has several advantages: i) it exploits the current execution data to start act autonomously as soon as it is confident on the task, making the teaching process shorter tedious as the robot can act autonomously when it is sure about the correct action to execute; ii) the new experience and feedback from the user are used to fix the learned behavior if some parts are wrong or adapt if the expected behavior changes.

Another advantage of the interactive setting is to make it easier for a naive user to use the system. In many cases a deep understanding of the learning problem and learning algorithms is needed to select parameters such as the size of the training dataset, the amount of feedback, and the length of the training. Indeed, if the size picked is too small the system will make mistakes with no possibility for improvement other than retraining the system with a bigger dataset. If the size is too big, the user will spend a lot of time recording useless demonstrations.

Recent works have considered interactive LfD in robotics, but mostly in the context of single-agent tasks and without concurrent actions and high-level representations. In this chapter, we propose a way to learn a collaborative task. The work presented in this chapter focuses on human-robot collaboration scenarios but could be applied more broadly, to any virtual agent that has to work with a human in a support fashion. Our contributions include the following: i) proposing and evaluating an interactive behavior learning framework for cooperative tasks with concurrent actions ii) the introduction of a potential error estimation mechanism for relational policy learning iii) the introduction of a formalism, based on relational semi-MDPs, to model concurrent decision problem and iv) how to learn preferences with the proposed framework.

We present the RAP framework and how to represent a cooperative task with it in Sec. 5.2. In Sec. 5.3, we present the interactive learning framework and the algorithms used. Lastly, Sec. 5.4 and Sec. 5.5 are the evaluation and the conclusion.

5.2 Team behavior modeling with RAPs

Using the RAP formalism, we can model teams of cooperating agents, where all agents are embedded in the same semi-MDP and the decision space is the *joint* space of human decisions and robot decisions, $\mathcal{D}$. Given a reward and using planning methods, for example, Value Iteration (VI), we can compute an optimal Q-function over the next decision $d \in \mathcal{D}$ in a given RAP state s. It provides values for decisions across agents. If there was a single central decision maker, it could read out the $\text{argmax}_d\, Q(d, s)$ and transmit the decision d to the agent it concerns. However, in real human-robot collaboration, without such a central decision maker, the readout and interpretation of this quality-function are non-trivial. With two decision makers, both might want to start an activity at the same time.

In the human-robot cooperation case, we assume that the joint decision space decomposes as $\mathcal{D} = \mathcal{D}_R \cup \mathcal{D}_H$ and $\mathcal{D}_R \cap \mathcal{D}_H = \{wait\}$, with $\mathcal{D}_R$ the robot's and $\mathcal{D}_H$ the human decision space.

Given the robot has a representation of the shared task as a Q-function, we propose the following procedure for the robot to decide on its own activities. If $\max_{d \in \mathcal{D}_R} Q(d, s) < \max_{d \in \mathcal{D}_H} Q(d, s)$, that is robot decisions have strictly less value than optimal human decisions, the robot does not start an activity and lets the initiative to the human. Otherwise, the robot samples uniformly from the set of optimal robot decisions $\subseteq \mathcal{D}_R$.

5.3 Interactive cooperative behavior learning

We now present the interactive learning framework. The basic idea is to have a system where the user can instruct the robot. But we make it also possible for the robot to act before being instructed when it is certain about what activity to execute. The underlying assumption is that, as the robot does the task again and again, it will become more and more certain about which activities to perform. To be certain about a decision we rely on potential error estimation (detailed in section 5.3.5).

5.3.1 The interactive framework

Fig 5.1. presents the interaction flow. We incrementally build a dataset of expert behavior during task solving and after each episode (completion of the task) relearn the correct behavior using batch learning. During an episode, for each decision, based on the predicted decision and the potential error associated, we distinguish three cases: confident, ask-before-act and waiting-robot. If the predicted activity is a robot one and the estimated error is inferior to a threshold, called *confident threshold*, the robot acts and eventually gets feedback, if no feedback is given before the end of the activity, the activity is considered to be correct. On the other hand, if the estimated error is over the *confident threshold*, the robot starts by asking the user if the predicted decision is correct and use the user feedback to execute a correct activity. If the predicted activity is a human one or the decision is wait, the robot does nothing and gets feedback either by observing a human activity or by getting a command to do an action. After each activity (start human activity, start robot activity or wait) D is updated.

Since our present goal is to produce an efficient learning procedure, we also added a memory system. If the robot recognizes the current state as part of

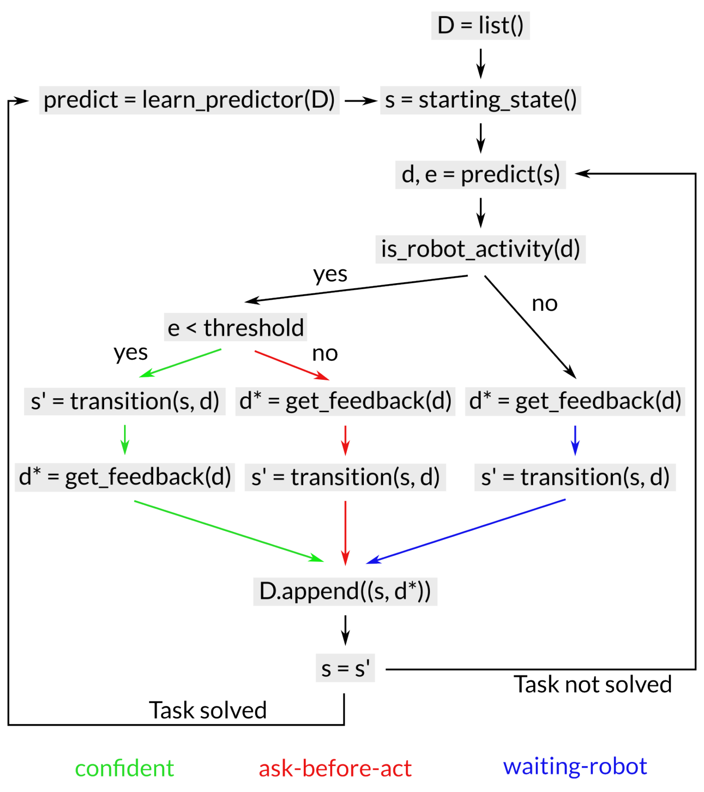


Figure 5.1: Schema of the interaction protocol. The three different cases (confident, ask-before-act and waiting-robot) are shown in color.

D, it predicts the corresponding activity with an estimated error of 0. This also allows avoiding loops during the execution.

5.3.2 Learning cooperative behavior

We learn the behavior based on the dataset gathered during task solving. It is composed of tuples $(state_i, decision_i)$ where $decision_i$ is a semi-MDP decision and can, therefore, be the activation of a human activity, the activation of a robot activity or the wait primitive.

We propose to use the TBRIL algorithm [Natarajan *et al.*, 2011] as it is, to our knowledge, the only policy learning algorithm for relational knowledge. TBRIL works by finding a quality function such that seen actions are optimal for their associated state.

5.3.3 Learning preferences

The previously defined algorithm can also be used to learn the user's preferences when the general task is known beforehand.

Given a general task with different ways to solve it, i.e. different optimal paths in the RAP, preferences are defined as the preferred subset of these paths. The task can be seen as prior knowledge whereas preference is the learned behavior. In the extreme case where no prior knowledge is available, this problem is reduced to task learning.

More formally, under a Markov Decision Process (MDP) the task is defined by the reward, R_{task}. Using Value Iteration, we can compute the optimal quality function Q^*_{task}. We can then represent the behavior of the team taking into account human preferences as another quality function $Q^*_{full} = Q^*_{task} + Q_p$. Where Q_p is a shaping function of the task optimal quality function such that Q^*_{full} maximizes task and human preference.

By initializing the parameter Q^*_{prior} of TBRIL to Q^*_{task} we can use the same algorithm to learn preferences. The advantage of learning the preferences is to, at the same time, leverage prior knowledge to be efficient early on while being able to adapt to different users.

5.3.4 Protocol of interaction

At this stage, we introduce three feedback types:

- Validation: the predicted decision is correct
- Modification: the predicted decision is not correct and feedback is given before decision is executed
- Correction: the predicted decision is not correct and feedback is given after decision is executed

Table 5.1: The different kind and ways to obtain feedback based on the predicted action and whether it is correct. Exp. stands for explicit and imp. stands for implicit. The cases to avoid, explicit (blue) and correction (red) are highlighted

correct decision	A robot decision is predicted		Human decision or
	confident	ask-before-act	wait is predicted
predicted	imp. validation	exp. validation	imp. validation
not predicted (robot)	exp. correction	exp. modification	exp. modification
not predicted (human)	exp. correction	imp. modification	imp. modification

and two ways to get feedback :

- implicit: the system can recover the information by observing the scene
- explicit: the system receives the information by the direct intervention of the human

Based on the kind (robot/human/wait) of decision that is predicted and if it is correct, different feedback will be gathered. Table 5.1 presents these different feedback types. For example, if the predicted decision is correct, a validation feedback is be gathered. If the potential error is under the *confident threshold*, it will be done implicitly as the robot will act in confident mode. On the other hand, if the predicted decision is a robot one, the robot is confident and this prediction is wrong the feedback will be an explicit correction as the only way for the system to get feedback is from the user after the activity started.

An efficient learning procedure aims to minimize explicit feedback, as it requires attention from the user and can break the workflow of the task, as well as correction feedback, meaning a mistake have been made which can also break the workflow of the task. The challenge is due to these two objectives being opposed.

5.3.5 Estimating potential error

We now present how the potential error is estimated allowing acting differently when the system has enough information to be confident or when it should acquire more data before acting.

Because we are using relational trees to learn the expected behavior, we propose to estimate confidence using Query by Bagging [Mamitsuka, 1998].

In particular, we propose to learn $N (= 50)$ models, each from a random subset, S_i, of D (the optimal behavior dataset), i.e. $|S_i| = 0.4 \times |D|$.

We need to consider a set of optimal decisions as in most cases there are more than one and the user might give different ones. We define the error of an optimal set of predicted decisions, a subset p of $D(s)$, as the averaged minimum distance between a quality vector such that every decision in p is optimal, q_{pos}, and the quality vectors predicted by the models.

$$e_p = \frac{1}{N} \sum_{n=0}^{N} \operatorname*{argmin}_{q_{pos} \in Q_p} |q_{pos} - q^n_{pred}|$$

with $q^n_{pred} = [Q^n_f(s,d)]_{d \in D(s)}$, the vector of quality values predicted by learner n, and $Q_p = \{\mathbb{R}^{|D(s)|} | \forall d \in p, d' \in D(s), t_d \geq t'_d\}$, the set of quality vectors such that all decisions from p are optimal.

Given the state, for each possible set of decisions, the error is computed. The set with the least error is the final prediction. Sometimes more than one set shares the same error (for example, if only two decision, d_1 and d_2 are optimal for every learner, three sets will minimize the error: $\{d_1\}$, $\{d_2\}$ and $\{d_1, d_2\}$) we pick the one with the higher cardinality.

For a given set of decisions, we cast the problem of finding the associated error as a quadratic optimization under constraints problem and use an off-the-shelf solver to solve it.

5.4 Evaluation

This section presents the different experiments we conducted to evaluate the framework. They are divided in simulation experiments and a robotic experiment. In simulations, we validate the approach and evaluate how it can handle different situations. In the robotic setup, we validate it can be used on a real world robot.

5.4.1 Simulation Experiments

The simulation experiments are conducted on two domains and evaluate different aspects: the impact of the *confident threshold*, how the interactive approach performs compared to a batch one, how sensitive to noise the system is, how well it can transfer and what is the quality of the potential error. We use two metrics for most of these experiments. The cumulated number of explicit instructions and cumulated number of errors across different task executions. As explained earlier, both explicit instructions and errors break the workflow of the task and as such should be avoided.

Domains

We test our system in two domains: the blocksworld because it allows to easily change its dimension, and a more realistic cooperative human-robot assembling task.

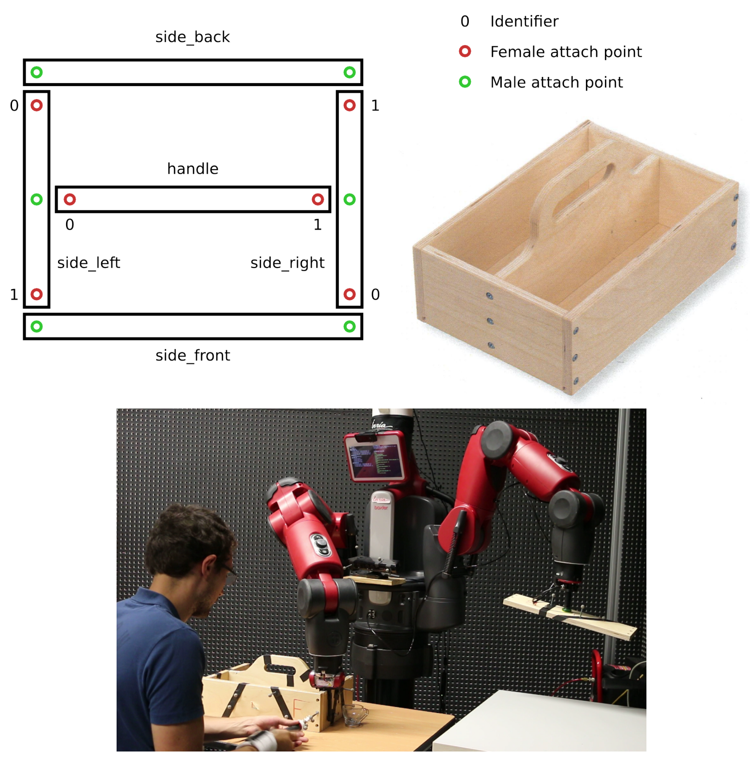


Figure 5.2: The toolbox domain. A box can be assembled in an easier and efficient way if a collaborator robot helps by providing the new parts and by holding parts in place to make screwing easier for the user. On the photo, the box is assembled by a human with the help of a robot. The robots hold the piece in place while the human is screwing. Concurrently, the robot is picking a piece in preparation for the next step.

Concurrent blocksworld This domain extends the standard blocksworld by allowing two activities (pick and put) to be executed at the same time. Blocks can be put on top of each other or on a surface (called floor) by a robot and by a human. Unless otherwise specified we use 5 blocks (2 red and 3 blue blocks).

The domain is represented with the predicates: *on/2*, *clear/1*, *busy/1*, *in_hand/2*, *blue/1* and *red/1*. The activities are *pick(agent, block)* and *put(agent, block, block)*, both of them last one unit of time and both of them can be realized by either the robot or the human.

The goal of the task is to stack all blocks in one tower. Starting states are generated by uniformly draw one state such that the number of towers is 4 and no activities are running.

Cooperative toolbox The cooperative toolbox domain is inspired by industrial tasks. In this domain, represented in Fig. 5.2, a robot must support a human in the assembly of a toolbox. The toolbox is constituted by five pieces: handle, side_left, side_right, side_front and side_back. The toolbox can be built in different ways, the side_left and side_right are interchangeable as well as side_front and side_back.

At the beginning of the task, all pieces are set in a location not accessible by the human. The robot has to realize consecutively two activities to put them in the human workspace : pick(piece) and give(piece). Once the human has pieces in his workspace he can start a positioning activity to put them in a correct disposition for screwing. Simultaneously, the robot should hold one of the pieces in order to allow the human to screw them together. Hold activity is done with the right arm of the robot whereas pick and give are done with the left arm so the robot can do different activities at the same time (which can be naturally represented with the RAP formalism). This domain has $240,000$ states. In this domain, the task is to build the toolbox.

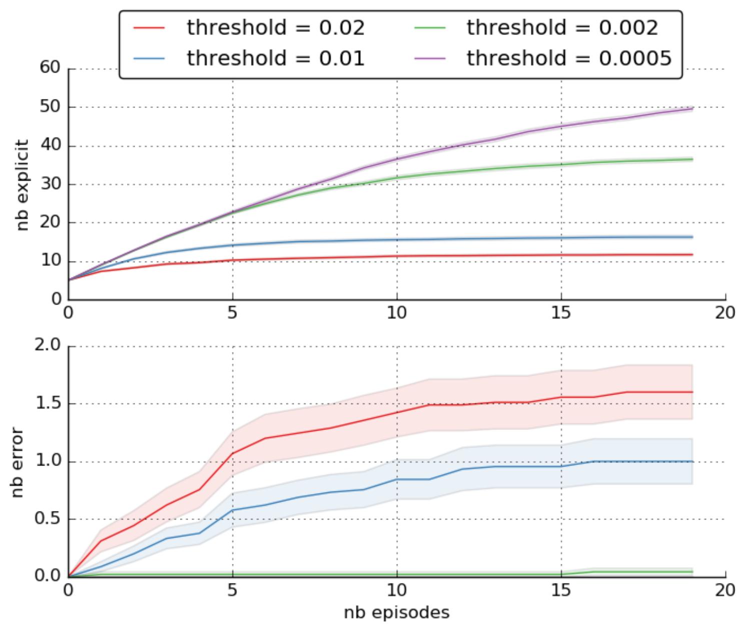
Results

Impact of the *confident threshold* parameter Previously we introduce the *confident threshold* that controls the trade-off between the number of explicit instructions and the number of error of the system. Fig. 5.3 presents the cumulative number of explicit instructions and errors for different value of *confident threshold*. We can see that, for both domains, with high values, the number of explicit instructions is low but the number of error is high and the other way around for low values of the *confident threshold*. It can indeed be used to control the trade-off between the number of explicit instructions and the number of error of the system. For application where errors are costly, it should be set to a low value whereas non-risky application should use a high value to increase learning speed.

Incremental vs. Batch A main claim of this chapter is that an interactive learning approach allows reducing the numbers of instructions and errors made by a system when compared to batch learning. Fig 5.4 displays the cumulative numbers of instructions and errors for interactive and batch after 20 task completions. Each approach has been run with different values for the parameter that controls for the trade-off between instructions and errors, *confident threshold* for interactive and number of demonstrations (full solve of the task) for batch.

For evaluating the batch learning, the interactive process is used except for the following modifications, with $nb\_demo = n$:

- The policy is only learned once, at the end of the n-th episode.
- Before the end of the n-th episode, the robot will never use the confident mode.



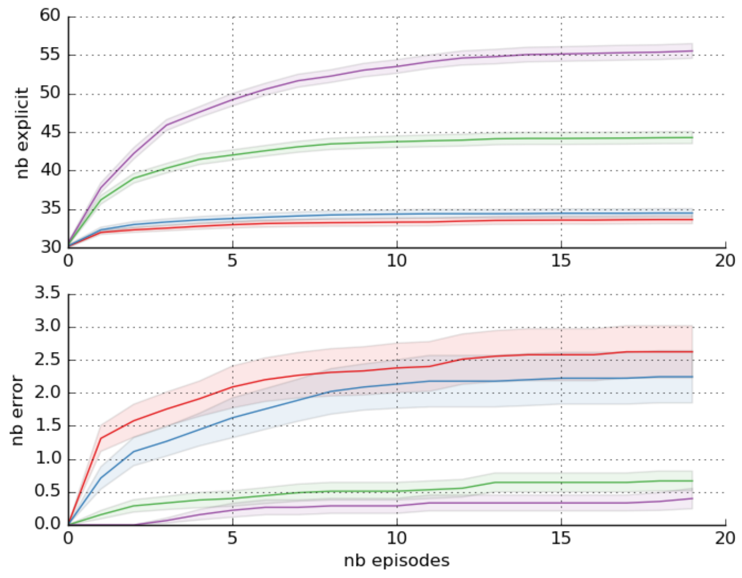


Figure 5.3: Impact of the *confident threshold* parameter. Top is for the blocks domain while bottom is for the toolbox domain.

- After the n-th episode, the robot will never use the ask-before-ask mode.
- After the n-th episode, the user will never give feedback to the robot.

The results show that for the blocksworld domain, for any values of *nb_demo*, there is a value of *confident threshold* such that the interactive approach is better both in terms of explicit instructions and errors. This also true for

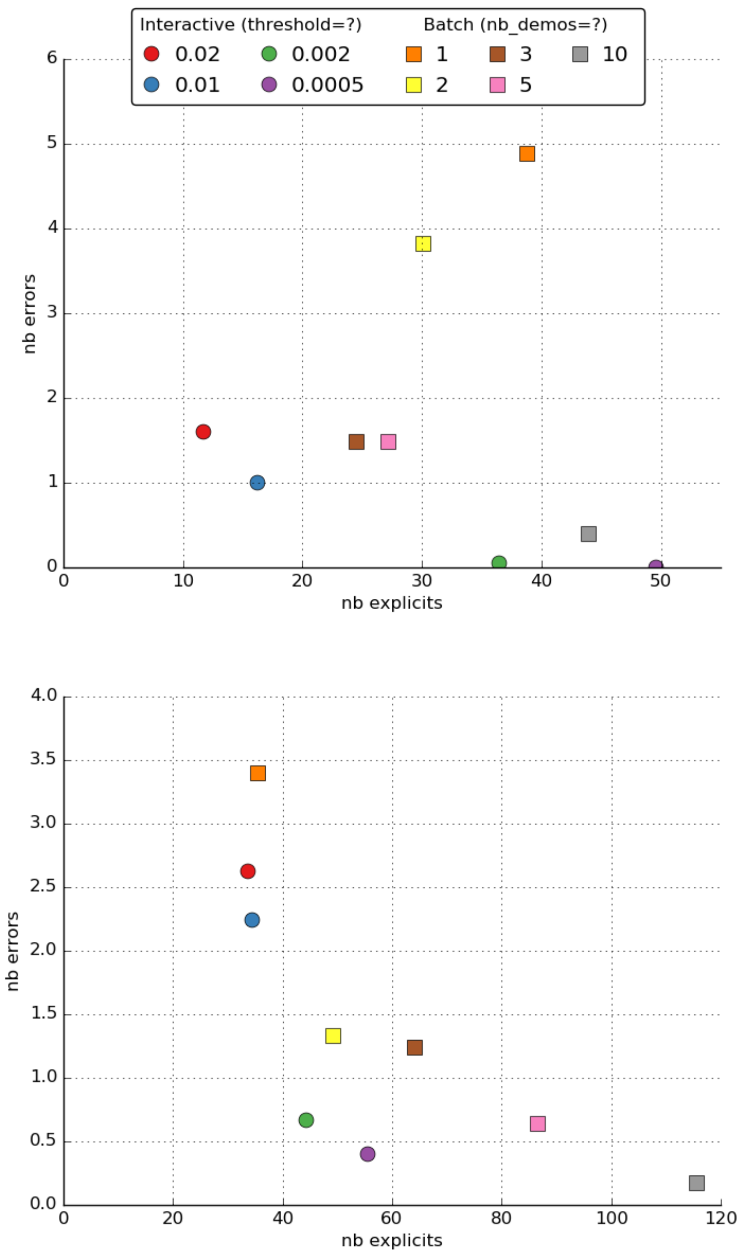


Figure 5.4: Comparison of different batch and interactive strategies after 20 iteration in terms of number of errors and number of explicit instructions. Top is for the blocks domain while bottom is for the toolbox domain.

the toolbox domain except for *nb_demo* = 10 where even a very low value of *confident threshold*, the interactive approach makes more mistakes. We explain later why this is the case. We argue that in most case, interactive with *threshold* = 0.0005 will be preferable to batch with *nb_demo* = 10, the number of errors is only 0.3 more while the number of instructions goes down to

56 (from 115).

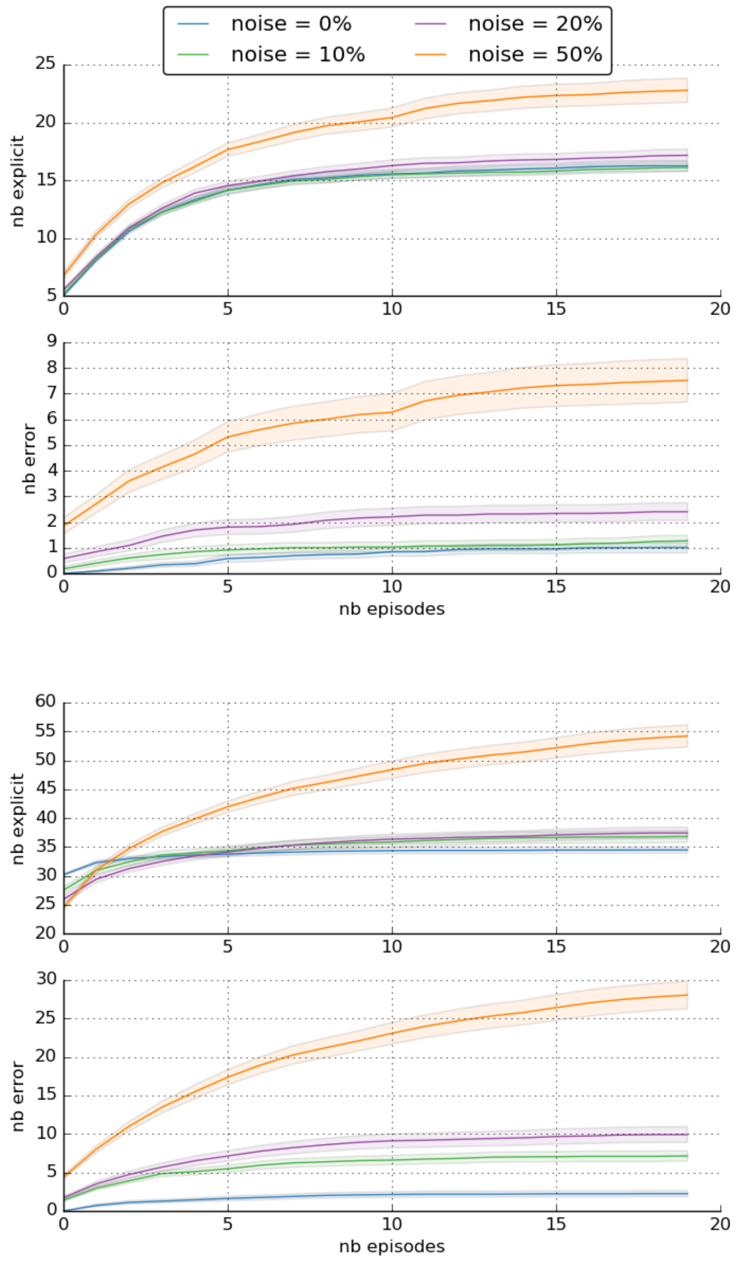


Figure 5.5: Cumulative numbers of explicit instructions and errors when the communication is noisy. Top is for the blocks domain while bottom is for the toolbox domain.

Noise Sensitivity In a real life scenario, the feedback might be noisy. For example, if the system is using speech recognition, some recognition error can occur or if the system is using a graphical interface, the user might miss-click. To evaluate the robustness of the proposed system to noise, we make the assumption that the communication to instruct the robot of the optimal action is noisy, for $noise = n\%$, in $n\%$ of the case a random decision is given in place of an optimal one. We can see in Fig. 5.5 that up to 20% noise the system is able to learn the policy efficiently. However, at 50% noise the performances decrease considerably.

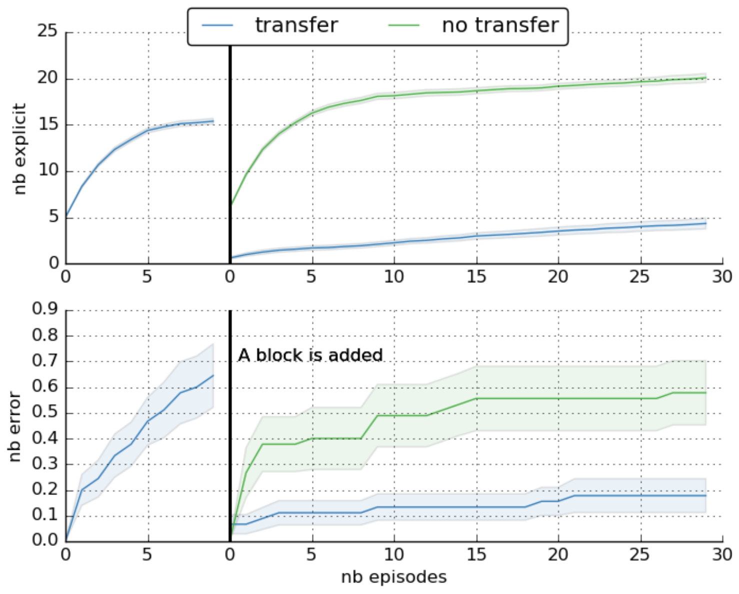


Figure 5.6: Impact of transferring a learn policy from a 5 blocks world to a 6 blocks world

Transfer One of the reasons that we propose to rely on relational representation is to naturally deal with domains where the number of objects can change. It is an important feature for robotics system that works with humans as the domain cannot be fully defined in advance. Human are unpredictable and very flexible (i.e having a second screwdriver around because the handle is more comfortable). Fig. 5.6 presents the results of an experiment in which we compare transferring a policy from a 5 to 6-blocksworld and learning a new policy from scratch on a 6-blocksworld. The transferred policy is trained for 10 episodes, we then add a block to the domain. The explicit instructions and errors counters are reset. The not transferred policy starts at episode 10 to allow comparing the two. The transferred policy requires fewer instructions while making fewer errors showing a clear benefit of transfer.

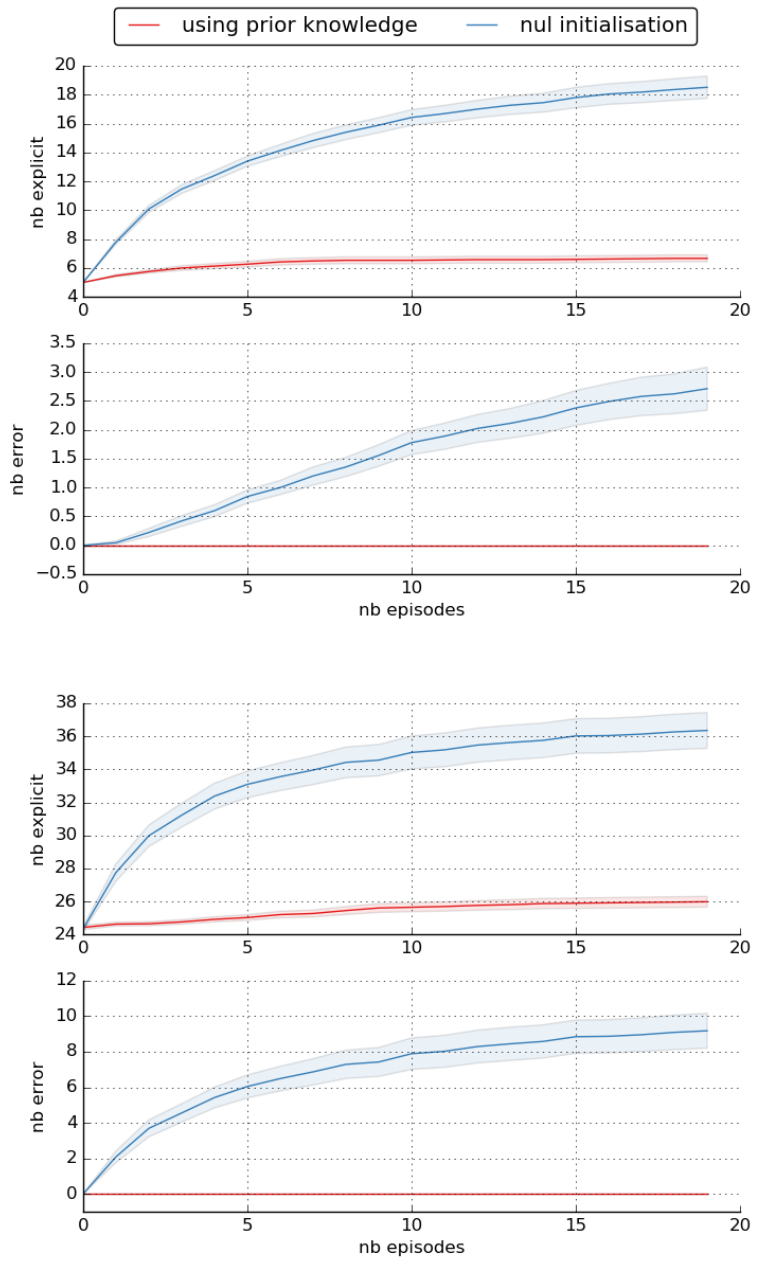
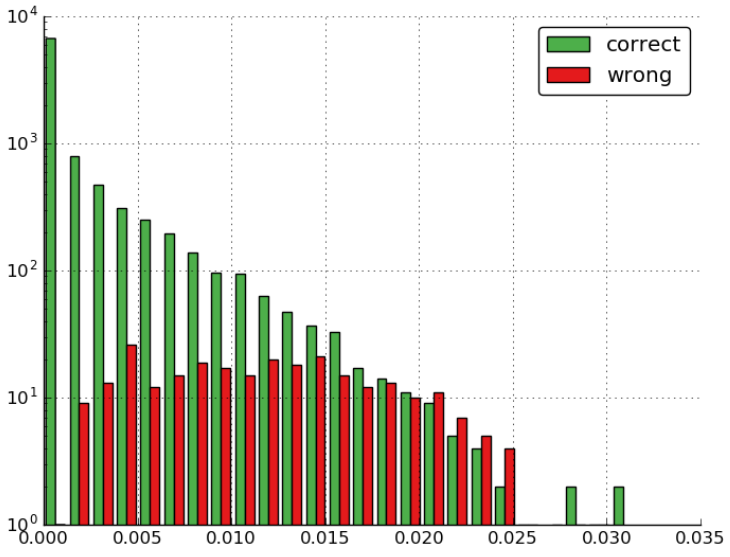


Figure 5.7: Impact of transferring a learn policy from a 5 blocks world to a 6 blocks world

Preference learning Fig. 5.7 evaluates the impact of leveraging prior knowledge about the task when such knowledge is available. In this experiment, we changed the task in both domains to add a preferences component. For the block domain, the user prefers to only handle red block. In the toolbox domain, the user prefers to get pieces one by one and as much as possible for the robot to have the arms in home position.

We can see that in both domains using the prior knowledge of the task allows learning faster while making no mistakes.



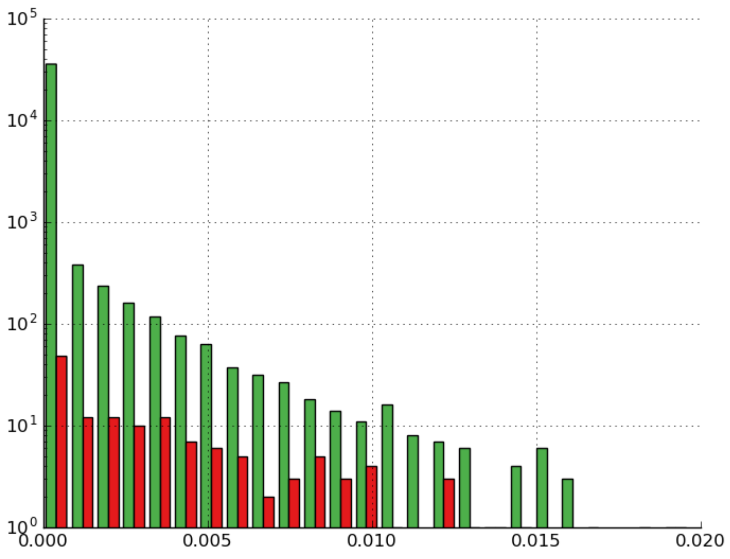


Figure 5.8: Distribution of the potential error for correctly and wrongly predicted decisions. Top is for the blocks domain while bottom is for the toolbox domain.

Potential Error Quality We are interested in evaluating if the measure used to estimate the potential error is clearly predicts the future error. In Fig.

5.8 we plot the distribution of the potential error for correctly and wrongly predicted decisions. We used the data of 45 runs of the interactive approach for 20 episodes with a *confident threshold* value of 0.01.

We can see that, for both domains, most of the correct decisions have a very low error (the y-axis uses a log scale). And, for the blocks world domain, the distribution of wrong decisions is clearly different. For the toolbox domain, however, the two distribution are more alike. It explains that, even for low values of the *confident threshold*, errors are made by the interactive approach.

5.4.2 Robotic implementation

We now present an experiment of a joint human-robot task realized with a user and the Baxter robot. This experiment uses the toolbox domain (figure 5.2) and consists of a collaborative assembly of a toolbox. We run the system for three episodes, each following a different RAP trajectory. The first two start with all the pieces not in the human workspace but the handle piece is assembled differently (it is reversed). The third one starts with the side_front already in the human workspace.

The perception system relies on Optitrack cameras for object tracking and human activity recognition, both outside the scope of this chapter. Based on this information, we compute the truth values of the domain predicates for all objects. The system is also programmed to recognize the different human activities.

Using an algorithm on a real robot is always more challenging than in simulation. The list of additional difficulties includes : an imperfect perception system and a model not conform with the reality. The imperfect perception system leads to predicates wrongly detected as true as well as the other way around (for example, because of occlusion). Having an algorithm robust to noise helps to cope with that. The mismatch of the model from the reality leads to making decision in states that are not predicted by the model. The presented algorithm allows learning what to do in those cases.

The results are presented in Fig. 5.9. We set the *confident threshold* low enough such that no error are made by the system. We can see that the algorithm allows the system to significantly reduce the number of modification feedback after only one assembly. A modification feedback is given when the robot suggests a nonoptimal activity. So, a decrease in the number of modification feedback shows the robot correctly learned the task and is capable of generalizing. The bottom graph shows that the number of use of the interface decrease with the number of assemblies. Which means that the robot is capable of correctly estimating its confidence. The video of the whole learning process can be found at https://vimeo.com/196631825.

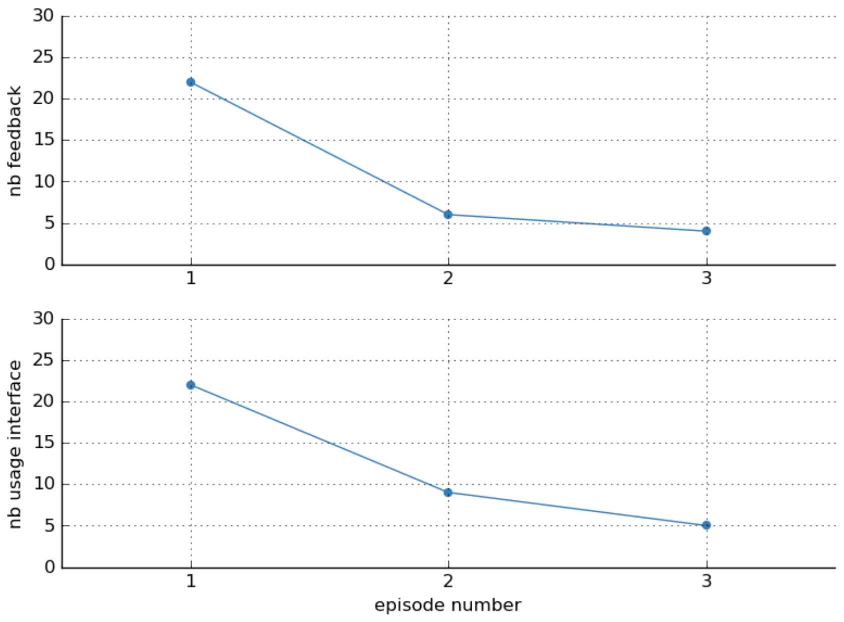


Figure 5.9: Number of modification feedbacks (top) and number of uses of the interface (bottom) during three consecutive assemblies of the toolbox with a real robot. Because there were no error made by the robot, a decrease in the number of modification feedback shows the user needs to instruct less and so that the robot correctly learned the task.

5.5 Conclusion

In this chapter we present the first approach for a robot to interactively learn a support behavior during concurrent human-robot collaboration. In our setting, the robot simultaneously executes the task and learns what the user expect it to do. Our main contribution is to consider such behavior learning in an interactive and concurrent multi-agent setting.

We first detailed the RAP framework and how to use it to represent concurrent and cooperative task realization. Because RAP is constructed on an underlying relational semi-MDP model, one can use pre-existing policy learning algorithm such as TBRIL. After introducing an interactive behavior learning framework that mixes the usual training and exploiting phase, we presented an estimate of the potential error for a given activity prediction.

We have shown that using the proposed framework leads to the robot making fewer errors and requiring less explicit instruction than more traditional batch learning. We also evaluate how robust our approach is for intra-domain transfer and noise. We demonstrate that we could change the dimension of the problem and still reuse parts of the information to learn fast. We describe how to use the proposed framework to learn user's preference during the execution of a collaborative task.

Lastly, we evaluated the implementation of this framework on a real robot and showed its viability. We believe interactive learning is an important feature to allow naive users to teach behavior to a robot.

This chapter presented the final product of this thesis in terms of algorithmic development. We believe we have proposed an architecture to efficiently learn behavior from a human operator in the context of collaborative tasks. The next chapter will present a user study of this architecture.

Chapter 6

Impact of Robot Initiative on Human-Robot Collaboration

This chapter presents a user study on the impact of robot initiative in cooperative task. It uses the framework proposed in Chapter 5. It has been published previously as a conference paper under the name "Impact of Robot Initiative on Human-Robot Collaboration" in Hri conference [Munzer *et al.*, 2017a] as a Late breaking report paper. Co-authors are Yoan Mollard and Manuel Lopes.

6.1 Introduction

Today industrial robots are still restricted to highly repetitive tasks where they work separated from humans. However, due to new technological developments both in robot safety and artificial intelligence, robots might soon see a more widespread adoption. Robot safety has recently been greatly improved thanks to more compliant designs and better human acknowledgment. This is allowing robots to work in the same workspace as humans without the risk to harm them. In the same time, advances in machine learning make it possible to have robots that can learn new tasks from non-expert human operators. It makes it economically possible to use robots for short-lived tasks.

Especially for collaborative tasks, there is still no definitive answer on how much autonomy is expected by the human coworker. This chapter focuses on studying how humans react to a robot that can take initiative. We have created a semi-autonomous robotic system that responds to instruction but can take initiatives when it is confident about the action to realize based on previous experiences. We compare it to a robot without such capacity.

Systems that progressively learn a behavior from human expert have been previously proposed. In Grollman and Jenkins [2007]; Mason and Lopes [2011], the user controls the robot. Once it stops the system will use past experiences to generalize to new situations. The user can take back the control at anytime to guide the robot when it judges the behavior is incorrect. Closer to what we propose, other works consider estimating the robot confidence to request guidance when the action to take is unsure [Chernova and Veloso, 2009]. Other works consider optimizing an external reward using social guidance [Knox and Stone, 2010; Thomaz and Breazeal, 2008].

The impact of robot initiative in human-robot collaboration has been previously studied. In Gombolay *et al.* [2015], the authors studied the impact of autonomy for a scheduling and executing task and found that user preferred to let control to the robot for the scheduling part. The present work differs on the following aspects : i) we study increasing autonomy and ii) in a collaborative task where human and robot share the same workspace.

We present the conducted study in section 6.2 and the results in section 6.3.

6.2 User Study

We perform a user study to check the following hypothesis: i) users find the robot more useful when it is able to take some initiatives and ii) users are less comfortable and more afraid when working with a semi-autonomous robot.

The study presented the subjects sequentially with two conditions *instructed* and *semi-autonomous*. Half of the subjects started with the *instructed* condition while the other half started with the *semi-autonomous* condition. After each condition, the subjects answered to twelve questions, presented as Likert scales about their experience. Finally, they were asked which condition they preferred.

The task considered was to assemble a toolbox with the help of a Baxter robot. The user could instruct the robot to do support actions using an interface on a tablet. The robot is equipped with 4 actions. It can pick a piece, give a piece, hold a piece in place (to help the user screw) and reset arms in home position. The pick and give actions are realized with the left arm while the hold action is realized with the right one. The robot is controlled using the Relational Activity Process (RAP) framework [Toussaint *et al.*, 2016] that allows concurrently running actions and uses relational representations for states and actions.

The *semi-autonomous* condition is composed of three assemblies of the toolbox. During the assemblies, the system was gathering information to learn a relational policy learner, in particular TBRIL [Natarajan *et al.*, 2011], to learn a mapping from states to actions. It also uses Query by Bagging [Mamitsuka, 1998] to estimate its confidence. During the second and third assemblies, each time it encountered a new state, it predicted the correct action and associated confidence. When the confidence was superior to a threshold, the robot started the action autonomously. Otherwise, it asked the user to confirm.

The *instructed* condition was also composed of three assemblies of the tool-

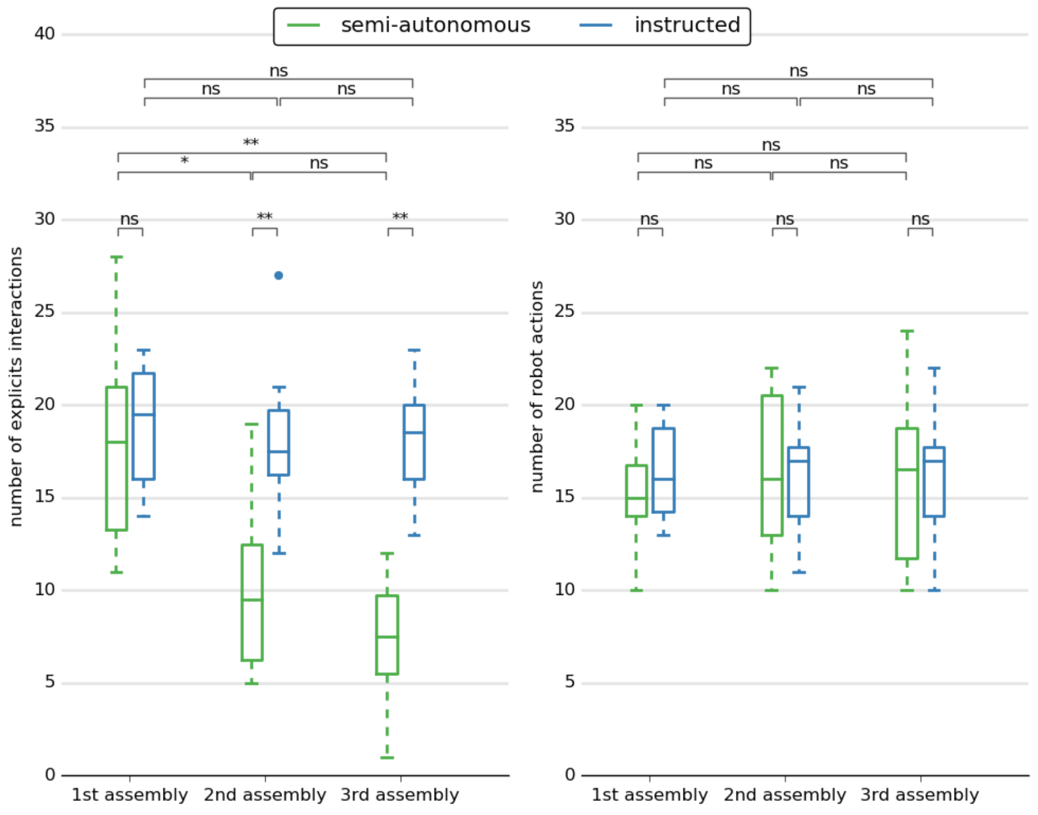


Figure 6.1: Number of interactions with the tablet (left) and number of action (right). Significance has been tested using the Wilcoxon test with the Pratt treatment and noted according to the standard defined by the APA(American Psychological Association)

box. However, no learning was involved. The user had to instruct all actions to the robot.

The study was conducted on 10 subjects (4 females) of age 30.1 ± 10. They self-reported an experience with robotics system at 3.1 ± 1.6 on a scale from 1 to 5.

6.3 Results

We first present quantifiable results before detailing results obtain through the questionnaires.

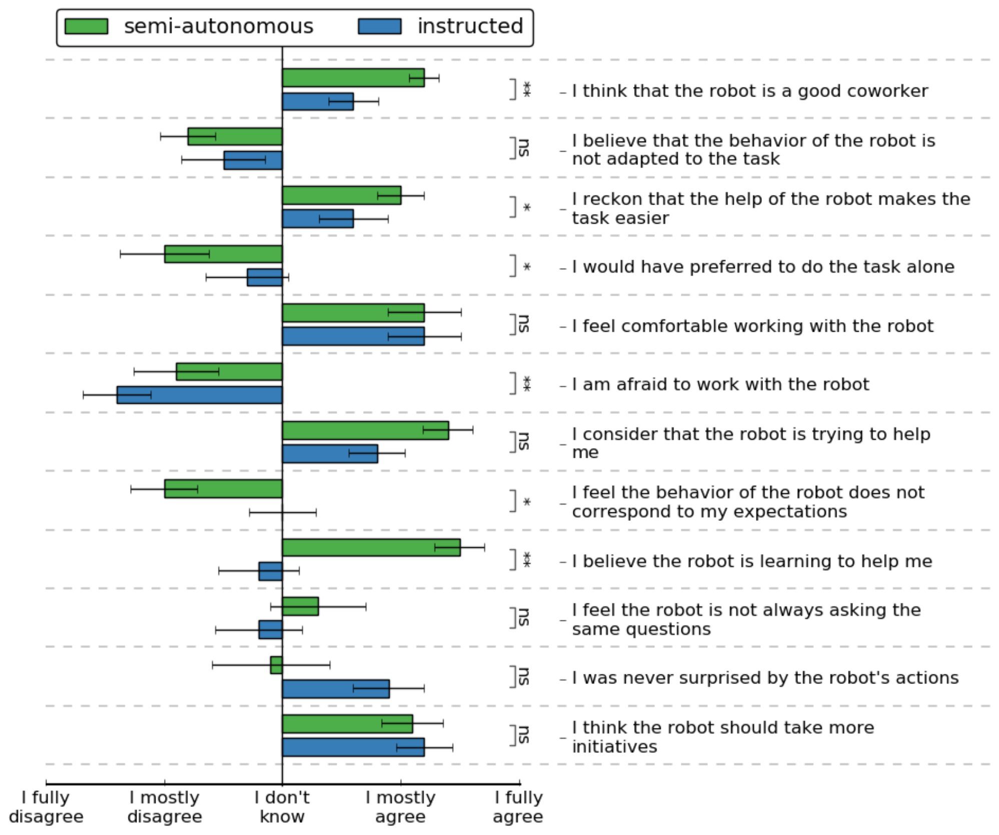


Figure 6.2: Answers to the questionnaire. Each subject experienced both conditions in random order. Significance has been tested using the Wilcoxon test with the Pratt treatment and noted according to the standard defined by the APA. Errors bars represent the standard error of the mean.

6.3.1 Quantifiable Results

Figure 6.1 presents the number of interactions with the tablet and number of robot action for both conditions. In the *semi-autonomous* condition the robot is able to learn the correct action, this can be seen as the number of interactions decreases with the number of assemblies while the number of robot actions stays constant. On the contrary, for the *instructed* condition both numbers stay constant. The number of interactions is sometimes higher than the number of actions because user tried to instruct actions before they were available (for example give, before pick finished).

6.3.2 Questionnaire Results

Results of the questionnaires are shown in Figure 6.2. The first four questions are related to how helpful the robot is. In three of these four questions, the results are statistically significantly better for the *semi-autonomous* condition. They considered that the robot was a good coworker, that it made the task easier and that it was better to do the task with the robot. Also when asked which condition they preferred 90% of people choose the *semi-autonomous* condition.

The next four questions are related to acceptability. While users reported being more afraid of the robot during the *semi-autonomous* condition, they also reported that the behavior was more conform to their expectations. This means that these two criteria are not strongly correlated. We want to precise that due to precision errors the robot was sometimes failing its actions. We hypothesize that people would be less afraid of a robot making fewer mistakes.

The last four questions treat about autonomy and learning. Users clearly noted that the robot was learning in the *semi-autonomous* condition. The agreement to the question "I think the robot should take more initiative." is positive and similar for both conditions despite the robot taking no initiative in the *instructed* condition and starting half of its action by itself in the other conditions (see section 6.3.1). This means users are expecting more autonomy from the robot.

6.3.3 Discussion and Future Work

This chapter presented a user experimentation to study the impact in terms of helpfulness and acceptability of a semi-autonomous robot for human-robot collaboration. The robot has shown to be seen as more helpful by the users. Users were also found to be more afraid of the semi-autonomous robot while its behavior was corresponding more closely to their expectations. As the users thought the semi-autonomous robot should take more initiative, future work includes comparing it to a fully autonomous (hard-coded) robot.

Chapter 7

Discussion

We presented in this thesis our work toward an efficient support robot for small scale industry. The work was focused on the high-level plan learning. We offer an approach based on interactive learning and relational representations allowing a robot to learn a cooperative behavior from a non-expert teacher. We validated the approach in simulation, on a real robot and by conducting a limited user study.

7.1 Contributions

Toward building such a system we introduce a number of contributions to the state of the art.

7.1.1 Algorithmic

First, we developed and evaluated the first Inverse Reinforcement Learning (IRL) algorithm for Relational Domains (see Chapter 3). We based our work on a previously developed IRL algorithm that we lifted to the relational setting. This required us to add an intermediary reward shaping step to create a compact reward that transfers well to different domains.

We then demonstrated how relational policy learning algorithms, both Imitation Learning (IL) and IRL, can be used in Relational Activity Processes (RAPs), allowing robots to learn concurrent policies in such domains (see sections 4.5 and 4.6).

We proposed a novel method for potential error estimation in tasks with multiple optimal actions (see section 5.3.5). It is based on query by committee and finds the best set of actions and its potential error based on a bag of learners. It contributed to the development of an architecture for interactive learning in relational domains (see section 5.3.1). We showed that compared to a batch approache, this architecture allows the robot to require fewer instructions from the user while making fewer errors.

7.1.2 Human-robot collaboration

Because this architecture is based on RAP, we were able to used it to learn cooperative behavior. We showed how the team behavior can be seen as a sequential decision making process while allowing both agent to start new activity at any time (see section 5.3.2). We also demonstrated, that if the task is known beforehand, only preferences of users can be learned in the cooperative context by decomposing the quality function (see section 5.3.3).

Lastly, we showed that these developments can be used in practice by creating a robotic setup and conducting a user study (see Chapter 6).

7.2 IRL vs IL

Throughout this thesis, IRL and IL have been compared in different scenarios. We would like to elaborate on the main differences between these two approaches when working with relational representations. One standard strength of IRL is to generalize very well to unseen states. However, we have found that when working with relational representations this advantage becomes less important as IL approaches already transfer well. Of course, as seen in chapter 3, for extreme cases IRL still has the advantage.

Major drawbacks of IRL are the need to have access to a transition model of the world and a higher computational cost. Not using IRL made it possible to use the proposed approach on a real robot. In real life experiments, the transition model will not match exactly the dynamic of the world. It is particularly problematic when perception problems are involved. The policy found from the reward might be wrong as a consequence. The computational cost is also limiting. As the world described by the representation have more states, finding the optimal policy from the learned reward takes more time. In the real robot setup, it can not run in real time preventing us to offer an interactive approach.

7.3 Limitations of RCSI

The curious reader might wonder why we don't present results for RCSI in chapter 5. We discovered a weakness in the algorithm when working with RMDP. The second step of the algorithm, shown in Figure 3.1, doesn't scale well for big datasets.

This step consists in stochastic optimization and the dimension of the vector to optimize is equal to the number of unique states in the training dataset. Because of the curse of dimensionality, stochastic optimization works better in low dimensionality problems. When using interactive learning the size of the dataset will grow after each task realization meaning the second step will work less and less.

The performance of RCSI never reaches a satisfying level. At the beginning, the dataset is too small to correctly learn the behavior while later on the stochastic optimization process loses its efficiency leading to a reward impossible to learn with a small tree.

One could argue that we should not use the second step of RCSI and simply use a more complex model to learn the reward from the quality-function. However, by removing the simplicity constraint on the reward, the interest of RCSI is lost. The quality-function learned from TBRIL and the quality-function obtained from the learned reward of RCSI becomes equivalent.

7.4 Future Work

7.4.1 Describing the world

Throughout this work, we have assumed that the robot is able to apply the high-level actions at the geometric level and that the world representation (the predicates) are given. While the first assumption seems reasonable to us the second is more of an open problem. Indeed, the research in both motion planning and trajectory learning is really rich. We believe it is today realistic for a non-expert to program most of the actions used in this work. It would require a short formation and a good interface using a combination of motion planning and trajectory learning.

Designing the world representation, on the other hand, is not possible for a non-expert. It requires a good knowledge about the algorithm and, often, several iterations. One necessary research step before it can be used in practice by non-expert to solve a wide variety of task is for the robot to design its own predicates.

Automatic symbols design is a hard problem and one could argue that it should be solved first. However, most researchs on symbol learning is task agnostic. The problem can be made simpler by taking in account the task. Using the solutions proposed in this thesis it should be possible to reduce the problem of learning symbols that describe the world to learning symbols that allow learning the current task.

7.4.2 Coordinated Actions

Another possible development of the work presented in this thesis can be the learning of coordinated action between the robot and the human on a geometric level. Research in concurrent motion primitive already exists, however, they often do not consider the task at hand. By using the high level knowledge of the task the team motion interaction could be made more efficient.

In a similar research direction the high level policy learning approach presented in this work could be improve by taking into account the geometric features of the different objects and their trajectory. Such an integrated logic-geometric approach would allow a very natural interaction between the member of the team and might improve the acceptance of the robot by the operator.

Bibliography

Abbeel, Pieter and Ng, Andrew Y, 2004. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning.*

Aberdeen, Douglas and Buffet, Olivier, 2007. Concurrent probabilistic temporal planning with policy-gradients. In *International Conference on Automated Planning and Scheduling.*

Akrour, Riad, Schoenauer, Marc and Sebag, Michele, 2011. Preference-based policy learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.*

Amato, Christopher, Konidaris, George, Cruz, Gabriel, Maynor, Christopher A, How, Jonathan P and Kaelbling, Leslie P, 2015. Planning for decentralized control of multiple robots under uncertainty. In *International Conference on Robotics and Automation.*

Amato, Christopher, Konidaris, George D and Kaelbling, Leslie P, 2014. Planning with macro-actions in decentralized pomdps. In *International Conference on Autonomous Agents and Multiagent Systems.*

Argall, Brenna D, Chernova, Sonia, Veloso, Manuela and Browning, Brett, 2009. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483.

Beaudry, Eric, Kabanza, Froduald and Michaud, François, 2010. Planning with concurrency under resources and time uncertainty. In *European Conference on Artificial Intelligence.*

Blockeel, Hendrik and De Raedt, Luc, 1998. Top-down induction of first-order logical decision trees. *Artificial intelligence*, 101(1).

Breiman, Leo, Friedman, Jerome, Stone, Charles J and Olshen, Richard A, 1984. *Classification and regression trees.* CRC press.

Browne, Cameron B., Powley, Edward, Whitehouse, Daniel, Lucas, Simon M., Cowling, Peter, Rohlfshagen, Philipp, Tavener, Stephen, Perez, Diego, Samothrakis, Spyridon, Colton, Simon and others,

2012. A survey of monte carlo tree search methods. *Transactions on Computational Intelligence and AI in Games*, 4(1).

Buffet, Olivier and Aberdeen, Douglas, 2009. The factored policy-gradient planner. *Artificial Intelligence*, 173(5-6).

Chandramohan, Senthilkumar, Geist, Matthieu, Lefevre, Fabrice and Pietquin, Olivier, 2011. User simulation in dialogue systems using inverse reinforcement learning. In *Interspeech*.

Chernova, Sonia and Veloso, Manuela, 2009. Interactive policy learning through confidence-based autonomy. *Journal of Artificial Intelligence Research*, 34(1).

De Raedt, Luc, 2008. *Logical and relational learning*. Springer Science & Business Media.

Džeroski, Sašo, De Raedt, Luc and Driessens, Kurt, 2001. Relational reinforcement learning. *Machine learning*, 43(1-2).

Fainekos, Georgios E., Kress-Gazit, Hadas and Pappas, George J., 2005. Hybrid controllers for path planning: A temporal logic approach. In *Conference on Decision and Control/European Control Conference*.

Friedman, Jerome H, 2001. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, 29(5).

Geurts, Pierre, Ernst, Damien and Wehenkel, Louis, 2006. Extremely randomized trees. *Machine learning*, 63(1).

Gombolay, Matthew C, Gutierrez, Reymundo A, Clarke, Shanelle G, Sturla, Giancarlo F and Shah, Julie A, 2015. Decision-making authority, team efficiency and human worker satisfaction in mixed human–robot teams. *Autonomous Robots*, 39(3).

Grollman, Daniel H and Jenkins, Odest Chadwicke, 2007. Dogged learning for robots. In *International Conference on Robotics and Automation*.

Hansen, Nikolaus, Müller, Sibylle and Koumoutsakos, Petros, 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). *Evolutionary Computation*, 11(1).

Hart, Stephen and Grupen, Roderic, 2011. Learning generalizable control programs. *Transactions on Autonomous Mental Development*, 3(3).

Jain, Ashesh, Sharma, Shikhar, Joachims, Thorsten and Saxena, Ashutosh, 2015. Learning preferences for manipulation tasks from online coactive feedback. *The International Journal of Robotics Research*, 34(10).

Jain, Ashesh, Wojcik, Brian, Joachims, Thorsten and Saxena, Ashutosh, 2013. Learning trajectory preferences for manipulators via iterative improvement. In *Conference on Neural Information Processing Systems*.

Keller, Thomas and Helmert, Malte, 2013. Trial-based heuristic tree search for finite horizon MDPs. In *International Conference on Automated Planning and Scheduling*.

Kersting, Kristian, Otterlo, Martijn Van and De Raedt, Luc, 2004. Bellman goes relational. In *International Conference on Machine Learning*.

Khardon, Roni, 1999. Learning action strategies for planning domains. *Artificial Intelligence*, 113(1).

Klein, Edouard, Geist, Matthieu, Piot, Bilal and Pietquin, Olivier, 2012. Inverse reinforcement learning through structured classification. In *Conference on Neural Information Processing Systems*.

Klein, Edouard, Piot, Bilal, Geist, Matthieu and Pietquin, Olivier, 2013. A cascaded supervised learning approach to inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

Knox, W Bradley and Stone, Peter, 2010. Combining manual feedback with subsequent mdp reward signals for reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*.

Knox, W Bradley, Stone, Peter and Breazeal, Cynthia, 2013. Training a robot via human feedback: A case study. In *International Conference on Social Robotics*.

Kober, Jens and Peters, Jan, 2009. Learning motor primitives for robotics. In *International Conference on Robotics and Automation*.

Kocsis, Levente and Szepesvári, Csaba, 2006. Bandit based monte-carlo planning. In *European Conference on Machine Learning*.

Koppula, Hema S., Jain, Ashesh and Saxena, Ashutosh, 2016. Anticipatory planning for human-robot teams. In *International Symposium on Experimental Robotics*.

Lang, Tobias and Toussaint, Marc, 2010. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39(1).

Lang, Tobias, Toussaint, Marc and Kersting, Kristian, 2012. Exploration in relational domains for model-based reinforcement learning. *The Journal of Machine Learning Research*, 13(1).

LEE, Min Kyung, FORLIZZI, Jodi, KIESLER, Sara, RYBSKI, Paul, ANTANITIS, John and SAVETSILA, Sarun, 2012. Personalization in hri: A longitudinal field experiment. In *Conference on Human-Robot Interaction*.

LOPES, Manuel, MELO, Francisco, KENWARD, Ben and SANTOS-VICTOR, José, 2009a. A computational model of social-learning mechanisms. *Adaptive Behavior*, 17(6).

LOPES, Manuel, MELO, Francisco and MONTESANO, Luis, 2009b. Active learning for reward estimation in inverse reinforcement learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*.

LUKSCH, Tobias, GIENGER, Michael, MUHLIG, Manuel and YOSHIIKE, Takahide, 2012. Adaptive movement sequences and predictive decisions based on hierarchical dynamical systems. In *International Conference on Intelligent Robots and Systems*.

MAMITSUKA, Naoki Abe Hiroshi, 1998. Query learning strategies using boosting and bagging. In *International Conference on Machine Learning*.

MASON, Martin and LOPES, Manuel, 2011. Robot self-initiative and personalization by learning through repeated interactions. In *Conference on Human-Robot Interaction*.

MAUSAM and WELD, Daniel S., 2008. Planning with durative actions in stochastic domains. *Journal of Artificial Intelligence Research*, 31.

MITSUNAGA, Noriaki, SMITH, Christian, KANDA, Takayuki, ISHIGURO, Hiroshi and HAGITA, Norihiro, 2008. Adapting robot behavior for human–robot interaction. *Transactions on Robotics*, 24(4).

MUNZER, Thibaut, MOLLARD, Yoan and LOPES, Manuel, 2017a. Impact of robot initiative on human-robot collaboration. In *Conference on Human-Robot Interaction*.

MUNZER, Thibaut, PIOT, Bilal, GEIST, Matthieu, PIETQUIN, Olivier and LOPES, Manuel, 2015. Inverse reinforcement learning in relational domains. In *International Joint Conference on Artificial Intelligence*.

MUNZER, Thibaut, TOUSSAINT, Marc and LOPES, Manuel, 2017b. Preference learning on the execution of collaborative human-robot tasks. In *International Conference on Robotics and Automation*.

NATARAJAN, Sriraam, JOSHI, Saket, TADEPALLI, Prasad, KERSTING, Kristian and SHAVLIK, Jude, 2011. Imitation learning in relational domains: A functional-gradient boosting approach. In *International Joint Conference on Artificial Intelligence.*

Natarajan, Sriraam, Khot, Tushar, Kersting, Kristian, Gutmann, Bernd and Shavlik, Jude, 2012. Gradient-based boosting for statistical relational learning: The relational dependency network case. *Machine Learning*, 86(1).

Natarajan, Sriraam, Kunapuli, Gautam, Judah, Kshitij, Tadepalli, Prasad, Kersting, Kristian and Shavlik, Jude, 2010. Multi-agent inverse reinforcement learning. In *International Conference on Machine Learning and Applications.*

Neu, Gergely and Szepesvári, Csaba, 2007. Apprenticeship learning using inverse reinforcement learning and gradient methods. In *Uncertainty in Artificial Intelligence.*

Neu, Gergely and Szepesvári, Csaba, 2009. Training parsers by inverse reinforcement learning. *Machine learning*, 77(2-3).

Ng, Andrew Y, Harada, Daishi and Russell, Stuart, 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *International Conference on Machine Learning.*

Ng, Andrew Y and Russell, Stuart J, 2000. Algorithms for inverse reinforcement learning. In *International Conference on Machine Learning.*

Niekum, Scott, Chitta, Sachin, Marthi, Bhaskara, Osentoski, Sarah and Barto, Andrew G, 2013. Incremental semantically grounded learning from demonstration. In *Robotics Science and Systems.*

Nikolaidis, Stefanos, Gu, Keren, Ramakrishnan, Ramya and Shah, Julie, 2014. Efficient model learning for human-robot collaborative tasks. *arXiv preprint arXiv:1405.6341.*

Nikolaidis, Stefanos and Shah, Julie, 2013. Human-robot cross-training: Computational formulation, modeling and evaluation of a human team training strategy. In *Conference on Human-Robot Interaction.*

Piot, Bilal, Geist, Matthieu and Pietquin, Olivier, 2013. Learning from demonstrations: Is it worth estimating a reward function? In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.*

Puterman, Martin L, 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* John Wiley & Sons.

QUINLAN, J Ross, 1993. *C4. 5: Programs for Machine Learning.* Morgan Kaufmann.

ROHANIMANESH, Khashayar and MAHADEVAN, Sridhar, 2002. Learning to take concurrent actions. In *Conference on Neural Information Processing Systems.*

ROHANIMANESH, Khashayar and MAHADEVAN, Sridhar, 2005. Coarticulation: An approach for generating concurrent plans in markov decision processes. In *International Conference on Machine Learning.*

RUSSELL, Stuart, 1998. Learning agents for uncertain environments. In *Conference on Learning Theory.*

SANNER, Scott, 2010. Relational dynamic influence diagram language (rddl): Language description. *Unpublished.*

SCHAAL, Stefan, 1999. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6).

SEGRE, Alberto and DEJONG, Gerald F, 1985. Explanation-based manipulator learning: Acquisition of planning ability through observation. In *International Conference on Robotics and Automation.*

SHAVLIK, Jude W and DEJONG, Gerald F, 1987. Bagger: an ebl system that extends and generalizes explanations. In *AAAI Conference on Artificial Intelligence.*

SHIVASWAMY, Pannaga and JOACHIMS, Thorsten, 2012. Online structured prediction via coactive learning. *arXiv preprint arXiv:1205.4213.*

SMITH, David E. and WELD, Daniel S., 1999. Temporal planning with mutual exclusion reasoning. In *International Joint Conference on Artificial Intelligence.*

THOMAZ, Andrea L and BREAZEAL, Cynthia, 2008. Teachable robots: Understanding human teaching behavior to build more effective robot learners. *Artificial Intelligence*, 172(6).

TOUSSAINT, Marc, MUNZER, Thibaut, MOLLARD, Yoan, WU, Li Yang, VIEN, Ngo Anh and LOPES, Manuel, 2016. Relational activity processes for modeling concurrent cooperation. In *International Conference on Robotics and Automation.*

WATKINS, Christopher J C H, 1989. *Learning from delayed rewards.* PhD thesis, University of Cambridge England.

WILCOX, Ronald, NIKOLAIDIS, Stefanos and SHAH, Julie, 2012. Optimization of temporal dynamics for adaptive human-robot interaction in assembly manufacturing. In *Robotics Science and Systems*.

YOON, Sungwook, FERN, Alan and GIVAN, Robert, 2002. Inductive policy selection for first-order mdps. In *Uncertainty in Artificial Intelligence*.

YOUNES, HLS and SIMMONS, Reid G., 2004. Policy generation for continuous-time stochastic domains with concurrency. In *International Conference on Automated Planning and Scheduling*.

ZETTLEMOYER, Luke S., PASULA, Hanna and KAELBLING, Leslie Pack, 2005. Learning planning rules in noisy stochastic worlds. In *AAAI Conference on Artificial Intelligence*.

ZIEBART, Brian D, MAAS, Andrew L, BAGNELL, J Andrew and DEY, Anind K, 2008. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*.