Freie Universität Berlin

Planning and Exploration in Stochastic Relational Worlds

Dissertation zur Erlangung des Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
am Fachbereich Mathematik und Informatik
der Freien Universität Berlin

von

Tobias Johannes Lang

Berlin
2011

FREIE UNIVERSITÄT BERLIN
Fachbereich Mathematik und Informatik

Institut für Informatik
Machine Learning and Robotics Lab
Prof. Dr. Marc Toussaint

Planning and Exploration in Stochastic Relational Worlds

Dissertation

Autor	Tobias Johannes Lang aus Mainz
Vorgelegt im	Februar 2011
Disputation am	30. Mai 2011
Erstgutachter	Prof. Dr. Marc Toussaint
Zweitgutachter	Prof. Dr. Klaus-Robert Müller

Summary

Goal-directed behavior is one of the most interesting aspects of human and animal intelligence. This thesis addresses planning and exploration in so called stochastic relational worlds which are characterized by two key attributes: they contain large numbers of objects whose properties and relationships can be manipulated, and the effects of actions are uncertain. Such worlds comprise many natural environments such as households, offices, or factories. We take up ideas from the emerging field of statistical relational artificial intelligence and combine rich symbolic representations with a probabilistic framework to learn and represent compact models of action effects which generalize across objects. We propose a variety of methods for *planning* with such models in ground relational domains on the level of concrete objects. Our methods are largely based on the information processing principle of probabilistic inference in graphical models. We introduce a framework for focusing on relevant objects in planning. We lift existing *exploration* theories to relational representations. This results in a novel form of exploration which focuses decidedly on objects to which the learned knowledge does not generalize yet. Combining our proposed techniques with existing methods enables goal-directed behavior of autonomous agents in stochastic relational worlds.

Acknowledgements
Danksagung

I have been very happy to have Marc Toussaint as my advisor. I have profited greatly from his impressively quick comprehension of complex matters and his benevolent helpfulness. His honest joy in doing research has strengthened my belief in science and research.

I thank Klaus-Robert Müller for his advice and ideas, for being the second referee of my thesis, and, most importantly, for his relaxed and friendly attitude which has made his machine learning lab at TU Berlin such an enjoyable environment to work.

I am happy to have gone to Berlin and to have spent my last years at the TU machine learning lab where I have felt comfortable and met great people. Thank you, Martijn, Katja, Nikolay, Stanio, Nils and all you others!

I thank Kristian Kersting for our joint research. Back in Freiburg, I would have never expected that this friendly “guy from the neighboring lab working on this strange logic stuff” would turn out to be so inspiring for my upcoming years.

I am grateful to many other people I met at conferences, summer schools and the like who have given me inspiration, advice and motivation for my research.

Mein besonderer Dank gilt all den Menschen, die sich womöglich etwas weniger als ich für “Planen und Erkundung in Stochastischen Relationalen Welten” begeistern, mit mir aber meine wichtigen Stunden der letzten Jahre verbracht und mir dadurch sehr bei meiner Dissertation geholfen haben: meinen Meenzern, meinen (Ex-) Osnabrückern, der Schivelcrew, den Berlinerinnen und Berlinern, die ich kennenlernen durfte und mir meine letzten drei Jahre so schön und intensiv gemacht haben, und dem unbekannten Kumpel im Himmel für seine Klappe. Und ich bin sehr froh über meine Familie: meine Eltern, meinen Bruder Tillmann und meine Schwester Elisa.

"If you test your programs not merely by what they can accomplish, but how they accomplish it, then you're really doing cognitive science; you're using AI to understand the human mind."

Herbert Simon

"Sehen Sie", erklärte Hohenegg, "nach meiner Auffassung gibt es drei mögliche Haltungen angesichts der Absurdität dieses Lebens. Zunächst die Haltung der Masse – hoï polloï –, die einfach nicht akzeptieren will, dass das Leben ein Scherz ist. Diese Leute lachen nicht darüber, sondern arbeiten, horten, kauen, verdauen, huren, pflanzen sich fort, werden alt und sterben wie die Ochsen im Joch – töricht, wie sie gelebt haben. Das ist die große Mehrheit. Dann gibt es diejenigen, die, wie ich, wissen, dass das Leben ein Scherz ist, und die den Mut haben, darüber zu lachen, wie die Taoisten oder Ihr Jude. Schließlich gibt es noch jene – und das trifft nach meiner Diagnose genau auf Sie zu –, die wissen, dass das Leben ein Scherz ist, die aber darunter leiden."

Jonathan Littel: *Les Bienveillantes*.
Translation from French to German by Hainer Kober.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The chimpanzee Grande spots a banana which hangs from the ceiling. She jumps for it, but the banana is out of her reach. She becomes frustrated and runs upset through her cage hitting the walls. All of a sudden, she stops, looks at the wooden boxes which are in her cage, looks at the banana, looks at the boxes again, and then starts to put one box on top of the other (Fig. 1.1). Grande knows that boxes are objects which she can manipulate and which she can climb upon. By reasoning, she seems to have suddenly understood how she can exploit these properties favorably to get the banana by building a tower. After this first experiment, Grande handles easily similar new situations with other boxes and is able to build towers of up to four boxes.

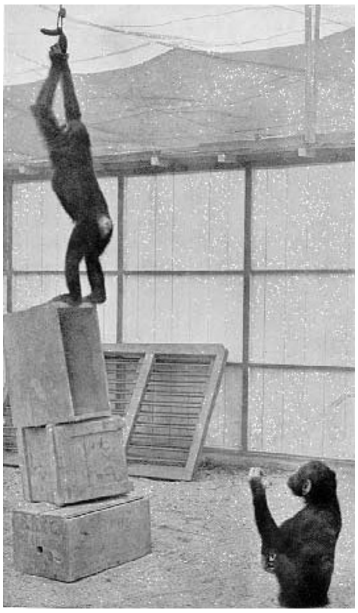


Figure 1.1: The chimpanzee Grande has successfully piled wooden boxes to reach a banana. (Source: Köhler, 1917)

Grande was one of the chimpanzees whose problem-solving skills the psychologist Wolfgang Köhler investigated in the 1910s in Tenerife, described in his book "Intelligenzprüfungen an Menschenaffen" ("The mentality of apes") (Köhler, 1917). Köhler's other chimpanzees showed likewise an understanding of how to use objects in their environment to achieve their goals and get what they want (that is, bananas). A key characteristic of the apes' behavior is the moment of insight, the "aha experience": suddenly, from reasoning the apes find a plan which they consider promising, and they start acting in a purposive way to execute this plan. This hints at the cognitive processing involved in problem-solving of animals: the chimpanzees seem to have a model of the properties of the boxes and what they can do with them—that they can move them and climb on them. Further, they appear to mentally find solutions before they actually manipulate the boxes. Such behavior shows every sign of involving insight and planning, at least on the first occasion.

What are the capabilities and prerequisites which permit such goal-directed behavior? What has Grande mastered to be able to get her banana? The experiment described above suggests that she

- represents the environment in terms of **objects and their properties and relationships**.
  There are wooden boxes and bananas in the world. Some boxes are on the ground, some on other boxes.
- has a **model of the effects of her actions** which tell her in general terms abstracting from individual objects how her actions change the world.
  Boxes can be lifted and put somewhere.
- has **learned** this knowledge from experience as it is hardly innate.
  An object is held inhand and let loose so that it falls on the object below or on the ground. A potential conclusion is that one can influence the position of objects.
- has **explored** her environment to gain the experience necessary for effective learning.
  Let's see what can be done with this wooden angular thing.
- copes with **uncertainty**.
  Sometimes, a tower of boxes is unstable and topples over.
- uses her model for **reasoning and planning** with the concrete objects in the current situation.
  Somehow, these boxes can be rearranged in such a way that by climbing them a closer position to this tempting banana can be reached.

This thesis is about computational theories and frameworks to solve these tasks and to enable goal-directed behavior such as Grande's tower-building:

> *The goal of this thesis is to understand and create computational principles for goal-directed behavior involving planning and exploration in natural environments composed of many manipulable objects.*

A household, offices and factories are examples of such environments. Such environments are often referred to as *stochastic relational worlds*. Before we discuss the concepts and contributions of this thesis on a more theoretical level, we take a closer look at stochastic relational worlds to get a better idea of our work.

1.1 Stochastic Relational Worlds

Stochastic relational worlds are a formalization of environments of objects where an autonomous agent performs actions to manipulate these objects and thus the state of the world. The general term agent subsumes humans, robots, chimpanzees, and other animals and is defined as a system that perceives its environment and takes actions which maximize its chances of success (Russell and Norvig, 2003). The two attributes of these worlds capture key properties of typical real-world environments:

- **Relational**. The environment contains many objects and can be described in terms of the properties of objects and their relations (Baum, 2004). Actions of the agent manipulate the properties and relations. Learning can abstract from individual objects: what has been learned about one object generalizes to others.
- **Stochastic**. An agent such as a robot operating in residential homes or in the open world in general has to cope with uncertainty about the outcomes of its actions for multiple reasons:
  - The world itself may be stochastic.
  - The agent's perceptions and actions are noisy. Sensors are limited in what they can perceive and often not fully accurate. Likewise, the motor control of the agent's body is affected by control noise and mechanical failure.
  - The abstraction level of the agent's world description neglects many details. Models are approximations of the environment whose degree of exactness is often sufficient, but sometimes fails to capture relevant properties implying uncertainty in prediction.
  - Allowing for uncertainty is crucial for learning compact models using regularization. For instance, learned models ignore deliberately rare situations ("outliers"). (We discuss this in more detail later.)

In the following, we present a prototypical domain of stochastic relational worlds which will form our running example in this thesis and on which we evaluate many of our methods (we also evaluate on other benchmarks). This domain makes the ideas presented thus far more concrete and provides a more specific idea of the challenges which we investigate in this thesis.

1.1.1 Motivating Example: Robot Manipulation

Inspired by the chimpanzee Grande and her towers of boxes, our prototypical domain describes worlds in a simulated complex robot manipulation scenario where a robot manipulates cubes, balls and boxes scattered on a table (Fig. 1.2). This is an important scenario in robotics: "Competent pick and place operations may provide a sufficient functional basis for the manipulation requirements of a many of the targeted applications [of autonomous manipulating robots]" (Christensen, 2009, p. 56). We use realistic robotics kinematics, path planning methods and a 3D rigid-body dynamics simulator (ODE) that enables a realistic behavior of the manipulated objects[1]. For instance, piles of

[1] This simulator is available at `http://userpage.fu-berlin.de/tlang/DWSim/` and was written by Marc Toussaint.

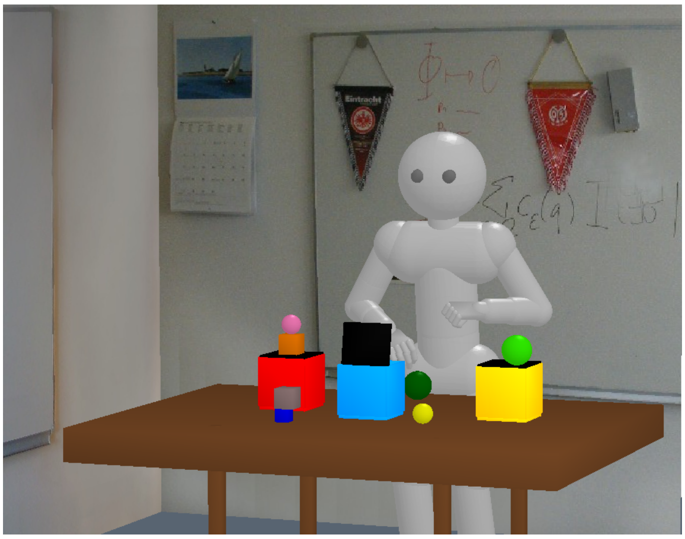


Figure 1.2: In our robot manipulation domain, a robot has to explore a 3D simulated desktop environment with cubes, balls and boxes of different sizes and colors to master various tasks.

objects may topple over or objects may even fall off the table (in which case they become out of reach for the robot). Depending on their type, objects show different characteristics. For example, it is almost impossible to successfully put an object on top of a ball, and building piles with small objects is more difficult. The robot can grab objects, try to put them on top of other objects, in a box or on the table. Boxes have a lid; special actions may open or close the lid; taking an object out of a box or putting it into it is possible only when the box is opened. The actions of the robot are affected by noise so that resulting object piles are not straight-aligned. We assume full observability of triples (s, a, s') that specify how the world changes to a new state s' when an action a was executed in a certain state s.

We take up the fundamental assumption from artificial intelligence and cognitive science that many aspects of higher intelligence can be achieved by reasoning on a symbolic level (Newell and Simon, 1976). We employ symbols to represent the structure and properties of states as well as to denote the actions of the agent. Hence, these symbols provide the mental language of the agent. In particular, we use relational logic symbols which permit to describe states in terms of objects and their relationships. We represent states with predicates $cube(X)$, $ball(X)$, $box(X)$, $table(X)$, $on(X, Y)$, $contains(X, Y)$, $out(X)$, $inhand(X)$, $upright(X)$, $closed(X)$, $clear(X) \equiv \forall Y.\neg on(Y, X)$, $inhandNil() \equiv \neg\exists X.inhand(X)$ and functions $size(X)$, $color(X)$, where the variables X and Y denote any world objects. These symbols are obtained by querying the state of the simulator and translating it according to simple hand-made guidelines, thereby sidestepping the difficult problem of converting the agent's observations into an internal representation. For instance, $on(a, b)$ holds if a and b exert friction forces on each other and a's z-coordinate is greater than the one of b, while their x- and y-coordinates are similar. The resulting space of possible world states, called the state space, is immense: if there are o objects and f different object sizes and colors in a world, the state space consists of $f^{2o}2^{2o^2+8o}$ different states (not excluding states one would classify as "impossible" given some intuition about real world physics). For instance, for $o = 10$ objects and $f = 3$ we get more than 10^{84} possible states. Clearly, methods which abstract over situations and objects are required for reasoning and acting in such worlds.

The actions of the robot are denoted by predicate symbols. These actions correspond to motor primitives whose effects we want to explore, learn and exploit. As mentioned above, the motor primitives are affected by noise. We define five different types of actions. The $grab(X)$ action triggers the robot to open its hand, move its hand next to X, let it grab X and raise the robot arm again. The execution of this action is not influenced by any further factors. For example, if a different object Y has been held in the hand before, it will fall down on either the table or a third object just below the robot hand; if there are objects on top of X, these are very likely to fall down. The $puton(X)$ action centers the robot's hand at a certain distance above X, opens it and raises the hand again. For instance, if there is an object Z on X, the object Y that was potentially inhand may end up on Z or Z might fall off X. The $openBox(X)$ and $closeBox(X)$ actions only apply to boxes. $openBox(X)$ triggers the robot to move its arm next to a box X and try to open its lid; this is only successful if there is no object on top of this box. If the box is already open, the position of the lid does not change. Similarly, $closeBox(X)$ triggers the robot to move its arm next to the lid of the box X and close it; if the box is already closed, the lid is not moved. The $doNothing()$ action triggers no movement of the robot's arm. The robot might choose this action if it is satisfied with the current state or thinks that any other action could be harmful with respect to its task. We emphasize again that actions always execute, regardless of the state of the world. Also, actions which are rather unintuitive for humans such as trying to grab the table or to put an object on top of itself are carried out. The robot has to learn by itself the effects of such motor primitives.

Our robot manipulation domain is related to the blocksworld scenario (Slaney and Thiébaux, 2001), perhaps the most popular—and, in the eyes of many students, infamous—scenario in classical artificial intelligence since the 1970s and an "almost ridiculously simplified proxy" (Pasula et al., 2007) of everyday-life environments. Many existing methods related to our work have been evaluated in the deterministic blocksworld or, over the last years, in extensions with stochastic action outcomes. Only very recently, realistic simulations of the blocksworld with intrinsic noise have been used in experiments (Pasula et al., 2007). Our robot manipulation scenario extends this simulated scenario by a manipulating articulated robot arm controlled by inverse kinematics and path planning and by multiple object types. While our domain is significantly more complex than previous testbeds, it is without doubt still a long way to full-fledged natural environments. In this thesis, however, we present a study where we investigate such a manipulation domain on a real robot platform.

Exploration, planning, and learning are challenging tasks in our robot manipulation domain due to its intrinsic noise and its complexity. In this thesis, we propose computational frameworks and algorithms for goal-directed behavior in such domains. Let us formalize the problem setting more precisely:

The agent is **given**:

- A **symbolic vocabulary** to describe **states**
  $on(X, Y)$, $inhand(X)$, $clear(X)$, $cube(X) \ldots$
- The ability to **convert** continuous perceptions into a symbolic representation
- **Symbolic actions** triggering motor primitives (always executed, effects depend on contexts)
  $grab(X)$, $puton(X)$, $openBox(X) \ldots$
- A (changing) **reward** function describing goal states and tasks

In contrast, the agent is **not** given and has to acquire **by itself**:

- **Policies and plans** specifying the actions to take to explore the environment, achieve tasks and get rewards
  "Which sequence of actions will lead to the goal?"
- A **model of the state transition dynamics** describing the effects of symbolic actions
  "How do the motor primitives triggered by symbolic actions change the world?"

This scenario has analogies to the decision-making of the chimpanzees from above: the chimpanzees seem to have symbols to refer to boxes and bananas and their relationships such as piles. From their experience, they must have learned the “transition model” for stacking and climbing boxes. Using this model for high-level reasoning, they derive appropriate plans to achieve their goals, namely to build towers of boxes to reach the banana.

Having illustrated the problem setting in a concrete scenario, we now turn to discussing goal-directed behavior in more general theoretical terms and setting our contributions in context.

1.2 Computational Principles for Intelligent Behavior

Research in artificial intelligence, robotics and cognitive science has laid many important foundations in our understanding of building intelligent systems and has led to impressive results. Nowadays, intelligent agents are on par with particularly intelligent humans in specialized tasks such as playing chess. They are hopelessly inferior to almost all humans, however, in deceivingly simple tasks of everyday-life, such as clearing a desktop, preparing a cup of tea or manipulating chess figures:

> *"The current state of the art in reasoning, planning, learning, perception, locomotion, and manipulation is so far removed from human-level abilities, that we cannot yet contemplate working in an actual domain of interest." (Pasula et al., 2007)*

For instance, high-level reasoning about common object manipulations of real-world robots is indeed a challenging task: we can choose from a large number of distinct actions with uncertain outcomes and the number of possible situations is basically unseizable. While the field of robotics has greatly advanced in learning of behavioral primitives and motor skills based on reward (Peters and Schaal, 2008), hardly any progress has been made to extract abstracting knowledge usable for goal-directed object manipulation. Abstract reasoning at the symbolic level is a bottleneck in current state-of-the-art robotics, and the ability to generalize from experiences is viewed as a key desired capability of robots: "It is largely perception and machine learning that distinguish a robot from an ordinary machine" (Christensen, 2009, p. 78).

So how can we tackle the challenges of autonomous goal-directed behavior? A promising approach is to learn a model of the environment from experience and exploit it for reasoning and acting. We describe this approach, and its alternative, next, before discussing how this translates to different aspects of intelligent behavior and stochastic relational worlds.

1.2.1 Reasoning with Models for Goal-Directed Behavior

The behavior of intelligent agents can be characterized with respect to how much active reasoning it involves. On the one hand, for many tasks agents maintain explicit knowledge which actions to take in a situation in form of direct mappings from states and sensory inputs to actions, for instance by associating task-dependent values with actions and choosing the most valuable actions. Finding appropriate actions is then simply a reaction to the current state and does not require reasoning about predicted action effects (Brooks, 1991). Such mappings can be genetically hard-wired (such as reflexes) or be acquired through extensive experience (such as habits or learned motor skills like peeling a banana).

Goal-directed (or deliberative) behavior with many changing tasks, however, cannot be achieved with habits or reflexes: an agent cannot maintain mappings for all potential tasks. Not only are the agent's memory resources limited, but also many tasks cannot be foreseen or the amount of experience with a task is insufficient to form habits. Instead, goal-directed behavior involves active reasoning: the agent needs to predict the effects of actions to plan for desirable situations. For this purpose, an agent requires a task-independent model of its environment and of the effects of its actions (Fikes and Nilsson, 1971). Such a model can be learned from the accumulated experience of the agent as the acquired knowledge is reusable and generalizes over tasks. The chimpanzees described above make the strong impression to have derived their solution of tower-building from reasoning with some model of their action effects. For instance, they might internally simulate action sequences and the resulting situations, or they might reason backwards from mental images of desired situations. Recent findings in neurobiology and cognitive science hint at the importance of internal simulation in intelligent behavior of animals (Hesslow, 2002; Grush, 2004). From a theoretical point of view, however, we require more rigorous computational frameworks and models of such processes. This thesis provides steps into this direction.

1.2.2 Aspects of Intelligent Behavior

Learning, planning (reasoning) and exploration are among the most interesting aspects of intelligent behavior in humans and animals and intrinsic to a model-based approach of goal-directed behavior. If in 100 years there are autonomous robots which have learned to manipulate objects in natural environments to achieve varying tasks, then these problems must have been solved. Let's take a closer look at each of these aspects.

Learning To understand how to manipulate its environment, an autonomous agent has to learn a model of the effects of its actions. For example, a pile of boxes is more stable if big boxes are placed at the bottom; it is a hard job to build a tower from balls; filling tea into a cup may lead to a dirty table cloth. Autonomous agents need to learn such models from their experience to adapt to new environments and not to rely on human hand-crafting.

Planning Once the agent can predict the effects of its actions by means of a model, the challenge is how to exploit this knowledge to reason and plan: how can the acquired model be used in reasonable time to find a sequence of actions suitable to achieve the current goal? This is a challenging task in natural environments where models are learned and thus imperfect and the outcomes of actions are uncertain. Furthermore, very many objects need to be considered so the number of potential actions in each timestep is huge and the search space of action sequences grows quickly with the length of sequences, namely exponentially in the number of actions.

Exploration In order to learn a model quickly, the agent needs to gather insightful experiences. It has to estimate the interestingness of actions and situations to decide where it will potentially learn the most useful things with respect to its tasks and goals. The generalization behavior of the learning system has strong implications on what is worth exploring. Exploiting generalization is especially important in the exploration of natural environments containing many objects. For instance, consider a hypothetical household robot which just needs to be taken out of the shipping box, turned on, and which then explores the environment to become able to attend its cleaning chores. Without a compact knowledge representation that supports abstraction and generalization of previous experiences to the current state and potential future states, it seems to be difficult—if not hopeless—for such a"robot-out-of-the-box" to explore one's home in reasonable time.

1.2.3 Intelligent Behavior in Stochastic Relational Worlds

The current state-of-the-art of methods in artificial intelligence and robotics is far from solving problems in many domains of interest (Pasula et al., 2007) such as natural environments. The recent years, however, have reassured and spurred the excitement and expectation of many researchers to come closer to the goal of building autonomous agents acting in relevant real-world scenarios. This hope is based on the advent of statistical relational learning techniques, resulting in the field of *statistical relational artificial intelligence*. These techniques combine two key benefits from different lines of research:

- They are probabilistic, deal with *uncertainty* on all levels of behavior and permit to learn models from experience.

- They make the structural assumption that the world is appropriately represented in terms of *objects and their properties and relationships*, namely by expressive relational representations using predicates and functions to capture object properties and relationships. The effects of actions are assumed to depend only on the attributes of objects and not on their identities. Such a prior assumption permits strong generalization which is required in the immense state spaces of relational domains.

Although methods from statistical relational artificial intelligence offer a great potential to approach goal-directed behavior, work into this direction is still relatively sparse. In this thesis, we introduce and discuss statistical relational methods for all three aspects of intelligent behavior mentioned above, learning, planning, and exploration, which we combine to enable goal-directed behavior of simulated and real-world robots.

1.3 Thesis Outline and Contributions

The organization of this thesis follows the mentioned aspects of goal-directed behavior:

- **Chapter 2: Basic Background**. Before diving into our frameworks and algorithms, we present the general basic background to readers unfamiliar with artificial intelligence and machine learning research on goal-directed behavior. We describe formal models for decision-making, in particular Markov decision processes, and several ways to represent states and actions; we discuss two prominent types of goal-directed behavior in artificial intelligence, namely planning and reinforcement learning; we show how the framework of graphical models can be used for probabilistic reasoning; and we discuss the limitations of classical methods for goal-directed behavior.

- **Chapter 3: Representing Relational Transition Models**. We investigate transition models which tell an agent how its actions change the state in stochastic relational worlds. We review the existing work on such models: starting from a general discussion of statistical relational learning, we describe the desired properties and established formalisms for relational transition models and the prerequisites for **learning** them from experience, before we discuss in detail an existing formalism based on probabilistic relational rules. Then, we introduce different graphical model formalisms to represent ground relational transition models.

Contributions:

- *We describe several novel ways to represent the **transition dynamics in ground relational domains by means of graphical models** and show how to build these from learned probabilistic relational rules.*
- *Furthermore, we set these rules in relation to the prominent description language PPDDL and provide more insights into learning such rules.*

- **Chapter 4: Planning in Ground Relational Domains**. We examine the problem of planning with a given relational transition model on the level of concrete objects in worlds with many objects and uncertain action outcomes. We present several planning algorithms based on different computational concepts and evaluate them extensively; we show how to cope with worlds containing a large number of objects; and we demonstrate some of our methods on a real-world robot.

  Contributions: *We introduce a variety of novel planning approaches for planning in ground relational domains:*

  - *We present **forward reasoning** approaches based on lookahead trees and our novel **PRADA** algorithm based on approximate inference in graphical models.*
  - *We extend the inference approach PRADA to backward reasoning enabling planning based on **bidirectional reasoning**.*
  - *We introduce the formal framework of **relevance grounding** for the idea of focusing on relevant objects in planning.*
  - *We are the first to frame **relational planning as a pure inference problem** and present a brief study in this direction.*
  - *We describe one of the first case studies on applying high-level relational planning on a **real-world robot**.*

- **Chapter 5: Relational Exploration**. We investigate how an intelligent agent can efficiently explore natural environments described by relational representations. We discuss why existing methods fail in these environments and how generalization exploiting relational abstraction can be used to steer exploration to gather insightful experiences. An extensive empirical evaluation completes our theoretical considerations.

  Contributions:

  - *We develop a **relational model-based reinforcement learning framework** generalizing over states, actions and objects.*

  - *We introduce the problem of* ***relational density estimation*** *in relational exploration strategies and present different methods to use it in a reinforcement learning context to formalize the novelty of relational states and actions.*
  - *We present our* ***relational exploration algorithm*** **REX**, *the first complete practical and implemented solution to exploration and model-based reinforcement learning in relational domains.*

- **Chapter 6: Conclusions**. We conclude this thesis with summarizing our main findings and giving some intuitions of what kinds of problems can be solved with our methods. Furthermore, we present major challenges for future work to achieve real-world goal-directed behavior.

1.4 Previously Published Work

The concepts, frameworks, algorithms and results presented in this thesis have partially been published in the following journal and conference papers:

Publications

T. Lang and M. Toussaint. Relevance grounding for planning in relational domains. In *Proc. of the European Conf. on Machine Learning (ECML)*, 2009a.

T. Lang and M. Toussaint. Approximate inference for planning in stochastic relational worlds. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2009b.

T. Lang and M. Toussaint. Probabilistic backward and forward reasoning in stochastic relational worlds. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2010a.

T. Lang and M. Toussaint. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research (JAIR)*, 39:1–49, 2010b.

T. Lang, M. Toussaint, and K. Kersting. Exploration in relational worlds. In *Proc. of the European Conf. on Machine Learning (ECML)*, 2010.

M. Toussaint, N. Plath, T. Lang, and N. Jetchev. Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

Submissions

T. Lang, M. Toussaint, and K. Kersting. Relational exploration. Submitted, 2011.

Chapter 2

Basic Background

In this chapter, we describe general background on goal-directed behavior of autonomous agents. This chapter is primarily targeted to readers not familiar with artificial intelligence and machine learning. We discuss the state-of-the-art work related to our proposed techniques in the subsequent chapters. First, we describe how an intelligent agent can represent its environment and the effects of its actions in formal models. In Sec. 2.2, we discuss types of goal-directed behavior which we investigate in this thesis. In Sec. 2.3, we explain different forms of graphical models and how they can be used for probabilistic reasoning. Finally, in Sec. 2.4 we discuss the limitations of classical work on goal-directed behavior.

2.1 Formal Models of the Environment

An intelligent agent needs an internal model of its environment to perform goal-directed behavior for varying tasks. A model is always only an approximation of the true underlying natural processes and needs to abstract the task-relevant structure from the natural world. The right level of abstraction is key for the agent's ability to reason and thus for its performance. For instance, a household robot cannot reason on the level of its sensory inputs and motor control to fulfill its cleaning chores: reasoning in the respective huge state spaces would be overly complex. Rather, it needs a symbolic description on an appropriate high level which describes the household structure in terms of the involved objects and how it can employ them for cleaning.

In AI research, the most prominent formal model of the interaction of an agent with its environment are Markov decision processes (Puterman, 1994). These allow arbitrary levels of abstraction and will be presented first. How to represent states and actions is a key question in formal models and will be discussed thereafter.

2.1.1 Markov Decision Processes

A Markov decision process (MDP) (Puterman, 1994; Boutilier et al., 1999) is a discrete-time stochastic control process used to model the interaction of an agent with its environment. At each time-step, the process is in one of a fixed set of discrete states $\mathcal{S}$ and the agent decides for an action from a set of actions $\mathcal{A}$. Each state $s \in \mathcal{S}$ is associated with an individual reward $\mathcal{R}(s)$. Rewards inform the agent which states it should achieve and which it has to avoid. Primarily for reasons of presentation, we do not consider the dependency of rewards $\mathcal{R}$ on actions; this assumption is not especially restrictive and our frameworks and algorithms can be augmented to deal with more general reward specifications taking both states and actions into account.

The key characteristic of MDPs is the Markov property: conditional on the present state and action, the future and past of the process are independent. Therefore, when the agent chooses an action in a state, the successor state depends only on the current state and the action and is independent of all previous states. The strong assumption of the Markov property allows to derive efficient inference methods. It needs to be justified by representing the underlying modeled process on the right level of abstraction.

More technically, the transition model $\mathcal{T}$ of an MDP specifies the transition distribution $P(s' \mid a, s)$ over successor states s' when executing an action a in a given state s. The agent receives rewards in states according to a function $\mathcal{R} : \mathcal{S} \to \mathbb{R}$. A (deterministic) policy $\pi : \mathcal{S} \to \mathcal{A}$ tells the agent which action to take in a given state. The objective of the agent is to maximize its collected rewards where future rewards are assumed to be of less worth than immediate rewards. This objective is formally expressed as the expected sum of discounted rewards

$$E[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0; \pi] \ .$$

for following the policy π starting in state s_0, where $0 < \gamma < 1$ is a discount factor and $r_t = \mathcal{R}(s_t)$ the reward obtained at time-step t. Often, we are only interested in the rewards until a fixed horizon d: $\sum_{t=0}^{d} \gamma^t r_t$. The value function $V : \mathcal{S} \to \mathbb{R}$ of a policy π is defined as the expected sum of discounted accumulated rewards when following π from a state s:

$$\begin{aligned} V^{\pi}(s) &= E[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s; \pi] \\ &= \mathcal{R}(s) + \gamma \sum_{s'} P(s' \mid \pi(s), s) V^{\pi}(s') \ . \end{aligned}$$

The optimal policy π^* which maximizes the value function for all states can be defined by the Bellman equation (Bellman, 1957) as

$$V^{\pi^*}(s) = \mathcal{R}(s) + \gamma \max_{a \in A} [\sum_{s'} P(s' \mid s, a) V^{\pi^*}(s')] \ .$$

Similarly, one can define the value $Q^{\pi}(s, a)$ of an action a in state s as the expected return after action a is taken in state s, using policy π to select all subsequent actions:

$$\begin{aligned} Q^{\pi}(s,a) &= E[\sum_{t=0}^{\infty} \gamma^t r_t \mid s_0 = s, a_0 = a; \pi] \\ &= \mathcal{R}(s) + \gamma \sum_{s'} P(s' \mid s,a) V^{\pi}(s') \ . \end{aligned}$$

The Q-values for the optimal policy π^* let us define the optimal action a^* and the optimal value of a state as

$$\begin{aligned} a^* &= \operatorname*{argmax}_{a \in A} Q^{\pi^*}(s,a) \quad \text{and} \\ V^{\pi^*}(s) &= \max_{a \in A} Q^{\pi^*}(s,a) \ . \end{aligned}$$

There exist many algorithms to compute the optimal policy in a given MDP, that is, where $\mathcal{T}$ and $\mathcal{R}$ are known, using for example dynamic programming techniques (Bellman, 1957; Howard, 1960). This is also called "solving" an MDP or *planning* in an MDP. Such algorithms depend strongly on the representation of states and actions which we discuss in the next subsection.

In general, and in most parts of this thesis, the agent does *not* know many details about the MDP model. In particular, the transition model $\mathcal{T}$ is unknown to the agent and needs to be learned from experience. Maximizing the expected rewards in an MDP with unknown transition model $\mathcal{T}$ is the problem of *reinforcement learning*. We discuss planning and reinforcement learning in Sec. 2.2.

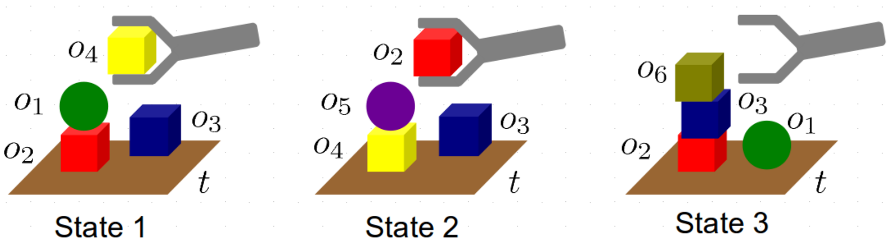
2.1.2 State and Action Representations

The states $\mathcal{S}$ in MDPs are descriptions of the agent's environment that capture information about the environment relevant to the problem at hand. How the agent represents these states and its actions $\mathcal{A}$ has important consequences on the conceptual and the algorithmic level of its behavior (Boutilier et al., 1999; Russell and Norvig, 2003). Three different representation types dominate AI research: *(i)* unstructured enumerated representations, *(ii)* factored propositional representations (Kearns and Koller, 1999; Guestrin et al., 2003) and *(iii)* relational representations (van Otterlo, 2009; Brachman and Levesque, 2004; McCarthy, 1963; de Raedt, 1997). Table 2.1 presents three states in a robot manipulation domain together with their translations to the respective representations.

The simplest representation of states and actions is the **enumerated**, or table-based or atomic or flat, representation. States and actions are represented by single distinct symbols. Hence, the state and action spaces $\mathcal{S}$ and $\mathcal{A}$ are simply enumerated lists or tables where states and actions respectively correspond to single entries. In Table 2.1, the three states are represented by symbols s_1, s_2 and s_3. This representation ignores the structure of states and does not provide a concept of objects. Therefore, it is impossible to express commonalities among states and actions. In the example, all three states appear equally different in this representation.

Table 2.1: Illustration of three world representation types in a robot manipulation domain

(a)



(b)

State	Enumerated	Factored	Relational
1	s_1	$on\_o_1\_o_2$, $on\_o_2\_t$, $on\_o_3\_t$, $inhand\_o_4$	$on(o_1, o_2)$, $on(o_2, t)$, $on(o_3, t)$, $inhand(o_4)$, $ball(o_1)$, $cube(o_2)$, $cube(o_3)$, $cube(o_4)$, $table(t)$
2	s_2	$on\_o_3\_t$, $on\_o_4\_t$, $on\_o_5\_o_4$, $inhand\_o_2$	$on(o_3, t)$, $on(o_4, t)$, $on(o_5, o_4)$, $inhand(o_2)$, $cube(o_2)$, $cube(o_3)$, $cube(o_4)$, $ball(o_5)$, $table(t)$
3	s_3	$on\_o_1\_t$, $on\_o_2\_t$, $on\_o_3\_o_2$, $on\_o_6\_o_3$	$on(o_1, t)$, $on(o_2, t)$, $on(o_3, o_2)$, $on(o_6, o_3)$, $cube(o_2)$, $cube(o_3)$, $cube(o_6)$, $ball(o_1)$, $table(t)$

In a **factored** propositional representation (Kearns and Koller, 1999; Guestrin et al., 2003), states are described by lists of state properties. Such properties represent, for instance, the battery status of a robot or whether a specific object is in the robot's hand. These properties are called state variables, state features or state attributes. By factoring states into individual state attributes, commonalities among states are captured. MDPs based on factored representations, called factored MDPs, have been investigated extensively in planning and reinforcement learning research. The disadvantage of factored representations is their lack of a notion of objects so that it is impossible to express commonalities among attributes. For instance, in Table 2.1 the attributes $on\_o_1\_o_2$ and $on\_o_5\_o_4$ are treated as fully distinct and therefore s_1 is perceived as equally different from s_2 as from s_3. Similar arguments hold for actions which are also represented by distinct symbols; for instance, formalizing the experience of grabbing object o_3 in s_1 in a factored representation might provide information about grabbing o_3 in s_2, but cannot provide information about grabbing any other object.

Relational representations account for state structure and objects explicitly (van Otterlo, 2009; Brachman and Levesque, 2004; McCarthy, 1963; de Raedt, 1997). The state space $\mathcal{S}$ is described by means of a relational vocabulary consisting of predicates $\mathcal{P}$ and functions $\mathcal{F}$ which describe the relationships and properties that can hold for domain objects $\mathcal{O}$. This is considered to be a key requirement for intelligent agents:

> *"It is hard to imagine a truly intelligent agent that does not conceive of the world in terms of objects and their properties and relations to other objects." Kaelbling et al. (2001)*

We distinguish between primitive and derived concepts (predicates and functions). The latter are defined in terms of formulas over primitive or other derived concepts. Let us briefly clarify more logic vocabulary: An atom is simply a predicate, or a function, applied to a tuple of terms. In this thesis, terms, which we also call arguments, are either logical variables or constants. Constants are also referred to as objects. If all of the terms in an atom are variables, then the atom is called an abstract atom. If all terms are constants, this is a ground atom. Thus, $\mathcal{P}$ and $\mathcal{F}$ yield the set of ground atoms with arguments taken from the set of domain objects $\mathcal{O}$. We will speak of grounding an abstract formula ψ if we apply a substitution σ that maps all of the variables appearing in ψ to objects in $\mathcal{O}$. A state is defined by a conjunction of true ground atoms. The action space $\mathcal{A}$ is defined by atoms from a special set of predicates $\mathcal{P}_{\mathcal{A}} \subset \mathcal{P}$ with arguments from $\mathcal{O}$. In MDPs based on relational representations, called relational MDPs, the commonalities of states, actions and objects can be expressed. Hence, we can generalize information about one object to reach conclusions about other, related objects. Using a relational representation state descriptions and models may be compact since we can abstract from concrete objects and situations by means of abstract atoms containing logical variables. In Table 2.1, abstract atoms let us capture the equivalence of s_1 and s_2 in contrast to the difference of s_1 and s_3: we can generalize s_1 and s_2 to the (semi-) abstract state $on(A, B), on(B, table), on(o_3, table), inhand(C)$, which cannot be done for s_3. Similar arguments hold for actions: for instance, the experience of grabbing o_1 in s_1 provides information about grabbing o_5 in s_2 when using a relational representation.

The choice of representation determines the expressivity and compactness of internal world models of an agent and thus strongly affects its learning, reasoning, generalization and exploration capabilities. We will discuss this in detail in Chapter 3 where we focus on the transition model $\mathcal{T}$ describing the effects of the agent's actions. Concepts and algorithms for goal-directed behavior based on expressive relational transition models are then the focus of Chapters 4 and 5. Which goal-directed behavior we will investigate is described next.

2.2 Goal-Directed Behavior

Goal-directed behavior (Botvinick and An, 2009; Niv et al., 2006; Tomasello et al., 2005) denotes the activity of an agent which attempts to achieve tasks and situations which satisfy its preferences and desires. Typically, goals and tasks change over time (Oudeyer et al., 2007; Dignum and Conte, 1997); if this is the case, an agent that is to behave in a goal-directed manner requires a model of its environment telling the agent how its actions change the environment (Fikes and Nilsson, 1971; Kaelbling et al., 1996; Boutilier et al., 1999). (Without a model, the agent would need to collect very many experiences to learn a reactive behavior for each new task.) In terms of MDPs, such a model corresponds to the transition model $\mathcal{T}$, while the goals and tasks are specified in the reward function $\mathcal{R}$ which in contrast to $\mathcal{T}$ is subject to frequent change. In this thesis, we assume $\mathcal{R}$ is always given to the agent and we focus on transition models $\mathcal{T}$. One should bear in mind that $\mathcal{T}$ is never correct in the sense of describing the natural world dynamics with full accuracy—instead, $\mathcal{T}$ is a model on an appropriate abstraction level and hence only approximates the true dynamics. Let's assume, however, that here $\mathcal{T}$ is the most accurate model for the given level of symbolic abstraction.

Usually, the agent which starts to behave in a goal-directed way does not know the "correct" model $\mathcal{T}$, but maintains an estimate $\mathcal{M}$ of $\mathcal{T}$. In this thesis, we will always refer to this estimate $\mathcal{M}$ when speaking of the agent's model; nonetheless, $\mathcal{M}$ may well correspond to the "correct" model $\mathcal{T}$.

The problem of how to use a (temporarily) fixed model $\mathcal{M}$ for reasoning and sequential decision-making is studied in the field of planning which we describe first in the following. Thus, planning is a computational process without interaction with the environment. In contrast, how to achieve high-reward states when starting to act without a confident estimate $\mathcal{M}$ is studied in the field of reinforcement learning (RL) which we discuss thereafter. In (model-based) RL, the agent has to learn $\mathcal{M}$ from its interaction with the environment and exploit it to achieve high rewards. Thus, a major focus in RL is how an agent can quickly gain an understanding of its environment, and planning is a computational subtask in RL.

2.2.1 Planning

Planning (Weld, 1999; Boutilier et al., 1999; Fikes and Nilsson, 1971; Helmert, 2003; Nau, 2007) is the problem of using a transition model $\mathcal{M}$ to compute an action sequence which leads to high reward states according to a reward function $\mathcal{R}$. Different planning scenarios can be distinguished along the following characteristics:

- *Certainty of action outcomes.* $\mathcal{M}$ may specify either deterministic or indeterministic action outcomes. In the latter case, these may be quantified with probabilities.

- *Type of reward function.* We distinguish two types of reward functions $\mathcal{R}$ based on the number of states which achieve reward and the variability of the assigned rewards.

  A *goal-oriented*, or task-oriented, reward function specifies non-zero rewards for a set of absorbing states where a clear-cut goal is fulfilled and the problem terminates. The cost-to-goal formalization, where we want to minimize the costs of steps to the goal, is symmetric and also falls under this category. In a factored propositional or relational representation, such a goal can expressed by means of a logical formula ϕ. Typically, this formula is a partial state description so that there exists more than one state where ϕ holds. For example, the goal might be to put all our romance books on a specific shelf, no matter where the remaining books are lying. In this case, planning involves finding a sequence of actions $\mathbf{a}$ such that executing $\mathbf{a}$ will result in a world state s with $s \models \phi$.

  Alternatively, the reward function may specify multiple, possibly competing objectives in the form of general preferences over states, called "utilities", by assigning varying rewards to different states so that the utility of desirable states is high, while it is low for unfavorable states. Such a *utility-based*, or process-oriented, reward function can be used to model an ongoing optimization process where the task of the agent is to continuously accumulate reward over an infinite horizon. For instance, our goal might be to keep the desktop tidy. Planning with utility-based rewards corresponds to the decision-theoretic paradigm which is also studied in operations research, control theory and economics.

Many real-world problems exhibit both task- and process-oriented behavior (Boutilier et al., 1999).

- *Number of start states.* We may either compute policies (or likewise, value functions) over the complete state space or we may focus on planning from a subset of distinguished start states. The latter problem is simpler as it requires examining only a small local subset of the complete state space. While algorithms computing full policies also solve the problem of planning from a single start state, they will be inefficient compared to specialized planners which can exploit knowledge of the start state to avoid searching through all states and to find local high-value plans.
- *Observability of states.* We may either have full access to the current state or we can only partially, or not at all, observe it. In this thesis, we assume that the state is always fully observable.

Given full observability of states, the framework of MDPs can accommodate all planning scenarios by appropriately choosing the transition model $\mathcal{M}$ and the reward function $\mathcal{R}$. Hence, MDPs generalize many of the planning paradigms found in the literature and have been adopted as a standard model for decision-theoretic planning with fully observable states in the field of AI (Boutilier et al., 1999; Bertsekas and Tsitsiklis, 1996). Although MDPs provide a general model, the concrete algorithms for planning depend strongly on the characteristics of the planning scenario. Likewise, the representation of states and actions and in turn of the transition model $\mathcal{M}$ and the reward function $\mathcal{R}$ have major consequences for planning algorithms. We will discuss this in detail in Chapter 3. We note here already that the potential compactness of $\mathcal{M}$ and $\mathcal{R}$ does not carry over to planning. In the following, we discuss prominent planning scenarios reflected in different lines of research.

In **classical planning** studied in the AI planning community, the objective is to find a sequence of deterministic actions that will lead from the initial state to a goal state (thus, the reward function is goal-oriented). Shorter plans might be favored, or more generally we might associate actions with costs which we want to minimize. Classical planning algorithms (Weld, 1999) exploit knowledge of the start state to avoid searching through all states.

Planning under uncertainty extends classical planning with probabilistic action outcomes and is inherently harder than its deterministic counterpart (Kushmerick et al., 1995; Littman, 1997; Littman et al., 1997). (We neglect here planning under uncertainty where the action outcomes are non-deterministic, but not probabilistically determined, which is for instance studied in conditional planning and in conformant planning (Russell and Norvig, 2003).) Achieving a goal state with certainty is typically unrealistic. The task is then to find a "satisficing" action sequence that will lead with high probability to states with large rewards. A further source of uncertainty next to stochastic action outcomes which we have ignored thus far is the uncertainty about the starting state. While we ignore it in the following and always assume deterministic initial states, planners based on probabilistic inference in graphical models, such as our approaches proposed in Chapter 4, incorporate this uncertainty in a straightforward manner. Research on classical deterministic and probabilistic planning has employed relational representations to a large extent as we discuss in more detail in Chapter 3.

In **decision-theoretic planning** (Boutilier et al., 1999) (or sequential decision-making under uncertainty), studied mostly in the machine learning and reinforcement learning communities, the task is to find sequences of stochastic action with (near-) optimal expected utility. Prominent "MDP solution algorithms" (Bertsekas and Tsitsiklis, 1996) compute policies (or value functions) over complete state spaces and are usually designed for enumerated representations. For instance, exact solution algorithms include linear programming and dynamic programming methods such as value iteration (Bellman, 1957) and policy iteration (Howard, 1960), while approximate value iteration and linear value function approximations are examples of approximate algorithms. In Chapter 4, we introduce different approaches for decision-theoretic planning from given start states in stochastic relational domains.

2.2.2 Reinforcement Learning

Reinforcement learning (RL) (Sutton and Barto, 1998; Kaelbling et al., 1996) is a learning paradigm where an agent needs to learn from its experience which actions to take in an unknown environment so as to accumulate high rewards. In contrast to supervised learning, an RL agent is never provided the correct outputs for the inputs (that is, the optimal actions for the given states) and its sub-optimal actions are not explicitly corrected. Instead, the agent needs to distribute received rewards, "reinforcements", over previous states and actions so as to understand which actions are advantageous where. For instance, a cook apprentice might be praised by its guests for cooking a delicious meal and then has to decide which actions, such as adding which spices, had significant impact on this reinforcement. Inspired by behaviorist psychology, reinforcement learning is a key scenario in AI: it needs to be solved by any autonomous agent who wants to continuously extend its ability and horizon to predict, control and manipulate its environment.

We formalize the problem of reinforcement learning in an MDP framework (Werbos, 1977). Then, the notion of an "unknown environment" implies that the correct transition model $\mathcal{T}$ is unknown and an estimate $\mathcal{M}$ needs to be (at least, implicitly) learned from interaction with the environment. In alternative formulations, the reward function $\mathcal{R}$ is also unknown, but in this thesis we always assume $\mathcal{R}$ be given.

Model-free RL approaches such as the prominent Q-learning algorithm (Watkins and Dayan, 1992) learn values of states and actions directly from the interaction with the environment (Brooks, 1991). Approaches for enumerated representations collect statistics for each individual state-action pair. In contrast, structured representations offer the potential to generalize experiences to yet unseen states and actions: this can be achieved by value function approximators which take sets of state features (and an action) as input and predict a value. The training instances for this supervised learning problem are provided by the experienced states in combination with the distributed collected rewards (reinforcements) of the agent. Using model-free approaches, an agent can learn to behave in a goal-directed way for a fixed reward function (which determines the specific task), but not in situations where the reward function of the agent changes quickly, for instance in case of varying tasks.

In contrast, **model-based** RL approaches learn a transition model $\mathcal{M}$ from the experienced state transitions in a supervised setting and then use $\mathcal{M}$ to distribute the reinforcements of high-reward states appropriately across the state space (Kaelbling et al., 1996; Russell and Norvig, 2003). This is achieved by calculating value functions of states and actions using planning algorithms (see Sec. 2.2.1). As the same $\mathcal{M}$ can be used for decision-theoretic planning with changing reward functions, model-based RL is a promising approach for goal-directed behavior with varying tasks. In this thesis we focus on model-based RL. The representational and structural choices for $\mathcal{M}$ have key effects on the performance of the agent. For instance, if $\mathcal{M}$ is based on an enumerated representation, reinforcements can only be distributed over already experienced states, while structured representations generalize over states and thereby can distribute rewards over yet unseen states. Transition models $\mathcal{M}$ and their implications on planning and reinforcement learning will be the focus of Chapter 3.

A central challenge of a reinforcement learning agent is to ensure that it explores its environment sufficiently to accurately understand the domain and to be able to plan for high-value states. At the same time, the agent has to ensure to exploit its learned knowledge and not to spend too much time in low-value parts of the state space. This is called the **exploration-exploitation tradeoff** (Sutton and Barto, 1998). For instance, consider a household robot which has found out how to wash dishes in the sink to achieve a reward for clean dishes. If it continues exploring, it may eventually find out that this can also be achieved by using the dish-washer which leaves it with the time to fulfill other tasks such as repairing the car, enabling to collect even more reward. Different concepts have been developed for describing explorative behavior, including curiosity (Schmidhuber, 1991a,b), counters on the occurrences of states and actions (Thrun, 1992), and assuming high rewards for unknown states and actions ("optimism in the face of uncertainty") (Szita and Lörincz, 2008). The frameworks of E^3 (Kearns and Singh, 2002), R^{max} (Brafman and Tennenholtz, 2002) and Bayesian Reinforcement Learning (Poupart et al., 2006) provide a rather explicit theoretical understanding of exploration and exploitation. For instance, E^3, which we will describe in detail in Chapter 5, has primarily been proposed to enable a proof that the exploration-exploitation problem can be solved in time which is polynomial in the number of states. While these methods provide clear theoretical insights, they have been formulated on enumerated representations and hence are not applicable to natural environments. In our view, the key challenge is thus not to devise new exploration-exploitation theories, but to realize the existing principles on non-trivial representations. A central aspect is to investigate how generalization and abstraction in structured representations interferes with these principles. We will address this in detail in the upcoming chapters, in particular in Chapter 5.

2.3 Probabilistic Reasoning in Graphical Models

Graphical models represent knowledge in uncertain domains by networks of coupled information units and provide a computational framework for reasoning under uncertainty (Pearl, 1988; Jordan, 1999). Graphs visualize compact factorized representations of joint distributions exploiting their conditional independences. From a conceptual point of view, inference in graphical models can be thought of as a theoretically grounded formalization of the idea of internal simulation. On a purely information processing level, several authors proposed that certain functions of neural substrates could be abstracted in terms of Bayesian information processing and inference (Doya et al., 2007; Ott and Stoop, 2007; Tenenbaum et al., 2011). In the following, we first describe different types of graphical models, before we discuss how inference in such models can be used for probabilistic reasoning and planning.

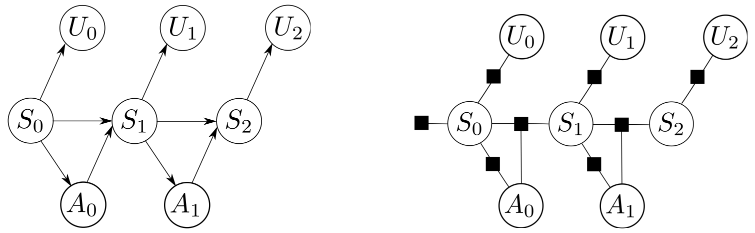
2.3.1 Graphical Models

Let us first clarify notation. We denote random variables by upper case letters (e.g., S), their values by the corresponding lower case letters ($s \in dom(S)$), variable sets by bold upper case letters ($\mathbf{S} = \{S_1, S_2, S_3\}$) and value sets by bold lower case letters (s $= \{s_1, s_2, s_3\}$). We also use column notation ($\mathbf{s}_{2:4} = \{s_2, s_3, s_4\}$).

Formally, a graphical model represents the joint probability distribution over a set of random variables $\mathbf{X}$ by means of a graph $\mathcal{G}$ which encodes how the joint factorizes over these variables. The nodes in $\mathcal{G}$ represent the random variables, while the edges define their dependencies and thereby express conditional independence assumptions among $\mathbf{X}$. Inference in graphical models is the process of computing a posterior marginal $P(\mathbf{H} \mid \mathbf{O})$ over the hidden variables $\mathbf{H} = \mathbf{X} \backslash \mathbf{O}$ given a set of observed variables $\mathbf{O}$. Two types of graphical models can be distinguished according to their underlying graph structure: in Bayesian networks $\mathcal{G}$ is a directed acyclic graph, while in factor graphs $\mathcal{G}$ is undirected. We discuss both in turn.

A **Bayesian network** (BN) (Jensen, 1996) uses a directed acyclic graph $\mathcal{G}$ to encode a compact representation of the joint probability distribution over random variables $\mathbf{X}$. The value x of a variable $X \in \mathbf{X}$ depends only on the values of its immediate ancestors in $\mathcal{G}$, which are called the parents $\mathrm{Pa}(X)$ of X. Conditional probability functions at each node define $P(X \mid \mathrm{Pa}(X))$. In case of discrete variables, they usually take the form of conditional probability tables. A BN is a compact representation of a distribution over $\mathbf{X}$ if all nodes have only few parents or their conditional probability functions have significant local structure. This will play a crucial role in our development of relational planning algorithms in Chapter 4.

An alternative graphical representation are **factor graphs** which represent the factorization of a probability distribution explicitly. Factor graphs are bi-partite graphs connecting variable nodes with factor nodes. Factors represent general couplings between variables. Mathematically, a factor graph is given by random variables $\mathbf{X} = \{X_1, \ldots, X_n\}$, a set of cliques $\mathcal{C} = \{C_1, \ldots, C_k\}$ which are tuples of variables and for



(a)

(b)

Figure 2.1: *Graphical models for representing a two-step MDP model in an enumerated state and action space.* The couplings between states S_t and actions A_t encode the policy. (a) A dynamic Bayesian network represents dependencies by means of arcs in a directed graph. (b) A factor graph represents the couplings of variables explicitly by factor nodes shown here as black squares.

each clique a factor ψ_i such that

$$P(X_1, \ldots, X_n) = \frac{1}{Z} \prod_{i=1}^{k} \psi_i(\mathbf{X}_{C_i}) \qquad \text{with} \quad Z = \sum_{\mathbf{x}} \prod_{i=1}^{k} \psi_i(\mathbf{X}_{C_i}) \; .$$

Though closely related, conditional independence and factorization are not the same concepts: for instance, a complete graph of three variables does not encode conditional independences, while the underlying distribution could be factored in three binary factors or one tertiary factor. Bayesian networks and a further type of undirected graphical models called **Markov random fields** (Kindermann and Snell, 1980) can be converted to factor graphs.

Graphical models can represent the evolution of stochastic systems over time by specifying the dependencies of variables across multiple time-steps. About the effects of actions can be reasoned by introducing random variables for actions. We will use both factor graphs as well as dynamic Bayesian networks (DBNs) for reasoning about actions over time. Fig. 2.1 shows a DBN and a factor graph for a two-step unstructured MDP. In the following, we describe DBNs in more detail; similar ideas apply to dynamic factor graphs. A DBN (Murphy, 2002) extends the BN formalism to model a dynamic system evolving over time. Usually, the focus is on discrete-time stochastic processes. The underlying system itself (in our case, a world state) is represented by a BN B, and the DBN maintains a copy of this BN for every time-step. A DBN can be defined as a pair of BNs $(B_0, B_\rightarrow)$, where B_0 is a (deterministic or uncertain) prior which defines the state of the system at the initial state $t=0$, and $B_\rightarrow$ is a two-slice BN which defines the dependencies between two successive time-steps t and $t+1$. The resulting DBN satisfies the Markov property: the variables at time $t+1$ depend only on other variables at time $t+1$ and on variables at t. In the DBN for a two-step MDP model using an enumerated representation shown in Fig. 2.1, the state at time-step t is a discrete random variable S_t whose values correspond to the state space of the MDP. Similarly, the discrete random variable A_t represents the action at time-step t and takes values according to the action space. The reward at time t is represented by the utility node U_t. It is possible to express arbitrary reward expectations $P(U_t \mid S_t)$ with binary U (Cooper, 1988). The transition model $\mathcal{T}$ of the MDP defines the probabilities $P(s_{t+1} \mid s_t, a_t)$ and the reward function $\mathcal{R}$ the probabilities $P(u_t \mid s_t)$. In addition, a distribution $P(s_0)$ for the initial state has to be provided. The policy π is represented by $P(a_t \mid s_t)$.

2.3.2 Inference for Planning

Inference in dynamic graphical models can be applied to the problem of planning in a stochastic domain in various ways. Let $\mathbf{H}_t$ denote the unobserved hidden variables and $\mathbf{O}_t$ the observed variables at time-step t such that $\mathbf{H}_t \cap \mathbf{O}_t = \emptyset$ and $\mathbf{S}_t = \mathbf{H}_t \cup \mathbf{O}_t$ where $\mathbf{S}_t$ are the variables representing time-step t. Inference tasks for reasoning include: inferring the hidden states, that is calculating the posterior marginals $P(\mathbf{h}_t \mid \mathbf{o}_{0:T})$; filtering, that is calculating the posterior $P(\mathbf{h}_t \mid \mathbf{o}_{0:t})$; calculating the likelihood of the observations, $P(\mathbf{o}_{0:T})$; and finding the most-likely sequence $\mathbf{h}^*_{0:T}$ maximizing $P(\mathbf{h}_{0:T} \mid \mathbf{o}_{0:T})$. In particular, filtering will be at the heart of one of our relational planning techniques proposed in Chapter 4.

The ultimate goal is to use inference to compute optimal policies in MDPs. This problem has been termed as "planning by/as inference". Toussaint and Storkey (2006) have recently shown that planning in MDPs can be reformulated as a problem of probabilistic inference in a DBN (see also Botvinick and An (2009); Hoffman et al. (2009)). This reformulation is based on several ideas: identifying the policy as a parameter of the DBN, namely as the conditional probability $P(a \mid s)$ coupling states to actions; using a mixture model to cope with different horizons; and computing posteriors over action variables conditioning on rewards. An expectation-maximization (EM) algorithm (Dempster et al., 1977) which employs inference in an internal loop can be used to optimize the parameters $P(a \mid s)$ such that the likelihood of the conditioned rewards is maximized, thereby computing the optimal policy. Intuitively, in this framework rewards correspond to "mental observations" of future events. Inferring posteriors over hidden action variables and optimizing the DBN parameters amounts to finding actions and a policy which is "coherent" with these "observations" (Toussaint, 2009b).

The mentioned EM algorithm (Toussaint and Storkey, 2006) solves planning in unstructured MDPs with enumerated state and action spaces. The corresponding DBN represents a state simply by a single variable S_t. What makes planning-by-inference promising is that there exists a variety of inference techniques which can be applied on more structured representations instead. For instance, variables can be discrete, continuous or mixed and may represent different features of states such as motor commands, future actions and constraints, even in hierarchies (Toussaint, 2009b). Planning corresponds to conditioning some of these variables to desired values (goals) and others to known state or sensor information and then to compute coherent estimates of the remaining variables (in particular actions) across the network. As random variables can represent states on arbitrary abstraction levels, in principle inference in graphical models is applicable from low-level continuous motor control to symbolic decision-theoretic planning. While there are already appealing successful applications, for instance to hierarchical controllers in partially observable MDPs (Toussaint et al., 2008), thus far applying inference for planning in symbolic relational representations has not been investigated. Standard inference techniques are hardly applicable in such scenarios. We will examine this challenge in detail in Chapters 3 and 4.

2.4 Discussion

Goal-directed behavior has been a major topic in the study of intelligent systems and agents from the first days (McCarthy, 1963; Fikes and Nilsson, 1971). Classical work in artificial intelligence (AI) and reinforcement learning (RL) laid important foundations in our understanding of goal-directed behavior. Both lines of classical research, however, face limitations. Interestingly, these are complementary: while classical AI methods employ expressive representations, they cannot cope with uncertainty and cannot adapt behavior from experience; while classical RL methods investigate learning to act from experience in domains with uncertain action outcomes, they focus on trivial non-generalizing state and action representations. We review this in more detail in the following.

Limitations of Classical Artificial Intelligence

The idea of using rich symbolic representations to model the world and the effects of the agent's actions has been pursued in artificial intelligence research from its beginnings (McCarthy, 1963). Various relational representations have been investigated, often in very general forms employing full (and undecidable) first-order logic. The "physical symbol systems hypothesis" (Newell and Simon, 1976) is central to classical AI approaches (also known as "good old-fashioned AI") which states that many aspects of higher-level intelligence can be achieved by the manipulation of symbols. The assumption that human problem solving consists primarily of such high-level symbol manipulation also laid the grounds for the field of cognitive science. Classical AI achieved great success at simulating high-level reasoning in small demonstration programs, in classical planning in deterministic domains and led to successful applications such as expert systems.

A major drawback of classical AI methods is that they typically rely on hand-crafted, laborious models of the world. Therefore, such methods often solve problems only in severely restricted problem domains. For instance, early work on relational action languages used hard-coded representations for representing deterministic domain dynamics (Fikes and Nilsson, 1971). Learning and coping with uncertainty, however, are key requirements for intelligent agents. Specifying in advance models by hand for all possible situations the agent may encounter is surely infeasible (Pasula et al., 2007). Besides, humans have difficulties to provide accurate models of complex domains, both on the structural as well as the parametrical level. Also, transition dynamics may change making existing models obsolete, so that agents have to adapt to such changes. Furthermore, the world by itself may be stochastic: there may be random events or hidden (latent) variables which cannot be accessed.

Hence, what is required to make classical AI approaches more applicable to real-world scenarios are generic tools to express uncertain information, to cope with lack of knowledge, to reason under uncertainty and to learn to act from experience. Indeed, uncertainty has become a major concern in AI research over the last two decades. Nonetheless, learning for goal-directed behavior in stochastic domains is still a largely unresolved challenge.

Limitations of Classical Reinforcement Learning

The field of reinforcement learning (RL) investigates the learning task of an autonomous agent which has to find a policy for action selection that maximizes its reward over the long run (Sec. 2.2.2). Classical (model-free and model-based) RL methods (Sutton and Barto, 1998) have laid important foundations for understanding and formalizing exploration, learning and planned behavior. In contrast to classical approaches in AI, these methods have often been designed to cope explicitly with uncertainty in action outcomes and observations. However, while research in AI from the early days on has investigated powerful generalizing relational representations, surprisingly, most classical RL methods instead rely on propositional, that is enumerated or factored, representations. The focus on learning and uncertainty and the neglect of expressive representations may be due to the historical development of RL which evolved as a subfield of machine learning rather than from AI research. Hence, the shortcomings of propositional representations (Sec. 2.1.2) transfer to classical RL methods, namely the difficulty to learn strongly generalizing knowledge and to be applicable to domains of real-world goal-directed behavior.

To overcome the limitations of classical RL, expressive representations of states and actions are required. The field of relational reinforcement learning (Džeroski et al., 2001; van Otterlo, 2009), which has evolved over the last decade, pursues this by means of relational representations, but has mostly focused on model-free approaches which are less appropriate for goal-directed behavior. How relational representations can be used for learning expressive transition models and acting in a goal-directed manner is a major focus of this thesis and will be discussed in more detail in the next chapters.

Chapter 3

Representing Relational Transition Models

Models of the transition dynamics of its environment enable an intelligent agent to perform goal-directed behavior. Transition models inform an agent how its actions modify the state of the world by encoding the distribution $P(s' \mid s, a)$ over successor states s' when the agent performs action a in state s. In this chapter, we investigate such models using relational representations to account for the transition dynamics in stochastic worlds with many objects. In Sec. 3.1, we describe the related work on relational transition models. Thereafter, in Sec. 3.2 we introduce graphical models to represent the transition dynamics of ground relational domains on which a wide variety of computational methods can be applied for inference. Finally, we discuss briefly the most important aspects of relational models with respect to the following chapters.

Our **contributions** in this chapter are the following:

- We describe several novel ways to represent the **transition dynamics in ground relational domains by means of graphical models** and show how to build these from learned probabilistic relational rules (Sec. 3.2) (Lang and Toussaint, 2009b, 2010b).
- Furthermore, we set these rules in relation to the prominent action description language PPDDL and provide more insights into learning such rules (Sec. 3.1.3 and Appendix B) (Lang and Toussaint, 2010b).

3.1 Related Work on Relational Transition Models

Our methods introduced in this thesis use relational transition models for goal-directed behavior. In the following, first we review briefly general work in the field of statistical relational learning which forms the background of research on relational transition models. Then, we describe various aspects of probabilistic relational transition models, including their required properties, the prerequisites to learn them, existing model types, and how they enable goal-directed behavior. Finally, we inspect a specific transition model type, probabilistic relational rules, in detail, discuss how to learn rules from experience and place this model type in context to the prominent PPDDL representation.

3.1.1 Statistical Relational Learning

The field of statistical relational learning (SRL) (Getoor and Taskar, 2007; de Raedt et al., 2008; de Raedt, 2008) combines statistical machine learning techniques with structured relational representations to learn generalizing, compact models from experience. For instance, an autonomous agent can then adapt to unknown stochastic relational environments and perform goal-directed behavior by learning symbolic transition models from experience. Here, we give a brief overview over the general field of statistical relational learning.

A variety of specific models with the corresponding computational methods for inference and learning has been developed. Almost all of them ignore dynamic aspects of natural environments and focus on static domains. For instance, Muggleton (1996) and Cussens (1999) have extended stochastic grammars to stochastic logic programs, while Taskar et al. (2002) have extended Markov networks towards relational Markov networks and Richardson and Domingos (2006) towards Markov logic networks (MLNs.) MLNs, for instance, combine logic representations with formalisms of uncertainty by assigning weights to logic formulas; these weights determine the degree to which a formula typically holds and in turn the probability of "worlds" (in the sense of world states) by combining the weights over all ground formulas. Likewise, dependency networks have been extended towards relational dependency networks (Neville and Jensen, 2007) and Bayesian networks to logical and relational structures, including probabilistic logic programs (Ngo and Haddawy, 1997), relational Bayesian networks (Jaeger, 1997), probabilistic relational models (Pfeffer, 2000; Getoor et al., 2001), Bayesian logic programs (Kersting and de Raedt, 2008), and Bayesian logic (Milch et al., 2005).

While the parameters of these models can often be estimated efficiently using expectation -maximization (EM) (Dempster et al., 1977) or gradient descent, structure learning is a particularly complex task: the number of potential structures is infinite; as learning the structure of (propositional) Bayesian networks is NP-complete, learning relational structures must be at least as hard. Many structure learning approaches take up the idea of traditional inductive logic programming methods (de Raedt, 2008) to greedily improve the current best model by syntactic manipulations with refinement operators and to choose a model according to some penalized likelihood scoring metric. Structure learning has been investigated in particular in the context of MLNs (Mihalkova and Mooney, 2007; Kok and Domingos, 2009, 2010) and relational dependency networks (Neville and Jensen, 2007; Natarajan et al., 2010). For autonomous goal-directed behavior we require a solution to the structure learning problem in dynamic domains, as we want to learn a probabilistic relational transition model. Fortunately, one of the few solutions to structure learning based on a heuristic algorithm (Pasula et al., 2007) applies to learning transition models. We will discuss it in depth below.

Apart from the problem of structure learning, how in general to model dynamic domains within the prominent SRL approaches mentioned above and how to perform the required inference then is not clear. While it is often argued that time could be modeled as an additional variable in relations and any known inference algorithm developed for the static cases could then be employed, successful applications of this idea are not widely known. In contrast to many static domains, in dynamic domains we need approximations even for sparse models as state attributes easily become correlated over time as they share the same influences in the past. Deriving efficient inference methods for specific statistical relational models and using them for planning will be the focus of Chapter 4.

3.1.2 Probabilistic Relational Transition Models

A model $\mathcal{M}$ of the transition dynamics specifies $P(s' \mid s, a)$, the probability of a successor state s' if action a is performed in state s (Boutilier et al., 1999). In this thesis, this distribution is usually non-deterministic. A model $\mathcal{M}$ has to satisfy the following requirements:

- *Expressivity*: $\mathcal{M}$ shall cover large parts of the state space $\mathcal{S}$ and the action space $\mathcal{A}$. Typically, these spaces are very large in relational domains, so that a model $\mathcal{M}$ needs to generalize strongly to cover significant parts of S and A.

- *Inference*: The point of a model $\mathcal{M}$ is to enable goal-directed behavior. For this purpose, it needs to be possible to use and exploit $\mathcal{M}$ by means of efficient inference techniques (Brachman and Levesque, 2004). Hence, we often prefer low-complexity models.

- *Learning*: $\mathcal{M}$ needs to be learned from the interaction with the environment. A learning algorithm for $\mathcal{M}$ has to be efficient and should only require few experiences to learn expressive models. Thus, $\mathcal{M}$ has to incorporate strong structural assumptions about the world and to depend on a comparatively small number of parameters to achieve generalization. This is achieved by penalizing model complexity in form of regularization which leads to compact and thus generalizing models (instead of merely storing data).

Relational models $\mathcal{M}$ permit strong generalization by encapsulating the transition probabilities $P(s' \mid s, a)$ in a compact way exploiting the relational structure of the domain in terms of objects and their properties and relationships. For example, probabilistic relational rules, described in detail in Sec. 3.1.3, employ generalized partial world state descriptions in the form of conjunctions of abstract literals. This enables abstraction from object identities and concrete domain instantiations. For instance, consider a set of N cups: the effects of trying to grab any of these cups may be described by the same single abstract model instead of using N individual models. The structural assumptions of relational representations may not always permit perfect models of the world. It is our view, however, that a model does not have to be a precise map of the truth—the point of a model is to abstract, to partition the state and action space in the right ways and to simplify things, such that it is a good basis for decision making.

Statistical relational learning methods (Getoor and Taskar, 2007; de Raedt, 2008) are required to learn relational models $\mathcal{M}$. As we will argue below, the assumption of uncertainty is key for learning. Statistical relational learning is not merely a marriage between logic and statistical learning—it is crucial to get logic working in real-world scenarios. Statistical relational learning and in particular probabilistic relational models do not make use of logic in the computational sense, in contrast to traditional work in AI investigating logic inference and theorem proving as a computational paradigm (for example, in Prolog). Instead, from our point of view, the logic descriptions are just complex feature descriptors which are used by a statistical model $\mathcal{M}$ for prediction.

The suitability of a model $\mathcal{M}$ depends on the computational process it is used for (Brachman and Levesque, 2004). For instance, some model representations are appropriate for efficient learning, while others are more appropriate for inference. In particular, the compactness of models and the existence of efficient learning techniques do not imply that the development of efficient inference techniques is straightforward; typically, efficient inference methods need to be developed separately. In this thesis, for instance, we employ probabilistic relational rules for learning and then convert them to graphical models for planning; using a rule representation is appealing for learning due to the intriguing regularized method to extract them from noisy experience. Before taking a closer look at such rules in Sec. 3.1.3, in the following we discuss the prerequisites for learning probabilistic relational transition models in general, review related work, including other prominent AI planning representations, and touch on using these models for goal-directed behavior which will be the focus of the upcoming chapters.

Learning Generalizing Transition Models

The central learning task in model-based reinforcement learning is to estimate a transition model $\mathcal{M}$ from a set of experiences $\mathcal{E} = \{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}$ which defines a conditional distribution $P(s' \mid s, a)$ and can be used for decision-making and planning (Kaelbling et al., 1996). An example of $\mathcal{E}$ is given in Table 3.1. We obtain compact models by compressing the experiences $\mathcal{E}$. In Sec. 3.1.3, we will describe a specific algorithm for learning rules. Here, we consider general conceptual points about learning and compression. Compression of experiences $\mathcal{E}$ and thereby learning of compact models can exploit three opportunities:

- The *frame assumption* (Russell and Norvig, 2003), expressing that all state properties which are not explicitly changed by an action persist over time, allows us to simplify the learning problem by deliberately ignoring large parts of the world.

- By *abstraction*, we can find general patterns in the experiences. Here, abstraction is based on the assumption that the world is made up of objects and the effects of actions manipulating the objects generally depend on the attributes of objects instead of their identities. We use relational representations to achieve such abstraction (de Raedt, 1997).

Table 3.1: The autonomous agent collects a *series* $\mathcal{E} = \{(s_t, a_t, s_{t+1})\}_{t=0}^{T-1}$ *of relational state transitions* consisting of an action a_t (on the left), a predecessor state s_t (first line) and a successor state s_{t+1} (second line after the arrow). The changing state features are underlined. The agent uses such experiences to learn a transition model $\mathcal{M}$ resulting in a compression of the state transitions.

$\mathcal{E}$ = {		
$grab(d)$:		$cube(a),\ cube(b),\ ball(c),\ ball(d),\ table(t),\ on(a,t),\ on(b,t), on(c,a),\ \underline{on(d,b)} \ldots$
	$\rightarrow$	$cube(a),\ cube(b),\ ball(c),\ ball(d),\ table(t),\ on(a,t),\ on(b,t), on(c,a), \underline{inhand(d)} \ldots$
$puton(t)$:		$cube(a),\ cube(b),\ ball(c),\ ball(d),\ table(t),\ on(a,t),\ on(b,t), on(c,a), \underline{inhand(d)} \ldots$
	$\rightarrow$	$cube(a),\ cube(b),\ ball(c),\ ball(d),\ table(t),\ on(a,t),\ on(b,t), on(c,a), \underline{on(d,t)} \ldots$
$grab(c)$:		$cube(a),\ cube(b),\ ball(c),\ ball(d),\ table(t),\ on(a,t),\ on(b,t),\ \underline{on(c,a)},\ on(d,t) \ldots$
	$\rightarrow$	$cube(a),\ cube(b),\ ball(c),\ ball(d),\ table(t),\ on(a,t),\ on(b,t),\ on(d,t),\ \underline{inhand(c)} \ldots$
$\vdots$		$\vdots$
}		

- Assuming *uncertainty* in the observations is essential to fit a generalizing, regularized function. The classic picture in statistical learning for this is illustrated in Fig. 3.1. Uncertainty allows us to tradeoff the exact modeling of every observation (that is, the model likelihood) with generalization capabilities and to find low-complexity explanations of the experiences. Singleton events can be "explained away" as noise. In our point of view, the assumption of uncertainty is crucial to be able to model realistic domains with relational representations: it "unleashes" the model and opens the door for simplification and abstraction. Even in deterministic domains, it may be advantageous to learn a probabilistic model because it can be more compact and neglect irrelevant details. In this sense, modeling uncertainty can also be understood as regularization.

The generalization capability of a model $\mathcal{M}$ and, closely related, the required number of experiences to learn it depend on the chosen representation (Boutilier et al., 1999). In an enumerated representation, we need to collect experiences for each relevant state-action pair (s, a) separately. In a factored propositional representation, one can—to some degree—generalize over states and actions by means of the state attributes, but not over objects. For instance in Table 3.1, in a factored representation we need to learn the effects for $grab(d)$ and $grab(c)$ independently. In contrast, a *relational* representation (de Raedt, 1997) enables compact transition models $P(s' \mid s, a)$ by using abstract formulas to generalize from concrete situations and object identities. For instance in Table 3.1, both $grab(d)$ and $grab(c)$ are used to learn one general model for the abstract action $grab(X)$. In turn, the learned model also applies to situations with previously unseen objects (which is impossible with enumerated and factored propositional representations). We discuss existing relational model types in the following.

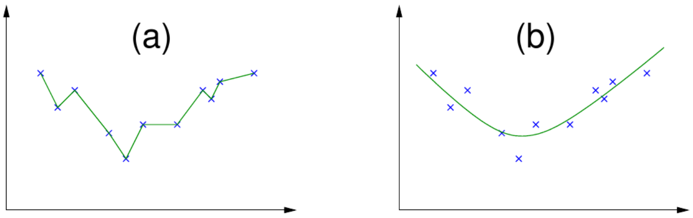


Figure 3.1: *Assuming uncertainty in the observations is crucial to learn compact models.* It corresponds to regularizing the learning algorithm which in turn enables compression and abstraction. (a) Modeling every observation exactly prohibits simplification and abstraction. (b) Assuming uncertainty allows for fitting generalizing smooth functions.

Prominent Relational Transition Models

Since the beginnings of AI research, relational languages were used to model state transitions induced by actions. A famous first deterministic approach is the situation calculus (SC) (McCarthy, 1963) which is an axiomatized logical framework. States are expressed by full first-order logic formulas (including quantification and disjunction) and state transitions are calculated by logical inference with successor and frame axioms applied to the current situation. Due to the complexity of SC, inference in SC is difficult, may be undecidable and involve a theorem prover.

In contrast, the representation language STRIPS (Stanford research institute problem solver) (Fikes and Nilsson, 1971) is restricted enough to allow for the development of efficient inference algorithms and thus became the basic representation language in classical planning. Our discussion of relational representations in Sec. 2.1.2 is based on this representation. In STRIPS, states are represented as conjunctions of positive ground literals with arguments taken from the set of domain objects $\mathcal{O}$; thus, in contrast to SC, state descriptions do not contain complex logical formulas. The state transitions triggered by symbolic actions are described by action schemata (also known as "STRIPS rules") instead of action axioms. STRIPS allows only for deterministic action effects. Table 3.2 illustrates an exemplary action schema. An action schema consists of an abstract action (with variables as arguments), a list of preconditions in form of a conjunction of literals with arguments taken from the set of action arguments, and add- and delete-lists describing the modified literals in the successor state: literals in the add-list are made true and literals in the delete-list are made false. All other ground literals persist according to the frame assumption. If an action is applied in a state where not all preconditions of its action schema hold, this results in a reported "failure signal" and the state will not change. Planning with STRIPS models is P-Space complete (Bylander, 1994). Stochastic STRIPS rules (Kushmerick et al., 1995; Blum and Langford, 1999) extend STRIPS by considering multiple outcomes with probabilities and separate add- and delete-lists. An extension of stochastic STRIPS are noisy indeterministic deictic (NID) rules (Pasula et al., 2007) which add some important features to the STRIPS representation. We will use NID rules extensively in this thesis and present them in detail below.

Many languages have extended deterministic STRIPS by more expressive constructs.

Table 3.2: Example STRIPS action schema for a simple deterministic robot manipulation scenario which models grabbing a ball X.

Action:		$grab(X,Y)$
Preconditions:		$on(X,Y),\ ball(X)$
Effect	— Add:	$inhand(X)$
	— Delete:	$on(X,Y)$

For instance, the popular action description language (ADL) (Pednault, 1989) introduced conditional effects and negative precondition literals into action schemata among other features. The planning domain definition language (PDDL) (Ghallab et al., 1998) provides a standard syntax for various planning formalisms used in AI including STRIPS and ADL. PPDDL (Younes and Littman, 2004) is a probabilistic extension of a subset of PDDL and is the language of the recent international probabilistic planning competitions. Object-oriented MDPs (OOMDPs) (Diuk et al., 2008) consider object attributes explicitly from which the true ground literals in a state are derived and whose predicted values are described in the schema effects.

In the discussion thus far, we have neglected how to learn relational transition models. For a long time, this problem was not considered prominently in AI and machine learning. Only with the advent of inductive logic programming (ILP) (Nienhuys-Cheng and de Wolf, 1997) in the 90s, which studies the supervised learning of relational concepts, significant work on (partial) learning of deterministic action schemata began. Much of this work applied ILP algorithms directly to sets of experienced transitions or transitions provided by a "teacher". For instance, Gil (1994) used operator refinement techniques to complete partial STRIPS-like domain descriptions (missing some preconditions and action effects) from experience by actively testing operators. Other algorithms such as OBSERVER (Wang, 1995) and TRAIL (Benson, 1996) used both experience and a teacher to learn deterministic action schemata. The first significant work on learning probabilistic action schemata is very recent: Pasula et al. (2007) presented a learning algorithm for their NID rules representation. Walsh (2010) showed that the problem of learning general action schemata is NP-hard, but he presented learning algorithms for many learning subtasks with suitable restrictions (such as bounding the number of preconditions and the number of effects) for which efficiency guarantees on the sample complexity can be derived.

Goal-directed Behavior with Probabilistic Relational Models

Probabilistic relational models have the potential to let an agent perform goal-directed behavior in relational domains. It is important to note, however, that while such models may describe the transition dynamics of the world in compact terms, this compactness does not carry over to planning and exploration. Independent methods need to be developed for these tasks.

Planning (Weld, 1999) with deterministic relational models has a long tradition in classical AI. During the last decades, probabilistic and decision-theoretic planning techniques (Kushmerick et al., 1995; Boutilier et al., 1999) for calculating the state and action values of either the partial or the complete states spaces have attracted increasing attention. Some approaches exploit the relational structure to plan in relational MDPs on an abstract level (without referring to concrete objects from $\mathcal{O}$) (Boutilier et al., 2001; Kersting et al., 2004). Alternatively, one may reason in the ground relational domain. This is one of the pursuits of this thesis. Planning in ground domains makes it straightforward to account for special constructs of some world models, such as the noise outcome and the uniqueness requirement of NID rules (Pasula et al., 2007) discussed below. Although grounding simplifies the problem, decision-theoretic planning in the propositionalized representation is a challenging task in complex stochastic domains. In Chapter 4, we introduce algorithms reasoning in the ground relational domain for approximating the optimal values of actions (and action-sequences) for a given state. We also discuss the related work on planning under uncertainty in relational domains in more detail then.

Research on *reinforcement learning* (Sutton and Barto, 1998) in relational representations has focused on model-free approaches which compute policies from experience with respect to fixed goals (Džeroski et al., 2001). Essentially, a number of relational regression algorithms have been developed to estimate a value function of states and actions. In contrast, the work on relational model-based RL is sparse. With the notable exception of Walsh (2010), in particular the important problem of relational exploration has not received significant attention. In Chapter 5, we present a conceptual framework for relational exploration, and we introduce an algorithm for relational model-based reinforcement learning using probabilistic relational rules. We describe related work on relational reinforcement learning in more detail there.

3.1.3 Probabilistic Relational Rules

Pasula et al. (2007) have recently introduced an appealing transition model representation based on noisy indeterministic deictic (NID) rules which combine several advantages:

- a *relational* representation enabling generalization over objects and situations,
- *indeterministic* action outcomes with probabilities to account for uncertainty,
- *deictic references* for actions to reduce the action space,
- *noise outcomes* to avoid explicit modeling of rare and overly complex outcomes, and
- the existence of an effective *learning algorithm.*

This representation gains abstracting knowledge by compressing experiences as shown in Table 3.1 into uncertain rules.

An example of NID rules for our robot manipulation domain is shown in Table 3.3. Fig. 3.2 depicts a situation where this rule can be used for prediction. Formally, a NID rule r is given as

$$a_r(\mathcal{X}) : \ \phi_r(\mathcal{X}) \quad \rightarrow \quad \begin{cases} p_{r,1} & : \ \Omega_{r,1}(\mathcal{X}) \\ & \vdots \\ p_{r,m_r} & : \ \Omega_{r,m_r}(\mathcal{X}) \\ p_{r,0} & : \ \Omega_{r,0} \end{cases} , \tag{3.1}$$

where $\mathcal{X}$ is a set of logical variables in the rule (which represent a (sub-)set of abstract objects). The rule r consists of preconditions, namely that action a_r is applied on $\mathcal{X}$ and that the abstract state context ϕ_r is fulfilled, and $m_r{+}1$ different abstract outcomes with associated probabilities $p_{r,i} > 0$, $\sum_{i=0} p_{r,i} = 1$. Each outcome $\Omega_{r,i}(\mathcal{X})$ describes which state attributes are predicted to change when the rule is applied and this outcome is chosen. The context $\phi_r(\mathcal{X})$ and outcomes $\Omega_{r,i}(\mathcal{X})$ are conjunctions of literals (that is, positive and negated atoms) constructed from the set of predicates $\mathcal{P}$ as well as equality statements comparing functions from the set of functions $\mathcal{F}$ to constant values. The so-called *noise outcome* $\Omega_{r,0}$ subsumes all possible action outcomes which are not explicitly specified by one of the other $\Omega_{r,i}$. This includes rare and overly complex outcomes typical for noisy domains, which we do not want to cover explicitly for compactness and generalization reasons. For instance, in Fig. 3.2 a potential, but highly improbable outcome is to grab the blue cube while pushing all other objects of the table: the noise outcome allows to account for this without the burden of explicitly stating it. The arguments of the action $a(\mathcal{X}_a)$ may be a true subset $\mathcal{X}_a \subset \mathcal{X}$ of the variables $\mathcal{X}$ of the rule. The remaining variables are called deictic references $\mathcal{D} := \mathcal{X} \setminus \mathcal{X}_a$ and denote objects relative to the agent or action being performed. Using deictic references has the advantage to decrease the arity of action predicates. This in turn reduces the size of the action space by at least an order of magnitude, which can have significant effects on learning and planning. For instance, consider a binary action predicate which in a world of n objects has n^2 groundings in contrast to a unary action predicate which has only n groundings.

So, how do we apply NID rules? Let σ denote a substitution that maps variables to constant objects, $\sigma : \mathcal{X} \rightarrow \mathcal{O}$. Applying σ to an abstract rule $r(\mathcal{X})$ yields a *ground rule* $r(\sigma(\mathcal{X}))$. We say a ground rule r *covers* a state s and a ground action a if $s \models \phi_r$ and $a = a_r$. Let Γ be our set of ground rules and $\Gamma(s,a) \subset \Gamma$ the set of rules covering (s,a). If there is a unique covering rule $r_{(s,a)} \in \Gamma(s,a)$, that is $|\Gamma(s,a)| = 1$, we use it to model the effects of action a in state s. We calculate $P(s' \,|\, s,a)$ by taking all outcomes of $r_{(s,a)}$ (omitting subscripts in the following) into account weighted by their respective probabilities,

$$P(s' \,|\, s,a) \ = \ P(s' \,|\, s,r) \ = \ \sum_{i=1}^{m_r} p_{r,i}\, P(s' \,|\, \Omega_{r,i}, s) \ + \ p_{r,0}\, P(s' \,|\, \Omega_{r,0}, s), \tag{3.2}$$

where, for $i > 0$, $P(s' \,|\, \Omega_{r,i}, s)$ is a deterministic distribution that is one for the unique state constructed from s taking the changes of $\Omega_{r,i}$ into account. The distribution given

Table 3.3: Example NID rule for our complex robot manipulation domain which models to try to grab a ball X. The cube Y is implicitly defined as the one below X (deictic referencing). X ends up in the robot's hand with high probability, but might also fall on the table. With a small probability something unpredictable happens. Confer Fig. 3.2 for an example application.

$$
\begin{aligned}
\mathit{grab}(X): \quad & \mathit{on}(X,Y),\ \mathit{ball}(X),\ \mathit{cube}(Y),\ \mathit{table}(Z) \\
\rightarrow \quad & \begin{cases} 0.7 & : \ \mathit{inhand}(X),\ \neg \mathit{on}(X,Y) \\ 0.2 & : \ \mathit{on}(X,Z),\ \neg \mathit{on}(X,Y) \\ 0.1 & : \ \text{noise} \end{cases}
\end{aligned}
$$

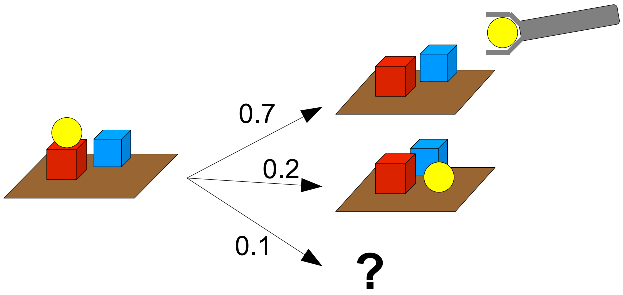


Figure 3.2: The NID rule defined in Table 3.3 can be used to predict the effects of action $grab(yellowBall)$ in the situation on the left side. The right side depicts the possible successor states as predicted by the rule. The noise outcome is indicated by a question mark and does not define a unique successor state.

the noise outcome, $P(s' \,|\, \Omega_{r,0}, s)$, is unknown and needs to be estimated. Pasula et al. use a worst case constant bound $p_{min} \leq P(s' \,|\, \Omega_{r,0}, s)$ to lower bound $P(s' \,|\, s, a)$. Alternatively, to come up with a well-defined distribution, one may assign very low probability to very many successor states. As described in more detail in Chapter 4, our planning algorithm based on approximate inference in graphical models achieves this by exploiting the factored state representation of a ground relational domain and predicting each state attribute to change with a very low probability.

If a state-action pair (s, a) does *not* have a unique covering rule r (including the case that more one rule covers the state-action pair, resulting in potentially conflicting predictions), we use a noisy default rule r_ν which predicts all effects as noise: $P(s'|s, r_\nu) = P(s' \,|\, \Omega_{r_\nu,0}, s)$. Using r_ν expresses that we do not know what will happen. This is not meaningful and thus disadvantageous for planning. (Hence, one should bias a NID rules learner to learn rules with contexts which are likely to be mutually exclusive.) For this reason, the concept of unique covering rules is crucial in planning with NID rules. Here, we have to pay the price for using deictic references: when using an abstract NID rule for prediction, we always have to ensure that its deictic references have unique groundings.

This may require examining a large part of the state representation, so that proper storage of the ground state and efficient indexing techniques for logical formula evaluation are needed.

The semantics of NID rules allow one to plan in relational domains, that is, to find a "satisficing" action sequence that will lead with high probability to states with high rewards. In Chapter 4, we introduce planning algorithms based on NID rules. Our major motivation for employing NID rules is that we can learn them from observed actions and state transitions. Furthermore, our planning approach PRADA can exploit their simple structure (which is similar to probabilistic STRIPS operators) and convert them into a DBN representation. We provide a detailed comparison of NID rules and PPDDL in Appendix B. While NID rules do not support all features of a sophisticated domain description language such as PPDDL, they can compactly capture the dynamics of many interesting planning domains as we will see in Chapters 4 and 5. In the following, we describe how we can actually learn NID rules from experience.

Learning NID Rules

As discussed above, the ability to learn transition models from experience is a crucial requirement for autonomous agents. The problem of learning rule-sets is in general NP-hard, but with suitable assumptions and restrictions efficiency guarantees on the sample complexity can be given for many learning subtasks (Walsh, 2010). Pasula et al. (2007) have proposed a supervised batch learning algorithm for *complete* NID rules. This algorithm learns the structure of rules as well as their parameters from experience triples $\mathcal{E} = \{(s_t, a_t, s_{t+1})_{t=0}^{T-1}\}$ (as illustrated in Table 3.1) relying on the frame assumption (assuming that nothing changes which is not explicitly stated to change). Following ideas from inductive logic programming, the learning algorithm performs a greedy search through the space of rule-sets. It optimizes the tradeoff between maximizing the likelihood of the experience triples and minimizing the complexity of the current hypothesis rule-set Γ by optimizing the scoring metric

$$S(\Gamma) = \sum_{(s,a,s')\in\mathcal{E}} \log P(s' \mid s, r_{(s,a)}) - \alpha \sum_{r\in\Gamma} \mathrm{PEN}(r) \ , \tag{3.3}$$

where $r_{(s,a)} \in \Gamma$ is either the unique covering rule for the state-action pair (s,a) or the noisy default rule r_ν. α is a scaling parameter that controls the influence of regularization. PEN(r) penalizes the complexity of a rule and is defined as the total number of literals in r. This has analogies to results in cognitive science suggesting that the subjective complexity of human intuitive theories is determined by the representation length (Kemp et al., 2008). The larger α is set, the more compact are the learned rule-sets—and thus, the more general, but also the more uncertain and potentially inaccurate. For instance, if we set $\alpha = \infty$, the resulting model will consist of only the default rule, explaining all state transitions as noise. In contrast, if we set $\alpha = 0$, the resulting model explains each experience as exactly as possible, potentially overfitting and badly generalizing the data.

The noise outcome of NID rules is crucial for learning. The learning algorithm is initialized with a rule-set comprising only the noisy default rule r_ν and then iteratively adds new rules or modifies existing ones using a set of search operators. The noise outcome allows avoiding overfitting, as we do not need to model rare and overly complex outcomes explicitly. Its drawback is that its successor state distribution $P(s' \mid \Omega_{r,0}, s)$ is unknown. To deal with this problem, the learning algorithm uses a lower bound p_{min} to approximate this distribution, as described above. In our experience, while the resulting rule-sets are robust to minor changes in the two parameters p_{min} and α, sensible settings for both are essential for learning expressive rule-sets and their mutual influence has to be taken into account. These parameters, however, cannot be optimized by the rule-learning algorithm itself. The required degree of rule compactness is not only determined by the complexity of the modeled domain in terms of its stochasticity and richness, such as the number and complexity of contexts which need to be distinguished to describe the effects of an action. But it is also affected by the purpose the rules have to serve; for instance, whether we want the rules to be as accurate as possible (for very reliable planning) or whether a coarse approximation of the transition dynamics is sufficient (for fast approximate planning).

This rule learning algorithm uses greedy heuristics in its attempt to learn complete rules, so no guarantees on its behavior can be given. It has been shown, however, that learned NID rules can provide accurate transition models in noisy manipulation domains (Pasula et al., 2007). In this thesis, we show that the transition dynamics in more complex robot manipulation scenarios and in many domains of the international planning competition can be learned effectively from experience with NID rules.

NID rules are an appropriate representation for learning a transition model $\mathcal{M}$ from experience. Different representation types may be more appropriate for other computational processes such as inference and planning. In the following section, we discuss graphical models for ground relational domains which can be built from NID rules.

3.2 Graphical Models for Ground Relational Domains

Decision-theoretic problems where agents need to choose appropriate actions can be represented by means of graphical models which are augmented by decision nodes to specify the agent's actions (Boutilier et al., 1999). Toussaint and Storkey (2006) have recently shown that the problem of planning in a Markov decision process, which is typically framed as a dynamic programming problem, can be reformulated as a problem of probabilistic inference in graphical models (see Sec. 2.3). While both the dynamic programming and the probabilistic inference view compute the same optimal solutions (on feasible problems), the interesting difference lies in the concrete computational methods that can be used to exploit structure for gaining efficiency or to compute approximations to infeasible problems. With the planning-by-inference paradigm, the whole variety of existing (exact and approximate) inference methods becomes available to solve hard planning problems in structured domains.

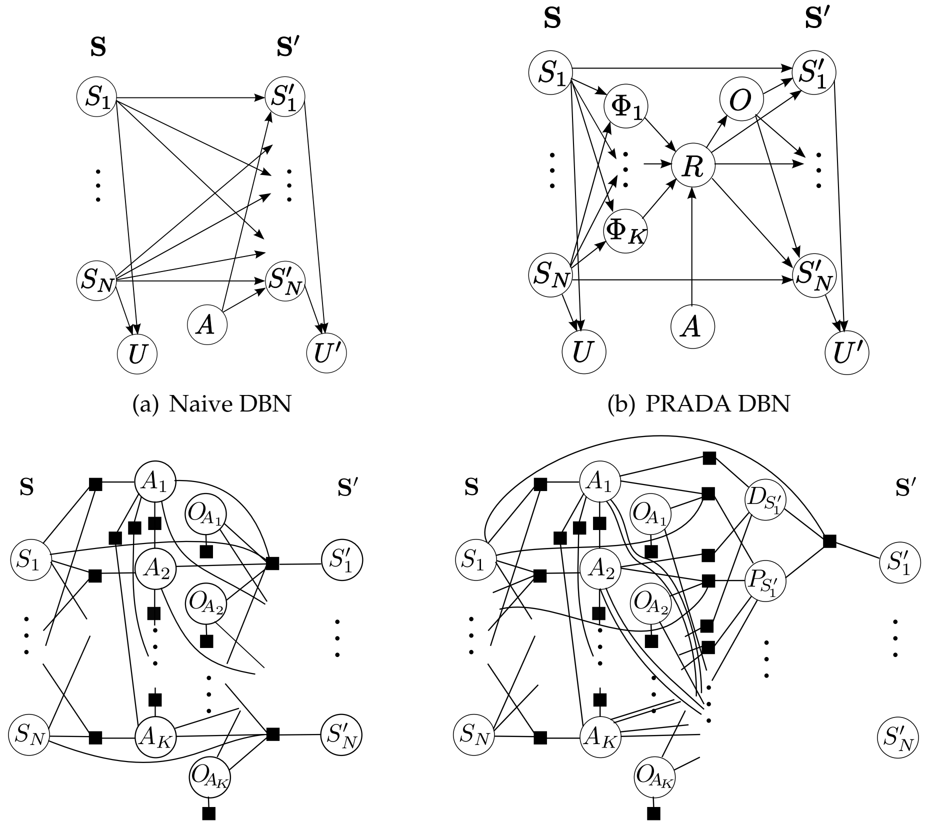
So far, however, planning-by-inference approaches have only been applied in non-relational domains with limited state spaces. It is an open question how to use inference techniques for relational planning. The existing non-relational approaches are not feasible in relational domains due to their very large state and action spaces. The crucial question is how to represent a relational planning problem in appropriate graphical models which permit efficient inference methods. Such models need to be highly structured to exploit the redundancies and symmetries in relational domains. For instance, many relational states share the same evidence and behave similarly in message propagation. When planning a sequence of actions, the similarity of the network structure across time-slices may cause many nodes to share a lot of information. While exact inference is probably always infeasible in non-trivial relational domains, appropriate graphical model representations may permit the development of efficient approximate inference techniques.

In this section, we discuss an existing and three novel graphical models to represent the transition dynamics of ground relational domains (Lang and Toussaint, 2009b, 2010b). We show how to convert NID rules, providing a relational transition model, to dynamic Bayesian networks and factor graphs. In Chapter 4, we discuss how to apply inference in these models for planning. In particular, the second proposed graphical model (Sec. 3.2.2) will provide the formal framework of our planning algorithm PRADA (Lang and Toussaint, 2009b, 2010b).

Before diving into the models, we restate our notational conventions: we denote random variables by upper case letters (e.g., S), their values by the corresponding lower case letters ($s \in dom(S)$), variable sets by bold upper case letters ($\mathbf{S} = \{S_1, S_2, S_3\}$) and value sets by bold lower case letters ($\mathbf{s} = \{s_1, s_2, s_3\}$). We also use column notation ($\mathbf{s}_{2:4} = \{s_2, s_3, s_4\}$).

3.2.1 Naive DBN

A naive way to convert NID rules to DBNs is shown in Fig. 3.3(a). States are represented by a set $\mathbf{S} = \{S_1, \ldots, S_N\}$ where for each ground predicate according to the set of predicates $\mathcal{P}$ there is a binary S_i and for each ground function according to the set of functions $\mathcal{F}$ there is an S_j with range according to the represented function. Actions are represented by an integer variable A which indicates the action out of a set of ground action predicates. The reward gained in a state is represented by U and may depend only on a subset of the state variables. It is possible to express arbitrary reward expectations $P(U \mid \mathbf{S})$ with binary U (Cooper, 1988). How can we define the transition dynamics using NID rules in this naive model? Assume we are given a set of fully abstract NID rules. We compute all groundings of these rules w.r.t. the objects of the domain and get the set Γ of K different ground NID rules. The parents of a state variable S'_i at the successor time-step include the action variable A and the respective variable S_i at the predecessor time-step. The other parents of S'_i are determined as follows: For each rule $r \in \Gamma$ where the literal corresponding to S'_i appears in the outcomes of r, all variables S_k corresponding to literals in the context of r are parents of S'_i. As typically S'_i can be manipulated by several actions which in turn are modeled by several rules, the total number of parents



(a) Naive DBN

(b) PRADA DBN

(c) Half-decomposed FG

(d) Fully decomposed FG

Figure 3.3: *Graphical models for ground relational domains.* We convert probabilistic relational rules to directed dynamic Bayesian networks (DBNs) and undirected factor graphs (FGs) to use inference for relational planning. We omit the utility nodes U in the factor graphs for visibility.

of S_i' can be very large. This problem becomes worse by the usage of deictic references in the NID rules, as they increase the total number K of ground rules in Γ. The resulting local structure of the conditional probability function of S_i' is very complex, as one has to account for the uniqueness of covering rules. These complex dependencies between two time-slices make this representation unfeasible for planning.

3.2.2 PRADA-DBN

A better way to capture the structure of NID rules is the graphical model shown in Fig. 3.3(b) (Lang and Toussaint, 2009b, 2010b). This model is more compact in terms of the number of parameters used to define the conditional probability tables. It will be used by our planning algorithm PRADA, hence we call it PRADA-DBN. This DBN represents the joint distribution

$$P(u', \mathbf{s}', o, r, \boldsymbol{\phi} \,|\, a, \mathbf{s}) \quad = \quad P(u' \,|\, \mathbf{s}')\; P(\mathbf{s}' \,|\, o, r, \mathbf{s})\; P(o \,|\, r)\; P(r \,|\, a, \boldsymbol{\phi})\; P(\boldsymbol{\phi} \,|\, \mathbf{s})\,, \tag{3.4}$$

which we will explain in detail in the following. As before, assume we are given a set of fully abstract NID rules, for which we compute the set Γ of K different ground NID rules w.r.t. the objects in the domain. In addition to $\mathbf{S}$, $\mathbf{S}'$, A, U and U' as above, we use a binary random variable Φ_i for each rule to model the event that its context holds, which is the case if all required literals hold. Let $I(\cdot)$ be the indicator function which is 1 if the argument evaluates to true and 0 otherwise. Then, we have

$$P(\boldsymbol{\phi}\,|\,\mathbf{s}) = \prod_{i=1}^{K} P(\phi_i|\mathbf{s}_{\pi(\Phi_i)}) = \prod_{i=1}^{K} I\left(\bigwedge_{j\in\pi(\Phi_i)} S_j = s_{r_i,j}\right). \tag{3.5}$$

We use $\bigwedge_i \rho_i$ to express a logical conjunction $\rho_1\wedge\cdots\wedge\rho_n$. The function $\pi(\Phi)$ yields the set of indices of the state variables in $\mathbf{s}$ on which Φ depends. $\mathbf{s}_{r_i}$ denotes the configuration of the state variables corresponding to the literals in the context of r_i. We use an integer-valued variable R ranging over $K+1$ possible values to identify the rule which predicts the effects of the action. If it exists, this is the unique covering rule for the current state-action pair, i.e., the only rule $r \in \Gamma(a)$ modeling action a whose context holds:

$$P(R=r|a,\boldsymbol{\phi}) = I\left(r \in \Gamma(a) \wedge \Phi_r = 1 \wedge \bigwedge_{r'\in\Gamma(a)\setminus\{r\}} \Phi_{r'} = 0\right). \tag{3.6}$$

If no unique covering rule exists, we predict no changes as indicated by the special value $R = 0$ (assuming not to execute the action):

$$P(R=0\,|\,a,\boldsymbol{\phi}) = \bigwedge_{r\in\Gamma(a)} \neg I\left(\Phi_r = 1 \wedge \bigwedge_{r'\in\Gamma(a)\setminus\{r\}} \Phi_{r'} = 0\right). \tag{3.7}$$

The integer-valued variable O represents the outcome of the action as predicted by the rule. It ranges over M possible values where M is the maximum number of outcomes all rules in Γ have. To ensure a sound semantics, we introduce empty dummy outcomes with zero-probability for those rules whose number of outcomes is less than M. The probability of an outcome is defined as in the corresponding rule:

$$P(O=o\,|\,r) = p_{r,o}\,. \tag{3.8}$$

We define the probability of the successor state as

$$P(\mathbf{s}'\,|\,o,\mathbf{s},r) = \prod_i P(s_i'\,|\,o,s_i,r)\,, \tag{3.9}$$

which is one for the unique state that is constructed from s taking the changes according to $\Omega_{r,o}$ into account: if outcome o specifies a value for S_i', this value will have probability one. Otherwise, the value of this state variable persists from the previous time-step. As rules usually change only a small subset of s, persistence most often applies. The resulting dependency $P(s_i'\,|\,o,r,s_i)$ of a variable S_i' at time-step $t+1$ is compact. In contrast to the naive DBN in Fig. 3.3(a), it has only three parents, namely the variables for the outcome, the rule and its predecessor at the previous time-step. This simplifies the specification of a conditional probability function for S' significantly and enables efficient inference, as we will see later. The probability of the reward is given by

$$P(U'\!=\!1 \,|\, \mathrm{s}') = I\left(\bigwedge_{j\in\pi(U')} S'_j\!=\!\tau_j\right) \; . \tag{3.10}$$

The function $\pi(U')$ yields the set of indices of the state variables in s' on which U' depends. The configuration of these variables that corresponds to our planning goal is denoted by $\boldsymbol{\tau}$. Alternatively, more general reward specifications for decision-theoretic planning can be defined by individual probabilities for U' conditioned on different state configurations (Cooper, 1988). Uncertain initial states can be naturally accounted for by specifying priors $P(\mathrm{s}^0)$. We renounce on the specification of a prior here, however, as the initial state s^0 will always be given in our experiments later to be able to compare to the lookahead-tree based approaches SST and UCT which require deterministic initial states. The distribution $P(a)$ used for sampling actions has major influence on the planning procedure. We describe it in the context of our planning algorithms in Sec. 4.2.

For simplicity we have ignored derived predicates and functions which are defined in terms of other predicates or functions. Derived concepts may increase the compactness of rules. If dependencies among concepts are acyclic, it is straightforward to include derived concepts in our model by intra-state dependencies for the corresponding variables. Indeed, we will use derived predicates in our experiments.

3.2.3 Half-decomposed Factor Graph

The structure of the PRADA-DBN model permits a specific efficient approximate inference method which we describe in the next chapter. However, this DBN is still highly connected due to the central high-dimensional nodes A, R and O which couple all the variables describing a state transition. While the symmetries inherent in relational domains are reflected in the parameters of these central nodes, exploiting symmetries in the associated factors for efficient inference is not straight-forward. In contrast, most approximate inference strategies exploit symmetries in the *structure*.

Therefore, we introduce two novel graphical models in the following which represent the symmetries of relational domains in their structure. Instead of directed Bayesian networks, we represent these models as undirected factor graphs as it is more general to express their parameters as coupling weights than as conditional probabilities. In our subsequent presentation, for ease of illustration we assume that there is exactly one NID rule without deictic references for each abstract action so that we can ignore the problem of rule uniqueness and identify each ground action with exactly one ground rule. An extension to multiple rules and deictic references is not difficult.

The novel factor graph shown in Fig. 3.3(c) decomposes a relational state transition by introducing for each ground action $\alpha \in \mathcal{A}$ a separate binary action variable A together with an outcome variable O_A with arity according to the number of outcomes of α's covering rule r_α. We abuse notation slightly and will denote r_α by r_A in the following. A value $a = 1$ of A denotes that the corresponding action α is executed, $a = 0$ that it is not executed. The value of O_A picks out the corresponding outcome in r_A. Similarly as before, a context factor f_Φ for each action variable A ensures that the context literals of r_A hold if A is true,

$$f_\Phi(\mathbf{s}_{\pi(\Phi_{r_A})}, a) = \begin{cases} I\left(\bigwedge_{j \in \pi(\Phi_{r_A})} S_j = s_{r_A,j}\right) & \text{if} \quad a = 1 \\ 1 & \text{otherwise} \end{cases} .$$

As above, the function $\pi(\Phi_{r_A})$ yields the set of indices of the state variables in $\mathbf{s}$ on which Φ_{r_A} depends, and $\mathbf{s}_{r_A}$ denotes the configuration of the state variables corresponding to the literals in r_A's context. If the action variable does not hold ($a = 0$, second line), this factor does not impose any constraints on the current state.

By means of mutual action exclusion factors f_{excl}, we ensure that two action variables are never true at the same time,

$$f_{excl}(a, a') = \begin{cases} 0 & \text{if} \quad a = 1 \wedge a' = 1 \\ 1 & \text{otherwise} \end{cases} .$$

The outcome factors f_o are simple priors for the outcome variables O_A according to the outcome probabilities of the covering rule r_A:

$$f_o(o_A) = p_{r_A, o_A} .$$

As in the DBNs presented above, the outcome factors define the only non-deterministic information in the graphical model.

Finally, the successor state factors f_{suc} couple the successor state with the predecessor state by taking actions and their outcomes into account. In more detail, f_{suc} couples the successor state variable S'_i with its predecessor S_i, the variables $\mathbf{A}_{S'_i}$ of all actions manipulating S'_i (as determined by their rules) and their corresponding outcome variables $\mathbf{O}_{S'_i}$ as follows:

$$f_{suc}(s'_i, s_i, \mathbf{a}_{S'_i}, \mathbf{o}_{S'_i}) = \begin{cases} 1 & \text{if} \quad s_i = s'_i \wedge \forall A \in \mathbf{A}_{S'_i} : a = 0 \qquad \text{(Ia)} \\ 0 & \text{if} \quad s_i \neq s'_i \wedge \forall A \in \mathbf{A}_{S'_i} : a = 0 \qquad \text{(Ib)} \\ 1 & \text{if} \quad \exists A_k \in \mathbf{A}_{S'_i} : a_k = 1 \wedge (\forall A_l \in \mathbf{A}_{S'_i}, l \neq k : a_l = 0) \\ & \qquad \wedge (s_i \circ \Omega_{r_{A_k}, o_k} = s'_i) \qquad \text{(IIa)} \\ 0 & \text{if} \quad \exists A_k \in \mathbf{A}_{S'_i} : a_k = 1 \wedge (\forall A_l \in \mathbf{A}_{S'_i}, l \neq k : a_l = 0) \\ & \qquad \wedge (s_i \circ \Omega_{r_{A_k}, o_k} \neq s'_i) \qquad \text{(IIb)} \\ 0 & \text{if} \quad \exists A_k, A_l \in \mathbf{A}_{S'_i}, k \neq l : a_k = 1, a_l = 1 \qquad \text{(III)} \end{cases} .$$

If all action variables $A \in \mathbf{A}_{S'_i}$ are false, then S'_i has to match its predecessor S_i (cases Ia and Ib). If there is exactly one true variable $A_k \in \mathbf{A}_{S'_i}$, then S'_i is set according to the prediction of rule r_{A_k} and the corresponding outcome variable O_k (cases IIa and IIb). To express this constraint, we abuse notation and let $\circ$ denote the operation to construct the unique successor value of S'_i according to S_i and rule outcome $\Omega_{r,o}$. Furthermore, f_{suc} forbids multiple true action variables $A \in \mathbf{A}_{S'_i}$ by setting the weight of such configurations to 0 (case III).

For brevity, we do not discuss the reward factor f_{reward} for a utility variable U which can be set straightforwardly along the lines of the conditional probabilities of the above DBNs.

This factor graph provides an encoding of the successor state distribution $P(s' \mid s, a)$ as defined by NID rules. Its drawback is that the factors f_{suc} couple all actions $\mathbf{A}_{S'_i}$ manipulating the state variable S'_i. As $\mathbf{A}_{S'_i}$ may become large, the factors f_{suc} easily become large as well, making it hard to develop efficient approximate inference methods. The next graphical model tries to overcome this limitation.

3.2.4 Fully Decomposed Factor Graph

Our novel model shown in Fig. 3.3(d) decomposes the model of the previous subsection further by introducing two additional variables D_i and P_i for each successor state attribute S'_i. This helps to avoid the large successor state factors f_{suc} of the previous model. First, the binary decision variable D_i denotes whether no action variable $A \in \mathbf{A}_{S'_i}$ of actions manipulating S'_i is true and thus the value of S_i persists ($D_i = 0$) or whether a relevant action variable $A \in \mathbf{A}_{S'_i}$ is true and thus potentially predicts a change (this depends on the outcome) ($D_i = 1$). If $D_i = 0$, then S'_i must match S_i, otherwise S'_i is set according to the action's prediction. Separate factors $f_{decision}$ for each variable $A \in \mathbf{A}_{S'_i}$ determine whether a relevant action variable is true:

$$f_{decision}(a, d_i) = \begin{cases} 0 & \text{if} \quad a=1 \wedge d_i=0 \\ \epsilon & \text{if} \quad a=0 \wedge d_i=1 \\ 1 & \text{otherwise} \end{cases} .$$

The first line prohibits the situation when a manipulating action variable is true ($a=1$), but the decision variable is set to persistence ($d_i=0$). The second line specifies the case that the action is not executed ($a = 0$), but an action manipulation for S'_i is signaled ($d_i=1$). A non-zero ϵ is necessary to allow for another action to manipulate S'_i. ϵ should be chosen small, however, to make the case less probable that for all relevant actions it holds that $a=0$ while $d_i=1$ signals manipulation. Let $m = |\mathbf{A}_{S'_i}|$ and l be the number of actions $A \in \mathbf{A}_{S'_i}$ for which $a = 1$. Then, the marginal over d_i of the product of the factors $f_{decision}$ is $P(d_i=0) \propto 1^{m-l}0^l = 0^l$ and $P(d_i=1) \propto \epsilon^{m-l}1^l = \epsilon^{m-l}$. Thus, in the limit of ϵ to 0, the marginal over d_i corresponds to a logical OR over the actions.

The prediction variables P_i have the same range as their corresponding state attribute variables S'_i and represent the predicted value according to an executed relevant action. For each combination of state attribute S'_i and manipulating action $A \in \mathbf{A}_{S'_i}$, there is a separate factor f_{pred} over the prediction variable P_i, the predecessor state attribute variable S_i, the action variable A and its outcome variable O:

$$f_{pred}(p_i, s_i, a, o) = \begin{cases} 0 & \text{if} \quad a=1 \wedge (s_i \circ \Omega_{r_A,o} \neq p_i) \\ 1 & \text{if} \quad a=1 \wedge (s_i \circ \Omega_{r_A,o} = p_i) \\ 1 & \text{otherwise} \quad (a=0) \end{cases} \quad .$$

As above, we abuse notation and let $\circ$ denote the operation to construct the unique predicted value according to S_i and outcome $\Omega_{r,o}$. If the action variable is true, f_{pred} prohibits other values than the one predicted by the action (first and second line). If the action does not hold, the other variables are not constrained (third line).

Apart from the two new factor types $f_{decision}$ and f_{pred}, the factors for rule contexts f_Φ, action mutual exclusion f_{excl} and outcome priors f_o are the same as in the previous factor graph. We need to modify the successor state factors f_{suc}, however, which now couple predecessor and successor state attribute variables S_i and S'_i with the corresponding decision and prediction variables S_i and P_i as

$$f_{suc}(s'_i, s_i, d_i, p_i) = \begin{cases} 1 & \text{if} \quad d_i=1 \wedge p_i=s'_i \quad \text{(Ia)} \\ 0 & \text{if} \quad d_i=1 \wedge p_i\neq s'_i \quad \text{(Ib)} \\ 1 & \text{if} \quad d_i=0 \wedge s_i=s'_i \quad \text{(IIa)} \\ 0 & \text{if} \quad d_i=0 \wedge s_i\neq s'_i \quad \text{(IIb)} \end{cases} \quad . \tag{3.11}$$

If the decision node signals action manipulation, S'_i has to match the predicted value (cases Ia and Ib). Otherwise, S'_i has to match the value of its predecessor (cases IIa and IIb). Now, we see the advantage of introducing the additional variables D_i and P_i for each state attribute S'_i: in contrast to the previous factor graph model, it allows for a compact specification of f_{suc} over only four variables. Apart from f_Φ whose size depends on the complexity of rule contexts, all factors in this graphical model involve maximally four variables so it is fully decomposed.

Whether this factor graph provides an encoding of the successor state distribution $P(s' \mid s, a)$ as defined by NID rules depends on the parameter ϵ in the factors $f_{decision}$. In the limit of ϵ to 0, the decision variables represent a logical OR over the actions manipulating the respective state attributes and in this case the overall factor graph is an equivalent representation.

3.3 Discussion

In this chapter, we have investigated models of the transition dynamics in stochastic relational worlds. These models inform an autonomous agent how its actions change the state of the world and can thus be exploited to perform goal-directed behavior. Probabilistic relational transition models permit to learn and represent the transition dynamics in worlds with many objects: they make the structural assumption that the world is appropriately represented in terms of objects and their properties and relationships and the effects of actions depend only on these properties and relationships and not on object identities; thereby, they permit strong generalization. We have discussed the requirements to be able to learn relational transition models and we have described popular relational transition model representations which are typically hand-crafted and not learned. We have focused on a specific model in form of probabilistic relational rules for which an effective learning algorithm is known. We have provided some novel insights concerning learning such rules and their relation to the general planning language PPDDL. Different types of representations are appropriate for different computational processes. For this reason, we have introduced novel graphical models for ground relational domains which can be derived from learned rules. Such models will build the backbone in our planning algorithms in Chapter 4, which in combination with algorithms for learning will provide the foundation for exploration in relational domains investigated in Chapter 5.

Chapter 4

Planning in Ground Relational Domains

To act in the real world, we have to accomplish two tasks. First, we need to understand how the world works: for example, a pile of cubes is more stable if we place the big cubes at its bottom; it is a hard job to build a tower from balls; something can only be put into a box if the box is open and not already filled. Probabilistic relational transition models discussed in Chapter 3 encode such knowledge. They enable to generalize over object identities to unencountered situations and objects of similar types and to account for indeterministic action effects and noise. Autonomous agents need to learn such models from experience to adapt to new environments and not to rely on human hand-crafting. Noisy indeterministic deictic (NID) rules (Pasula et al., 2007), described in Sec. 3.1.3, are an appealing relational transition model as they can be learned effectively from experience and are therefore our choice in this chapter.

Once we know about the possible effects of our actions, we face a second challenging task: how can we use our acquired knowledge in reasonable time to find a sequence of actions suitable to achieve our goals? This chapter investigates this second task, namely planning. The previously used approach for planning with probabilistic relational rules relies on growing full look-ahead trees in the ground domain (Kearns et al., 2002; Pasula et al., 2007). Due to the very large action space and the stochasticity of the world, the computational burden to plan just a single action with this method in a given situation can be overwhelmingly large. In this chapter, we propose novel ways for reasoning efficiently in the ground domain using learned NID rules, enabling fast planning in complex environments with varying goals.

We proceed as follows. After having discussed related work on planning in stochastic relational domains, we propose two novel methods for *forward* reasoning in ground relational domains (Sec. 4.2): we apply the existing *Upper Confidence bounds applied to Trees* (UCT) algorithm (Kocsis and Szepesvari, 2006) with NID rules which cuts suboptimal parts of lookahead-trees early, and we introduce the *Probabilistic Relational Action-sampling in DBNs planning Algorithm* (PRADA) (Lang and Toussaint, 2009b, 2010b) which uses probabilistic inference to cope with uncertain action outcomes. Thereafter, we introduce a *bidirectional* reasoning approach (Lang and Toussaint, 2010a) based on employing PRADA's approximate inference for both backward and forward reasoning (Sec. 4.3). Bidirectional reasoning may lead to significant efficiency gains in comparison to pure unidirectional reasoning. All these planning approaches rely on grounding the relational domain, that is, they reason on the level of concrete objects in a propositionalised relational representation. Planning in the fully grounded representation may become inefficient with a large number of objects. We show how to avoid this problem by means of *relevance grounding* (Lang and Toussaint, 2009a) which grounds a relational domain only with respect to the objects which are relevant for the planning task at hand (Sec. 4.4). Thereafter, we explore briefly the full planning-by-inference paradigm in ground relational domains, which has never been investigated thus far (Sec. 4.5). Finally, we present a study on applying our forward reasoning algorithm PRADA on a *real-world robot platform* (Sec. 4.6) (Toussaint, Plath, Lang, and Jetchev, 2010). To our knowledge, this is the first relational planning approach for object manipulation on a real robot, where in addition the underlying knowledge has been learned from experience in a simulator.

Our **contributions** in this chapter are the following novel approaches for planning in ground relational domains:

- We present **forward** reasoning approaches, in particular we introduce our **PRADA** algorithm based on approximate inference in graphical models (Sec. 4.2) (Lang and Toussaint, 2009b, 2010b).
- We extend the inference approach PRADA to backward reasoning enabling planning based on **bidirectional** reasoning (Sec. 4.3) (Lang and Toussaint, 2010a).
- We introduce the formal framework of **relevance grounding** for the idea of focusing on relevant objects in planning (Sec. 4.4) (Lang and Toussaint, 2009a).
- We are the first to frame **relational planning as a pure inference problem** and present a brief study in this direction (Sec. 4.5).
- We describe one of the first case studies on applying high-level relational planning on a **real-world robot** (Sec. 4.6) (Toussaint, Plath, Lang, and Jetchev, 2010).

4.1 Related Work on Planning in Stochastic Relational Domains

The problem of decision-making and planning in stochastic relational domains has been approached in different ways. One can renounce on the use of relational transition models and try to estimate a policy or the values of actions with respect to a fixed goal type (such as $on(X, Y)$) directly from state attributes. This is the approach taken in model-free reinforcement learning (see Sec. 2.2) and has been investigated thoroughly in the context of relational representations (Džeroski et al., 2001; Driessens et al., 2006) using abstract logical formulas to generalize over objects and situations. In contrast, if a probabilistic relational transition model is available (either learned or handcrafted), one can pursue decision-theoretic planning in several other ways. This is the focus of the current chapter and of our discussion of related work in the following.

Planning in Abstract Representations

Within the machine learning community, a popular direction of research formalizes the problem as a relational Markov decision process (RMDP) (see Sec. 2.1.2) and develops dynamic programming algorithms to compute solutions, that is policies over complete state and action spaces. Many algorithms reason in the *lifted* abstract representation without grounding or referring to particular problem instances. Boutilier et al. (2001) introduce symbolic dynamic programming, the first exact solution technique for RMDPs which uses logical regression to construct minimal logical partitions of the state space required to make all necessary value function distinctions. This approach has not been implemented as it is difficult to keep the first-order state formulas consistent and of manageable size. Based on these ideas, Kersting et al. (2004) propose an exact value iteration algorithm for RMDPs using logic-programming, called ReBel. They employ a restricted language to represent RMDPs so that they can reason efficiently over state formulas. Hölldobler et al. (2006) present a first-order value iteration algorithm (FOVIA) using a different restricted language. Karabaev and Skvortsova (2005) extend FOVIA by combining first-order reasoning about actions with a heuristic search restricted to those states that are reachable from the initial state. Wang et al. (2008) derive a value iteration algorithm based on using first-order decision diagrams (FODDs) for goal regression. They introduce reduction operators for FODDs to keep the representation small, which may require complex reasoning; an empirical evaluation has not been provided. Joshi et al. (2009) apply model checking to reduce FODDs and generalize them to arbitrary quantification.

These techniques form an interesting research direction as they reason exactly about abstract RMDPs. They employ different methods to ensure exact regression such as theorem proving, logical simplification, or consistency checking. Therefore, principled approximations that can discover good policies in more difficult domains are likewise worth investigating. Sanner and Boutilier (2007, 2009) present a first-order approximate linear programming approach (FOALP). They approximate the value function based on linear combinations of first-order functions which they deduce from the relational transition model by abstract symbolic reasoning without grounding, showing impressive results on solving RMDPs with millions of states. Other lifted planning approaches based on relational transition models sample trajectories from ground domain instantiations to induce first-order value function and policy representations. For instance, Fern et al. (2006) consider a variant of approximate policy iteration (API) where they replace the value-function learning step with a learning step in policy space. They make use of a policy-space bias as described by a generic relational knowledge representation and simulate trajectories to improve the learned policy. Kersting and Driessens (2008) describe a non-parametric policy gradient approach which can deal with propositional, continuous and relational domains in a unified way. Gretton and Thiébaux (2004) use abstract symbolic reasoning to generate a suitable hypothesis language which they then use for policy induction; thereby, their approach avoids formula rewriting and theorem proving, while still requiring model-checking.

Planning in the Ground Domain

Instead of working in the abstract representation, one may reason in the ground domain. This makes it straightforward to account for two special characteristics of NID rules: the noise outcome and the uniqueness requirement of rules. When grounding an RMDP which specifies rewards only for a set of goal states, one might in principle apply any of the traditional AI planning methods used for propositional representations (Weld, 1999; Boutilier et al., 1999). Much research within the planning community has focused on deterministic domains and thus cannot be applied straightforwardly in stochastic worlds. A common approach for probabilistic planning, however, is to make the planning problem deterministic and apply deterministic planners (Kuter et al., 2008). Indeed, FF-Replan (Yoon et al., 2007) and its extension using hindsight optimization (Yoon et al., 2008) have shown impressive performance on many probabilistic planning competition domains. The common variant of FF-Replan considers each probabilistic outcome of an action as a separate deterministic action, ignoring the respective probabilities. It then runs the deterministic Fast-Forward (FF) planner (Hoffmann and Nebel, 2001) on the determinized problem. FF uses a relaxation of the planning problem: it ignores the delete effects of actions and applies clever heuristics to prune the search space. FF-Replan outputs a sequence of actions and expected states. Each time an action execution leads to a state which is not in the plan, FF-Replan has to replan, that is, to recompute a new plan from scratch in the current state. The good performance of FF-Replan in many probabilistic domains has been explained by the structure of these problems (Little and Thiébaux, 2007). It has been argued that FF-Replan should be less appropriate in domains in which the probability of reaching a dead-end is non-negligible and where the outcome probabilities of actions need to be taken into account to construct a good policy.

Probabilistic Planning Many participants of the most recent probabilistic planning competition (IPPC, 2008) extend FF-Replan to deal with the probabilities of action outcomes (see the competition website for brief descriptions of the algorithms). The winner of the competition, RFF (Teichteil-Konigsbuch et al., 2010), computes a robust policy offline by generating successive execution paths leading to the goal using FF. The resulting policy has a low probability of failing. LPPFF uses subgoals generated from a determinization of the probabilistic planning problem to divide it into smaller manageable problems. HMDPP's strategy is similar to the all-outcomes-determinization of FF-Replan, but accounts for the probability associated with each outcome. SEH (Wu et al., 2008) extends a heuristic function of FF-Replan to cope with local optima in plans by using stochastic enforced hill-climbing.

A common approach to reasoning in a more general reward-maximization context which avoids explicitly dealing with uncertainty is to build look-ahead trees by sampling successor states. Two algorithms which follow this idea, namely SST (Kearns et al., 2002) and UCT (Kocsis and Szepesvari, 2006), are investigated in this chapter.

Another approach by Buffet and Aberdeen (2009) directly optimizes a parameterized policy using gradient descent. They factor the global policy into simple approximate policies for starting each action and sample trajectories to cope with probabilistic effects.

Planning in an uncertain world has also been investigated in more general contexts (Gray et al., 2000).

Planning using Approximate Inference Instead of sampling state transitions, we propose the planning algorithm PRADA in this chapter which accounts for uncertainty in a principled way using approximate inference. Domshlak and Hoffmann (2007) propose an interesting planning approach which shares ideas with our work. They introduce a probabilistic extension of the FF planner, using complex algorithms for building probabilistic relaxed planning graphs. They construct dynamic Bayesian networks (DBNs) from hand-crafted STRIPS operators and reason about actions and states using weighted model counting. Their DBN representation, however, is inadequate for the type of probabilistic relational rules that we use, for the same reasons why the naive DBN model discussed in Sec. 3.2 is inappropriate. Planning by inference approaches (Toussaint and Storkey, 2006), which have been limited to non-relational domains thus far, spread information also backwards through DBNs and calculate posteriors over actions (resulting in policies over complete state spaces). How to use full planning by inference in relational domains is an open issue which we will briefly explore in this chapter.

Ground Domains with Many Objects All approaches working in the grounded representation have in common that the number of states and actions will grow exponentially with the number of objects. To apply them in domains with very many objects, these approaches need to be combined with complementary methods that reduce the state and action space complexity in relational domains. For instance, one can focus on envelopes of states which are high-utility subsets of the state space (Gardiol and Kaelbling, 2003), or one can exploit the equivalence of actions (Gardiol and Kaelbling, 2007), which is particularly useful in combination with ignoring certain predicates and functions of the relational logic language (Gardiol and Kaelbling, 2008). In this chapter, we investigate an orthogonal approach which grounds the representation only with respect to relevant objects.

4.2 Forward Reasoning

The existing approach for planning with probabilistic relational rules, the *Sparse Sampling Tree* (SST) algorithm (Kearns et al., 2002), reasons in a forward direction by growing full look-ahead trees in the ground domain. The computational burden to plan just a single action with this method in a given situation can be overwhelmingly large due to the very large action space and the stochasticity of the world. In this section, we propose two novel methods for efficient forward reasoning in the ground domain with learned probabilistic relational rules.

First, we apply the existing *Upper Confidence bounds applied to Trees* (UCT) algorithm (Kocsis and Szepesvari, 2006) with NID rules. In contrast to full-grown look-ahead trees, UCT samples actions selectively, thereby cutting suboptimal parts of the tree early. Second, we introduce the *Probabilistic Relational Action-sampling in DBNs planning Algorithm* (PRADA) (Lang and Toussaint, 2009b, 2010b) which uses probabilistic inference in dynamic Bayesian networks to cope with uncertain action outcomes. Instead of growing look-ahead trees with sampled successor states, PRADA applies approximate inference techniques to propagate the effects of actions. In particular, we make two contributions with PRADA: *(i)* We derive an approximate inference method to cope with the state complexity of a time-slice of the ground relational DBN. Thereby, we can efficiently predict the effects of action sequences. *(ii)* For planning based on sampling action-sequences, we propose a sampling distribution for plans which takes predicted state distributions into account.

We evaluate our planning approaches in our simulated complex 3D robot manipulation environment with realistic physics (see Sec. 1.1.1), with an articulated humanoid manipulating objects of different types (see Fig. 4.12). This domain contains billions of world states and a large number of potential actions. We learn NID rules from experience in this environment and apply them with our planning approaches in different planning scenarios of increasing difficulty. Furthermore, we provide results of our approaches on the planning domains of the most recent international probabilistic planning competition.

4.2.1 Planning with Look-Ahead Trees

To plan with NID rules, one can treat the domain described by the relational logic vocabulary as a relational Markov decision process as discussed in Sec. 2.1. In the following, we present two algorithms which employ NID rules as a generative model to build look-ahead trees starting from the initial state. These trees are used to estimate the values of actions and states.

Sparse Sampling Trees

The *Sparse Sampling Tree* (SST) algorithm (Kearns et al., 2002) for MDP planning samples randomly sparse, but full-grown look-ahead trees of states starting with the given state as root. This suffices to compute near-optimal actions for any state of an MDP. Given a planning horizon d and a branching factor b, SST works as follows (see Fig. 4.1): In each tree node (representing a state), *(i)* SST takes all possible actions into account, and *(ii)* for each action it takes b samples from the successor state distribution using a generative model for the transitions, in our case the NID rules defining the transition model $\mathcal{M}$, to build tree nodes at the next level. Values of the tree nodes are computed recursively from

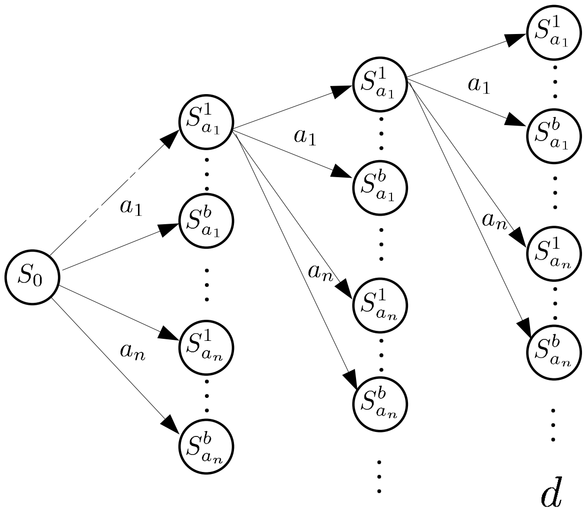


Figure 4.1: The SST planning algorithm samples sparse, but full-grown look-ahead trees to estimate the values of actions and states.

the leaves to the root using the Bellman equation: in a given node, the Q-value of each possible action is estimated by averaging over all values of the b children states for this action; then, the maximizing Q-value over all actions is chosen to estimate the value of the given node. SST has the favorable property that it is independent of the total number of states of the MDP, as it only examines a restricted subset of the state space; but it is exponential in the time horizon taken into account.

Pasula et al. (2007) apply SST for planning with NID rules. When sampling the noise outcome while planning with SST, they assume to stay in the same state, but discount the estimated value. We refer to this adaptation when we speak of SST planning in the remainder of the chapter. If an action does not have a unique covering rule, we use the noisy default rule r_ν to predict its effects. It is always better to perform a $doNothing$ action instead where staying in the same state does not get punished. Hence, in SST planning one can discard all actions for a given state which do not have unique covering rules.

While SST is near-optimal, in practice it is only feasible for very small branching factor b and planning horizon d. Let the number of actions be a. Then the number of nodes at horizon d is $(ba)^d$. (This number can be reduced if the same outcome of a rule is sampled multiple times.) As an illustration, assume we have 10 possible actions per time-step and set parameters $d = 4$ and $b = 4$ (the choice of Pasula et al. in their experiments). To plan a single action for a given state, one has to visit $(10 * 4)^4 = 2,560,000$ states. While smaller choices of b lead to faster planning, they result in a significant accuracy loss in realistic domains. As Kearns et al. note, SST is only useful if no special structure that permits compact representation is available. In Sec. 4.2.2, we will introduce an alternative planning approach based on approximate inference that exploits the structure of NID rules.

Sampling Trees with Upper Confidence Bounds

The *Upper Confidence Bounds applied to Trees* (UCT) algorithm (Kocsis and Szepesvari, 2006) also samples a search tree of subsequent states starting with the current state as root. In contrast to SST which generates b successor states for every action in a state, the idea of UCT is to choose actions selectively in a given state and thus to sample selectively from the successor state distribution. UCT tries to identify large subsets of suboptimal actions early in the sampling procedure and to focus on promising parts of the look-ahead tree instead.

UCT builds its look-ahead tree by repeatedly sampling simulated *episodes* from the initial state using a generative model, for instance the transition model $\mathcal{M}$. An episode is a sequence of states, rewards and actions until a limited horizon d: $s_0, r_0, a_1, s_1, r_1, a_2 \ldots s_d, r_d$. After each simulated episode, the values of the tree nodes (representing states) are updated online and the simulation policy is improved with respect to the new values. As a result, a distinct value is estimated for each state-action pair in the tree by Monte-Carlo simulation.

More precisely, UCT follows the following policy in tree node s: If there exist actions from s which have not been explored yet, then UCT samples one of these using a uniform distribution. Otherwise, if all actions have been explored at least once, then UCT selects the action that maximizes an upper confidence bound $Q^{\triangledown}_{UCT}(s,a)$ on the estimated action value $Q_{UCT}(s,a)$,

$$Q^{\triangledown}_{UCT}(s,a) \;=\; Q_{UCT}(s,a) + c\sqrt{\frac{\log n_s}{n_{s,a}}}\ , \tag{4.1}$$

$$\pi_{UCT}(s) \;=\; \underset{a}{\text{argmax}}\ Q^{\triangledown}_{UCT}(s,a)\ , \tag{4.2}$$

where $n_{s,a}$ counts the number of times that action a has been selected from state s, and n_s counts the total number of visits to state s, $n_s = \sum_a n_{s,a}$. The bias parameter c defines the influence of the number of previous action selections and thereby controls the extent of the upper confidence bound.

At the end of an episode, the value of each encountered state-action pair (s_t, a_t), $0 \le t < d$, is updated using the total discounted rewards:

$$n_{s_t,a_t} \;\leftarrow\; n_{s_t,a_t} + 1\ , \tag{4.3}$$

$$Q_{UCT}(s_t,a_t) \;\leftarrow\; Q_{UCT}(s_t,a_t) + \frac{1}{n_{s_t,a_t}}\Big[\sum_{t'=t}^{d} \gamma^{t'-t} r_{t'} - Q_{UCT}(s_t,a_t)\Big]\ . \tag{4.4}$$

The policy of UCT implements an exploration-exploitation tradeoff for planning: It balances between exploring currently suboptimal-looking actions that have been selected seldom thus far and exploiting currently best-looking actions to get more precise estimates of their values. The total number of episodes controls the accuracy of UCT's estimates and has to be balanced with its overall running time.

UCT has achieved remarkable results in challenging domains such as the game of Go (Gelly and Silver, 2007). To the best of our knowledge, we are the first to apply UCT for planning in stochastic relational domains, using NID rules as a generative model. We adapt UCT to cope with noise outcomes in the same fashion as SST: we assume to stay in the same state and discount the obtained rewards. Thus, UCT takes only actions with unique covering rules into account, for the same reasons as SST does.

4.2.2 Planning with Approximate Inference

Uncertain action outcomes characterize compact transition models of complex environments, but make planning in relational domains substantially more difficult. The sampling-based approaches discussed in the previous subsection tackle this problem by repeatedly generating samples from the outcome distribution of an action using the transition probabilities of an MDP. This leads to look-ahead trees that easily blow up with the planning horizon. Instead of sampling successor states, one may maintain a distribution over states, a so-called "belief". In the following, we introduce an approach for planning in ground stochastic relational domains which propagates beliefs over states in the sense of state monitoring. We develop an approximate inference method to efficiently propagate beliefs in the compact graphical model for NID rules of the PRADA-DBN type described in Sec. 3.2 and shown again in Fig. 4.2 for convenience. Based on our approximate inference method, we describe our *Probabilistic Relational Action-sampling in DBNs planning Algorithm* (PRADA) (Lang and Toussaint, 2009b, 2010b), which samples action-sequences in an informed way and evaluates these using approximate inference in DBNs. An example is presented to illustrate the reasoning of PRADA. Finally, we describe some theoretical considerations concerning PRADA, compare it to the approaches of the previous subsection, SST and UCT, and introduce a simple extension of PRADA.

Approximate Inference

We are interested in inferring posterior state distributions $P(\mathrm{s}^t \,|\, \mathbf{a}^{0:t-1})$ given the sequence of previous actions (where we omit conditioning on the initial state for simplicity) in our graphical model type called PRADA-DBN (Fig. 4.2). Exact inference is intractable in this graphical model. When constructing a junction tree, we will get cliques that comprise whole Markov slices (all variables representing the state at a certain timestep). Consider eliminating all state variables $\mathbf{S}^{t+1}$. Due to moralization, the outcome variable O will be connected to all state variables in $\mathbf{S}^t$. After elimination of O, all variables in $\mathbf{S}^t$ will form a clique. Thus, we have to make use of approximate inference techniques. General loopy belief propagation (LBP) is unfeasible due to the deterministic dependencies in small cycles which inhibit convergence. We also conducted some preliminary tests in small networks with a damping factor, but without success. It is an interesting open question whether there are ways to alternate between propagating deterministic information and running LBP on the remaining parts of the network, for example, whether methods such as MC-SAT (Poon and Domingos, 2007) can be successfully applied in decision-making contexts as ours. Here, we propose a different approximate inference scheme using a factored frontier (FF). The FF algorithm (Murphy and Weiss, 2001) describes an inference procedure that computes exact marginals in the

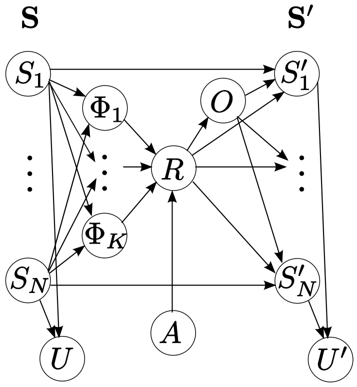


Figure 4.2: The dynamic Bayesian network "PRADA-DBN" used by our planning algorithm PRADA (see Sec. 3.2).

next time-step subject to a factored approximation of the previous time-step. We take up this idea in our approximate inference procedure to calculate the beliefs over successor states: we exploit the model structure of our PRADA-DBNs to come up with formulas for the state attribute marginals which take into account the central high-dimensional variables for actions, rules and outcomes. In contrast to LBP and the FF algorithm, we do not propagate information backwards: our approach does not condition on rewards (as in full planning by inference) and samples actions, so that backward reasoning is uninformative.

We follow the idea of the factored frontier (FF) algorithm (Murphy and Weiss, 2001) and approximate the belief with a product of marginals:

$$P(\mathbf{s}^t \,|\, \mathbf{a}^{0:t-1}) \;\approx\; \prod_i P(s_i^t \,|\, \mathbf{a}^{0:t-1}) \; . \tag{4.5}$$

We define

$$\alpha(s_i^t) \;:=\; P(s_i^t \,|\, \mathbf{a}^{0:t-1}) \;\; \text{and} \tag{4.6}$$

$$\alpha(\mathbf{s}^t) \;:=\; P(\mathbf{s}^t \,|\, \mathbf{a}^{0:t-1}) \;\approx\; \prod_{i=1}^{N} \alpha(s_i^t) \tag{4.7}$$

and derive a FF filter for the PRADA-DBN model. In the following, we focus on the mathematical derivations. An illustrative example will be provided later. We are interested in inferring the state distribution at time $t+1$ given an action sequence $\mathbf{a}^{0:t}$ and calculate the marginals of the state attributes as

$$\alpha(s_i^{t+1}) = P(s_i^{t+1} \,|\, \mathbf{a}^{0:t}) \tag{4.8}$$

$$= \sum_{r^t} P(s_i^{t+1} \,|\, r^t, \mathbf{a}^{0:t-1}) \; P(r^t \,|\, \mathbf{a}^{0:t}) \; . \tag{4.9}$$

In Eq. (4.9), we use all rules for prediction, weighted by their respective posteriors $P(r^t \,|\, \mathbf{a}^{0:t})$. This reflects the fact that depending on the state we use different rules to model the same action. The weight $P(r^t \,|\, \mathbf{a}^{0:t})$ is 0 for all rules not modeling action a^t. For the remaining rules which do model a^t, the weights correspond to the posterior over those parts of the state space where the according rule is used for prediction.

We compute the first term in (4.9) as

$$\begin{aligned} P(s_i^{t+1} \,|\, r^t, \mathbf{a}^{0:t-1}) &= \sum_{s_i^t} P(s_i^{t+1} \,|\, r^t, s_i^t)\; P(s_i^t \,|\, r^t, \mathbf{a}^{0:t-1}) \\ &\approx \sum_{s_i^t} P(s_i^{t+1} \,|\, r^t, s_i^t)\; \alpha(s_i^t)\;. \end{aligned} \tag{4.10}$$

Here, we sum over all possible values of the variable S_i at the previous time-step t. Intuitively, we take into account all potential "pasts" to arrive at the value s_i^{t+1} at the next time-step. The resulting term $P(s_i^{t+1} \,|\, r^t, s_i^t)$ enables us to easily predict the probabilities at the next time-step as discussed below. Each such prediction is weighted by the marginal $\alpha(s_i^t)$ of the respective previous value. The approximation in (4.10) assumes that s_i^t is conditionally independent of r^t. This is not true in general as the choice of a rule for prediction depends on the current state and thus also on attribute S_i. To improve on this approximation one can examine whether s_i^t is part of the context of r^t: if this is the case, we can infer the state of s_i^t from knowing r^t. However, we found our approximation to be sufficient.

As one would expect, we calculate the successor state distribution $P(s_i^{t+1} \,|\, r^t, s_i^t)$ by taking the different outcomes o of r^t into account weighted by their respective probabilities $P(o \,|\, r^t)$,

$$P(s_i^{t+1} \,|\, r^t, s_i^t) = \sum_{o} P(s_i^{t+1} \,|\, o, r^t, s_i^t)\; P(o \,|\, r^t)\;. \tag{4.11}$$

This shows us how to update the belief over S_i^{t+1} if we predict with rule r^t. $P(s_i^{t+1} \,|\, o, r^t, s_i^t)$ is a deterministic distribution. If o changes the value of S_i, s_i^{t+1} is set accordingly. Otherwise, the value s_i^t persists.

Let's turn to the computation of the second term in Eq. (4.9), $P(r^t \,|\, \mathbf{a}^{0:t})$, the posterior over rules. The trick is to use the context variables $\boldsymbol{\Phi}$ and to exploit the assumption that a rule r models the state transition if and only if it uniquely covers $(a^t, \mathbf{s}^t)$, which is indicated by an appropriate assignment of the $\boldsymbol{\Phi}$. This can then be further reduced to an expression involving only the marginals $\alpha(\cdot)$. We start with

$$\begin{aligned} P(R^t\!=\!r \,|\, \mathbf{a}^{0:t}) &= \sum_{\boldsymbol{\phi}^t} P(R^t\!=\!r \,|\, \boldsymbol{\phi}^t, \mathbf{a}^{0:t})\; P(\boldsymbol{\phi}^t \,|\, \mathbf{a}^{0:t}) \\ &= I(r \!\in\! \Gamma(a^t))\; P\left(\Phi_r^t\!=\!1, \bigwedge_{r' \in \Gamma(a^t) \setminus \{r\}} \Phi_{r'}^t\!=\!0 \,|\, \mathbf{a}^{0:t-1}\right) \\ &= I(r \!\in\! \Gamma(a^t))\;\; P(\Phi_r^t\!=\!1 \,|\, \mathbf{a}^{0:t-1})\; P\left(\bigwedge_{r' \in \Gamma(a^t) \setminus \{r\}} \Phi_{r'}^t\!=\!0 \,|\, \Phi_r^t\!=\!1, \mathbf{a}^{0:t-1}\right)\;. \end{aligned} \tag{4.12}$$

To simplify the summation over $\boldsymbol{\phi}^t$, we only have to consider the unique assignment of the context variables when r is used for prediction: provided it models the action, as indicated by $I(r \in \Gamma(a^t))$, this is the case if its context Φ^t_r holds, while the contexts $\Phi^t_{r'}$ of all other "competing" rules r' for action a^t do not hold.

We calculate the second term in (4.12) by summing over all states s as

$$P(\Phi^t_r \!=\! 1 \,|\, \mathbf{a}^{0:t-1}) = \sum_{\mathbf{s}^t} P(\Phi^t_r \!=\! 1 \,|\, \mathbf{s}^t)\ \alpha(\mathbf{s}^t) \approx \sum_{\mathbf{s}^t} P(\Phi^t_r \!=\! 1 \,|\, \mathbf{s}^t)\ \prod_j \alpha(s^t_j) \tag{4.13}$$

$$= \prod_{j \in \pi(\Phi^t_r)} \alpha(S^t_j \!=\! s_{r,j}) \quad . \tag{4.14}$$

The approximation in (4.13) is the FF assumption. In (4.14), $\mathbf{s}_r$ denotes the configuration of the state variables according to the context of r like in (3.5). We sum out all variables not in the context of r. Only the variables in r's context remain: the terms $\alpha(S^t_j = s_{r,j})$ correspond to the probabilities of the respective literals.

The third term in (4.12) is the joint posterior over the contexts of the competing rules r' given that r's context already holds. We are interested in the situation where none of these other contexts hold. We calculate this as

$$P\left(\bigwedge_{r' \in \Gamma(a^t) \setminus \{r\}} \Phi^t_{r'} \!=\! 0 \,|\, \Phi^t_r \!=\! 1, \mathbf{a}^{0:t-1} \right) \approx \prod_{r' \in \Gamma(a^t) \setminus \{r\}} P(\Phi^t_{r'} \!=\! 0 \,|\, \Phi^t_r \!=\! 1, \mathbf{a}^{0:t-1}) \quad , \tag{4.15}$$

approximating it by the product of the individual posteriors. The latter are computed as

$$P(\Phi^t_{r'} \!=\! 0 \,|\, \Phi^t_r \!=\! 1, \mathbf{a}^{0:t-1}) = \sum_{\mathbf{s}^t} P(\Phi^t_{r'} \!=\! 0 \,|\, \mathbf{s}^t)\ P(\mathbf{s}^t \,|\, \Phi^t_r \!=\! 1, \mathbf{a}^{0:t-1}) \tag{4.16}$$

$$\approx \begin{cases} 1.0 & \text{if} \quad \Phi_r \wedge \Phi_{r'} \rightarrow \bot \\ 1.0 - \prod_{\substack{i \in \pi(\Phi^t_{r'}), \\ i \notin \pi(\Phi^t_r)}} \alpha(S^t_i \!=\! s_{r',i}) & \text{otherwise} \end{cases} \quad , \tag{4.17}$$

where the if-condition expresses a logical contradiction of the contexts of r and r'. If their contexts contradict, then r''s context will surely not hold given that r's context holds. Otherwise, we know that the state attributes apppearing in the contexts of both r and r' do hold as we condition on $\Phi_r = 1$. Therefore, we only have to examine the remaining state attributes of r''s context. Again, we approximate this posterior with the FF marginals.

Finally, we compute the reward probability straightforwardly as

$$P(U^t \!=\! 1 \,|\, \mathbf{a}^{0:t-1}) = \sum_{\mathbf{s}_t} P(U^t \!=\! 1 \,|\, \mathbf{s}_t) P(\mathbf{s}^t \,|\, \mathbf{a}^{0:t-1}, \mathbf{s}^0) \approx \prod_{i \in \pi(U^t)} \alpha(S^t_i \!=\! \tau_i) \ , \tag{4.18}$$

where $\boldsymbol{\tau}$ denotes the configuration of state variables corresponding to the planning goal as in (3.10). As above, the summation over states is simplified by the FF assumption resulting in a product of the marginals of the required state attributes.

The overall computational costs of propagating the effects of an action are quadratic in the number of rules for this action (for each such rule we have to calculate the probability that none of the others applies) and linear in the maximum numbers of context literals and manipulated state attributes of those rules.

Our inference framework requires an approximation for the distribution $P(s' \mid \Omega_{r,0}, s)$ (cf. Eq. (3.2)) to cope with the noise outcome of NID rules. From the training data used to learn rules, we estimate which predicates and functions change value over time as follows: let $\mathbf{S}_c \subset \mathbf{S}$ contain the corresponding variables. We estimate for each rule r the average number N^r of changed state attributes when the noise outcome applies. Due to our factored frontier approach, we can consider the noise effects for each variable independently. We approximate the probability that $S_i \in \mathbf{S}_c$ changes in r's noise outcome by $\frac{N^r}{|\mathbf{S}_c|}$. In case of change, all changed values of S_i have equal probability.

Planning

The PRADA-DBN representation in Fig. 4.2 together with the approximate inference method described before enable us to derive a novel planning algorithm for stochastic relational domains: The *Probabilistic Relational Action-sampling in DBNs planning Algorithm* (PRADA) plans by sampling action sequences in an informed way based on predicted beliefs over states and evaluating these action sequences using approximate inference.

More precisely, we sample sequences of actions $\mathbf{a}^{0:T-1}$ of length T. For $0 < t \leq T$, we infer the posteriors over states $P(\mathbf{s}^t \mid \mathbf{a}^{0:t-1}, \mathbf{s}^0)$ and rewards $P(u^t \mid \mathbf{a}^{0:t-1}, \mathbf{s}^0)$ (in the sense of filtering or state monitoring). Then, we calculate the value of an action sequence with a discount factor $0 < \gamma < 1$ as

$$Q(\mathbf{s}^0, \mathbf{a}^{0:T-1}) := \sum_{t=0}^{T} \gamma^t P(U^t = 1 \mid \mathbf{a}^{0:t-1}, \mathbf{s}^0) \; . \tag{4.19}$$

We choose the first action of the best sequence $\mathbf{a}^* = \text{argmax}_{\mathbf{a}^{0:T-1}} Q(\mathbf{a}^{0:T-1}, s^0)$, if its value exceeds a certain threshold θ (for instance, $\theta = 0$). Otherwise, we continue sampling action-sequences until either an action is found or planning is given up. The quality of the found plan can be controlled by the total number of action-sequence samples and has to be traded off with the time that is available for planning.

We aim for a strategy to sample good action sequences with high probability. We propose to choose with equal probability among the actions that have a unique covering rule for the current state. Thereby, we avoid the use of the noisy default rule r_ν which models action effects as noise and is thus of poor use in planning. For the action at time t, PRADA samples from the distribution

$$P^t_{sample}(a) \propto \sum_{r \in \Gamma(a)} P\left(\phi^t_r = 1, \bigwedge_{r' \in \Gamma(a) \setminus \{r\}} \phi^t_{r'} = 0 \mid \mathbf{a}^{0:t-1} \right) . \tag{4.20}$$

This is a sum over all rules for action a: for each such rule we add the posterior that it is the unique covering rule, that is that its context ϕ^t_r holds, while the contexts $\phi^t_{r'}$ of the competing rules r' do not hold. This sampling distribution takes the current state distribution into account. Thus, the probability to sample an action sequence $\mathbf{a}$ predicting the state sequence $s_0, \ldots, s_T$ depends on the likelihood of the state sequence given $\mathbf{a}$: the more likely the required outcomes are, the more likely the next actions will be sampled (see Appendix C.3). Using this policy, PRADA does not miss actions which SST and UCT explore, as the following proposition states (proof in Appendix A.1).

Proposition 4.2.1 *The set of action sequences PRADA samples with non-zero probability is a super-set of the ones of SST and UCT.*

In our experiments, we replan after each action is executed without reusing the knowledge of previous time-steps. This simple strategy helps to get a general impression of PRADA's planning performance and complexity. Other strategies are easily conceivable. For instance, one might execute the entire sequence without replanning, trading off faster computation times with a potential loss in the achieved reward. In noisy environments, it might seem a better strategy to combine the reuse of previous plans with replanning. For instance, one could omit the first action of the previous plan, which has just been executed, and examine the suitability of the remaining actions in the new state. While we consider only the single best action sequence, in many planning domains it might also be beneficial to marginalize over all sequences with the same first action. For instance, an action a_1 might lead to a number of reasonable sequences, none of which are the best, while another action a_2 is the first of one very good sequence, but also many bad ones—in which case one might favor a_1.

Illustrative Example

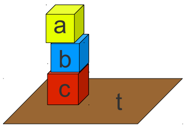
Let us consider the small planning problem in Table 4.1 to illustrate the reasoning procedure of PRADA. Our domain is a noisy robot manipulation domain represented by predicates $table(X)$, $cube(X)$, $on(X, Y)$, $inhand(X)$ and $clear(X) \equiv \forall Y. \neg on(Y, X)$ where a robot can perform two types of actions: it may either lift a cube X by means of action $grab(X)$ or put the cube which is held in hand on top of another object X using $puton(X)$. The start state s_0 shown in 4.1(a) contains three cubes a, b and c stacked in a pile on table t. The goal shown in 4.1(b) is to get the middle cube b on-top of the top cube a. Our transition model provides three abstract NID rules to predict action effects, shown in Table 4.1(c). Only the first rule has uncertain outcomes: it models to grab an object which is below another object. In contrast, grabbing a clear object (Rule 2) and putting an object somewhere (Rule 3) always leads to the same successor state.

First, PRADA constructs a DBN to represent the planning problem. For this purpose, it computes the grounded rules with respect to the objects $\mathcal{O} = \{a, b, c, t\}$ shown in 4.1(d). Most potential grounded rules can be ignored: one can deduce from the abstract rules which predicates are changeable. In combination with the specifications in s_0, this prunes most grounded rules. For instance, we know from s_0 that t is the table. Thus, no ground rule with action argument $X = t$ needs to be constructed as all rules require $cube(X)$.

Table 4.1: Example of PRADA's factored frontier inference

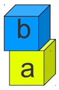
(a) Start state

$s_0 = \{on(a,b), on(b,c), on(c,t), cube(a), cube(b), cube(c), table(t)\}$



(b) Goal

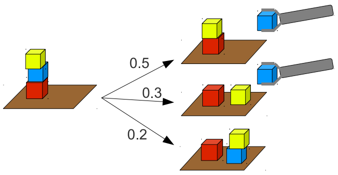
$\tau = \{on(b,a)\}$



(c) Abstract NID rules with example situations

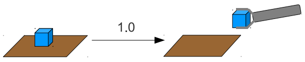
Rule 1:

$$grab(X): \quad on(Y,X),\ on(X,Z),\ cube(X),\ cube(Y),\ table(T)$$
$$\rightarrow \begin{cases} 0.5: & inhand(X),\ on(Y,Z),\ \neg on(Y,X),\ \neg on(X,Z) \\ 0.3: & inhand(X),\ on(Y,T),\ \neg on(Y,X),\ \neg on(X,Z) \\ 0.2: & on(X,T),\ \neg on(X,Z) \end{cases}$$



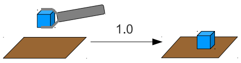
Rule 2:

$$grab(X): \quad cube(X),\ clear(X),\ on(X,Y)$$
$$\rightarrow \{\ 1.0: \quad inhand(X),\ \neg on(X,Y)$$



Rule 3:

$$puton(X): \quad inhand(Y),\ cube(Y)$$
$$\rightarrow \{\ 1.0: \quad on(Y,X),\ \neg inhand(X)$$



(d) Grounded NID rules

Grounded Rule	Action	Substitution
$(1, a/bbt)$	$grab(a)$	$\{X \to a, Y \to b, Z \to b, T \to t\}$
$(1, a/bct)$	$grab(a)$	$\{X \to a, Y \to b, Z \to c, T \to t\}$
...		
$(1, c/bbt)$	$grab(c)$	$\{X \to c, Y \to b, Z \to b, T \to t\}$
$(2, a/b)$	$grab(a)$	$\{X \to a, Y \to b\}$
$(2, a/c)$	$grab(a)$	$\{X \to a, Y \to c\}$
$(2, a/t)$	$grab(a)$	$\{X \to a, Y \to t\}$
...		
$(2, c/t)$	$grab(c)$	$\{X \to c, Y \to t\}$
$(3, a/b)$	$puton(a)$	$\{X \to a, Y \to b\}$
$(3, a/c)$	$puton(a)$	$\{X \to a, Y \to c\}$
...		
$(3, t/c)$	$puton(t)$	$\{X \to a, Y \to c\}$

(e) Inferred posteriors in PRADA's FF inference for action-sequence $(grab(b), puton(a))$

	$t=0$	$t=1$	$t=2$
State marginals α			
$on(a,b)$	1.0	0.2	0.2
$on(a,c)$	0.0	0.5	0.5
$on(a,t)$	0.0	0.3	0.3
$on(b,a)$	0.0	0.0	0.8
$on(b,c)$	1.0	0.0	0.0
$on(b,t)$	0.0	0.2	0.2
$on(c,t)$	1.0	1.0	1.0
$inhand(b)$	0.0	0.8	0.16
$clear(a)$	1.0	1.0	0.2
$clear(b)$	0.0	0.8	0.8
$clear(c)$	0.0	0.5	0.5
Goal U	0.0	0.0	0.8
$P(\Phi \mid \mathbf{a}^{0:t-1})$			
$\Phi_{(1,b/act)}$	1.0	0.0	
$\Phi_{(1,b/att)}$	0.0	0.04	
$\Phi_{(1,c/btt)}$	1.0	0.5	
$\Phi_{(2,a/b)}$	1.0	0.2	
$\Phi_{(2,a/c)}$	0.0	0.5	
$\Phi_{(2,a/t)}$	0.0	0.3	
$\Phi_{(2,b/t)}$	0.0	0.16	
$\Phi_{(2,c/t)}$	0.0	0.5	
$\Phi_{(3,a/b)}$	0.0	0.8	
$\Phi_{(3,c/b)}$	0.0	0.8	
$\Phi_{(3,t/b)}$	0.0	0.8	
Unique rule			
$(1, b/act)$	1.0	0.0	
$(1, b/att)$	0.0	0.0336	
$(1, c/att)$	0.0	0.25	
$(1, c/btt)$	1.0	0.0	
$(2, a/b)$	1.0	0.07	
$(2, a/c)$	0.0	0.28	
$(2, a/t)$	0.0	0.12	
$(2, b/t)$	0.0	0.154	
$(2, c/t)$	0.0	0.25	
$(3, a/b)$	0.0	0.8	
$(3, c/b)$	0.0	0.8	
$(3, t/b)$	0.0	0.8	
Action coverage			
$grab(a)$	1.0	0.47	
$grab(b)$	**1.0**	0.187	
$grab(c)$	1.0	0.5	
$puton(a)$	0.0	**0.8**	
$puton(c)$	0.0	0.8	
$puton(t)$	0.0	0.8	
Sample distribution			
$P_{sample}(grab(a))$	0.33	0.132	
$P_{sample}(grab(b))$	**0.33**	0.0526	
$P_{sample}(grab(c))$	0.33	0.141	
$P_{sample}(puton(a))$	0.0	**0.225**	
$P_{sample}(puton(c))$	0.0	0.225	
$P_{sample}(puton(t))$	0.0	0.225	
$P(R^t = r^t \mid \mathbf{a}^{0:t})$			
$R^t = (1, b/act)$	1.0	0.0	
$R^t = (3, a/b)$	0.0	0.8	
$R^t = 0$	0.0	0.2	

Based on the DBN, PRADA samples action-sequences and evaluates their expected rewards. In the following, we investigate this procedure for the sampling of action-sequence $(grab(b), puton(a))$. Table 4.1(e) presents the inferred values of the DBN variables and other auxiliary quantities. The marginals α (Eq. (4.6)) of the state variables at $t = 0$ are set deterministically according to s_0. We calculate the posteriors over context variables $P(\Phi \,|\, \mathbf{a}^{0:t-1})$ according to Eq. (4.14). In our example, at $t = 0$ there is one rule with probability 1.0 for each of the actions $grab(a)$, $grab(b)$ and $grab(c)$. In contrast, there are no rules with non-zero probability for the various $puton(\cdot)$ actions. By the help of Eq. (4.17), we calculate the probability of each rule r to be the unique covering rule for the respective action (listed under *Unique rule*; note that we do not condition on a fixed action a^t thus far): this is the case if context Φ_r of r holds, while all contexts $\Phi_{r'}$ of the competing rules r' for the same action do not hold. At $t = 0$, this is the same as the posterior of Φ_r alone. The resulting probabilities are used to calculate the sampling distribution of Eq. (4.20): first, we compute the probability for each action to have a unique covering rule which is a simple sum over probabilities of the previous step (listed under *Action coverage* in the table); then, we normalize these values to get a sampling distribution $P_{sample}(\cdot)$. At $t = 0$, this results in a sampling distribution which is uniform over the three actions with unique rules. Assume we sample $a^0 = grab(b)$ (grabbing blue cube b). Variable R specifies the ground rules to use for predicting the state marginals at the next time-step. We can infer its posterior according to Eq. (4.12). Here, $P(R^0 = (1, b/act) \,|\, a^0) = 1.0$.

Things get more interesting at $t = 1$. Here, we observe the effects of the factored frontier. For instance, consider calculating the posterior over context Φ_r for ground rule $r = (1, b/att)$ (grabbing blue cube b in the context that it is on the table t and below the yellow cube a) using Eq. (4.14),

$$\begin{aligned} P(\Phi_{(1,b/att)} \,|\, a^0) &\approx \alpha(on(a,b)) \cdot \alpha(on(b,t)) \cdot \alpha(cube(a)) \cdot \alpha(cube(b)) \cdot \alpha(table(t)) \\ &= 0.2 \cdot 0.2 \cdot 1.0 \cdot 1.0 \cdot 1.0 = 0.04. \end{aligned}$$

In contrast, the exact value is $P(\Phi_{(1,b/att)} \,|\, a^0) = 0.2$, according to the third outcome of abstract Rule 1 used to predict a^0. The imprecision is due to ignoring the correlations: FF regards the marginals for $on(a, b)$ and $on(b, t)$ as independent, while in fact they are fully correlated.

At $t = 1$, the action $grab(a)$ has three ground rules with non-zero context probabilities (grabbing a from either b, c or t). This is due to the three different outcomes of abstract Rule 1. As an example, we calculate the probability of rule $(2, a/c)$ (grabbing a from c) to be the unique covering rule for $grab(a)$ at $t = 1$ as

$$\begin{aligned} &P(\Phi_{(2,a/c)}, \neg\Phi_{(2,a/b)}, \neg\Phi_{(2,a/t)} \,|\, a^0) \\ &\qquad \approx P(\Phi_{(2,a/c)} \,|\, a^0) \cdot (1. - P(\Phi_{(2,a/b)} \,|\, a^0)) \cdot (1. - P(\Phi_{(2,a/t)} \,|\, a^0)) \\ &\qquad = 0.5 \cdot (1. - 0.2) \cdot (1. - 0.3) = 0.28 \;. \end{aligned}$$

After some more calculations, we determine the sampling distribution at $t = 1$. Assume we sample action $puton(a)$. This results in rule $(3/a, b)$ (putting b on a) being used for prediction with 0.8 probability—since this is its probability to be the unique covering rule for action $puton(a)$. The remaining mass 0.2 of the posterior is assigned to those parts of the state space where no unique covering rule is available for $puton(a)$. In this case, we use the default rule $R = 0$ (corresponding to not performing the action) so that with probability 0.2 the values of the state variables persist.

Finally, let us infer the marginals at $t = 2$ using Eq. (4.9). As an example, we calculate $\alpha(inhand(b)^{t=2})$. Let $i(b)$ be brief for $inhand(b)$. We sum over the ground rules $r^{t=1}$ taking the potential values $i(b)^{t=1}$ and $\neg i(b)^{t=1}$ at the previous time-step $t = 1$ into account,

$$\begin{aligned}\alpha(i(b)^{t=2}) \approx \sum_{r^{t=1}} P(r^{t=1} \,|\, \mathbf{a}^{0:1}) \; (\; P(i(b)^{t=2} \,|\, r^{t=1}, \neg i(b)^{t=1}) \; \alpha(\neg i(b)^{t=1}) \\ + \;\; P(i(b)^{t=2} \,|\, r^{t=1}, i(b)^{t=1}) \; \alpha(i(b)^{t=1}) \;) \\ = 0.8 \; (0.0 * 0.2 + 0.0 * 0.8) \; + \; 0.2 \; (0.0 * 0.2 + 1.0 * 0.8) = 0.16 \; .\end{aligned}$$

As discussed above, only the ground rule $(3/a, b)$ (first summand) and the default rule (second summand) play a role in this prediction. In effect, the belief that b is inhand decreases from 0.8 to 0.16 after having tried to put b on a, as expected. Similarly, we calculate the posterior of $on(b, a)$ as 0.8. This is also the expected probability to reach the goal when performing the actions $grab(b)$ and $puton(a)$. (Here, PRADA's inferred value coincides with the true posterior.)

For comparison, the probability to reach the goal is 1.0 when performing the actions $grab(a)$, $puton(t)$, $grab(b)$ and $puton(a)$, that is, when we clear b before we grab it. This plan is safer, as it has higher probability, but takes more actions.

Theoretical Considerations concerning PRADA

We provide some basic theoretical considerations concerning PRADA in Appendix C:

- We prove the simple result that PRADA almost surely finds the optimal action sequence with an increasing number of samples provided it uses exact inference (instead of the approximate inference procedure described above based on a factored frontier) (Appendix C.1).
- We present sufficient conditions under which the inference based on a factored frontier computes exact posteriors (Appendix C.2).
- We claim that PRADA's sampling strategy exploits the assumption that rewards are probable if the optimal action sequence $\mathbf{a}^*$ is performed, that is, that $P(u \,|\, \mathbf{a}^*)$ is large; we provide some analytical and empirical results in toy scenarios to support this claim (Appendix C.3).

Comparison of the Forward Reasoning Approaches

The most prominent difference between the presented forward reasoning approaches is in their way to account for the stochasticity of action effects. On the one hand, SST and UCT repeatedly take samples from successor state distributions and estimate the value of an action by building look-ahead trees. On the other hand, PRADA maintains beliefs over states and propagates indetermistic action effects forward. More precisely, PRADA and SST follow opposite approaches: PRADA samples actions and calculates the posteriors over successor states by (approximate) probabilistic inference, while SST considers all actions (and thus is exact in its action search) and samples state transitions to approximate the posteriors. The price for considering all actions is SST's overwhelmingly large computational cost. UCT remedies this issue and samples action sequences and thus state transitions selectively: it uses previously sampled episodes to build upper confidence bounds on the estimates for action values in specific states, which are used to adapt the policy for the next episode. It is not straightforward to translate this adaptive policy to PRADA since PRADA works on beliefs over states instead of states directly. Therefore, we chose the simple policy for PRADA to sample randomly from all actions with a unique covering rule in a state (in the form of a sampling distribution to account for beliefs over states).

PRADA returns a whole plan that will transform the world state into one where the goal is fulfilled with a probability exceeding a given threshold θ, in the spirit of conformant planning or probabilistic planning with no observability (Kushmerick et al., 1995). Due to their outcome-sampling, SST and UCT cannot return such a plan in a straightforward way. Instead, they provide a policy for many successor states based on their estimates of the action-values in their look-ahead tree. The estimates of states deeper in the tree are less reliable as they have been built from less episodes. If an action has been executed and a new state is observed, these estimates can be reused. Thus far, PRADA does not take any knowledge gained in previous action-sequence samples into account to adapt its policy. An elegant way to achieve this and to better exploit goal knowledge might use backpropagation through our DBNs to plan completely by inference (Toussaint and Storkey, 2006). This is not straightforward to do in a principled way in the large state and action spaces of relational domains; we will explore it in Sec. 4.3 and Sec. 4.5. Alternatively, PRADA could give high weight to the second action of the previous best plan. Below, we introduce another simple way called A-PRADA to make use of previous episodes to find better plans.

PRADA can afford its simple action-sampling strategy as it evaluates large numbers of action-sequences efficiently and does not have to grow look-ahead trees to account for indeterministic effects. This points at an important difference, illustrated in Fig. 4.3: all three algorithms are faced with search spaces of action sequences which are exponential in the horizon. To calculate the value of a *given* action sequence, however, SST and UCT still need exponential time due to their outcome sampling. In contrast, PRADA propagates the state transitions forward and thus is linear in the horizon.

Like all approximate planning algorithms, neither SST, UCT nor PRADA can be expected to perform ideally in all situations. SST and UCT sample action outcomes and hence face problems if important outcomes only have small probability. For instance, consider an agent that wants to escape a room with two locked doors. If it hits the first door which is made of wood it has a chance of 0.05 to break it and escape. The second

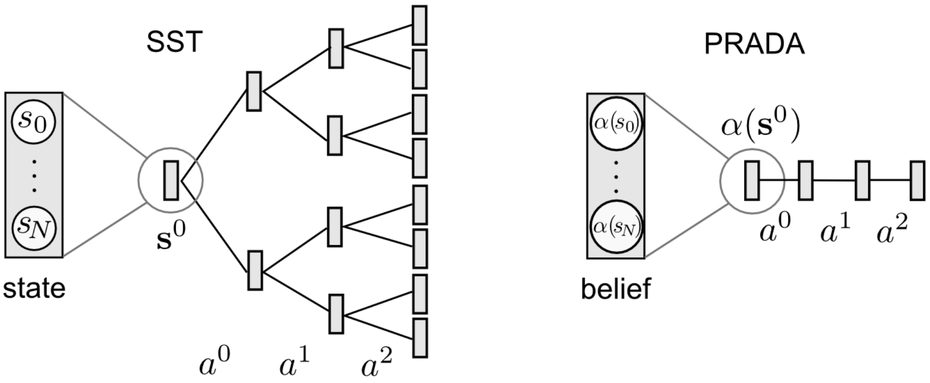


Figure 4.3: *Two different methods to account for stochastic actions.* To evaluate a *given* action sequence $\mathbf{a}$, lookahead-tree methods like SST and UCT sample from the successor state distribution $P(\mathbf{s}' \,|\, \mathbf{s}, a)$ for each state; here, two samples are used. This results in a lookahead-tree which is exponential in the length of $\mathbf{a}$. In contrast, PRADA propagates beliefs $\alpha(\mathbf{s})$ over states and is only linear in the length of $\mathbf{a}$. Maintaining a belief, however, is exponential in the number of state attributes; this necessitates appropriate approximate inference techniques.

door is made of iron and has only a chance of 0.001 to break. SST and UCT may take a very long time to detect that it is 50 times better to repeatedly hit the wooden door. In contrast, PRADA recognizes this immediately after having reasoned about each of the actions once as it takes all outcomes into account. On the other hand, in PRADA's approximate inference procedure the correlations among state variables get lost while SST and UCT preserve them as they sample complete successor states. This can impair PRADA's planning performance in situations where correlations are crucial. Consider the following simple domain with two state attributes a and b. The agent can choose from two actions modeled by the rules

$$action1: \quad - \quad \rightarrow \quad \begin{cases} 0.5 & : \quad a,\ b \\ 0.5 & : \quad \neg a,\ \neg b \end{cases} \text{, and}$$

$$action2: \quad - \quad \rightarrow \quad \begin{cases} 0.5 & : \quad a,\ \neg b \\ 0.5 & : \quad b,\ \neg a \end{cases} .$$

The goal is to make both attributes either true or false, i.e., $\boldsymbol{\tau} = (a \wedge b) \vee (\neg a \wedge \neg b)$. For both actions, the resulting marginals will be $\alpha(a) = 0.5$, $\alpha(\neg a) = 0.5$, $\alpha(b) = 0.5$ and $\alpha(\neg b) = 0.5$. Due to its factored frontier, PRADA cannot distinguish between both actions although $action1$ will achieve the goal, while $action2$ will not.

PRADA's estimated probabilities of states and rewards may differ significantly from their true values. This does not harm its performance in many domains as our experiments indicate (cf. our evaluation below). We suppose the reason for this is that while PRADA's estimated probabilities can be imprecise, they enable a correct ranking of action sequences—and in planning, we are interested in choosing the best action instead of calculating correctly its value.

A further difference between the proposed algorithms is in their way to handle the noise outcome of rules: PRADA assigns very small probability to all successor states—in the spirit of the noise outcome. In contrast, for SST and UCT it does not make sense to sample from such a distribution over all possible states, as any single successor state has extremely low probability and will be inadequate to estimate state and action values. Hence, they use the described workaround to assume to stay in the same state, while discounting obtained rewards.

It is straightforward for PRADA to deal with uncertain initial states. Uncertainty of initial states is common in complex environments and may for instance be caused by partial observability or noisy sensors. This uncertainty has its natural representation in the belief state PRADA works on. In contrast, SST and UCT cannot account for uncertain initial states directly, but would have to sample from the prior distribution.

An Extension: Adaptive PRADA

We present a simple extension of PRADA to increase its planning accuracy. We exploit the fact that PRADA evaluates complete sequences of actions—in contrast to SST and UCT where the actions taken at $t > 0$ depend on the sampled outcomes. *Adaptive* PRADA (A-PRADA) (Lang and Toussaint, 2009b, 2010b) examines the best action sequence found by PRADA. While PRADA chooses the first action of this sequence without further reasoning, A-PRADA inspects each single action of this sequence and decides by simulation whether it can be deleted. The resulting shortened sequence may lead to an increased expected reward. This is the case if actions do not have significant effects on achieving the goal or if they decrease the success probability. If such actions are omitted, the states with high reward are reached earlier and their rewards are discounted less. For instance, consider the goal to grab a blue ball: an action sequence that grabs a red cube, puts it onto the table and only then grabs the blue ball can be improved by omitting the first two actions which are unrelated to the goal.

More precisely, A-PRADA takes PRADA's action sequence $\mathbf{a}_P$ with the highest value (which is not necessarily the optimal action sequence) and investigates iteratively for each action whether it can be deleted. An action can be deleted from the plan if the resulting plan has a higher reward likelihood. This idea is formalized in Algorithm 1. The crucial calculation of this algorithm is to compute values $Q(s^0, \mathbf{a}^{0:T-1})$ as defined in Eq. (28) and restated here for convenience:

$$Q(\mathbf{s}^0, \mathbf{a}^{0:T-1}) = \sum_{t=1}^{T} \gamma^t P(U^t = 1 \,|\, \mathbf{a}^{0:t-1}, \mathbf{s}^0) \ .$$

PRADA's approximate inference procedure is particularly suitable for calculating all required $P(U^t = 1 \,|\, \mathbf{a}^{0:t-1}, \mathbf{s}^0)$. It performs this calculation in time linear in the length T of the action sequence, while SST and UCT would require time exponential in T because of their outcome sampling.

Algorithm 1 Adaptive PRADA (A-PRADA)

```
Input: PRADA's plan $\mathbf{a}_P$
Output: A-PRADA's plan $\mathbf{a}_A$
$\mathbf{a}_A \leftarrow \mathbf{a}_P$
for $t = 0$ to $t = T - 1$ do
    while true do
        Let $\mathbf{a}$ be a plan of length $T$.
        $\mathbf{a}^{0:t-1} \leftarrow \mathbf{a}_A^{0:t-1}$        ▷ Omit $a^t$
        $\mathbf{a}^{t:T-2} \leftarrow \mathbf{a}_A^{t+1:T-1}$
        $a^{T-1} \leftarrow doNothing$
        if $Q(s^0, \mathbf{a}) > Q(s^0, \mathbf{a}_A)$ then
            $\mathbf{a}_A \leftarrow \mathbf{a}$
        else
            break
        end if
    end while
end for
return $\mathbf{a}_A$
```

4.2.3 Evaluation

We have implemented all presented forward reasoning algorithms and the learning algorithm for NID rules in C++[1]. We evaluate our approaches in two different scenarios. The first is an intrinsically noisy complex simulated environment where we learn NID rules from experience and use these to plan. Second, we apply our algorithms on the benchmarks of the Uncertainty Part of the International Planning Competition 2008.

Robot Manipulation Domain

We perform experiments in our simulated complex robot manipulation environment (see Sec. 1.1.1) where a robot manipulates objects scattered on a table. Due to its intrinsic noise and its complexity, this simulated robot manipulation scenario is a challenging domain for both learning compact transition models as well as planning.

We use the rule learning algorithm of Pasula et al. (2007) with the same parameter settings to learn three different sets of fully abstract NID rules. Each rule-set is learned from independent training sets of 500 experience triples (s, a, s') that specify how the world changed from state s to successor state s' when an action a was executed, assuming full observability. Training data to learn rules are generated in a world of six cubes and four balls of two different sizes by performing random actions with a slight bias to build high piles. Our resulting rule-sets contain 9, 10 and 10 rules respectively. These rule-sets provide approximate partial transition models. They generalize over the situations of the experiences, but may not account for situations that are completely different from what the agent has seen before. To enforce compactness and avoid overfitting, rules are regularized; hence, the learning algorithm may sometimes favor to model rarely ex-

[1] Our code is available at `http://userpage.fu-berlin.de/tlang/prada/`.

perienced state transitions as low-probability outcomes in more general rules, thereby trading off accuracy for compactness. This in combination with the general noisiness of the world causes the need to carefully account for the probabilities of the world when reasoning with these rules.

We perform three series of experiments with planning tasks of increasing difficulty. In each series, we test the planners in different worlds with varying numbers of cubes and balls. Thus, we transfer the knowledge gained in the training world to different, but similar worlds by using abstract NID rules. For each object number, we create five different worlds. Per rule-set and world, we perform three independent runs with different random seeds. To evaluate the different planning approaches, we compute the mean performances and planning times over the fixed (but randomly generated) set of 45 trials (3 *learned* rule-sets, 5 worlds, 3 random seeds).

We choose the parameters of the planning algorithms as follows. For SST, we report results for different branching factors b, as far as the resulting runtimes allow. Similarly, UCT and (A-)PRADA each have a parameter that balances their planning time and the quality of their found actions. For UCT this is the number of episodes, while for (A-)PRADA this is the number of sampled action-sequences. Depending on the experiment, we set both heuristically such that the tradeoff between planning time and quality is reasonable. In particular, for a fair comparison we pay attention that UCT, PRADA and A-PRADA get about the same planning times, if not reported otherwise. Furthermore, for UCT we set the bias parameter c to 1.0 which we found heuristically to perform best. For all planners and experiments, we set the discounting factor for future rewards to $\gamma = 0.95$. A crucial parameter is the planning horizon d, which heavily influences planning time. Of course, d cannot be known a-priori. Therefore, if not reported otherwise, we deliberately set d larger than required for UCT and (A-)PRADA to suggest that our algorithms are also effective when d can only be estimated. Indeed, we found in all our experiments that as long as d is not too small, its exact choice does not have significant effects on UCT's and (A-)PRADA's planning quality—unlike its effects on planning times. In contrast, we set the horizon d for SST always as small as possible, in which case its planning times are still very large. If a planning algorithm does not find a suitable action in a given situation, we restart the planning procedure: SST builds a new tree, UCT runs more episodes and (A-)PRADA takes new action-sequence samples. If in a given situation after 10 planning runs a suitable action still is not found, the trial fails.

Furthermore, we use FF-Replan (Yoon et al., 2007) as a baseline. As discussed in detail with the related work in Sec. 4.1, FF-Replan determinizes the planning problem, thereby ignoring outcome probabilities. FF-Replan has shown impressive results on the domains of the probabilistic planning competitions. These domains are carefully designed by humans: their action dynamics definitions are complete, accurate and consistent and are used as the true world dynamics in the according experiments—in contrast to the learned NID rules we use here which estimate approximate partial models of our robot manipulation domain. To be able to use the derived predicate *clear(X)* in the FF-Replan implementation of our experiments, we included the appropriate literals of this predicate by hand in the outcomes of the rules—while our SST, UCT and (A-)PRADA implementations infer these values automatically from the definition of *clear(X)*. We report results of FF-Replan with these (slightly modified) learned rules using the all-outcomes determinization scheme, denoted by FF-Replan-All below. (Using single-outcome schemes always led to worse performance.) Some of these rules are very general (putting only few restrictions on the arguments and deictic references); in this case, more actions appear applicable in a given state than make sense from an intuitive human perspective which hurts FF-Replan much more than the other methods, resulting in large planning times for FF-Replan. For instance, a rule may model the toppling over of a small tower including object X when trying to put an object Y on top of the tower: one outcome with a small probability might specify Y to end up below X. While this is only possible if Y is a cube, of course, the learning algorithm may choose to omit a typing predicate $cube(X)$ due to regularization, as it prefers compact rules and none of its experiences might require this additional predicate. As this may pose severe problems for FF-Replan, we created modified rule-sets by hand where we introduced typing predicates where appropriate to make contexts more distinct. Below, we denote our results with these modified rule-sets as FF-Replan-All* and FF-Replan-Single*, using all-outcomes and single most-probable outcome determinization schemes.

High Towers In our first series of experiments, we investigate building high towers which was the planning task in the work of Pasula et al. (2007). More precisely, the reward in a state is defined as the average height of objects. This constitutes an easy planning problem as many different actions may increase the reward (object identities do not matter) and a small planning horizon d is sufficient. We set SST to horizon $d = 4$ (Pasula et al. 's choice) with different branching factors b and UCT and (A-)PRADA to horizon $d = 6$. In our experiments, initial states do not contain already stacked objects, so the reward for performing no actions is 0. Table 4.2 and Fig. 4.4 present our results. SST is not competitive. For a branching factor $b > 1$, it is slower than UCT and (A-)PRADA by at least an order of magnitude. For $b = 1$, its performance is poor. In this series of experiments, we designed the worlds of 10 objects to contain many big cubes. This explains the relatively good performance of SST in these worlds, as the number of good plans is large. As mentioned above, we control UCT, PRADA and A-PRADA to have about the same times available for planning. All three approaches perform far better than SST in almost all experiments. The difference between UCT, PRADA and A-PRADA is never significant.

This series of experiments indicates that planning approaches using full-grown look-ahead trees like SST are inappropriate even for easy planning problems. In contrast, approaches that exploit look-ahead trees in a clever way such as UCT seem to be the best choice for easy tasks which require a small planning horizon and can be solved by many alternative good plans. The performance of the planning approaches using approximate inference, PRADA and A-PRADA, however, comes close to the one of UCT, showing also their suitability for such scenarios.

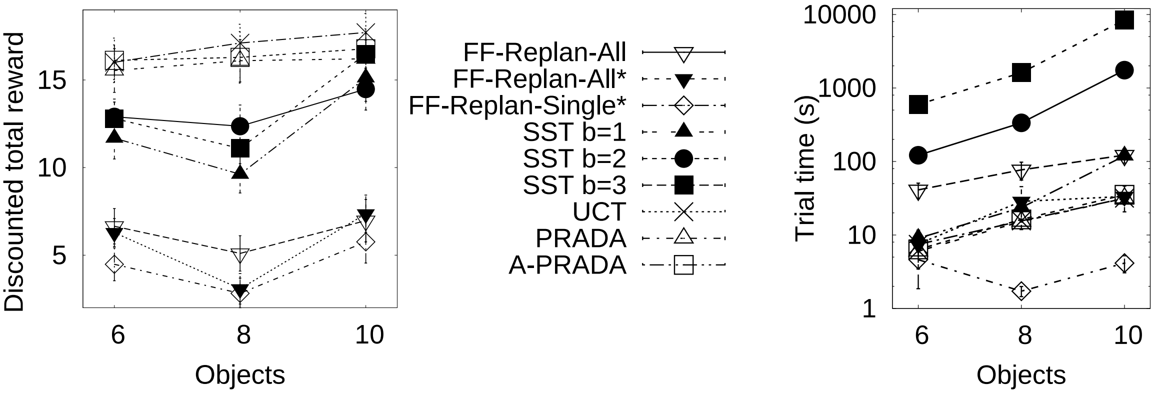
FF-Replan focuses on exploiting conjunctive goal structures and cannot deal with quantified goals. As the grounded reward structure of this task consists of a disjunction

Table 4.2: *High towers problem. Reward* denotes the discounted total reward for different numbers of objects (cubes/balls and table). The reward for performing no actions is 0. All data points are averages over 45 trials created from 3 learned rule-sets, 5 worlds and 3 random seeds. Standard deviations of the mean estimators are shown. *FF-Replan-All* and FF-Replan-Single* use hand-made modifications of the original learned rule-sets.* Fig. 4.4 visualizes these results.

Objects	Planner	Reward	Trial time (s)
6+1	FF-Replan-All	6.65 ± 1.01	41.07 ± 9.63
	FF-Replan-All*	6.29 ± 0.80	7.54 ± 4.09
	FF-Replan-Single*	4.48 ± 0.94	**4.61** ± 2.75
	SST (b=1)	11.68 ± 1.19	9.03 ± 0.80
	SST (b=2)	12.90 ± 1.01	121.40 ± 11.12
	SST (b=3)	12.80 ± 0.94	595.43 ± 55.95
	UCT	**16.01** ± 0.99	7.45 ± 0.19
	PRADA	15.54 ± 1.25	6.01 ± 0.07
	A-PRADA	**16.12** ± 1.27	6.36 ± 0.07
8+1	FF-Replan-All	5.10 ± 1.01	76.86 ± 20.98
	FF-Replan-All*	3.08 ± 0.87	28.65 ± 16.81
	FF-Replan-Single*	2.82 ± 0.87	**1.72** ± 0.27
	SST (b=1)	9.62 ± 1.07	23.57 ± 3.48
	SST (b=2)	12.36 ± 1.21	335.5 ± 52.4
	SST (b=3)	11.09 ± 0.87	1613.3 ± 249.2
	UCT	**17.11** ± 1.07	15.54 ± 0.40
	PRADA	16.10 ± 1.21	15.24 ± 0.27
	A-PRADA	16.29 ± 1.47	16.30 ± 0.27
10+1	FF-Replan-All	6.97 ± 1.21	121.99 ± 27.43
	FF-Replan-All*	7.36 ± 1.07	33.45 ± 12.80
	FF-Replan-Single*	5.76 ± 1.21	**4.14** ± 1.08
	SST (b=1)	15.12 ± 1.34	119.26 ± 10.59
	SST (b=2)	14.48 ± 1.20	1748.7 ± 170.2
	SST (b=3)	16.48 ± 1.19	8424 ± 851
	UCT	**17.71** ± 1.08	31.71 ± 5.83
	PRADA	16.21 ± 1.07	31.58 ± 1.14
	A-PRADA	16.78 ± 1.14	35.22 ± 0.40

of different tower combinations, FF-Replan has to pick an arbitrary tower combination as its goal. Therefore, to apply FF-Replan we sample tower combinations according to the rewards they achieve (i.e., situations with high towers are more probable) and do not exclude combinations with balls at the bottom of towers as they are not prohibited by the reward structure. As Yoon et al. (2007) note, "the obvious pitfall of this [goal formula sampling] approach is that some groundings of the goal are not reachable or are much more expensive to reach from the initial state". When FF-Replan cannot find a plan, we do not execute an action, but sample a new ground goal formula at the next time-step, preserving already achieved tower structures.

FF-Replan performs significantly worse than the previous planning approaches. The major reason for this is that FF-Replan often comes up with plans exploiting low-probability outcomes of rules—in contrast to SST, UCT and (A-)PRADA which reason over the probabilities. To illustrate this, consider the example rule in Table 3.3 (page 36) which models



(a) Reward (b) Time

Figure 4.4: *High towers problem.* Visualization of the results presented in Table 4.2. The reward for performing no actions is 0. All data points are averages over 45 trials created from 3 learned rule-sets, 5 worlds and 3 random seeds. Error bars for the standard deviations of the mean estimators are shown. Please note the log-scale in (b).

putting a ball on top of a cube. It has two explicit outcomes: the ball usually ends up on the cube; sometimes, however, it falls on the table. FF-Replan can misuse this rule as a tricky way to put a ball on the table—ignoring that this often will fail. As the results of FF-Replan-Single* show, taking only most probable outcomes into account does not remedy this problem: there are often two to three outcomes with similar probabilities so such a choice seems unjustified; sometimes, the "intuitively expected" outcome is split up into different outcomes with low probabilities, which however vary only in features irrelevant for the planning problem (such as $upright(\cdot)$).

Clearance The task in our second series of experiments is to clear up the desktop. Objects are lying scattered all over the table in the beginning. An object is cleared if it is part of a tower containing all other objects of the same class. An object class is simply defined in terms of color which is additionally provided to the state representation of the robot. The reward of the robot is defined as the number of cleared objects. In our experiments, classes contain 2-4 objects with at most 1 ball (in order to enable successful piling). Our starting situations contain some piles, but only with objects of different classes. Thus, the reward for performing no actions is 0. This clearance task is more difficult than building high towers, as the number of good plans yielding high rewards is significantly reduced.

We set the planning horizon $d = 6$ optimal for SST which is required to clear up a class of 4 objects, namely grabing and putting three objects. As above, by contrast we set $d = 10$ for UCT and (A-)PRADA to show that they can deal with overestimation of the usually unknown optimal horizon d. Table 4.3 and Fig. 4.5 present our results. The horizon $d = 6$ overburdens SST as can be seen from its large planning times. Even for $b = 1$, SST takes almost 40 minutes on average in worlds of 6 objects, while over 2 hours in worlds of 8 objects. Therefore, we did not try SST for greater b. In contrast, the

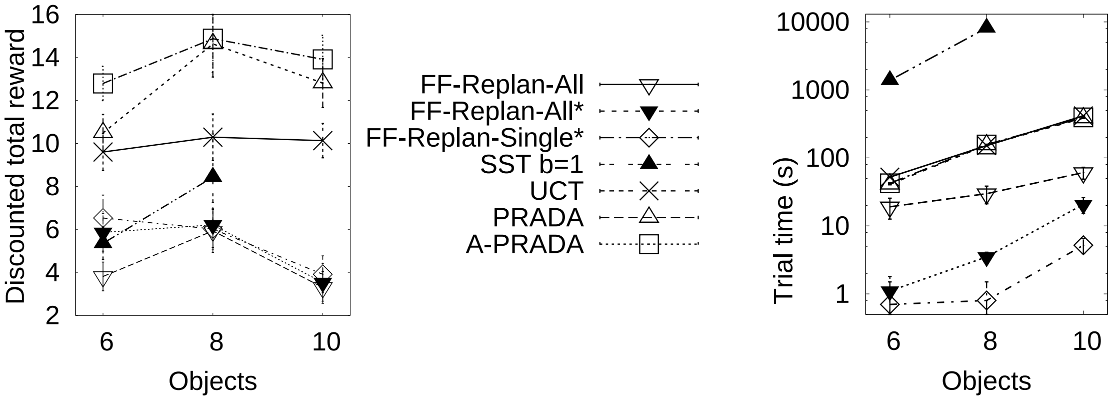
Table 4.3: *Clearance problem*. *Reward* denotes the discounted total reward for different numbers of objects (cubes/balls and table). The reward for performing no actions is 0. All data points are averages over 45 trials created from 3 learned rule-sets, 5 worlds and 3 random seeds. Standard deviations of the mean estimators are shown. *FF-Replan-All* and FF-Replan-Single* use hand-made modifications of the original learned rule-sets.* Fig. 4.5 visualizes these results.

Obj.	Planner	Reward	Trial time (s)
6+1	FF-Replan-All	3.81 ± 0.67	19.1 ± 6.5
	FF-Replan-All*	5.86 ± 0.87	1.1 ± 0.7
	FF-Replan-Single*	6.53 ± 1.07	**0.7** ± 0.8
	SST (b=1)	5.35 ± 0.75	1382.6 ± 80.4
	UCT	9.60 ± 0.86	52.2 ± 0.7
	PRADA	10.94 ± 0.86	40.9 ± 0.7
	A-PRADA	**12.79** ± 0.80	42.3 ± 0.7
8+1	FF-Replan-All	5.93 ± 1.00	29.8 ± 8.7
	FF-Replan-All*	6.21 ± 1.05	3.5 ± 0.6
	FF-Replan-Single*	6.02 ± 0.94	**0.8** ± 0.7
	SST (b=1)	8.43 ± 2.01	8157 ± 978
	UCT	10.29 ± 1.08	151.4 ± 2.0
	PRADA	**14.63** ± 1.54	154.5 ± 1.9
	A-PRADA	**14.87** ± 1.57	157.4 ± 2.0
10+1	FF-Replan-All	3.30 ± 0.74	60.9 ± 12.1
	FF-Replan-All*	3.53 ± 0.87	20.7 ± 5.4
	FF-Replan-Single*	3.91 ± 0.86	**5.2** ± 1.3
	SST (b=1)	–	> 8h
	UCT	10.13 ± 0.80	415.7 ± 7.4
	PRADA	12.81 ± 1.14	385.3 ± 4.7
	A-PRADA	**13.91** ± 1.12	394.5 ± 4.0

planning times of UCT, PRADA and A-PRADA, again controlled to be about the same and to enable reasonable performance, are two orders of magnitude smaller, although overestimating the planning horizon: for a trial they take on average about 45s in worlds of 6 objects, $2\frac{1}{2}$ minutes in worlds of 8 objects and 6-7 minutes in worlds of 10 objects. Nonetheless, UCT, PRADA and A-PRADA perform significantly better than SST. In all worlds, PRADA and A-PRADA in turn outperform UCT, in particular in worlds with many objects. A-PRADA finds the best plans among all planners. All planners gain more reward in worlds of 8 objects in comparison to worlds of 6 objects, as the number of objects that can be cleared increases as well as the number of classes and thus of good plans. The worlds of 10 objects contain the same numbers of object classes like the worlds of 8 objects, but with more objects, making planning more difficult.

Overall, our findings in the *Clearance* experiments indicate that while SST is inappropriate, UCT achieves good performance in planning scenarios which require medium planning horizons and where there are several, but not many alternative plans. Approaches using approximate inference like PRADA and A-PRADA, however, seem to be more appropriate in such scenarios of intermediate difficulty.

Furthermore, our results indicate that FF-Replan is inadequate for the clearance task.



(a) Reward (b) Time

Figure 4.5: *Clearance problem.* Visualization of the results presented in Table 4.3. The reward for performing no actions is 0. All data points are averages over 45 trials created from 3 learned rule-sets, 5 worlds and 3 random seeds. Error bars for the standard deviations of the mean estimators are shown. Note the log-scale in (b).

We sample target classes randomly to provide a goal structure to FF-Replan; the tower structure within a target class in turn is also randomly chosen. The bad performance of FF-Replan is due to the reasons described in the previous experiments; in particular the plans of FF-Replan often rely on low-probability outcomes.

Reverse Tower To explore the limits of UCT, PRADA and A-PRADA, we conducted a final series of experiments where the task is to reverse towers of C cubes which requires at least $2C$ actions (each cube needs to be grabbed and put somewhere at least once). Apart from the long planning horizon, this is difficult due to the noise in the simulated world: towers can become unstable and topple over with cubes falling off the table. To decrease this noise slightly to obtain more reliable results, we forbid the robot to grab objects that are not clear (i.e., below other objects). We set a limit of 50 executed actions on each trial. If thereafter the reversed tower still is not built, the trial fails. The trial also fails if one of the required objects falls off the table.

Table 4.4 presents our results. We cannot get SST with optimal planning horizon $d = 10$ to solve this problem even for five cubes. Although the space of possible actions is reduced due to the mentioned restriction, SST has enormous runtimes. With $b = 1$, SST does not find suitable actions (no leaves with the goal state) in several starting situations—the increased planning horizon leads to a high probability of sampling at least one unfavorable outcome for a required action. For $b \geq 2$, a single tree traversal of SST takes more than a day. We found UCT to also require large planning times in order to achieve a reasonable success rate. Therefore, we set the planning horizons optimal for UCT. In worlds of 5 cubes, UCT with optimal $d = 10$ has a success rate of about 40% while taking on average more than 40 minutes in case of success. For 6 cubes, however, UCT with optimal $d = 12$ never succeeds even when planning times exceed 4

Table 4.4: *Reverse tower problem.* The trial times and numbers of executed actions are given for the successful trials for different numbers of objects (cubes and table). All data points are averages over 45 trials created from 3 learned rule-sets, 5 worlds and 3 random seeds. Standard deviations of the mean estimators are shown. *FF-Replan-All* and FF-Replan-Single* use hand-made modifications of the original learned rule-sets.*

Objects	Planner	Success rate	Trial time (s)	Executed actions
5+1	FF-Replan-All	0.02	7.1 ± 0.0	12.0 ± 0.10
	FF-Replan-All*	**1.00**	26.7 ± 2.7	13.1 ± 0.9
	FF-Replan-Single*	0.67	**7.0** ± 0.9	13.6 ± 1.1
	SST (b=1)	0.00	-	-
	SST (b=2)	0.00	> 1 day	-
	UCT	0.38	2504.9 ± 491.1	19.5 ± 4.0
	PRADA	0.71	27.0 ± 1.8	13.2 ± 0.7
	A-PRADA	0.82	25.4 ± 0.8	**10.9** ± 0.8
6+1	FF-Replan-All	0.00	-	-
	FF-Replan-All*	**1.00**	589.2 ± 73.7	**12.0** ± 0.8
	FF-Replan-Single*	0.64	**52.7** ± 5.3	17.3 ± 2.1
	UCT	0.00	> 4 h	-
	PRADA	0.47	66.4 ± 3.9	13.6 ± 0.9
	A-PRADA	0.56	77.5 ± 8.3	14.4 ± 2.5
7+1	FF-Replan-All	0.00	-	-
	FF-Replan-All*	0.42	2234.2 ± 81.1	**15.1** ± 1.3
	FF-Replan-Single*	**0.56**	**687.4** ± 86.4	17.5 ± 2.0
	PRADA	0.24	871.3 ± 126.6	18.2 ± 1.2
	A-PRADA	0.23	783.7 ± 132.6	**15.1** ± 1.8

hours. In contrast, we can afford an overestimating horizon $d = 20$ for PRADA and A-PRADA. In worlds of 5 cubes, PRADA and A-PRADA achieve success rates of 71% and 82% respectively in less than half a minute. A-PRADA's average number of executed actions in case of success is almost optimal. In worlds of 6 cubes, the success rates of PRADA and A-PRADA are still about 50%, taking a bit more than a minute on average in case of success. When their trials fail, this is most often due to cubes falling off the table and not because they cannot find appropriate actions. Cubes falling off the table is also a main reason why the success rates of PRADA and A-PRADA drop to 23% and 24% respectively in worlds of 7 cubes when towers become rather unstable. Planning times in successful trials, however, also increase to more than 13 minutes indicating the limitations of these planning approaches. Nonetheless, the mean number of executed actions in successful trials is still almost optimal for A-PRADA.

Overall, the *Reverse tower* experiments indicate that planning approaches using look-ahead trees fail in tasks that require long planning horizons and can only be achieved by very few plans. Given the huge action and state spaces in relational domains, the chances that UCT simulates an episode with exactly the required actions and successor states are very small. Planning approaches using approximate inference like PRADA and A-PRADA have the crucial advantage that the stochasticity of actions does not affect their runtime exponentially in the planning horizon. Of course, their search space of action-sequences still is exponential in the planning horizon so that problems requiring long horizons are hard to solve also for them. Our experiments show that by using the very simple, though principled extension A-PRADA, we can gain significant performance improvements.

Our results also show that FF-Replan fails to provide good plans when using the original learned rule-sets. This is surprising as the characteristics of the *Reverse tower* task seem to favor FF-Replan in comparison to the other methods: there is a single conjunctive goal structure and the number of good plans is very small while these plans require long horizons. As the results of FF-Replan-All* and FF-Replan-Single* indicate, FF-Replan can achieve a good performance with the adapted rule-sets that have been modified by hand to restrict the number of possible actions in a state. While this constitutes a proof of concept of FF-Replan, it shows the difficulty of applying FF-Replan with learned rule-sets.

Summary Our results demonstrate that successful planning with learned transition models (here in the form of rules) may require to explicitly account for the quantification of predictive uncertainty. More concretely, methods applying look-ahead trees (UCT) and approximate inference ((A-)PRADA) outperform FF-Replan on different tasks of varying difficulty. Furthermore, (A-)PRADA can solve planning tasks with long horizons, where UCT fails. Only if one post-processes the learned rules by hand to clarify their application contexts and the planning problem uses a conjunctive goal structure and requires few and long plans, FF-Replan performs better than UCT and (A-)PRADA.

IPPC 2008 Benchmarks

In the second part of our evaluation, we apply our proposed approaches on the benchmarks of the latest international probabilistic planning competition, the Uncertainty Part of the International Planning Competition in 2008 (IPPC, 2008). The involved domains differ in many characteristics, such as the number of actions, the required planning horizons and the reward structures. As the competition results show, no planning algorithm performs best everywhere. Thus, these benchmarks give an idea for what types of problems SST, UCT and (A-)PRADA may be useful. We convert the PPDDL domain specifications into NID rules along the lines described in Appendix B. The resulting rule-sets are used to run our implementations of SST, UCT and (A-)PRADA on the benchmark problems.

Each of the seven benchmark domains consists of 15 problem instances. An instance specifies a goal and a starting state. Instances vary not only in problem size, but also in their reward structures (including action costs), so a direct comparison is not always possible. In the competition, each instance was considered independently: planners were given a restricted amount of time (10 minutes for problems 1-5 of each domain and 40 minutes for the others) to cover as many repetitions of the very same problem instance as possible up to a maximum of a 100 trials. Trials differed in the random seeds resulting in potentially different state transitions. The planners were evaluated with respect to the number of trials ending in a goal state and the collected reward averaged over all trials.

Eight planners entered in the competition (see Sec. 4.1), including FF-Replan which was not an official participant. For their results, which are too voluminous to be presented here, we refer the reader to the website of the competition. Below, we provide a *qualitative* comparison of our methods to the results of these planners. We do not attempt a direct *quantitative* comparison for several reasons. First, the different hardware prevents timing comparisons. Second, competition participants have frequently not been able to successfully cover trials of a single or all instances of a domain. It is difficult to tell the reasons for this from the results tables: the planner might have been overburdened by the problem, might have faced temporary technical problems with the client-server architecture framework of the competition or could not cope with certain PPDDL constructs which could have been rewritten in a simpler format.

Third and most importantly, we have not optimized our implementations to reuse previous planning efforts. Instead, we fully replan for each single action (within a trial and across trials). The competition evaluation scheme puts replanners at a disadvantage (in particular those which replan each single action). Instead of replanning, a good strategy for the competition is to spend most planning time before starting the first trial and then reuse the resulting insights (such as conditional plans and value functions) for all subsequent trials with a minimum of additional planning. Indeed, this strategy has often been adopted as many trial time results indicate. We acknowledge that this is a fair procedure to evaluate planners which compute policies over large parts of the state-space before acting. We feel, however, that this is counter to the idea of our approaches: UCT and (A-)PRADA are meant for flexible planning with varying goals and different situations. Thus, what we are interested in is the average time to compute good actions and successfully solve a problem instance when there is no prior knowledge available.

Therefore, for each single problem instance we perform 100 trials with different random seeds using full replanning. A trial is aborted if a goal state is not reached within some maximum number of actions varying slightly for each benchmark (about 50 actions). We present the success rates and the mean estimators of trial times, executed actions and rewards with their standard deviations in Table 4.5 for the problem instances where at least one trial was successfully covered in reasonable time.

Search and Rescue (Table 4.5(a)) is the only domain where SST (with branching factor 1) is able to find plans within reasonable time—with significantly larger runtimes than UCT and (A-)PRADA. The success rates and the rewards indicate that PRADA and A-PRADA are superior to UCT and scale up to rather big problem instances. To give an idea w.r.t. the IPPC evaluation scheme: UCT solves successfully 54 trials of the first instance within 10 minutes with *full replanning*, while PRADA and A-PRADA solve all trials with full replanning. In fact, despite of replanning each single action, PRADA and A-PRADA show the same success rates as the best planners of the benchmark except for the very large problem instances (within the competition, only the participants FSP-RBH and FSP-RDH achieved comparably satisfactory results). We conjecture that the success of our methods is due to that fact that this domain requires to account carefully for the outcome probabilities, but does not involve very long planning horizons.

Triangle-Tireworld (Table 4.5(b)) is the only domain where UCT outperforms PRADA

Table 4.5: *Benchmarks of the IPPC 2008.* The first column of a table specifies the problem instance. *Suc.* is the success rate. The trial time and the number of executed actions are given for the successful trials. Where applicable, the average reward for all trials is shown. All results are achieved with *full replanning* within a trial and across trials.

(a) Search and Rescue

	Planner	Suc.	Trial Time (s)	Actions	Reward
01	SST	100	37.9±0.1	9.2±0.2	1440±90
	UCT	54	1.4±0.1	11.4±0.3	900±70
	PRADA	100	1.1±0.1	10.5±0.4	1460±89
	A-PRADA	100	1.1±0.1	10.4±0.4	1460±89
02	SST	100	220.2±0.1	9.8±0.2	1560±83
	UCT	56	4.1±0.3	12.2±0.6	880±100
	PRADA	100	1.6±0.1	12.9±0.7	1460±89
	A-PRADA	100	1.6±0.1	12.8±0.4	1440±90
03	SST	71	955.5±0.5	9.8±0.2	1662±85
	UCT	57	12.9±0.6	13.6±0.6	680±63
	PRADA	99	1.4±0.1	18.0±1.0	1480±88
	A-PRADA	99	1.4±0.1	17.9±1.1	1480±88
04	UCT	61	24.9±1.6	16.1±0.8	7200±57
	PRADA	100	1.4±0.0	11.9±0.4	1460±89
	A-PRADA	100	1.4±0.0	11.5±0.3	1500±87
05	UCT	46	40.1±2.1	16.8±1.4	600±64
	PRADA	89	6.8±0.3	21.8±0.9	1240±83
	A-PRADA	92	6.5±0.3	21.0±0.9	1320±81
06	UCT	39	71.7±5.6	19.5±1.3	410±59
	PRADA	83	10.1±0.9	24.3±1.3	1240±90
	A-PRADA	84	10.0±0.9	23.7±1.2	1240±90
07	UCT	53	230.3±13.2	21.5±1.4	540±62
	PRADA	98	10.1±0.4	18.5±0.8	1470±88
	A-PRADA	98	9.9±0.4	18.0±0.8	1490±87
08	UCT	34	332.9±24.1	21.71±1.5	360±59
	PRADA	59	20.2±0.8	30.4±1.7	910±82
	A-PRADA	59	19.9±0.8	29.9±1.7	910±82
09	UCT	30	752.8±72.3	26.4±2.4	360±48
	PRADA	63	30.2±1.2	27.5±1.6	930±80
	A-PRADA	65	30.0±1.1	27.5±1.6	1010±84
10	PRADA	21	97.9±10.2	26.8±2.8	180±27
	A-PRADA	21	92.1±9.8	26.7±2.8	180±27
11	PRADA	17	151.7±12.3	30±2.5	250±29
	A-PRADA	18	154.1±11.9	30.2±2.6	250±29
12	PRADA	38	210.8±72.1	30.1±10.5	636±253
	A-PRADA	21	219.8±28.5	30.7±2.8	556±55

(b) Triangle-Tireworld

	Planner	Suc.	Trial Time (s)	Actions
01	SST	0	–	–
	UCT	100	9.9±0.3	6.9±0.2
	PRADA	100	8.5±0.2	6.4±0.2
	A-PRADA	100	8.0±0.2	6.1±0.2
02	UCT	100	64.1±2.2	12.4±0.3
	PRADA	57	30.1±0.7	9±0.2
	A-PRADA	65	33.7±0.8	11.4±0.3
03	UCT	89	390.5±8.5	18.6±0.4
	PRADA	19	119.2±4.9	12.3±0.5
	A-PRADA	21	121.0±5.3	14.3±0.7
04	UCT	82	1497±19	26.0±0.5
	PRADA	6	2967±143	17.5±1.1
	A-PRADA	4	244.2±43.6	15.5±2.8

(c) Blocksworld

	Planner	Suc.	Trial Time (s)	Actions	Reward
01	SST	0	–	–	–
	UCT	0	–	–	–
	PRADA	53	17.8±0.4	23.0±0.7	0.8±0.0
	A-PRADA	63	18.4±0.5	22.3±0.8	0.6±0.0
03	PRADA	10	57.0±3.3	21.5±1.8	-9.6±0.0

(d) Boxworld

	Planner	Suc.	Trial Time (s)	Actions	Reward
01	SST	0	–	–	–
	UCT	0	–	–	–
	PRADA	100	257.8±6.3	46.8±1.0	1.00±0.0
	A-PRADA	100	143.8±3.1	43.1±1.1	1.00±0.0
02	PRADA	100	285.2±7.8	46.2±1.3	20.00±0.0
	A-PRADA	100	215.8±4.2	39.6±0.9	20.00±0.0
03	UCT	100	1285.2±8.1	32.8±0.0	929.8±2.1
	PRADA	100	165.7±2.9	52.5±1.1	865.1±3.3
	A-PRADA	50	457.8±7.1	35.0±0.7	754.1±21.5
04	PRADA	28	959.0±35.5	76.1±3.2	0.3±0.5
	A-PRADA	60	519.2±15.3	72.0±2.4	0.6±0.1
05	UCT	54	9972±776	37.9±3.5	606±149
	PRADA	61	345.4±8.5	68.4±1.6	465±24
	A-PRADA	2	528.6±38.8	38.0±0.0	411±34
08	PRADA	3	3361±88	87.0±2.3	0.19±0.1
	A-PRADA	10	1579±48	85.3±2.7	0.29±0.3
09	PRADA	28	1449±25	85.9±1.5	1365±31
	A-PRADA	0	– (1750.3)	–	1126±30

(e) Exploding Blocksworld

	Planner	Suc.	Trial Time (s)	Actions
01	SST	5	8607±1224	9.6±0.6
	UCT	3	111.8±14.0	9.3±0.4
	PRADA	62	3.6±0.0	8.6±0.8
	A-PRADA	61	3.9±0.0	8.4±0.8
02	PRADA	28	11.9±0.3	14.4±0.5
	A-PRADA	29	12.7±0.2	13.2±0.5
03	PRADA	36	14.3±0.3	12.6±0.6
	A-PRADA	30	16.8±0.3	12.5±0.5
04	PRADA	27	30.3±1.2	14.8±0.5
	A-PRADA	26	14.9±1.1	15.2±0.5
05	PRADA	100	5.5±0.1	6.6±0.1
	A-PRADA	100	5.5±0.1	6.6±0.1
06	PRADA	51	128.5±2.9	16.9±0.7
	A-PRADA	61	97.5±5.3	17.3±0.8
07	PRADA	14	125.0±6.9	15.3±0.4
	A-PRADA	72	154.8±5.5	17.6±1.0

and A-PRADA, although at a higher computational cost. The more depth-first-like style of planning of UCT seems useful in this domain. To give an idea w.r.t. the IPPC evaluation scheme: UCT performs 60 successful trials of the first instance within 10 minutes, while PRADA and A-PRADA achieve 72 and 74 trials resp. using full replanning; but UCT solves more trials in the more difficult instances. The required planning horizons increase quickly with the problem instances. Our approaches cannot cope with the large problem instances, which only three competition participants (RFF-BG, RFF-PG, HMDPP) could cover.

Our methods face problems when the required planning horizons are very large, while the number of plans with non-zero probability is small. This becomes evident in the **Blocksworld** benchmark (Table 4.5(c)). This domain is different from the robot manipulation environment used in our first evaluation described above. The latter is considerably more stochastic and provides more actions in a given situation (e.g., we may grab objects within a pile). Blocksworld is the only domain where our approaches are inferior to FF-Replan. To give an idea w.r.t. the IPPC evaluation scheme: UCT does not perform a single successful trial of the first instance within 10 minutes, while PRADA and A-PRADA achieve 16 and 17 trials resp. using full replanning.

In the **Boxworld** domain (Table 4.5(d)), our approaches can exploit the fact that the delivery of boxes is (almost) independent of the delivery of other boxes (in most problem instances this is further helped by the intermediate rewards for delivered boxes). In contrast to UCT, PRADA and A-PRADA scale up to relatively large problem instances. PRADA and A-PRADA solve all 100 trials of the first problem instance, requiring on average 4.3 min and 2.4 min resp. with full replanning. Only two competition participants solved trials successfully in this domain (RFF-BG and RFF-PG). To give an idea w.r.t. the IPPC evaluation scheme: UCT does not perform a single successful trial within 10 minutes, while PRADA completes 2 and A-PRADA 4 trials. This small number can be explained by the large plan lengths where each single action is computed with full replanning.

Finally, in the **Exploding Blocksworld** domain (Table 4.5(e)) PRADA and A-PRADA perform better or as good as the competition participants. To give an idea w.r.t. the IPPC evaluation scheme: UCT achieves only a single successful trial within 10 minutes, while PRADA and A-PRADA complete 56 and 61 trials resp..

We did not perform any experiments in either the **SysAdmin** or the **Schedule** domain. Their PPDDL specifications cannot be converted into NID rules due to the involved universal effects. In contrast, this has been possible for the Boxworld domain despite of the universal effects there: in the Boxworld problem instances, the universally quantified variables always refer to exactly one object which we exploit for conversion to NID rules. (Note that this can be understood as a trick to implement deictic references in PPDDL by means of universal effects. The according action operator, however, has odd semantics: boxes could end up in two different cities at the same time.) Furthermore, we ignored the **Rectangle-Tireworld** domain, which together with the Triangle-Tireworld domain makes up the 2-Tireworlds benchmark, as its problem instances have faulty goal descriptions: They should include *not(dead)* (this has not been critical to name a winner in the competition as personally communicated by Olivier Buffet).

Summary The majority of the PPDDL descriptions of the IPPC benchmarks can be converted into NID rules, indicating the broad spectrum of planning problems which can be covered by NID rules. Our results demonstrate that our approaches perform comparably to or better than state-of-the-art planners on many traditional hand-crafted planning problems. This hints at the generality of our methods for probabilistic planning beyond the type of robotic manipulation domains considered above. Our methods perform particularly well in domains where outcome probabilities need to be carefully accounted for. They face problems when the required planning horizons are very large, while the number of plans with non-zero probability is small; this can be avoided by intermediate rewards.

4.2.4 Conclusions and Future Work

We have presented two approaches for planning based on forward reasoning with probabilistic relational rules in ground relational domains. Our methods are designed to work on *learned* rules which provide approximate partial models of noisy worlds. Our first approach is an adaptation of the UCT algorithm which samples look-ahead trees to cope with action stochasticity. Our second approach, called PRADA, models the uncertainty over states explicitly in terms of beliefs and employs approximate inference in graphical models for planning. When we combine our planning algorithms with an existing rule learning algorithm, an intelligent agent can *(i)* learn a compact transition model of the dynamics of a complex noisy environment and *(ii)* quickly derive appropriate actions for varying goals. Results in a complex simulated robotics domain show that our methods outperform the state-of-the-art planner FF-Replan on a number of different planning tasks. In contrast to FF-Replan, our methods reason over the probabilities of action outcomes. This is necessary if the world dynamics are noisy and only partial and approximate transition models are available.

However, our planners also perform remarkably well on many traditional probabilistic planning problems. This is demonstrated by our results on IPPC benchmarks, where we have shown that PPDDL descriptions can be converted to a large extent to the kind of rules our planners use. This hints at the general-purpose character of particularly PRADA and the potential benefits of its techniques for probabilistic planning. For instance, our methods can be expected to perform similarly well in large propositional MDPs which do not exhibit a relational structure.

So far, our planning approaches deal in reasonable time with problems containing up to 10-15 objects (implying billions of world states) and requiring planning horizons of up to 15-20 time-steps. Nonetheless, our approaches are still limited in that they rely on reasoning in the ground representation. If very many objects need to be represented, our approaches need to be combined with other methods that reduce state and action space complexity (confer Sec. 4.4).

In its current form, the approximate inference procedure of PRADA relies on the specific compact DBNs compiled from rules. The development of similar factored frontier filters for arbitrary DBNs, for example derived from more general PPDDL descriptions, is promising. Similarly, the adaptation of PRADA's factored frontier techniques into existing probabilistic planners is worth of investigation. An important direction for improving PRADA is to make it adapt its action-sequence sampling strategy to the experience of previous samples. We have introduced a very simple extension, A-PRADA, to achieve this, but more sophisticated methods are conceivable. Learning rule-sets online and exploiting them immediately by our planning method is also an important direction of future research in order to enable acting in the real world, where we want to behave effectively right from the start. Approaches in this direction are presented in Chapter 5.

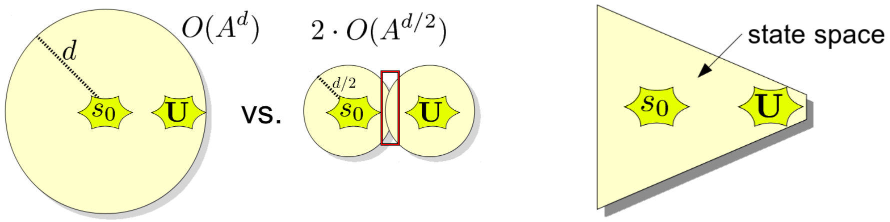
We explore two promising extensions of using approximate inference for planning in the next sections. In Sec. 4.3, we investigate using probabilistic relational rules and inference for backward reasoning. In Sec. 4.5, we investigate relational planning by inference which conditions on rewards and uses inference to compute a policy in form of posteriors over actions, instead of using inference to evaluate action sequences as done by PRADA.

4.3 Probabilistic Backward and Forward Reasoning

Combining forward with backward reasoning has the potential to greatly improve planning accuracy and efficiency. First, backward reasoning may drastically prune the search space of action sequences, as illustrated in Fig. 4.6(a): instead of search spaces which are exponential in the length d of the complete plans, we only have to consider search spaces which are exponential in $\frac{d}{2}$. Furthermore, backward reasoning is particularly useful in problems where the number of possible actions close to goal states is small in comparison to the start state, as illustrated in Fig. 4.6(b). Consider for instance the goal to build a tower from objects which are initially scattered over a table. The last action has to put the top object on the tower, while in the initial state any object could be grabbed.

So far, however, backward reasoning in ground relational domains has largely focused on non-probabilistic domains where action outcomes are not probabilistically determined. Classical approaches use regression techniques to map action operators and state variables to formulas describing the conditions under which a variable becomes true (Rintanen, 2008).

Bidirectional inference in graphical models has emerged as a promising technique for planning in non-relational domains. The forward reasoning approach PRADA described in the previous section uses forward inference in dynamic Bayesian networks compiled from learned probabilistic relational rules. Backward reasoning in ground relational domains using probabilistic inference has not been investigated yet. Previous approaches such as the forward-backward algorithm in Hidden Markov models (HMMs) (Rabiner, 1989) and the planning by inference paradigm (Toussaint and Storkey, 2006) in non-relational MDPs work in limited small state spaces and are not applicable in DBNs for ground relational domains, where exact inference is infeasible. The factored frontier inference of PRADA cannot be used directly for backpropagation, either. A core challenge



(a) (b)

Figure 4.6: Backward reasoning can improve the accuracy and efficiency of planning from a start state s_0 to a goal state $\mathbf{U}$ through the search space of states (light yellow areas). (a) Instead of the larger space of forward only search (left), combined forward and backward reasoning (right) has to search only through two search spaces exponential in $\frac{d}{2}$ (A is the number of actions) . (b) The number of potential actions close to goal states, that is the search space around $\mathbf{U}$, may be smaller than close to s_0.

is how to condition the state distribution at the last time-step on receiving a high reward when using approximate inference. This problem arises in particular for complex abstract reward dependencies such as partial goal descriptions.

We make three contributions to overcome these problems (Lang and Toussaint, 2010a): *(i)* We show how to use NID rules to learn a probabilistic backward model. *(ii)* We model arbitrary (partial) goal descriptions with a mixture state distribution and derive a probabilistic backward reasoning procedure. *(iii)* We introduce a two-filter (Solo, 1982) inference method to use bidirectional reasoning for planning in stochastic relational domains. We perform experiments in the robot manipulation domain (see Sec. 1.1.1). Empirical results show that bidirectional probabilistic reasoning can lead to more efficient and accurate planning in comparison to pure forward reasoning.

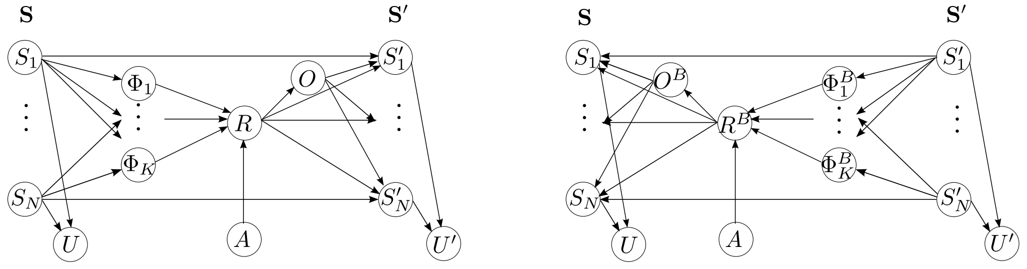
We proceed as follows. In Sec. 4.3.1, we present our two-filter using backward reasoning with NID rules. In Sec. 4.3.2, we introduce the bidirectional reasoning approach. Then, we show our experimental results in Sec. 4.3.3, before we discuss the main points.

4.3.1 Two-Filter Smoothing using Backward NID Rules

We propose to adapt PRADA for backward reasoning using NID rules. Then, we can exploit the knowledge about the reward for planning.

Backward Messages for a Two-Filter

Existing approaches for combining probabilistic backward and forward reasoning calculate smoothed state posteriors conditioned on a sequence of observed variables. Examples are the forward-backward algorithm in HMMs (Rabiner, 1989) or Expectation-Maximization used for planning by inference in small state spaces (Toussaint and Storkey, 2006). Here, we want to calculate posteriors $P(\mathbf{s}^t, U^T \mid \mathbf{a}^{0:T-1})$ where T is the last time-



(a) Forward DBN

(b) Backward DBN

Figure 4.7: (a) PRADA converts NID rules into a forward DBN to predict the effects of action sequences. (b) For backward reasoning, we use a second DBN with the same state, action and reward variables, but different rule variables (Φ_i^B, Γ^B and O^B) according to the backward rules.

step which we can use to evaluate an action-sequence by

$$P(U^T \,|\, \mathbf{a}^{0:T-1}) = \sum_{\mathbf{s}^t} P(\mathbf{s}^t, U^T \,|\, \mathbf{a}^{0:T-1}) \ .$$

These posteriors can be calculated by means of forward messages

$$\alpha_{\mathbf{a}^{0:t-1}}(\mathbf{s}^t) := P(\mathbf{s}^t \,|\, \mathbf{a}^{0:t-1})$$

and backward messages

$$\beta_{\mathbf{a}^{t:T-1}}(\mathbf{s}^t) := P(U^T \,|\, \mathbf{s}^t, \mathbf{a}^{t:T-1})$$

such that $P(U^T, \mathbf{s}^t \,|\, \mathbf{a}^{0:T-1}) = \alpha_{\mathbf{a}^{0:t-1}}(\mathbf{s}^t) \cdot \beta_{\mathbf{a}^{t:T-1}}(\mathbf{s}^t)$.

It is intractable to calculate these messages exactly in relational domains due to the immense state spaces. PRADA calculates the forward messages α approximately using a factored frontier filter. Unfortunately, PRADA's specific factored frontier equations only work for forward reasoning in the graphical model re-shown for convenience in Fig. 4.7(a) and cannot be applied for calculating the likelihood backward messages β. We might use rejection sampling in PRADA's forward DBN, but this is highly inefficient. It is in general unclear how to calculate the β even approximately in a tractable way in all but the smallest state spaces. Therefore, as an alternative we propose a filtering approach for backward reasoning. We use PRADA in reversed order, providing us with messages $\hat{\beta}_{\mathbf{a}^{t:T-1}}(\mathbf{s}^t) := P(\mathbf{s}^t \,|\, \mathbf{a}^{t:T-1}, U^T)$. This requires a set of backward NID rules from which we can build a backward DBN as shown in Fig. 4.7(b) and apply PRADA's factored frontier inference.

A *two-filter* (Solo, 1982; Briers et al., 2009) uses the resulting messages $\hat{\beta}$ to approximate the likelihood messages β,

$$\begin{aligned}\beta_{\mathbf{a}^{t:T-1}}(\mathbf{s}^t) &= P(U^T \,|\, \mathbf{s}^t, \mathbf{a}^{t:T-1})\\ &= \frac{P(\mathbf{s}^t \,|\, U^T, \mathbf{a}^{t:T-1})P(U^T \,|\, \mathbf{a}^{t:T-1})}{P(\mathbf{s}^t \,|\, \mathbf{a}^{t:T-1})}\\ &= \frac{P(\mathbf{s}^t \,|\, U^T, \mathbf{a}^{t:T-1})P(U^T \,|\, \mathbf{a}^{t:T-1})}{P(\mathbf{s}^t)}\\ &\approx \frac{P(\mathbf{s}^t \,|\, U^T, \mathbf{a}^{t:T-1})P(U^T)}{P(\mathbf{s})}\\ &\propto P(\mathbf{s}^t \,|\, U^T, \mathbf{a}^{t:T-1}) = \hat{\beta}_{\mathbf{a}^{t:T-1}}(\mathbf{s}^t)\ .\end{aligned}$$

The approximations of this two-filter are due to the intractable state distributions $P(\mathbf{s}^t)$ at specific time-steps t, also required to account for the dependencies of U^T on $\mathbf{a}^{t:T-1}$. Since in planning we are interested in ranking different action sequences $\mathbf{a}$, we can drop the likewise intractable reward marginal $P(U^T)$, as we can drop $P(\mathbf{s})$ assuming a uniform state prior.

Backward Rules

To use PRADA for backward filtering, we require a set of backward NID rules to define a distribution $P(s \,|\, s', a)$ over predecessor states s if an action a was applied before the current state s'. These rules take exactly the same form as in the forward case, only the semantics are changed. Given a forward model, one might try to invert the according rules. How to account for the special characteristics of NID rules such as uniqueness and noise outcomes in this case is unclear, however. As our forward models are learned and thus in any event approximations of the true underlying dynamics, we propose to learn the backward rules directly from data as well. This has the advantage that we can use the same algorithm which we already use for learning the forward rules. We only have to provide the experience triples in reversed order (s', a, s).

Depending on the domain, state transitions may be easier to model in one direction than the other. This may affect the deictic references in NID rules which may be unique only in one direction. Consider for instance the unary predicate $\mathit{pickup}(X)$ (Fig. 4.8). A forward rule could use a deictic reference Y to describe where X was taken from which is required to conclude $\neg on(X, Y)$ for the successor state. When reasoning backward, looking only at the successor state s', it is impossible to determine Y. This can be solved by increasing the arity of the action predicate so that less deictic references need to be resolved. For this reason, in our experiments we will use binary action predicates $takefrom(X, Y)$ and $dropabove(X, Y)$ instead of $\mathit{pickup}(X)$ and $\mathit{puton}(Y)$—while still allowing for deictic referencing to third or fourth objects. An exemplary rule for $dropabove(X, Y)$ is shown in Table 4.6.

At first glance, one might suspect that extending the action predicate arity increases the planning complexity due to the increased action space. This is resolved, however, when using a policy that only considers actions with unique rules. As we regularize

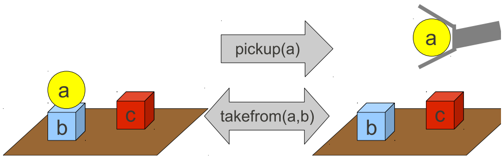


Figure 4.8: Using a unary action predicate $pickup(a)$, it is impossible to deduce from the successor state (via a deictic reference) whether a was taken from b or c. This information is captured explicitly in the binary action predicate $takefrom(a, b)$. Due to its action sampling strategy, extending the arity of actions does not influence PRADA's planning efficiency.

our rule learning procedure, the learned rules model typical state transitions. Thus, a planner using these rules takes actions only in frequently observed contexts into account, effectively pruning large parts of the action space in a given situation. Furthermore, determining all unique covering rules has the same computational cost, independently of whether Y is used as a second action argument or as a deictic reference.

4.3.2 Backward-Forward Reasoning

We use a two-filter to plan in stochastic relational domains. Given a backward model $B \equiv P(s \,|\, s', a)$ in form of NID rules for a state s, an action a and successor state s', we apply PRADA first to reason backward to estimate a distribution of states backward-reachable from goal states. Then, we use a second set of NID rules specifying a forward model $A \equiv P(s' \,|\, s, a)$ to reason forward from the initial state to find action sequences leading to states close to a goal state.

Table 4.6: Example NID rule for a robot manipulation scenario with binary actions, which models dropping an object X over an object Y in a situation where Y is below an object Z (deictic referencing). With high probability, X will land on Z, but might also fall on the table. With a small probability something unpredictable happens.

$$
\begin{array}{rl}
dropabove(X,Y): & inhand(X),\ on(Z,Y),\ table(T) \\
\rightarrow & \left\{ \begin{array}{lll} 0.6 & : & on(X,Z),\ \neg inhand(X) \\ 0.3 & : & on(X,T),\ \neg inhand(X) \\ 0.1 & : & \text{noise} \end{array} \right.
\end{array}
$$

Goal State Distributions

For probabilistic backward reasoning, we require a state distribution at the last time-step T. We are interested in states achieving a high reward, namely

$$\hat{\beta}^T(\mathbf{s}) := P(\mathbf{s} \,|\, U^T) \;.$$

In contrast to previous work on planning by inference, we cannot calculate $\hat{\beta}^T(\mathbf{s})$ exactly in relational domains due to the large state spaces. Thus, we approximate it with a factored frontier $\hat{\beta}^T(\mathbf{s}) \approx \prod_i \hat{\beta}^T(s_i)$.

If the goal fully specifies the final state, setting the marginals $\hat{\beta}^T(s_i)$ to their deterministic values is straightforward. If the goal is defined in terms of a partial state description in form of a conjunction ξ of literals, only some state attributes $\mathbf{s}_\xi \subset \mathbf{s}$ have deterministic values. The situation becomes more difficult to deal with if the goal is specified in terms of literals of a derived predicate, corresponding to formulas over primitive predicates, such as existentially quantified goals. Consider for instance the goal to stack the cubes $\{a, b, c\}$ in any order. In this case, the clearly dissimilar states $\mathbf{s}_1 = \{on(a, b), on(b, c)\}$ and $\mathbf{s}_2 = \{on(c, b), on(b, a)\}$ yield the same reward. If we approximate the final state belief by marginals, we lose the crucial correlations among the variables. This is a general problem in backward reasoning and arises likewise in non-probabilistic and propositional domains. A common strategy there is to pick arbitrary grounded forms of the goal, for instance choosing $\mathbf{s}_1$ in the example above. This has the pitfall that some goal groundings may not be reachable or more costly to reach from the given state. To avoid these problems and achieve a closer approximation of the goal state distribution $\hat{\beta}^T$, we approximate it by means of a mixture model with individual components $\hat{\beta}^T_c$,

$$\hat{\beta}^T(\mathbf{s}^T) \approx \frac{1}{C} \sum_{c=1}^{C} \hat{\beta}^T_c(\mathbf{s}^T). \tag{4.21}$$

The components c are built from conjunctions ξ_c over ground literals of primitive predicates and functions which partially describe world states achieving high reward. For instance, ξ_c might define the tower in $\mathbf{s}_2$ above. We choose these formulas ξ_c without taking the initial state $\mathbf{s}_0$ or knowledge about actions in terms of rules into account—to separate this clearly from planning. Concerning unspecified properties of the final state, we use a prior $P^F(\mathbf{s})$ and define the component $\hat{\beta}^T_c$ as

$$\hat{\beta}^T_c(\mathbf{s}) \;\propto\; \delta_{\mathbf{s},\xi_c}\, P^F(\mathbf{s}), \quad \delta_{\mathbf{s},\xi_c} = \begin{cases} 1 & \text{if} \quad \mathbf{s} \Vdash \xi_c \\ 0 & \text{otherwise} \end{cases} \;.$$

We choose $P^F(\mathbf{s})$ such that states close to the initial state $\mathbf{s}^0$ are highly probable. This is inspired by traditional AI backward reasoning: there, state variables not in ξ_c are left unspecified until either they need to be set as required by the preconditions of a rule during backward search or until the initial state is achieved in which case all unspecified variables in the final state implicitly get set to their values in the initial state. This assumption is also advantageous for the factored frontier as the repeated multiplication of small probabilities (such as uninformed 0.5 for binary variables) may lead to very small rule context probabilities, decreasing the probabilities of unique rules.

Backward Messages

For each component $\hat{\beta}^T_c$ of the mixture model approximation of $\hat{\beta}^T$ given in Eq. (4.21), we sample N backward action sequences $\mathbf{b}_{ci} = (b^{T-1}_{ci}, \ldots, b^{T-D_{\leftarrow}}_{ci})$ of horizon $D_{\leftarrow}$ using PRADA and the backward model B, where we set $\hat{\beta}^T_c$ as the initial state distribution. One such sample $\mathbf{b}_{ci}$ results in the distribution $\hat{\beta}^t_{\mathbf{b}_{ci}}(\mathbf{s}^t) = P(\mathbf{s}^t \,|\, \mathbf{b}_{ci}, \xi_c, U^T)$. We do not want to evaluate a forward action sequence $\mathbf{a}$ with each backward action sequence $\mathbf{b}_{ci}$ individually. Hence, we approximate state posteriors $\hat{\beta}^t(\mathbf{s}^t) = P(\mathbf{s}^t \,|\, U^T)$ generalizing over concrete action sequences as

$$\hat{\beta}^t(\mathbf{s}^t) \approx \frac{1}{C} \sum_{c=1}^{C} \frac{1}{N} \sum_{i=1}^{N} \hat{\beta}^t_{\mathbf{b}_{ci}}(\mathbf{s}^t) \; .$$

The resulting $\hat{\beta}$ define the probability of states according to backward reasoning from the goal state mixture distribution $\hat{\beta}^T$ using PRADA's action sampling strategy. They quantify which states are actually backward reachable from goal states.

Evaluating Forward Sequences

Having calculated the backward messages $\hat{\beta}^t(\mathbf{s}^t)$, we sample forward action sequences $\mathbf{a}^{0:t-1}$ using the forward model A and PRADA yielding messages $\alpha_{\mathbf{a}^{0:t-1}}(\mathbf{s}^t) = P(\mathbf{s}^t \,|\, \mathbf{a}^{0:t-1})$. For each $\mathbf{a}$ we are interested in its suitability to achieve a goal state at time T with $t \leq T$. We use the two-filter of Sec. 4.3.1 with the backward state distribution from above to calculate

$$\begin{aligned} P(U^T \,|\, \mathbf{a}^{0:t-1}) &= \sum_{\mathbf{s}^t} P(\mathbf{s}^t \,|\, \mathbf{a}^{0:t-1}) \, P(U^T \,|\, \mathbf{s}^t) \\ &\underset{\sim}{\propto} \sum_{\mathbf{s}^t} \alpha_{\mathbf{a}^{0:t-1}}(\mathbf{s}^t) \, \hat{\beta}^{T-t}(\mathbf{s}^t) \; . \end{aligned}$$

Representing the messages by means of factored frontiers $\alpha(\mathbf{s}) = \prod_i \alpha(s_i)$ and $\hat{\beta}(\mathbf{s}) = \prod_i \hat{\beta}(s_i)$ (dropping indices for clarity), where i ranges over the individual state attributes, we calculate this sum over message products as

$$\begin{aligned} \sum_{\mathbf{s}} \alpha(\mathbf{s})\hat{\beta}(\mathbf{s}) &= \sum_{\mathbf{s}} \prod_i \alpha(s_i)\hat{\beta}(s_i) \\ &= \prod_i \sum_{s_i} \alpha(s_i)\hat{\beta}(s_i) \\ &= \prod_i \sum_{s_i} \alpha(s_i) \frac{1}{C} \sum_{c=1}^{C} \hat{\beta}_c(s_i) \\ &= \prod_i \frac{1}{C} \sum_{s_i} \alpha(s_i) \sum_{c=1}^{C} \hat{\beta}_c(s_i) \; . \end{aligned}$$

Action Selection

For a set $\boldsymbol{\mathcal{A}} = \{\mathbf{a}_1, \ldots, \mathbf{a}_M\}$ of forward action sequence samples of length $D_{\rightarrow}$, we determine the best action sequence $\mathbf{a}^*$ defined as

$$\begin{aligned}\mathbf{a}^* &= \underset{\mathbf{a} \in \boldsymbol{\mathcal{A}}}{\operatorname{argmax}}\ P(U \mid \mathbf{a}) \\ &= \underset{\mathbf{a} \in \boldsymbol{\mathcal{A}}}{\operatorname{argmax}}\ \max_{0 < t \leq D_{\rightarrow}} \sum_T P(T)\ P(U^T \mid \mathbf{a}^{0:t-1}), \\ &\approx \underset{\mathbf{a} \in \boldsymbol{\mathcal{A}}}{\operatorname{argmax}}\ \max_{0 < t \leq D_{\rightarrow}} \sum_T \gamma^T \sum_{\mathbf{s}_t} \alpha_{\mathbf{a}^{0:t-1}}(\mathbf{s}^t) \hat{\beta}^{T-t}(\mathbf{s}^t)\ ,\end{aligned}$$

where we take different horizons T to achieve a goal state into account, discounting them with $P(T) = \gamma^T$ with $0 < \gamma < 1$ to favour smaller horizons.

4.3.3 Evaluation

We compare our backward-forward reasoning approach PRADA$\leftrightarrows$ to the purely forward approaches Upper Confidence Bounds on Trees (UCT) (Kocsis and Szepesvari, 2006), PRADA$\rightarrow$ and A-PRADA$\rightarrow$ (see Sec. 4.2). If rules contain probabilistically dominant outcomes, UCT can be viewed as almost making the corresponding state transitions deterministic. Our test domain is the robot manipulation scenario where a robot manipulates cubes, balls and boxes scattered on a table.

We employ the rule learning algorithm of Pasula et al. with the same parameter settings to learn forward and backward action models in form of fully abstract NID rules from training sets of 500 experience triples each. Training data to learn rules are generated in a world of two boxes, six cubes and four balls of two different sizes by performing random actions with a slight bias to build high piles. Learning a backward model is more difficult as deictic references can often not be uniquely resolved (cf. Sec. 4.3.1). We suspect this to be a domain-specific characteristic rather than a general directional bias (see Massey (1999) for a discussion of the metaphysics of directionality in planning). The resulting backward models are compact and cover the standard situations that arise in the tasks (such as lifting a clear object). We learn one backward model (9 abstract rules) and three different forward models (12-14 abstract rules) from independent training data.

We perform three experiments. In each experiment, we investigate different worlds with varying numbers of objects. For each object number we create five start situations with different objects. Per rule-set combination and start situation, we perform three independent runs with different random seeds. For evaluation, we compute the mean planning times and performances over the fixed (but randomly generated) set of 45 test scenarios (3 *learned* forward rule-sets, 1 *learned* backward rule-set, 5 situations, 3 random seeds). In all experiments, we use deliberately overestimated planning horizons D as these can't be known apriori. For PRADA$\leftrightarrows$, we set $D_{\leftarrow}$ and $D_{\rightarrow}$ each equal to $\frac{1}{2}D$.

Clearance The goal in our first experiment is to clear up the objects which are scattered over the desktop. An object is defined to be cleared if it is piled with all objects of the

Table 4.7: *Clearance problem. Obj.* denotes the object number (cubes/balls and table) and *Reward* the discounted total reward, which is 0 for performing no actions. PRADA⇆ is the proposed bidirectional reasoning approach.

Obj.	Planner	Reward	Trial time (s)
6+1	UCT	32.13±0.41	31.85±1.47
6+1	PRADA→	53.76±0.45	**7.64±1.34**
6+1	A-PRADA→	53.11±0.35	17.11±1.34
6+1	PRADA⇆	**54.10±0.48**	14.48±1.41
8+1	UCT	15.05±0.70	166.05±6.36
8+1	PRADA→	31.33±0.94	**65.90±1.00**
8+1	A-PRADA→	32.23±0.97	76.97±1.47
8+1	PRADA⇆	**33.12±1.09**	**65.91±2.01**
10+1	UCT	32.15±0.97	1148.81±29.83
10+1	PRADA→	97.25±1.96	426.84±16.41
10+1	A-PRADA→	88.40±1.75	444.46±11.10
10+1	PRADA⇆	**111.34±1.96**	**399.04±6.04**

Table 4.8: *Reverse tower problem. Suc.* is the success rate and *Actions* the number of used actions in case of success. PRADA⇆ is the proposed bidirectional reasoning approach.

Obj.	Planner	Suc.	Trial time (s)	Actions
5+1	UCT	0.0	> 1h	–
5+1	PRADA→	0.91	16.38±1.74	11.85±1.21
5+1	A-PRADA→	0.89	18.12±1.88	12.43±1.27
5+1	PRADA⇆	**0.93**	**10.69±0.47**	**10.12±0.47**
6+1	PRADA→	0.80	24.27±1.27	**12.06±0.67**
6+1	A-PRADA→	**0.89**	27.59±2.28	12.62±0.93
6+1	PRADA⇆	0.83	**18.20±0.80**	12.26±0.47
7+1	PRADA→	**0.62**	129.83±8.44	14.75±0.80
7+1	A-PRADA→	0.60	123.20±5.70	**13.70±0.60**
7+1	PRADA⇆	0.58	**99.91±5.23**	14.77±0.87

same color. In our experiments, 2-4 objects have the same color with at most 1 ball (to enable successful piling). The starting situations contain piles, but only with objects of different colors. We let the robot perform 20 actions in worlds of 6 objects (in addition to the table), 30 for 8 and 40 for 10 objects. We emphasize that we did not use any world knowledge to set the goal state mixture distribution for PRADA⇆. In particular, the mixtures also contain clearly impossible situations (as could be deduced from the rules), for examples piles where balls are the lowest objects. Table 4.7 shows our results. UCT performs worst even when admitted very long planning times. We controlled the other approaches to have about the same planning time. In this rather easy planning problem not requiring long horizons, the additional computational overhead of combing forward and backward reasoning starts to pay off in worlds of 10 objects. Then, the planning problem has achieved a certain level of complexity (a very large state space) and PRADA⇆ performs significantly better than the pure forward approaches.

Table 4.9: *Box tower problem. Obj.* denotes the number of objects (cubes/balls, boxes and table), *Suc.* the success rate and *Actions* the number of used actions in case of success. PRADA⇆ is the proposed bidirectional reasoning approach.

Obj.	Planner	Suc.	Trial time (s)	Actions
3+3+1	UCT	0.29	331.94±2.13	**7.62±0.27**
3+3+1	PRADA→	0.84	12.64±0.20	11.53±1.14
3+3+1	A-PRADA→	0.82	11.89±0.17	10.51±0.94
3+3+1	PRADA⇆	**0.91**	**10.09±0.22**	14.66±2.14
4+3+1	UCT	0.0	> 1h	–
4+3+1	PRADA→	0.67	29.33±0.51	17.50±1.94
4+3+1	A-PRADA→	0.67	31.52±0.39	14.90±1.21
4+3+1	PRADA⇆	**0.73**	**22.98±0.46**	**12.36±1.68**
5+3+1	PRADA→	0.40	72.09±1.39	25.22±2.35
5+3+1	A-PRADA→	0.38	64.58±1.68	21.88±2.75
5+3+1	PRADA⇆	**0.51**	**61.03±1.06**	**17.35±1.55**

Reverse Tower The goal is to reverse a tower of c cubes. This is a difficult planning task requiring a long planning horizon. (Depending on the available rule-sets, the minimum horizon may be less than $2c$ as the robot may predict that a cube on top of the grabbed cube may land on the table.) We set a limit of 50 actions on each trial. Table 4.8 presents our results. UCT cannot be used for this task requiring over an hour for a trial. To achieve about the same performance as PRADA⇆, the forward PRADA approaches need 25-70% more planning time. Backward reasoning prunes the search-space and hence speeds up planning.

Box Tower The goal is to build a specific tower of cubes and balls on one of three available boxes, no matter which one. All boxes are closed in the beginning. One of them contains the object which shall be on top of the goal tower. All the other objects are scattered on the table. This is a difficult planning problem as the robot may erroneously start building the desired tower just on the filled box before taking out the required object. As above, no specific world knowledge is used to construct the goal state mixture of PRADA⇆ which may also contain components where the desired tower is built on the filled box, increasing planning difficulty. The minimum number of required actions is $1 + 2 \cdot o$ where o is the number of objects (besides the box) in the target tower. We set a limit of 50 actions on each trial. Table 4.9 presents our results. As above, UCT is not competitive. PRADA⇆ always has the highest success rate—while at the same time requiring the smallest planning times. Backward reasoning is particularly useful in this scenario as the number of possible actions is comparatively small in goal states in comparison to start states. This is also reflected in the smaller number of executed actions to achieve a goal state in worlds with many objects.

4.3.4 Conclusions and Future Work

We have introduced an approach for bidirectional reasoning in ground stochastic relational domains based on probabilistic two-filter inference which combines forward and backward reasoning. Our empirical results show that by exploiting the knowledge about high reward states, we can significantly increase both planning accuracy and efficiency. Finding appropriate mixture models to approximate goal state beliefs to account for partial goal descriptions is a major topic of future research. Also, in a more traditional planning sense one might investigate using the backward messages as proposal distribution for biasing the forward messages for search. Furthermore, investigating the relationships to lifted backward reasoning as is done in symbolic dynamic programming (Boutilier et al., 2001) would provide many insights.

4.4 Relevance Grounding

Complex environments typically contain very many objects. Consider for example a household robot that has to represent all kinds of furniture, dishes, house inventory and the like together with their properties and relationships. Such realistic domains comprise state spaces that are exponential in the number of represented objects and large sets of stochastic actions. Probabilistic relational models describe the action effects and state transitions compactly in terms of abstract logical formulas, but how to exploit this model compactness for planning remains a major challenge. Planning in the fully grounded representation can become inefficient if the number of objects grows very large. This problem is often simply ignored by designing the domain carefully to only contain those domain aspects which are relevant for successful planning. For truly autonomous agents operating continuously with changing tasks, however, we require principled ways to make planning in complex environments tractable.

Models of human cognition provide an inspiring idea of how one may plan in a highly complex world. Humans are often assumed to possess declarative world knowledge about the types of objects they encounter in daily life (Anderson, 1993) which is presumably stored in the long-term memory. For instance, they know that piling dishes succeeds the better the more exactly aligned these dishes are. This abstract knowledge is independent of any concrete dish instance or other unrelated objects (such as lamps and cars) and is akin to abstract probabilistic relational models. When planning, human beings may reason about objects according to their abstract world knowledge (Botvinick and An, 2009) by grounding their abstract world model with respect to these objects (Fig. 4.9). Such reasoning is often assumed to take place in the working memory, a cognitive system functioning as a work-space in which recently acquired sensory information and information from long-term memory are processed for further action such as decision-making (Baddeley, 1999; Ruchkin et al., 2003). This system has *limited capacity* and thus humans are limited in their capacity to concurrently reason about or mentally manipulate several explicit objects at the same time.[2] But concurrent reasoning about

[2] For instance, the classical literature on the working memory capacity for concurrent reasoning speaks

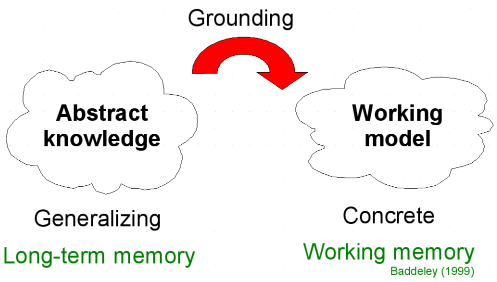


Figure 4.9: *A model of human cognition.* To solve a planning task, humans use their abstract knowledge in the long-term memory to build a concrete working model with the task-relevant objects.

very many objects may—in real-world tasks—not be necessary. Rather, successful planning typically involves only a small subset of relevant objects: humans only take those objects into account which they deem relevant for the problem at hand. For example, when planning to prepare a cup of tea, they do not consider the frying pan in the shelf or a soccer ball in the garage. One can view this as grounding the abstract world knowledge only with respect to these relevant objects, thereby enabling tractable planning. In the planning-by-inference paradigm this corresponds to the approach of instantiating (grounding) the abstract knowledge only for a limited number of explicit objects to form a grounded dynamic Bayesian network.

In this section, we take up this idea and exploit the great advantage of abstract relational world models to be applicable to arbitrary subsets of objects and introduce the framework of relevance grounding (Lang and Toussaint, 2009a). First, we define object relevance in terms of a graphical model. This allows us then to prove consistency between repeated planning in partially grounded models restricted to relevant objects and planning in the fully grounded model. Thereby, we reformulate the original intractable problem into tractable versions where we can apply any efficient planning method to solve our problem at hand, enabling real-time planning and planning with quickly changing goals. Empirical results in our robotic manipulation scenario using a learned world model show the effectiveness of our approach.

The remainder of this section is organized as follows. In the next section, we introduce a formal model of relevance grounding in terms of a graphical model. In Sec. 4.4.2, we provide a definition of object relevance. In Sec. 4.4.3, we present our algorithm which exploits relevance grounding. We briefly touch on learning object relevance in Sec. 4.4.4 and then provide an empirical evaluation. Finally, in Sec. 4.4.6 we discuss related work before we conclude.

of the "magic number" being about seven (Miller, 1956). Many more modern studies argue for different numbers in different contexts, but confirm the capacity limitation in principle.

4.4.1 A Formal Model of Relevance Grounding

Grounding a relational representation language $\mathcal{L}$ w.r.t. all objects $\mathcal{O}$ of a domain results in a state space that is exponential in $|\mathcal{O}|$. Thus, evaluating an action sequence is exponential in $|\mathcal{O}|$. Furthermore, the set of ground actions and thus the search space of plans scales with the number of objects. (Planning is even further complicated due to the stochasticity of actions.) Planning is only tractable in case $|\mathcal{O}|$ is very small. In most realistic scenarios, $|\mathcal{O}|$ is rather large, however. Fortunately, it often suffices to take only the objects that are relevant for the planning problem into account. In the following model, we formalize the idea *relevance grounding* in a systematic way. We introduce a probabilistic model which expresses the coupling between state sequences, action sequences, objects and rewards. This model will help to formalize what planning with subsets of objects implies. In particular, we will be able to derive results on planning with subsets of objects—which corresponds to conditioning on object sets $\mathbf{o}$.

By Γ we denote the model grounded for all objects, including the complete state and action space (all ground atoms w.r.t. $\mathcal{O}$), which defines the state transition dynamics according to some given relational transition model $\mathcal{M}$. Let $\mathbf{a} = (a_1, \ldots, a_T)$ denote a *plan*, i.e., a sequence of actions. Let $\mathbf{s} = (s_1, \ldots, s_T)$ denote a sequence of encountered states. We assume that in a given trial $(\mathbf{s}, \mathbf{a})$ certain objects are relevant while others are not. For example, an object o is relevant if it is an argument of one of the actions in $\mathbf{a}$. We give a concrete definition of object relevance in Sec. 4.4.2. For now, we only assume that, in general, object relevance can be expressed by a conditional probability $P(\mathbf{o}|\mathbf{s}, \mathbf{a})$ where $\mathbf{o}$ is a random variable referring to a subset of $\mathcal{O}$. Let u denote the event of achieving a reward at the end of a trial[3]. We assume the following joint distribution over these random variables representing the graphical model shown in Fig. 4.10:

$$P(u, \mathbf{o}, \mathbf{s}, \mathbf{a}; \Gamma) = P(u \,|\, \mathbf{s}, \mathbf{a}; \Gamma)\, P(\mathbf{o} \,|\, \mathbf{s}, \mathbf{a}; \Gamma)\, P(\mathbf{s} \,|\, \mathbf{a}; \Gamma)\, P(\mathbf{a}; \Gamma)\ . \tag{4.22}$$

Note that all conditional distributions depend on the model Γ. In absence of goals or rewards, we assume a uniform prior over plans $P(\mathbf{a}; \Gamma)$, discounted by their length T by a discount factor $0 < \gamma < 1$ (see footnote 3). In the following, we will eliminate $\mathbf{s}$, that is, we consider

$$P(u, \mathbf{o}, \mathbf{a}; \Gamma) = \sum_{\mathbf{s}} P(u, \mathbf{o}, \mathbf{s}, \mathbf{a}; \Gamma) = P(u \,|\, \mathbf{o}, \mathbf{a}; \Gamma)\, P(\mathbf{o} \,|\, \mathbf{a}; \Gamma)\, P(\mathbf{a}; \Gamma)\ ,$$

where u now conditionally depends on $\mathbf{o}$.

Generally, planning requires finding the maximizing argument $\mathbf{a}^*$ of the following distribution:

$$P(\mathbf{a} \,|\, u; \Gamma)\ \propto\ P(u \,|\, \mathbf{a}\,;\, \Gamma)\, P(\mathbf{a}; \Gamma)\,.$$

Finding plans with high $P(\mathbf{a} \,|\, u; \Gamma)$ is a difficult task for the following reasons: *(i)* the search space of $\mathbf{a}$ scales with the number of objects; *(ii)* evaluating $P(u \,|\, \mathbf{a}; \Gamma)$ is difficult

[3] When we assume a geometric prior on the trial length, the expected reward is equivalent to the sum of discounted rewards when rewards are given in each time-step—see Toussaint and Storkey (2006) for details.

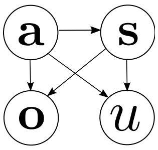


Figure 4.10: The graphical model of relevance grounding couples actions $\mathbf{a}$, states $\mathbf{s}$, objects $\mathbf{o}$ and the reward u.

as Γ's state space is exponential in the number of objects $\mathcal{O}$. To overcome this problem, we observe that we can decompose

$$P(\mathbf{a} \,|\, u; \Gamma) = \sum_{\mathbf{o}} P(\mathbf{a} \,|\, \mathbf{o}, u; \Gamma) P(\mathbf{o} \,|\, u; \Gamma) \tag{4.23}$$

where $P(\mathbf{o} \,|\, u; \Gamma)$ is defined as

$$P(\mathbf{o} \,|\, u \,;\, \Gamma) \propto \sum_{\mathbf{a}} P(u \,|\, \mathbf{o}, \mathbf{a} \,;\, \Gamma)\, P(\mathbf{o} \,|\, \mathbf{a}; \Gamma)\, P(\mathbf{a}; \Gamma)\ . \tag{4.24}$$

$P(\mathbf{o} \,|\, u; \Gamma)$ is a measure for the relevance of object sets with respect to the reward. Note that this posterior favors small object sets due to the prior over plan lengths in $P(\mathbf{a}; \Gamma)$. If every successful plan makes use of object o, then for each $\mathbf{o}$ with $P(\mathbf{o} \,|\, u \,;\, \Gamma) > 0$ we have $o \in \mathbf{o}$. In this case, we call o *necessary* for u. Using this formalization of the relevance of object sets, Eq. (4.23) provides us a way to decompose the above planning problem into two stages: *(i)* sampling of object sets using $P(\mathbf{o} \,|\, u; \Gamma)$; *(ii)* finding plans with high $P(\mathbf{a} \,|\, \mathbf{o}, u; \Gamma)$ corresponding to planning conditioned on a set of relevant objects. The key idea is that the conditioning on $\mathbf{o}$ in stage *(ii)* may significantly reduce the cost of planning, as we will discuss below.

4.4.2 A Sufficient Definition of Relevance

In the following, we provide a definition of $P(\mathbf{o} \,|\, \mathbf{s}, \mathbf{a})$ which we have neglected thus far. For a given pair $(\mathbf{s}, \mathbf{a})$, we define the set Ω of relevant object sets as

$$\begin{aligned}\Omega(\mathbf{s}, \mathbf{a}) = \{&\mathbf{o} \subseteq \mathcal{O} \,|\, \forall t, 0 \le t < T : P(s_{t+1} \,|\, s_t, a_t; \Gamma) = P(s_{t+1} \,|\, s_t, a_t; \Gamma_{\mathbf{o}}) \\ \wedge\ \ &\forall \mathbf{o}' \subset \mathbf{o}, \mathbf{o}' \neq \mathbf{o}\ \ \exists t, 0 \le t < T : P(s_{t+1} \,|\, s_t, a_t; \Gamma) \neq P(s_{t+1} \,|\, s_t, a_t; \Gamma_{\mathbf{o}'})\}\end{aligned} \tag{4.25}$$

where $\Gamma_{\mathbf{o}}$ is the *reduced model* including only the objects $\mathbf{o}$ with their ground atoms. $P(\mathbf{s}_{t+1} \,|\, \mathbf{s}_t, \mathbf{a}_t; \Gamma_{\mathbf{o}})$ is defined such that all atoms with at least one argument $o \notin \mathbf{o}$ persist from $\mathbf{s}_t$ and are ignored while calculating transition probabilities, and we define the action prior in the reduced model as $P(\mathbf{a}; \Gamma_{\mathbf{o}}) = P(\mathbf{a} \,|\, \mathbf{o}; \Gamma)$. $\Omega(\mathbf{s}, \mathbf{a})$ comprises minimal object sets that are required to predict the state transitions correctly for a specific trial $(\mathbf{s}, \mathbf{a})$. Note that $|\Omega(\mathbf{s}, \mathbf{a})| \ge 1$ for all $(\mathbf{s}, \mathbf{a})$. We define $P(\mathbf{o} \,|\, \mathbf{s}, \mathbf{a})$ as

$$P(\mathbf{o} \,|\, \mathbf{s}, \mathbf{a}) := \frac{I(\mathbf{o} \in \Omega(\mathbf{s}, \mathbf{a}))}{|\Omega(\mathbf{s}, \mathbf{a})|}\ . \tag{4.26}$$

Intuitively, relevant object sets are those that are taken into account to calculate the transition probabilities in s for a given $\mathbf{a}$. Clearly, they include the objects which are manipulated, i.e., whose properties or relationships change. We call these *actively relevant*. There are also *passively relevant* objects which are taken into account by the transition model $\mathcal{M}$. For instance, imagine the task to go to the kitchen and prepare a cup of tea. The tea bag, the cup and the water heater are actively relevant objects. If the kitchen has two doors and one of them is locked, then the latter is passively relevant: we cannot manipulate, that is open, it, but it plays a role in planning as its being locked determines the other door to be necessary. A more technical example of object relevance is given below.

In general, there might be alternative interesting definitions of object relevance, for example where the transition probabilities in Eq. (4.25) only hold approximately. We chose the above definition because it is sufficient to a certain consistency for planning in reduced models (the proof can be found in Appendix A.2):

Lemma 4.4.1 *When conditioning on a subset* $\mathbf{o}$ *of relevant objects, the following probabilities in the reduced model* $\Gamma_{\mathbf{o}}$ *are the same as in the full model* Γ*:*

- *State sequences:* $P(\mathbf{s} \,|\, \mathbf{o}, \mathbf{a}; \Gamma) = P(\mathbf{s} \,|\, \mathbf{a}; \Gamma_{\mathbf{o}})$
- *Rewards:* $P(u \,|\, \mathbf{o}, \mathbf{a}; \Gamma) = P(u \,|\, \mathbf{a}; \Gamma_{\mathbf{o}})$
- *Action sequences:* $P(\mathbf{a} \,|\, \mathbf{o}, u; \Gamma) = P(\mathbf{a} \,|\, u; \Gamma_{\mathbf{o}})$

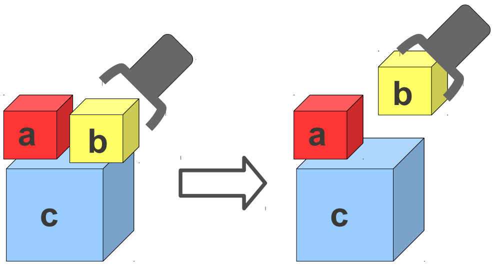
From Lemma 4.4.1 and Eq. (4.23) the following proposition follows directly:

Proposition 4.4.2 *Given the joint in Eq. (4.22) and the definition of* $P(\mathbf{o} \,|\, \mathbf{s}, \mathbf{a})$ *in Eq. (4.26), it holds:*

$$P(\mathbf{a} \,|\, u; \Gamma) = \sum_{\mathbf{o}} P(\mathbf{a} \,|\, u; \Gamma_{\mathbf{o}}) \; P(\mathbf{o} \,|\, u; \Gamma) \, .$$

Illustrative Example of Object Relevance

The scenario in Table 4.10 illustrates active and passive object relevance. Two small cubes a and b are on top of a big cube c. Our goal is to hold b inhand. This can be achieved by means of a plan consisting of a single action $grab(b)$. Our transition model $\mathcal{M}$ contains two rules to model the $grab$-action. Rule 1 applies if the target cube is the only cube on top, in which case $grab$ always succeeds. Rule 2 applies if the target cube is not the only cube on top. In this case, $grab$ only succeeds with probability 0.8; otherwise with probability 0.2, grabbing fails due to lack of space and the target cube is pushed off the big cube instead. Clearly, in our situation we have to use Rule 2. Cubes b and c are manipulated and thus are actively relevant, whereas a is passively relevant as it determines Rule 2 to apply. If a was ignored, we would use Rule 1—yielding an erroneous higher success probability. Similar scenarios are typical in physical worlds: the probabilities of successful planning change when objects (for example, potential obstacles) are added to or removed from the scene even when they are not actively manipulated.



Action: $grab(B)$

Rule 1:
$grab(X): \quad on(X,Y),\ \forall Z \neq X: \neg on(Z,Y)$
$\rightarrow 1.0 : inhand(X),\ \neg on(X,Y)$

Rule 2:
$grab(X): \quad on(X,Y),\ on(Z,Y)$
$\rightarrow 0.8 : inhand(X),\ \neg on(X,Y)$
$\quad 0.2 : \neg on(X,Y)$

Table 4.10: Example of active and passive object relevance. Objects b and c are actively relevant as their properties are changed. a is passively relevant as it determines Rule 2 to model the transition. If a was ignored, Rule 1 would be used instead yielding wrong state transition probabilities.

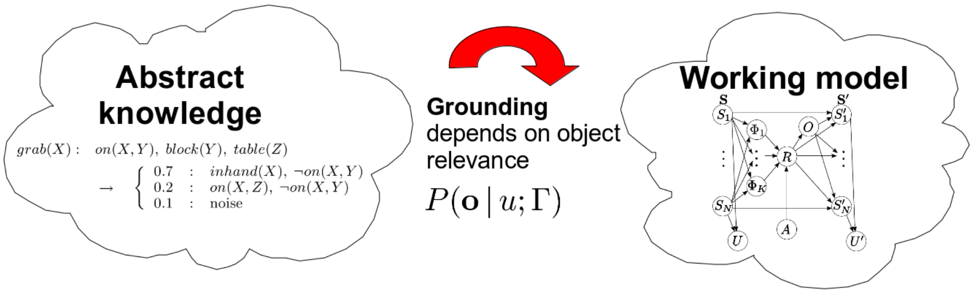

Abstract knowledge
$grab(X): \quad on(X,Y),\ block(Y),\ table(Z)$
$\rightarrow \begin{cases} 0.7 : inhand(X), \neg on(X,Y) \\ 0.2 : on(X,Z), \neg on(X,Y) \\ 0.1 : \text{noise} \end{cases}$
Grounding
depends on object relevance
$P(\mathbf{o} \mid u; \Gamma)$
Working model
s
s'
S_1
S'_1
Φ_1
O
R
Φ_K
S_N
S'_N
U
A
U'

Figure 4.11: *A model of relevance grounding.* Analogous to the model of human cognition shown in Fig. 4.9, the abstract model Γ is grounded with respect to a set $\mathbf{o}$ of task-relevant objects resulting in a reduced model $\Gamma_{\mathbf{o}}$ with smaller state and action spaces. In our case, the abstract model consists of probabilistic relational rules, while the concrete model is a ground DBN.

4.4.3 Planning with Relevant Objects

Our definition of object relevance and the subsequent discussion led to a crucial observation: to find plans with high $P(\mathbf{a} \mid u; \Gamma)$, it is not necessary to use the full model Γ including all objects $\mathcal{O}$. As Proposition 4.4.2 shows, an alternative is to find plans in the reduced models $P(\mathbf{a} \mid u; \Gamma_{\mathbf{o}})$ for object sets $\mathbf{o}$ with high relevance $P(\mathbf{o} \mid u; \Gamma)$. This makes planning more efficient due to the reduced state and action spaces in $\Gamma_{\mathbf{o}}$. The analogy to the model of human cognition of Fig. 4.9 is visualized in Fig. 4.11.

Obviously, we do not know $P(\mathbf{o} \mid u; \Gamma)$. If we knew the reward likelihoods for all plans, in other words, if we had already planned, we could calculate this quantity according to Eq. (4.24). However, planning is just the problem we are trying to solve. Thus, we have to estimate this quantity by some distribution $q(\cdot)$ over object sets resulting in

Algorithm 2 Relevance grounding

```
Input: objects $\mathcal{O}$, goal $\tau$, relational transition model $\mathcal{M}$, relevance distribution $q(\cdot)$, number of relevance groundings $N_{rel}$, number of verifications $N_{ver}$
Output: action sequence $\mathbf{a}$
for $i = 1$ to $N_{rel}$ do        ▷ Relevance grounding
    Sample object set $\mathbf{o}_i \subset \mathcal{O}$ according to $q$
    Build reduced model $\Gamma_{\mathbf{o}_i}$
    $\mathbf{a}_i = plan(\tau; \Gamma_{\mathbf{o}_i}, \mathcal{M})$        ▷ Plan in reduced model
    $\psi(\alpha_i) = P(u \,|\, \mathbf{a}_i; \Gamma_{\mathbf{o}_i}, \mathcal{M})$        ▷ Value in reduced model
end for
for $i = 1$ to $N_{ver}$ do        ▷ Verifying in original model
    Let $\mathbf{a}$ denote plan with $i$-th largest $\psi$
    Calculate $\Psi(\mathbf{a}) = P(u \,|\, \mathbf{a}; M, \mathcal{M})$        ▷ Value in original model
end for
return $\text{argmax}_{\mathbf{a}} \Psi(\mathbf{a})$
```

the approximate distribution $Q(\cdot)$ over plans defined as

$$Q(\mathbf{a}; u, \Gamma) \quad = \textstyle\sum_{\mathbf{o}} P(\mathbf{a} \,|\, u; \Gamma_{\mathbf{o}}) \; q(\mathbf{o}; u, \Gamma) \approx P(\mathbf{a} \,|\, u; \Gamma) \; . \tag{4.27}$$

The quality of $Q(\cdot)$ depends on the quality of the approximate relevance distribution $q(\cdot)$. If $q(\cdot)$ is not exact, then a plan found in $\Gamma_{\mathbf{o}}$ may have lower success probability when planning in Γ instead (see the example in Table 4.10). Therefore, it is a good idea to verify the quality of the proposed plan in the original model Γ or in a less reduced model $\Gamma_{\mathbf{o}'}$ with $\mathbf{o} \subset \mathbf{o}'$. This requires algorithms that can exploit the transition model $\mathcal{M}$ to efficiently calculate $P(u \,|\, \mathbf{a})$ also in large models.

Algorithm 2 presents our complete *relevance grounding* method. Given an estimator for object set relevance $q(\cdot)$, we can find plans with approximately high $P(\mathbf{a} \,|\, u; \Gamma)$ as follows: *(i)* we take samples $\mathbf{o}$ from $q(\cdot)$; *(ii)* we plan in the reduced models $\Gamma_{\mathbf{o}}$; *(iii)* we verify the resulting plans in the original or a less reduced model; *(iv)* we return the plan with the best verified value. In our experiments, we employ NID rules as transition model $\mathcal{M}$ and use the PRADA algorithm for planning which is in particular appropriate for verification as it evaluates an action sequence in time linear in its length.

4.4.4 Learning Object Relevance

A crucial part in our proposed method is the relevance estimator of object sets. Learning such an estimator is a novel and interesting machine learning problem. For a given goal τ, we can use our transition model $\mathcal{M}$ to create training instances $(\sigma_0, \tau, \mathbf{o}, P(\mathbf{o} \,|\, u))$. σ_0 is a description of the start state s_0 and may involve all types of information, such as discrete, relational and continuous features. We can employ any regressor that can make use of the chosen features to learn a function $q(\mathbf{o}; \sigma_0, \tau) \to \mathbb{R}$, mapping an object set $\mathbf{o}$ to its reward likelihood $P(\mathbf{o} \,|\, u)$ and parameterized according to the start situation σ_0

and the goal τ. As we are using relational representations, we can generalize over object identities in our planning goals τ and start situations σ_0 and transfer the knowledge gained in previous planning trials to new, but similar problems. We can create the training instances $(\sigma_0, \tau, \mathbf{o}, P(\mathbf{o} \,|\, u))$ from reasoning based on $\mathcal{M}$ without acting. Thus, we can learn object relevance based on nothing more than internal simulation (in contrast to "real" experiences)—akin to human reflection about a problem. A full approach to learning object relevance is beyond the scope of this section, but in our first experiment we will present an example of how to learn object relevance in a straightforward way, based purely on internal simulation.

4.4.5 Evaluation

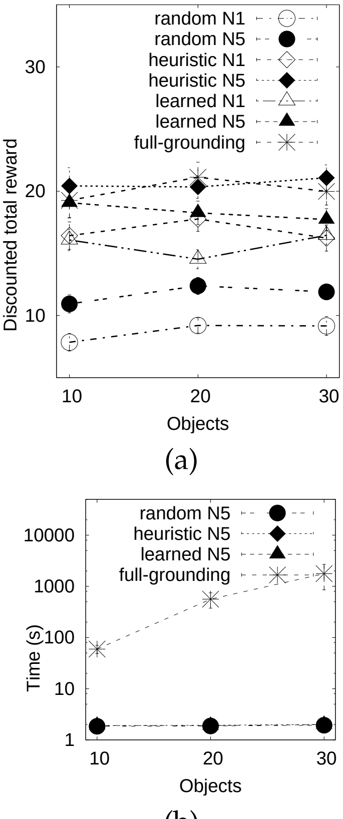
We test our *relevance grounding* approach in our robot manipulation scenario (see Sec. 1.1.1). We use NID rules to model the state transitions. We employ the rule learning algorithm of Pasula et al. (2007) with the same parameter settings to learn three different sets of fully abstract NID rules from independent training sets of 500 experience triples each. Training data to learn rules are generated in a world of ten objects (six cubes, four balls) of two different sizes by performing random actions with a slight bias to build high piles. The resulting rule-sets contain 11, 12 and 12 rules respectively. We use the PRADA algorithm (Sec. 4.2) for planning. We test our approach in worlds with varying numbers of cubes and balls of two different sizes. In each experiment, for each object number we create five start situations with different objects. Per rule-set and start situation, we perform three independent runs with different random seeds. In all scenarios, we make the assumption that relevant object sets contain 5 objects. We investigate different estimators to determine these 5 objects. To evaluate each approach, we compute the planning times and the mean performance over the fixed (but randomly generated) set of 45 test scenarios (3 *learned* rule-sets, 5 situations, 3 seeds).

High Towers In our first experiment, we investigate building high piles. Our starting situations are chosen such that all objects have height 0 (are on the table) and our reward is the total change in object heights. We let the algorithm run for 10 time-steps. We set PRADA's planning horizon to $d = 6$ and use a discount factor of $\gamma = 0.95$. If the world was deterministic and objects could be stacked perfectly (such that objects could also be stacked on balls), the optimal discounted total reward would be 37.04.

We investigate three different relevance estimators to determine the sets of relevant objects. The *random* estimator samples objects randomly and independently. The *handmade heuristic* assigns high probability to big cubes (since these are best to build with) and to objects that are either part of a high pile or on the table (in order to build higher piles). Once it has sampled an object, it assigns high probability to objects within the same pile as these might be required for deictic referencing in the NID rules (passive object relevance).

Furthermore, we investigate a simple *learned* estimator of object relevance from which we sample objects independently. We use linear regression to learn from discrete and

Table 4.11: *High towers problem*: (a) Mean rewards (changes in tower heights), (b) planning times, (c) details over 45 runs (3 rule-sets, 5 start situations, 3 seeds). Error bars for the rewards give the std. dev. of the mean estimator. N_{rel} denotes the number of relevant reduced models (cf. Algorithm 2). Performing no actions gives a reward of 0.



(a)

(b)

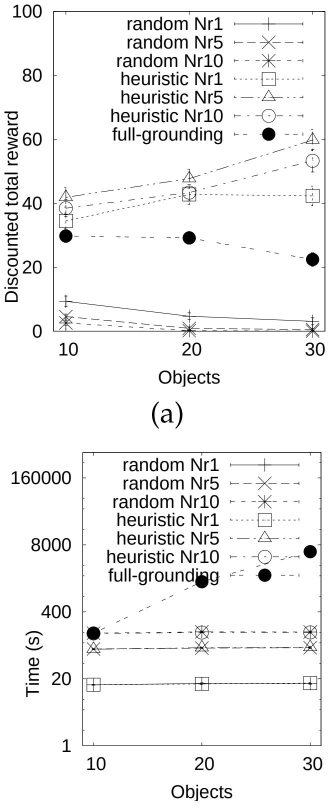
Obj.	Config	Reward	Time
10	random $N_{rel}=1$	7.85 ± 0.70	**0.37 ± 0.03**
	random $N_{rel}=5$	10.94 ± 0.73	1.84 ± 0.18
	random $N_{rel}=10$	14.59 ± 1.20	3.63 ± 0.26
	heuristic $N_{rel}=1$	16.42 ± 1.10	0.38 ± 0.02
	heuristic $N_{rel}=5$	20.43 ± 1.47	1.89 ± 0.11
	heuristic $N_{rel}=10$	**20.83 ± 1.32**	3.78 ± 0.23
	learned $N_{rel}=1$	16.07 ± 0.85	0.39 ± 0.02
	learned $N_{rel}=5$	19.07 ± 1.23	1.91 ± 0.11
	learned $N_{rel}=10$	18.63 ± 1.12	3.76 ± 0.29
	full-grounding	19.26 ± 1.38	59.51 ± 10.72
20	random $N_{rel}=1$	9.20 ± 0.62	0.39 ± 0.03
	random $N_{rel}=5$	12.38 ± 0.69	1.87 ± 0.15
	random $N_{rel}=10$	14.93 ± 0.74	3.78 ± 0.21
	heuristic $N_{rel}=1$	17.77 ± 0.99	**0.39 ± 0.02**
	heuristic $N_{rel}=5$	20.34 ± 0.91	1.93 ± 0.11
	heuristic $N_{rel}=10$	20.69 ± 1.16	3.85 ± 0.15
	learned $N_{rel}=1$	14.53 ± 0.77	0.42 ± 0.03
	learned $N_{rel}=5$	18.26 ± 0.94	1.90 ± 0.13
	learned $N_{rel}=10$	18.34 ± 0.82	3.81 ± 0.11
	full-grounding	**21.12 ± 1.21**	561.78 ± 186.76
30	random $N_{rel}=1$	9.16 ± 0.76	**0.38 ± 0.03**
	random $N_{rel}=5$	11.90 ± 0.64	1.93 ± 0.12
	random $N_{rel}=10$	14.00 ± 0.69	3.84 ± 0.25
	heuristic $N_{rel}=1$	16.23 ± 1.05	0.39 ± 0.02
	heuristic $N_{rel}=5$	**21.08 ± 1.03**	2.01 ± 0.13
	heuristic $N_{rel}=10$	20.21 ± 1.10	3.84 ± 0.16
	learned $N_{rel}=1$	16.45 ± 0.77	0.42 ± 0.04
	learned $N_{rel}=5$	17.72 ± 0.88	1.99 ± 0.04
	learned $N_{rel}=10$	18.44 ± 0.75	3.78 ± 0.24
	full-grounding	19.99 ± 1.11	1770.55 ± 916.44

(c)

logical object features, namely object size, type, color, height and clearedness. Training data are generated solely by internal simulation with the PRADA algorithm (in contrast to using the "real" ODE simulator) as follows: for a given situation, we randomly sample 5 objects and derive a plan in the partially grounded network; this plan is then evaluated in the full network and the resulting value is used as relevance estimate for these 5 objects. Note that this procedure does not require real experiences as it is fully based on internal reasoning about which features make an object relevant according to the learned world model (the NID rules in our case). The resulting learned estimator ignores object color as expected, but takes all other features into account, favoring clear big cubes at high heights. We compare these three relevance estimators to the *full-grounding* baseline which plans in the fully grounded model.

Table 4.11 presents our results. The mean performance of the heuristic is comparable to the full-grounding baseline. The performance of the learned estimator is comparable or only slightly worse than the heuristic, depending on the number of objects and

Table 4.12: *Clearance problem*: (a) Mean rewards, (b) planning times, (c) details over 45 runs (3 rule-sets, 5 start sit., 3 seeds). Error bars for the rewards give the std. dev. of the mean estimator. N_r is the number of relevant reduced models N_{rel}, N_v of partial plans N_{ver} verified in a less reduced model. Performing no actions gives a reward of 0.



(a)

(b)

Obj.	Config	Reward	Time
10	random N_r= 1	9.33 ± 1.67	**15.26 ± 0.21**
	rand. N_r= 5	4.59 ± 0.78	75.73 ± 1.53
	rand. N_r= 10	2.58 ± 0.73	153.86 ± 3.37
	rand. N_r= 5, N_v=3	17.54 ± 1.92	84.63 ± 2.42
	rand. N_r= 10, N_v=3	14.76 ± 1.85	162.25 ± 7.60
	heuristic N_r= 1	34.50 ± 2.44	15.31 ± 0.27
	heur. N_r= 5	41.88 ± 3.08	75.02 ± 1.64
	heur. N_r= 10	38.48 ± 2.67	151.92 ± 2.94
	heur. N_r= 5, N_v=3	46.46 ± 3.12	84.98 ± 1.83
	heur. N_r= 10, N_{vr}=3	**49.15 ± 2.81**	161.79 ± 3.63
	full-grounding	29.77 ± 2.02	153.93 ± 13.51
20	random N_r= 1	4.68 ± 1.07	16.24 ± 0.44
	rand. N_r= 5	0.88 ± 0.45	78.92 ± 1.08
	rand. N_r= 10	0.16 ± 0.11	163.21 ± 4.93
	rand. N_r= 5, N_v=3	2.62 ± 0.90	90.25 ± 1.74
	rand. N_r= 10, N_v=3	0.72 ± 0.58	168.91 ± 3.31
	heuristic N_r= 1	42.77 ± 3.19	**15.89 ± 0.48**
	heur. N_r= 5	47.72 ± 2.96	80.24 ± 1.38
	heur. N_r= 10	43.34 ± 2.60	158.53 ± 2.06
	heur. N_r= 5, N_v=3	51.03 ± 3.12	88.94 ± 1.99
	heur. N_r= 10, N_v=3	**56.76 ± 2.68**	172.27 ± 5.01
	full-grounding	29.19 ± 1.98	1537.37 ± 225.96
30	random N_r= 1	3.09 ± 1.09	**16.06 ± 0.41**
	rand. N_r= 5	0.52 ± 0.26	80.11 ± 1.47
	rand. N_r= 10	0.02 ± 0.02	162.17 ± 6.34
	rand. N_r= 5, N_v=3	1.16 ± 0.51	92.17 ± 2.52
	rand. N_r= 10, N_v=3	0.16 ± 0.10	178.96 ± 3.56
	heuristic N_r= 1	42.31 ± 3.06	16.47 ± 0.43
	heur. N_r= 5	**59.79 ± 3.33**	81.34 ± 3.87
	heur. N_r= 10	53.29 ± 3.50	159.01 ± 3.39
	heur. N_r= 5, N_v=3	55.00 ± 3.54	90.26 ± 3.46
	heur. N_r= 10, N_v=3	58.01 ± 3.42	168.51 ± 5.16
	full-grounding	22.42 ± 2.14	5893.81 ± 1006.56

(c)

the number N_{rel} of partially grounded models, but always significantly better than the random estimator. The performance of all estimators improves with increasing N_{rel}, but this effect diminishes if N_{rel} is large. Planning in the fully grounded model is hopelessly inefficient, in particular for large worlds. The same performance levels can be achieved by means of Relevance Grounding in only tiny fractions of this planning time, which is independent of the object number and thus constant over all investigated domain sizes.

Clearance The goal in our second experiment is to clear up the desktop (see Fig. 4.12). Objects are lying scattered all over the desktop. An object is cleared if it is part of a pile containing all other objects of the same class, which can be defined as

$$cleared(X) \equiv \forall Y : sameClass(X,Y) \to samePile(X,Y) \ . \tag{4.28}$$

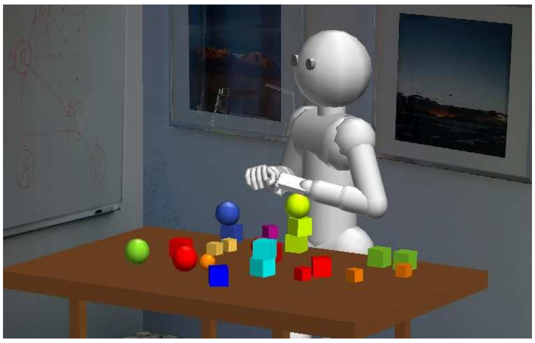


Figure 4.12: *Clearance task with relevance grounding.* The robot has to clear up the desktop by piling objects of the same size and color. Focusing on relevant objects permits efficient reasoning in this scenario containing many objects.

A class is defined in terms of color and size, but not type so that a class contains both cubes and balls. In our experiments, classes are made up of 2-4 objects with at most 1 ball (in order to enable successful piling). Our starting situations contain some piles, but only with objects of different classes. We let the algorithm run for 30 time-steps. For planning, we set PRADA's planning horizon to $d = 20$ and use a discount factor of $\gamma = 0.95$. If the world was deterministic and objects could be stacked perfectly, the optimal values would be 86.91 for worlds with 10 objects and 110.85 for worlds with 20 and 30 objects. We investigate two relevance estimators. The random estimator samples randomly and independently among all objects. The heuristic estimator chooses randomly among the objects which are not cleared yet and then takes all other objects of the same class into account. Nearest neighbors are used to fill up the object set. While this heuristic is hand-made, its idea can be derived from the logical reward description in Eq. (4.28) which states the importance of classes in relevant object sets on the left side of the implication. How this can be done in principled ways is a major direction of future work.

Table 4.12 presents our results. The random estimator performs poorly since its reduced models contain mostly only single instances of a class. This is disadvantageous as planning requires at least a second object of the same class and singleton instances are always cleared within reduced models which are thus a bad approximation of the full model. The mean performance of the heuristic estimator is significantly better than the full-grounding baseline, in particular in worlds with many objects. Note that the full-grounding baseline cannot find an optimal solution due to the huge search space. In contrast to planning in the fully ground model, the relevance grounding planning approaches are independent of the number of objects and thus several orders of magnitude faster.

We also investigate the use of verification of the plans found in the reduced models (cf. Algorithm 2). We evaluate the $N_{ver} = 3$ best reduced-model plans in a less reduced model containing 10 objects where the missing 5 slots are filled in by nearest neighbors. Thereby, information about objects within the same piles may be taken into account. Our results show that this improves the mean performance for both relevance estimators significantly at only a small increase in computational cost. In particular, this greatly increases the performance of the random estimator in worlds with 10 objects. In larger worlds, the random estimator almost never finds good plans in which case verification cannot help.

4.4.6 Related Work on Reduced Ground Relational Models

An alternative strategy to relevance grounding for reasoning in a ground relational model is to consider only a small relevant subset of the state space which is derived from the start state and the planning goal. In contrast to our approach, the resulting subspace still represents all objects, thus the action space size is not decreased. A straight-forward way to create such a subspace are look-ahead trees for the start state that estimate the value of an action by taking samples of the corresponding successor state distribution (as done by the algorithms SST and UCT described in Sec. 4.2.1). Another idea is to maintain an *envelope* of states, a high-utility subset of the state space which can be used to define a relational MDP (Gardiol and Kaelbling, 2003). This envelope can be further refined by incorporating nearby states in order to improve planning quality. A crucial part of this approach is the initialization of the envelope which is based on an initial straight-line path from the start state to a goal state using a heuristic forward planner (for example, by making this planning problem deterministic by only considering the most-probable successor state of an action). The envelope-based approach depends strongly on the efficiency and quality of this initial planner which is still faced with the complexity of the action space and its dependence on the number of objects, thus being applicable only for rather small planning horizons.

Action space complexity can be decreased by noting that if the identities of objects do not matter but only their relationships, then different equivalent actions may lead to equivalent successor states (Gardiol and Kaelbling, 2007). These are states where the same relationships hold, but not necessarily with the same objects. Relevance grounding accounts for this idea by defining different object subsets to be relevant for the planning problem at hand. Action equivalence can be exploited during planning by only considering one sampled action per action equivalence class which significantly reduces the search space. If identity matters for a large number of objects, however, then this approach does not yield significant improvements. Another way to reduce the state space complexity is to look only at a subset of the logical vocabulary, that is, to ignore certain predicates and functions (Gardiol and Kaelbling, 2008). This is inspired by work on abstract decision-theoretic planning in factored propositional MDPs where omitting state features to build abstract MDPs and policies has been shown to lead to great computational savings, while guarantees on the found solution can still be provided (Dearden and Boutilier, 1997). Ignoring predicates and functions helps in particular when combined with the action equivalence approach as state descriptions become shorter and more approximate and the number of state equivalences increases. All these methods just discussed are complementary to our approach and when applied in a reduced ground model within the relevance grounding framework might yield a promising way to plan efficiently in highly complex domains.

4.4.7 Conclusions and Future Work

We have presented an approach for efficient planning in stochastic relational worlds based on exploiting object relevance. In particular in environments with very many objects (> 1000) we believe that organizing goal-directed behavior without such a focus on relevant objects is infeasible. We define object relevance in terms of a graphical model. We have derived a systematic framework to plan in partially grounded models which we have proven to be consistent with planning in the fully grounded model. Empirical results show our approach to be effective in complex relational environments. Also, we have argued that our approach has interesting analogies to human cognition. Our framework is independent of the concrete planning algorithm used within the reduced models. In particular, it can well be combined with other approaches to increase planning efficiency in stochastic relational domains that have recently been introduced, such as envelope-based methods (Gardiol and Kaelbling, 2003). Our approach is orthogonal to the goal of developing (lifted) inference methods that can cope with many objects.

A key part for our framework and the major direction of future research is the estimator of object relevance. We have provided a successful example of how relevance can be learned from object features by means of nothing more than internal simulation, but this is clearly only preliminary. In our point of view, learning to estimate the relevance of objects is a formidable problem for machine learning, as a huge variety of methods using discrete, continuous, and logical features can be applied. Clearly, this is a difficult problem, bearing in mind that human beings often take a long time until they master certain types of planning problems. However, estimating object relevance appears to us to be a crucial prerequisite to be able to plan in the highly complex real world.

4.5 Relational Planning By Inference

Our algorithm PRADA described in Sec. 4.2 uses approximate inference for forward reasoning. It builds a graphical model, the PRADA-DBN, from a set of NID rules and the domain objects $\mathcal{O}$; conditioning on actions, it propagates the effects of actions by means of a factored frontier filter; the actions are sampled according to an informed sampling strategy; it evaluates several sampled action-sequences and chooses the one with the highest expected reward. In contrast, full planning-by-inference approaches condition on rewards, propagate information also backwards, infer posteriors over actions and thereby calculate a policy. For this purpose, the underlying graphical models require a coupling between predecessor states and actions. Learning a policy amounts to calculating the parameters of this coupling: these parameters define the conditional probability functions in DBNs and the factors in factor graphs.

Full planning-by-inference is a hard task in relational domains due to the very large state and action spaces. Exact inference is infeasible as the majority of state variables typically gets correlated after inference over only few time-steps: each variable representing a ground atom can be manipulated by several actions and for combinatorial reasons beliefs over many, if not all variables, need to be maintained. Thus, we need approximate inference techniques such as loopy belief propagation (LBP). As described above, LBP fails in the graphical models of the PRADA-DBN type, even when using a damping factor. We attribute this to the centralized structure of PRADA-DBN with its central high-dimensional nodes for actions, rules and outcomes.

As an alternative, here we explore approximate inference in the highly decomposed factor graph presented in Sec. 3.2 and shown in Fig. 3.3(d) on page 40. We perform tests with this graphical model structure in a simple world of cubes using the following two NID rules to describe the transition dynamics,

$$\begin{array}{ll} \mathit{move}(X,Y,Z): & \mathit{on}(X,Y),\ \mathit{clear}(X),\ \mathit{clear}(Z),\ \mathit{cube}(Y) \\ \quad \rightarrow & \left\{ \begin{array}{lll} 0.8 & : & \mathit{on}(X,Z),\ \mathit{clear}(Y),\ \neg \mathit{on}(X,Y),\ \neg \mathit{clear}(Z) \\ 0.2 & : & - \end{array} \right. \quad \text{and} \\ \mathit{moveFromTable}(X,Y,Z): & \mathit{on}(X,Y),\ \mathit{clear}(X),\ \mathit{clear}(Z),\ \mathit{table}(Y) \\ \quad \rightarrow & \left\{ \begin{array}{lll} 0.8 & : & \mathit{on}(X,Z),\ \neg \mathit{on}(X,Y),\ \neg \mathit{clear}(Z) \\ 0.2 & : & - \end{array} \right. \quad , \end{array}$$

where the first rule describes moving a cube X from a second cube Y to a position on-top of object Z while in the second rule the origin object Y is the table (the table is always "clear"). Our planning task is to build specific towers. We assign full evidence on the first and last time-slices in the factor graph, corresponding to the first state and the goal state (hence, the belief at the last time-step corresponds to the posterior $P(s_T \mid U^T = 1)$ when U assigns reward for exactly one goal state). We apply standard LBP to infer the posteriors over action variables.

In a tiny domain consisting of 3 objects (2 cubes, 1 table) a state consists of 12 state variables (9 for on-atoms, 3 for clear-atoms, omitting typing predicates). LBP works perfectly for a single state transition where the planning problem involves moving exactly one cube: the correct action has clearly the highest posterior. LBP still works in a slightly bigger problem consisting of 4 objects (3 cubes, 1 table) and 2 state transitions: the correct actions have the highest posteriors, but some false actions also have large posteriors. In problems with more than 4 objects and more than two state transitions, however, LBP in our graphical model fails: too many actions have high posteriors then. We played with the usage of damping, but it did not help to get LBP working in these bigger settings. In other test domains, we experienced similar problems.

A known problem of LBP is coping with deterministic graph structures. This might also affect our setting since the only non-deterministic information in our graphs is in the outcome factors. To get a better understanding of our problems with approximate inference, we examined the messages calculated during LBP: it appears that the approximation in particular of the backward messages is disadvantageous; also it seems to be particularly harmful for the approximate inference in our graphs that LBP ignores correlations.

These problems indicate that it may not be straightforward to apply LBP techniques to DBNs representing ground relational domains. Nonetheless, we think that the symmetries inherent in relational domains and the decoupling of nodes in our graphical models pose a great potential for the development of efficient techniques for relational planning-by-inference. Modified graphical models and different inference methods, in particular lifted methods, are worth investigation.

4.6 Relational Planning on a Real Robot

Autonomous robots deal with information on different levels of abstraction: they process low-level sensory input to gain the perceptual information they are interested in, reason about their high-level goals and actions, and translate abstract actions into low-level motor control. A central problem of modern robotics is how to integrate these different levels of abstraction for decision-making, planning and control, which requires a coherent principle of information processing. A general framework for information processing is provided by inference in graphical models which provides a principled way to define the couplings of variables with the corresponding uncertainties. Inference can be viewed as internal simulation for control, planning and decision making (see Sec. 2.3). In previous work, this approach has been applied on low-level motor control (Toussaint and Goerick, 2007; Toussaint, 2009a). In this thesis, we introduce probabilistic inference methods for high-level planning. So far, we performed successful experiments in simulated environments.

In this section, we describe a real-world demonstration where a robot integrates information across multiple levels of abstraction (Toussaint, Plath, Lang, and Jetchev, 2010). Our contribution to this demonstration is the high-level abstract reasoning capacity of the robot. We focus on **qualitative** aspects: our goal here is to demonstrate the general feasibility of the inference approach in a real-world scenario where an autonomous robot manipulates several objects in a goal-directed way. A quantitative investigation with multiple experiments and runs requires major efforts with respect to the non-high-level aspects of the robot (such as vision and motor control) which are beyond the scope of this thesis.

Our target scenario is a real blocksworld (Fig. 4.13): a robot with a 14 degrees-of-freedom (DoF) Schunk arm and hand with tactile sensors and a stereo camera manipulates objects on top of a table. By addressing this scenario we want to bring the blocksworld, perhaps the most popular scenario in classical AI since the 1970s, to real life. We decompose the problem of acting in the real blocksworld according to the different levels of abstraction and apply appropriate algorithms based on approximate inference on the level of motor control and trajectory optimization as well as of high-level planning.

We proceed as follows. First, we describe our target scenario in more detail. We present the different methods of our approach in Sec. 4.6.2. In Sec. 4.6.3, we discuss our qualitative evaluation on a real robot before we conclude. A video of the experiments and additional material such as source code can be found at the following web-page:

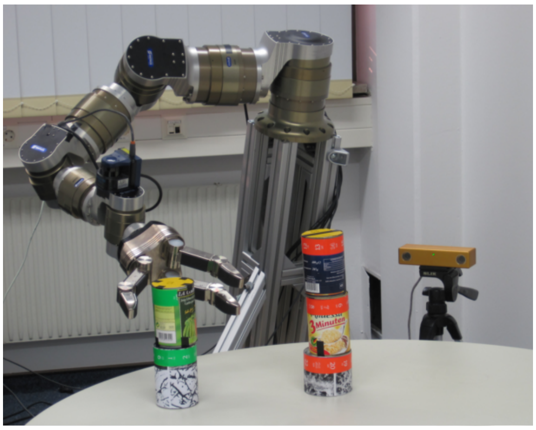


Figure 4.13: The robot has successfully put objects with green and red labels into separate piles, using probabilistic inference on different levels of abstraction for planning and control.

`http://userpage.fu-berlin.de/mtoussai/10-ICRA/` .

4.6.1 Target Scenario

Our overall goal is autonomous goal-directed manipulation in environments with multiple objects. In this chapter, we have introduced methods for planning in stochastic relational worlds and demonstrated these methods in physically simulated robot manipulation problems like clearing the desktop or building towers from objects of different sizes and shapes. Here we want to address similar scenarios, but on a real robotic platform. To solve the scenario we require a series of methods for learning, perception, planning and control: Eventually, the robot will need to

1. learn a high-level stochastic model of the effects of actions like grabbing and placing an object,
2. use vision to identify and localize objects,
3. use a stochastic relational planner to compute a sequence of actions,
4. use trajectory optimization to compute dynamically smooth reaching and pre-grasp motions,
5. use a controller to follow the computed trajectories,
6. and use a tactile feedback controller to execute the grasp.

Our robotic platform is shown in Figure 4.13 and includes the following hardware components:

- Schunk Light Weight Arm (LWA) with 7 DoF

- Schunk Dextrous Hand (SDH) with 7 DoF
- 6×14 tactile arrays on each of the 6 finger segments
- Bumblebee stereo camera

4.6.2 Methods

High-level Relational Planning

We employ the same symbolic representation as in the previous sections to describe our domain, using predicates such as $inhand(\cdot)$, $upright(\cdot)$, $on(\cdot, \cdot)$ and functions such as $size(\cdot)$ for world states and predicates such as $grab(\cdot)$ and $puton(\cdot)$ for actions. We learn NID rules (Sec. 3.1.3) from simulation to define a transition model $\mathcal{M}$ and plan with PRADA (Sec. 4.2) (Lang and Toussaint, 2009b, 2010b).

A major challenge in developing intelligent robots is how to couple high-level reasoning with sensors and low-level motor control. We approach the first problem by a set of simple heuristics to translate object information from vision and tactile sensors into our symbolic representation. For instance, we derive $on(a, b)$ if the x/y-coordinates of objects a and b are sufficiently similar while b's z-coordinate is slightly smaller, and $inhand(b)$ if b is closest to the robot's fingers and we get significant tactile feedback. To translate the high-level action symbols to concrete robot action, we set them to trigger the execution of the corresponding low-level motor control routines for grasping and placing.

Low-level Control and Sensing

The control framework follows in detail the approach of Toussaint (2010). We control the robot on a dynamic level. Let $q_t \in \mathbb{R}^{14}$ be the vector of all joint angles in the LWA arm and SDH hand at time t. The control operates on the phase state

$$x_t = (q_t, \dot{q}_t) \in \mathbb{R}^{28} \tag{4.29}$$

comprising joint angles and velocities. The control framework is based on having many *task variables* concurrently active with various precisions. Such task variables are used to define goal and constraints on robot movements. We use the *Approximate Inference Control* (AICO) algorithm (Toussaint, 2009a) for trajectory planning, optimizing reach, pre-grasp and place trajectories, following the optimized trajectories and grasping and releasing objects. AICO solves a stochastic optimal control problem based on probabilistic inference. It uses planning-by-inference to compute a posterior over the whole trajectory conditioned on all desired task variables. For robot vision, the identification and localization of objects is based on SURF interest points and descriptors (Bay et al., 2008).

4.6.3 Evaluation

Our experiments are designed to focus on **qualitative** aspects: we investigate whether the different methods can indeed be successfully applied in a real world domain and our approach is suitable to fully control an autonomous robot to achieve its goals. A quantitative investigation with multiple experiments and runs is beyond the scope of this thesis as explained in the introduction. For quantitative studies with respect to the individual methods, we refer the reader for a comparison and discussion of different low-level robot control approaches to the respective papers (Toussaint, 2009a; Toussaint and Goerick, 2007) and for high-level reasoning to the previous sections.

In our experimental setup, the Schunk robot is placed in front of a table with cylindric objects of two different sizes and colors. In the scenario of the cited video, the goal is to "clear up" the desktop: to stack objects of the same color onto each other. There are two big and one small red object and two big green objects stacked in three piles, where two piles contain two objects of different colors. To achieve the goal of a cleared desktop, the robot needs to grab and place three objects in total.

The first problem is to localize the objects using the stereo camera which is placed next to the robot arm. The objects have individual patterns which are used to identify them. Once the objects are recognized, their coordinates are calculated and a symbolic world state representation is derived. Then, the robot uses the PRADA algorithm (Lang and Toussaint, 2009b, 2010b) to derive a high-level plan of actions to achieve a cleared desktop. A set of 11 abstract NID rules has been learned beforehand using the algorithm of Pasula et al. (2007) with the same parameter settings from a set of 500 experiences of state transitions. We have generated these experiences in a 3D rigid-body dynamics simulator (ODE) of the scenario including the robot, the objects and the table by performing random actions with a slight bias to build high towers.

After a suitable plan has been found, the individual abstract actions trigger the respective low-level motor control routines, namely AICO, to generate grasp and placing trajectories. Fig. 4.14 (left) illustrates the start posture (central hand position) and the end posture of an optimized reach and pre-grasp trajectory. The pictures are taken from the simulator that is internally used by AICO. The red distance markers illustrate the output of the collision (distance proximity) detection engine. The end posture is a good pre-grasp with the fingers wrapping around the object with about 3 cm distance. When this pre-grasp posture has been reached the grasp is executed based on a tactile feedback loop. Fig. 4.14 (right) displays the start posture (hand at left pile) and end posture (hand at center pile) of an object placing movement.

The video shows a complete demonstration to solve the clearance task with five objects. PRADA found the correct sequence of actions necessary to stack the two objects with green labels on one pile. To reach this state, two objects with red labels first have to be removed from them by placing them on the center pile for the red-labeled objects. Fig. 4.13 shows the end posture after the whole trial.

The experiments also revealed limitations on the precision of our kinematic model of the arm and the object localization, causing some placed objects not to align as perfectly as they do in the simulation. This is exactly one of the situations why stochastic

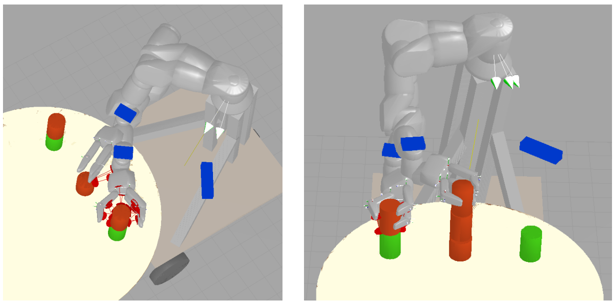


Figure 4.14: Visualization of start and end postures of calculated trajectories. The blue bars are the stereo camera (to the right of the robot) and a laser scanner (not used in these experiments, mounted on the arm close to the hand). *(left)* A grab action moving the hand from the center to the right pile. *(right)* A place action moving the hand with the object from the left to the center pile. The red distance markers indicate critical proximities between collidable objects.

action models are investigated in reinforcement learning and probabilistic reasoning is necessary.

4.6.4 Conclusions and Future Work

Our work is the first step towards fully integrated autonomous acting in an environment with multiple objects. We have focused on qualitative aspects and shown the general feasibility of using inference in graphical models to integrate information across multiples levels of abstraction in a real-world scenario. We chose the blocksworld scenario in analogy to typical AI benchmarks to demonstrate that the methods developed in this thesis, such as PRADA, can be transferred to the real world. While the fields of robot learning and probabilistic robotics have focused mainly on problems such as motor skill learning, system identification and state estimation, our work is among the first approaches to robot learning of abstract knowledge and object manipulation on the level of relational models. Related recent work has investigated robot navigation based on policy learning using relational decision trees (Cocora et al., 2006) and learning kinematic structures from physical interaction (Katz and Brock, 2009, 2008; Sturm et al., 2009), which may likewise be considered as a first form of relational learning on real robots.

Future work should apply our approach to scenarios that are more challenging for both the high-level planning as well as the low-level control: scenarios which include different types of objects and more cluttered scenes with big obstacles the robot has to avoid. It is worth examining how our approach can handle perceptional uncertainty which could lead to movement failures such as accidentally pushing objects off piles; this has direct consequences on reasoning with stochastic relational models.

4.7 Discussion

In natural environments with many objects, an autonomous agent needs to plan with a given or learned probabilistic relational model to achieve its goals or actively explore its domain. In this chapter, we have investigated planning approaches reasoning on the level of concrete objects in the ground relational domain. Existing state-sampling based search methods such as SST (Kearns et al., 2002) and UCT (Kocsis and Szepesvari, 2006) have difficulties to scale to the exponential spaces of relational domains. Likewise, exact planning-by-inference in a fully grounded relational model is infeasible. Therefore, we have proposed the algorithm PRADA for forward reasoning in stochastic relational domains which converts learned noisy probabilistic relational rules to a ground DBN representation and uses approximate inference in the DBN for planning. PRADA is the first practical planning method for learned relational rules and we have demonstrated its high performance in simulated complex robot manipulation domains of balls, cubes and boxes, where rules have been learned from experience, and on benchmarks of the international planning competition, where hand-made rules are provided. We have extended PRADA to a bidirectional reasoning method where we apply its approximate inference also for backward reasoning. Our experimental results show that by exploiting the knowledge about high reward states, we can significantly increase both planning accuracy and efficiency. We also succeeded in a first demonstration of probabilistic inference methods for organizing sequential manipulation on a real-world robotics platform, where the model was learned from simulator experiences (but used on the real robot) and high-level planning was achieved with PRADA.

Abstracting from the details of PRADA, we believe the reason for the success of PRADA is the way knowledge is transformed from a rule-based representation (suitable for learning) to a graphical-model-based representation (suitable for inference and planning). In our view, such transformations of knowledge between different representations, each suitable for different computational tasks, are fundamentally important for finding efficient solutions. In our case, the learner of compact rule-sets by Pasula et al. (2007) motivated the probabilistic rule-based representation; the planning-by-inference theory motivated the graphical-model representation for planning. The success of PRADA hints at the great potential of (approximate) inference techniques for reasoning in stochastic relational domains and opens the door to a large variety of methods based on inference in different representations. We have briefly explored full planning-by-inference in factor graphs which decompose the structure of ground relational domains. We have seen that in principle relational planning-by-inference using approximation techniques is possible, but that dealing with the large state and action spaces of relational domains requires more study. In particular, more advanced techniques for lifted inference in factor graphs, such as lifted belief propagation, have the potential to greatly improve efficiency and thus effectiveness of relational planning.

All our methods reason in the ground relational domain. Such methods may become inefficient with very large numbers of represented objects. Inspired by psychological concepts of human cognition, we have proposed the framework of relevance grounding to avoid this problem where we ground a relational domain only with respect to task-relevant objects. Thereby, our planning methods become applicable also to worlds with very large object numbers. In our view, the relevance of objects is of vital importance for organizing goal-directed behavior. Surprisingly, research in artificial intelligence and machine learning has largely ignored this important concept. Future work should investigate potential priors for object relevance in real world manipulation tasks as well as other domains. An obvious candidate is spatial proximity to goal objects, but there are surely other potential priors. We have shown first promising results on learning models of object relevance.

The effectiveness of our methods for planning in the real world is determined by the underlying transition model. Thus, future research on planning should investigate extensions of the rule framework or other representations to account for more powerful features as well as algorithms to learn them. This would in turn require appropriate modifications of our techniques. For instance, instead of using a noisy default rule, one may use mixture models to deal with actions with several (non-unique) covering rules, or in general use parallel rules that work on different aspects of the underlying system or on different hierarchical levels, permitting for instance multi-time-step abstraction.

The planning approaches introduced in this chapter assume that a transition model of the effects of the agent's actions is provided. Typically, an autonomous agent is not given such a model in advance and instead has to learn it from interaction with the environment. How an environment with many objects can be explored efficiently is the focus of the next chapter.

Chapter 5

Relational Exploration

In reinforcement learning, an autonomous agent's learning task is to find a policy for action selection that maximizes its reward over the long run. In the following, we assume the agent is told about the states where it receives reward, but has no idea in the beginning how its actions change the state of the world and how it can get to high reward states. We pursue a model-based approach where the agent learns models of the transition process from its interactions with the environment. These models are then analyzed to compute plans to drive active exploration, achieve high reward states and thereby perform tasks.

Typically, the agent solves tasks in similar situations of the same unknown environment. While doing so, the agent may find a good model based on its experience up to the current moment, but there might be an even better model based on states and actions unexplored so far. Thus, one of the key challenges in reinforcement learning is the exploration-exploitation tradeoff, which strives to balance two competing types of behavior of an autonomous agent in an unknown environment: the agent can either make use of its current knowledge about the environment to maximize its cumulative reward (that is, to exploit), or sacrifice short-term rewards to gather information about the environment (that is, to explore) in the hope of increasing future long-term return, for instance by improving its current world model. This exploration-exploitation tradeoff has received considerable attention in unstructured domains and several powerful techniques have been developed such as E^3 (Kearns and Singh, 2002), R^{max} (Brafman and Tennenholtz, 2002) and Bayesian reinforcement learning (Poupart et al., 2006). However, while these methods give a clear idea of how in principle exploration and exploitation can be organized, the corresponding basic algorithms only work on enumerated state spaces—toy worlds. In this chapter, we take up the principle ideas of these methods and investigate how we can realize efficient exploration in non-trivial relational representations.

The environment of the agent typically contains varying numbers of objects and relations among them. Learning and acting in such large relational domains is the second key challenge in reinforcement learning. Such relational domains are hard—or even impossible—to represent meaningfully using an enumerated state space. For instance, consider a hypothetical household robot which is taken out of its shipping box, switched on and then has to get acquainted with its new environment to be able to fulfill its duties. Without a compact knowledge representation that supports abstraction and generalization of previous experiences to the current state and potential future states, it seems to be difficult—if not hopeless—for such a "robot-out-of-the-box" to explore one's home in reasonable time. There are too many objects such as doors, plates and water-taps. For instance, after having opened one or two water-taps in bathrooms, the priority for exploring further water-taps in bathrooms, and also in other rooms such as the kitchen, should be reduced. This is impossible to express in a propositional setting where we would simply encounter a new and therefore non-modeled situation.

So far, however, the important problem of exploration in stochastic relational worlds has received little attention. This is exactly the problem we address in this chapter. Applying existing, propositional exploration techniques is likely to fail: what in a propositional setting would be considered a novel situation and worth exploration may in the relational setting be an instance of a well-known context in which exploitation is promising. This is the key insight of the current chapter:

> *The inherent generalization of learned knowledge in the relational representation has profound implications also on the exploration strategy.*

Consequently, we develop relational exploration strategies (Lang, Toussaint, and Kersting, 2010, 2011) in this chapter, inspired by Kearns and Singh's basic exploration technique E^3 (Explicit Explore or Exploit, discussed in detail below). In more detail, this includes the following steps:

- We develop a relational model-based reinforcement learning framework lifting ideas from E^3 to the relational case.
- We apply relational density estimation to estimate a smoothed empirical density for relational data which we exploit in our relational exploration strategies. While the problem of relational density estimation has general importance in many applications, we present different methods to use it in the reinforcement learning context to formalize the novelty of relational states and actions.
- We present our algorithm REX, the first complete practical and implemented solution to exploration and model-based reinforcement learning in relational domains with explicit explore-exploit mechanism. We integrate our relational planner PRADA (Chapter 4) and a learner of probabilistic relational rules (Pasula et al., 2007) into our framework. Based on actively generated training trajectories, the exploration strategy and the relational planner together produce in each round a learned transition model and in turn a policy that either reduces uncertainty about the environment and improves the current model, or exploits the current knowledge to maximize the utility of the agent.

Our extensive experimental evaluation in our robot manipulation domain and in domains of the international planning competition IPPC shows that our approaches can solve tasks in complex worlds where non-relational methods fail.

We proceed as follows. After having described related work, we discuss the existing E^3 framework for exploration in enumerated state spaces in Sec. 5.2. We develop our relational density estimation techniques to estimate the interestingness of relational states and actions in Sec. 5.3 which we use then in our relational exploration framework introduced in Sec. 5.4. As an instantiation of this theoretical framework, we propose a complete relational model-based reinforcement learning algorithm called Rex in Sec. 5.5. We finish this chapter with drawing conclusions and pointing at future work.

Our **contributions** in this chapter are the following (Lang, Toussaint, and Kersting, 2010, 2011):

- We develop a **relational model-based reinforcement learning framework** generalizing over states, actions and objects (Sec. 5.4).
- We introduce the problem of **relational density estimation** in relational exploration strategies and present different methods to use it in a reinforcement learning context to formalize the novelty of relational states and actions (Sec. 5.3).
- We present our **relational exploration algorithm Rex**, the first complete practical and implemented relational model-based reinforcement learning algorithm with explicit explore-exploit mechanism (Sec. 5.5).

5.1 Related Work on Exploration

Exploration has been studied intensely in the classical reinforcement learning scenario based on non-relational representations, but has been largely ignored thus far in relational reinforcement learning as our following presentation of related work shows. We also take a look at active learning which will be important for our relational density estimation framework and at the field of robotics.

Exploration in Non-relational Domains

The first studies on effective exploration in multi-state control problems developed a number of concepts for describing explorative behavior, including curiosity (Schmidhuber, 1991a), seeking to minimize the variance of action value estimates (Kaelbling et al., 1996) and counters on the occurrences of states and actions (Thrun, 1992). Thereafter, (near-)optimal exploration solutions have been developed for unstructured propositional and continuous domains where the environment is represented as an enumerated or vector space. Bayesian reinforcement learning (Poupart et al., 2006) provides an optimal solution to the exploration-exploitation problem in a Bayesian framework by taking all potential models weighted by their posteriors into account at once. This solution is comparable to optimal experimental design in statistics or operations research, but typically infeasible to compute. An alternative approach to optimal exploration are algorithms studied in the probabilistically approximately correct (PAC) framework applied to Markov decision processes (MDPs). The seminal algorithms E^3 (Kearns and Singh, 2002) and R^{max} (Brafman and Tennenholtz, 2002) achieve near-optimal polynomial runtimes in enumerated domains. These powerful theoretical limits refer to the number of states, however, and thus E^3 and R^{max} still scale exponentially in the number of objects. E^3 has been extended to parameter learning in factored propositional MDPs with a fixed structure (Kearns and Koller, 1999) where it scales polynomially in the number of parameters of the underlying DBN. A further concern why such algorithms are hard to apply in practice is that they rely on efficient (polynomial) approximate planers which provide compact μ-optimal policies—however, general algorithms with these guarantees are not known. Therefore, Guestrin et al. (2002) propose an exploration strategy for factored propositional MDPs which is directed towards a specific planning algorithm based on linear programming. Epshteyn et al. (2008) investigate active reinforcement learning in enumerated domains to focus the exploration on the regions of the state space to which the optimal policy is most sensitive.

Relational Reinforcement Learning

In recent years, there has been a growing interest in using rich representations such as relational languages for reinforcement learning (RL). Most work in this context has focused on model-free approaches (estimating a value function) and has not developed relational exploration strategies. Essentially, a number of relational regression algorithms have been developed for use in these relational RL systems such as relational regression trees (Džeroski et al., 2001) and Gaussian processes with graph kernels (Driessens et al., 2006). Kersting and Driessens (2008) introduce relational policy gradients. All of these approaches use some form of ϵ-greedy strategy to handle exploration; no special attention has been paid to the exploration-exploitation problem as done in the current chapter. Driessens and Džeroski (2004) propose the use of "reasonable policies" in model-free relational RL to provide guidance, that is, to increase the chance to discover sparse rewards in large relational state spaces, also known as reward shaping. While our approach described in this chapter can make use of such guidance, it decides autonomously for actions which reveal the relevant structures of the world and thus does not depend on a teacher. Sanner (2005, 2006) combines feature discovery with relational reinforcement learning but does not discuss the problem of relational density estimation for the exploration-exploitation tradeoff in general terms which will form the backbone of our approach. Ramon et al. (2007) present an incremental relational regression tree algorithm that is capable of dealing with concept drift and showed that it enables a relational Q-learner to transfer knowledge from one task to another. They, however, do not learn a model of the domain and, again, relational exploration strategies were not developed.

Exploration with Relational Models

The early approaches on deterministic action schema learning (Benson, 1996; Wang, 1995; Gil, 1994) in relational representations can be seen as first attempts to relational model-based RL. They apply learning techniques from inductive logic programming (ILP) (Nienhuys-Cheng and de Wolf, 1997), which studies the supervised learning of relational concepts from sets of positive and negative examples of relational predicates. ILP algorithms often try to represent the most specific concept that covers the positive training data and excludes the negative data (which are usually separable so uncertainty is no concern). Exploration, however, was no special focus in the first approaches on action schema learning or experience was provided by a teacher. Croonenborghs et al. (2007) learn a probabilistic relational transition model online in form of relational probability trees for individual state attributes and sample look-ahead trees of state transitions to give the agent more informed estimates of the values of action sequences. Exploration is based on sampling random actions instead of informed exploration. The pioneering work of Walsh (2010) is the one closest to ours. Walsh provides the first principled investigation into the exploration-exploitation tradeoff in relational domains and establishes sample complexity bounds for specific relational MDP learning problems, lifting ideas from R^{max} to relational domains. These bounds scale polynomially in the number of parameters and components of the models describing the transitions of the relational domain, such as the number of action schemata and effects, and thus sublinearly in the size of the state space. In contrast to his work, our conceptual focus here is on the relational exploration strategies themselves. Furthermore, our algorithms use a more expressive language and we provide the first empirical evaluation for relational model-based RL with fully learned models. We also show that relational representations are a promising technique to formalize the idea of curriculum learning (Bengio et al., 2009) where first simpler tasks are explored before more difficult tasks are tackled. Our work has interesting parallels in cognitive science: Windridge and Kittler (2010) employ ideas of relational exploration for cognitive bootstrapping, that is, to progressively learn more abstract representations of an agent's environment based on its action capabilities. In Chapter 4, we have shown that successful planning typically involves only a small subset of relevant objects, and Joshi et al. (2010) have shown that it typically also involves only a small subset of relevant states and how to make use of this fact to speed up symbolic dynamic programming significantly. A principled approach to exploration, however, has not been developed.

Active Learning

The question of optimal exploration in model-based RL where we learn a transition model has similarities to the problem of active learning (Cohn et al., 1996; Fedorov, 1972). In statistical relational learning (SRL) (Getoor and Taskar, 2007; de Raedt et al., 2008; de Raedt, 2008), active learning has only recently started to attract attention. Bilgic et al. (2010) investigate the use of relations to identify uncertain regions in the learning space by means of characteristic representatives. Xu et al. (2010) present an active exploration scheme for link-based preference learning. Nevertheless, the idea of relational density estimation in active concept learning can be traced back to Cussens (1998).

Exploration in Robotics

In a robotics learning context in continuous domains, active exploration has been investigated for localization and mapping (Stachniss et al., 2005). In developmental robotics addressing the idea of building systems and robots that mimic human and animal development (Weng et al., 2001; Weng, 2004; Lungarella et al., 2003), the drive for exploration is typically assumed to rely on some form of maximization of the learning progress, which implicitly leads the system to aim for novel situations (Oudeyer et al., 2007). Many of the ideas developed in this area can, in retrospect, be viewed as instances of the theoretically grounded recent work on exploration-exploitation in reinforcement learning. The empirical demonstrations of development and exploration have been limited to domains of low-level sensorimotor contingencies (Oudeyer et al., 2007). It remains an open challenge to organize robot exploration and learning on higher levels of abstraction that reflect the relational structure of natural environments. This is what we pursue in this chapter.

5.2 Background on E^3 in Enumerated State Spaces

The E^3 (Explicit Explore or Exploit) algorithm (Kearns and Singh, 2002) provides a near-optimal model-based solution to the exploration-exploitation problem in unstructured enumerated state spaces. It distinguishes explicitly between exploitation and exploration phases. The central concept are *known states* where all actions have been observed sufficiently often. For this purpose, E^3 maintains state-action counts for all state-action pairs (s, a). A sketch of the algorithm is provided in Algorithm 3; the situations when the concept of known states plays a role in the sketch are emphasized. If E^3 enters an *unknown* state, it takes the action it has tried the fewest times there (**direct exploration**). If it enters a *known* state, it tries to calculate a high-value policy within an MDP built from all known states where its model estimates are sufficiently accurate. If it finds such a policy which stays with high probability in the set of known states, this policy is executed (**exploitation**). Otherwise, E^3 plans in a different MDP in which the unknown states are assumed to have maximal value ("optimism in the face of uncertainty"), ensuring that the agent explores unknown states efficiently (**planned exploration**).

One can prove that with high probability E^3 performs near optimally for all but a polynomial number of time-steps. The theoretical guarantees of E^3 and similar algorithms such as R^{max} are strong. In practice, however, the required numbers of exploratory actions are large so that in case of large state spaces it is unrealistic to meet the theoretical thresholds of state visits. Furthermore, the polynomial bounds of E^3 on the sample complexity refer to the number of states. In relational domains, however, the state space is exponential in the number of objects, and hence, E^3 scales exponentially in the number of objects. To address this drawback, variants of E^3 for factored but proposi-

Algorithm 3 Sketch of E^3

```
Input: State $s$
Output: Action $a$
if $s$ is known then
    Plan in space of known states        ▷ where model estimates are sufficiently accurate
    if resulting plan has value above some threshold then
        return first action of plan        ▷ exploitation
    else
        Plan in modified state space where unknown states get maximum reward
        return first action of plan        ▷ planned exploration
    end if
else
    return action with the least observations in $s$        ▷ direct exploration
end if
```

tional MDP representations have been explored (Kearns and Koller, 1999; Guestrin et al., 2002). Our evaluations will include variants of factored exploration strategies where the factorization is based on ground relational formulas. However, factored MDPs still do not enable generalization over objects. In this chapter, we are investigating exploration strategies for relational representations and lift E^3 to relational domains. This may strongly improve the efficiency and the performance of a reinforcement learning agent. The central idea is to generalize the state-action counts of the original E^3 algorithm over states, actions and objects. We achieve this by modeling the novelty of states and actions as a relational density estimation problem. This is described in the next section.

5.3 A Density Estimation View on Known States and Actions

The theoretical derivations of the non-relational near-optimal exploration algorithms E^3 and R^{max} show that the concept of known states is crucial. On the one hand, the confidence in estimates in known states drives exploitation. On the other hand, exploration is guided by seeking for novel (yet unknown) states and actions. For instance, the direct exploration phase in E^3 chooses novel actions, which have been tried the fewest; the planned exploration phase seeks to visit novel states, which are labeled as yet unknown.

In the case of the original E^3 algorithm (and R^{max} and similar methods) operating in an enumerated state space, states and actions are considered known based directly on the number of times they have been visited. In relational domains, there are two reasons for why we should go beyond simply counting state-action visits to estimate the novelty of states and actions:

1. The size of the state space is exponential in the number of objects. If we base our notion of known states directly on visitation counts, then the overwhelming majority of all states will be labeled yet-unknown and the exploration time required to meet the criteria for known states of E^3 even for a small relevant fraction of the

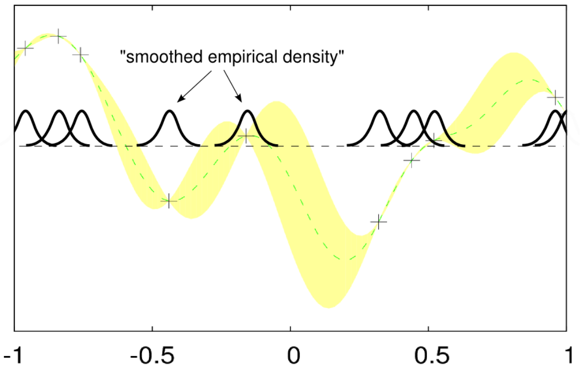


Figure 5.1: In active learning, a density model of previously seen points (crosses) is used to estimate which areas of the input space are known. Here, Gaussians, placed at the observations, enable a smooth estimate of known input points. This density is used to quantify the confidence in the estimated target function (green line with yellow strap).

state space becomes exponential in the number of objects.

2. The key benefit of relational learning is the ability to generalize over yet unobserved instances of the world based on relational abstractions. This implies a fundamentally different perspective on what is novel and what is known and permits qualitatively different exploration strategies compared to the propositional view.

A constructive approach to pinpoint the differences between propositional and relational notions of exploration, novelty and known states is to focus on a density estimation view. This is also inspired by the work on active learning (Fedorov, 1972) which typically selects points that, according to some density model of previously seen points, are novel (see for example Cohn et al. (1996) where the density model is an implicit mixture of Gaussians). This is illustrated in Fig. 5.1. Given some observations, the goal is to estimate a smoothed empirical density which defines the areas of the input space where we are confident about our target function estimates. Usually, we are confident (that is, the density is high) in areas which are close to an observation based on some distance function, while in areas far from any observations we are uncertain (the density is low).

A popular way to estimate densities are probabilistic models using mixture distributions. Given sets of mixture components f_k and mixture weights w_k, a mixture model can be written as

$$p(x) \propto \sum_k w_k f_k(x) \ . \tag{5.1}$$

The components can be understood as describing different features of x and can be arbitrary—for instance, relational as in our case: binary tests that have value 1 if some relational query is true for x; otherwise they have value 0. Estimating such mixture models is a type of unsupervised learning or clustering and involves two problems: estimating the feature functions f_k (structure learning) and estimating the feature weights w_k (parameter learning).

While the use of relational features offers a great compactness and generalization it complicates the feature selection (structure learning) problem: relational features are non-parametric functional representations. There are no prior restrictions on their length and complexity and hence we have essentially infinitely many factors to choose from. Consider a given relational feature respectively query. The longer the query is and the more variables it contains, the larger is the number of possible ways to bind the variables and the larger is the set of refined features that we potentially have to consider.

In addition, the space of the instances x is very large in our case. Recall that even very small relational models can have hundreds of ground atoms and it would be impossible to represent all possible states and in turn the exact distribution, which in its fully enumerated form would require roughly one probability entry for every distinct truth assignment to ground atoms. For 100 ground atoms, this would require approximately 2^{100} distinct probability entries, which is clearly intractable. Thus, we need to focus on a compact, factored representation of the density. Unfortunately, we have only a finite set of positive examples, that is experienced states. From this set we have to generalize to other states, but run the risk of over-generalization: it is unclear to which states we do not want to generalize in order not to over-generalize. Thus, in a very large space we essentially face the problem of structure learning from positive examples only (Muggleton, 1997) that is known to be more difficult than the traditional relational learning setting that additionally assumes that negative, that is impossible, examples are given.

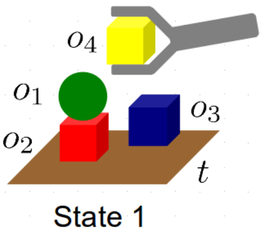
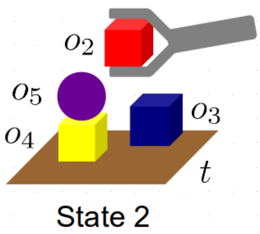
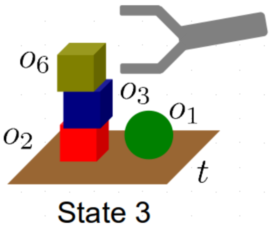
If the feature functions are given, however, the density estimation becomes simpler: only the weights need to be learned. For instance, in 1-class SVMs assumptions about the feature structure are embedded in the kernel function; in the mixture of Gaussians model, the functional form of Gaussians is given a-priori and provides the structure. Triggered by this simple insight, we propose a "patch up" approach in this chapter[1]. We examine different choices of feature functions in the relational setting whose weights can be estimated based on empirical counts. While they are only approximations, we then show that we can "patch up" and improve some of these approximations, namely the context-based features (see below) through learning NID rules. In our relational model-based RL algorithms as well as in our evaluation, we focus on context-based features. In other words, our methods solve both structure and parameter learning from positive examples only.

Let us now introduce different choices of feature functions for relational density estimation. From this, we derive different approaches to model a distribution of known states and actions in a relational RL setting. These methods estimate which relational states are considered known with some useful confidence measure according to a set of experiences $\mathcal{E}$ and a model $\mathcal{M}$ of the transition dynamics.

We formalize the problem of relational density estimation of known states s in the following general model based on queries $q \in Q$ (the density estimation of known actions

[1] In general, the problem of relational density estimation in an online fashion is an important open and intriguing problem with many applications. It has only briefly been touched in previous work (Cussens, 1998).

Table 5.1: Three relational states in a robot manipulation domain

		
$s_1 = \{\, on(o_1, o_2), on(o_2, t), on(o_3, t), inhand(o_4), ball(o_1), cube(o_2), cube(o_3), cube(o_4), table(t)\, \}$	$s_2 = \{\, on(o_3, t), on(o_4, t), on(o_5, o_4), inhand(o_2), cube(o_2), cube(o_3), cube(o_4), ball(o_5), table(t)\, \}$	$s_3 = \{\, on(o_1, t), on(o_2, t), on(o_3, o_2), on(o_6, o_3), ball(o_1), cube(o_2), cube(o_3), cube(o_6), table(t)\, \}$

is analogous):

$$P_Q(s) \;\propto\; \textstyle\sum_{q \in Q} c_{\mathcal{E}}(q)\; I(\exists \sigma : s \models \sigma(q)) \tag{5.2}$$

$$\text{with} \quad c_{\mathcal{E}}(q) \;=\; \textstyle\sum_{(s_e, a_e, s'_e) \in \mathcal{E}}\; I(\exists \sigma : s_e \models \sigma(q)) \;. \tag{5.3}$$

$I(\cdot)$ is the indicator function which is 1 if the argument evaluates to true and 0 otherwise. What is the set of queries Q? It is a set of relational formulas, usually in the form of a conjunction of ground or abstract atoms. We discuss different choices of Q in detail below. The substitution σ examines whether the query q is fulfilled in s, potentially grounding q. A state s has a high probability $P_Q(s)$ to be known if it satisfies queries $q \in Q$ with large counts $c_{\mathcal{E}}(q)$. The counts $c_{\mathcal{E}}(q)$ are based on the previously experienced states in the agent's set of observed state transitions $\mathcal{E} = \{(s_t, a_t, s_{t+1})\}_{t=1}^{T-1}$ and denote the number of times query q held in predecessor states of the experiences. This density implies that all states with low $P(s)$ are considered novel and should be explored, as in E^3. In the following, we discuss different choices for queries q and the accompanying density estimates which emphasize different aspects of relational data. As our running example we use the three states shown in Table 2.1 and for convenience shown again in Table 5.1. We assume that our experiences consist of exactly State 1, that is $\mathcal{E} = \{s_1\}$, while State 2 and State 3 have not been experienced.

Enumerated: Let us first consider briefly the propositional enumerated setting from a density estimation point of view. We have a finite enumerated state space S and action space A. The set of queries,

$$Q_{enum} = \{s \,|\, \exists (s_e, a_e, s'_e) \in \mathcal{E} : s_e = s\} \,,$$

corresponds to predecessor states $s \in S$ which have been visited in $\mathcal{E}$. Thus, queries are long conjunctions of ground atoms. This translates directly to the density

$$P_{enum}(s) \propto c_{\mathcal{E}}(s) \;, \quad \text{with } c_{\mathcal{E}}(s) = \sum_{(s_e, a_e, s'_e) \in \mathcal{E}} I(s_e = s) \;. \tag{5.4}$$

The count $c_{\mathcal{E}}(q)$ in Eq. (5.4) counts the number of occasions state s has been visited in $\mathcal{E}$ (in the spirit of Thrun (1992)). There is no generalization in this notion of known states.

Similar arguments can be applied on the level of state-action counts and the joint density $P(s, a)$. For instance, in Table 2.1 both states s_2 and s_3 are equally unknown and novel as they are both not the experienced state s_1.

Literal-based: Given a relational structure with the set of logical predicates $\mathcal{P}$, an alternative approach to describe what are known states is based on counting how often a literal (a potentially negated atom) has been observed true or false in the experiences $\mathcal{E}$ (all statements equally apply to functions $\mathcal{F}$, but we neglect this case here). A literal l for predicate $P \in \mathcal{P}$ containing variables is abstract in that it represents the set of all corresponding ground, that is variable-free, literals for P. Ground literals then play the role of the traditional factors used in mixture models. First, we consider the set of ground literals $\mathcal{L}^G$ with arguments taken from the domain objects $\mathcal{O}$. This leads to the set of queries $Q_{lit} = \mathcal{L}^G$ and in turn to the density

$$\begin{aligned} P_{lit}(s) \;\propto\; & \textstyle\sum_{l \in \mathcal{L}^G} \; c_{\mathcal{E}}(l)\; I(s \models l) \\ \text{with} \quad & c_{\mathcal{E}}(l) = \textstyle\sum_{(s_e, a_e, s'_e) \in \mathcal{E}} I(s_e \models l). \end{aligned} \tag{5.5}$$

Each query $l \in \mathcal{L}^G$ examines whether l has the same truth values in s as in experienced states. This implies that a state is considered familiar (with non-zero $P_{lit}(s)$) if a ground literal that is true (false) in this state has been observed true (false) before. Thus abstraction over states can be achieved by means of ground literals. We can follow the same approach for abstract literals $\mathcal{L}^A$ and set $Q_{lit} = \mathcal{L}^A$. For $l \in \mathcal{L}^A$ and a state s, we examine whether there are groundings of the logical variables in l such that s covers l. More formally, we replace $s \models l$ by $\exists \sigma : s \models \sigma(l)$ in the indicator function. For instance, we may count how often the blue ball was on top of *some* other object. If this was rarely the case this implies a notion of novelty which guides exploration. For example, in Table 5.1 states s_1 and s_3 share the ground literal $on(o_2, t)$ while this literal does not hold in s_2. Thus, if $Q_{lit} = \{on(o_2, t)\}$ and s_1 is the sole experienced state, then s_3 is perceived as better known than s_2. In contrast, if we use the abstract query $inhand(X)$ (expressing there was *something* held inhand) and set $Q_{lit} = \{inhand(X)\}$, then s_3 is perceived as more novel, since in both s_1 and s_2 some object was held inhand. Note that this second query abstracts from the identities of the inhand held objects.

Context-based / query-based: Assume that we are given a finite set Φ of contexts, which are queries consisting of formulas over predicates and functions. While many relational knowledge representations have some notion of context or rule precondition, in our running example of NID rules these may correspond to the set of NID rule contexts $\{\phi_r\}$. These are learned from the experiences $\mathcal{E}$ and have specifically been optimized to be a compact context representation (Sec. 3.1.3). Given a set Φ of such queries, setting $Q_\Phi = \Phi$ results in the density

$$\begin{aligned} P_\Phi(s) \;\propto\; & \textstyle\sum_{\phi \in \Phi} c_{\mathcal{E}}(\phi)\; I(\exists \sigma : s \models \sigma(\phi)) \\ \text{with} \quad & c_{\mathcal{E}}(\phi) \;=\; \textstyle\sum_{(s_e, a_e, s'_e) \in \mathcal{E}} \; I(\exists \sigma : s_e \models \sigma(\phi)). \end{aligned} \tag{5.6}$$

$c_{\mathcal{E}}(\phi)$ counts in how many experiences $\mathcal{E}$ the context respectively query ϕ was covered with arbitrary groundings. Intuitively, contexts/queries may be understood as describing situation classes based on whether the same abstract prediction models can be applied. Taking this approach, states are considered novel if they are not covered by any existing context/query ($P_\Phi(s) = 0$) or covered by a context/query that has rarely occurred in $\mathcal{E}$ ($P_\Phi(s)$ is low). That is, the description of novelty which drives exploration is lifted to the level of abstraction of these relational contexts. Similarly, we formulate a density estimation over states and actions based on state-action contexts/queries. For instance, in the case of a set Γ of NID rules, each rule defines a state-action context, resulting in the density

$$P_\Phi(s,a) \;\propto\; \textstyle\sum_{r\in\Gamma} c_{\mathcal{E}}(r)\; I(r = r_{(s,a)}), \quad \text{with} \quad c_{\mathcal{E}}(r) \;=\; |\mathcal{E}(r)|, \tag{5.7}$$

which is based on counting how many experiences are covered by the unique covering rule $r_{(s,a)}$ for a in s. $\mathcal{E}(r)$ denotes the experiences which are covered by r. Thus, the number of experiences covered by the rule $r_{(s,a)}$ modeling (s,a) can be understood as a measure of confidence in $r_{(s,a)}$ and thus determines $P_\Phi(s,a)$. We will use $P_\Phi(s,a)$ in our proposed algorithm below. In the example of Table 5.1, assume in s_1 we put the inhand cube o_4 successfully on o_3. From this experience, we learn a rule for the action $puton(X)$ with the context $\phi = inhand(Y) \wedge clear(X)$. This context is covered in s_2, while in s_3 nothing is held inhand and thus the context is not covered there. Therefore, s_3 is perceived as more novel than s_2, as the effects of less actions can be predicted.

Kernel-based: Different methods to estimate the similarity of relational states exist. For instance, Driessens et al. (2006) and Halbritter and Geibel (2007) present relational reinforcement learning approaches which use relational graph kernels to estimate the similarity of relational states. These can be used for relational density estimation (in the sense of 1-class SVMs) which, when applied in our context, would readily imply alternative notions of novelty and thereby exploration strategies. Applying such a method to model $P(s)$ from $\mathcal{E}$ implies that states are considered novel (with low $P(s)$) if they have a low kernel value (high "distance") to previous explored states. Let $k(\cdot,\cdot) \in [0,1]$ denote an appropriate kernel for the queries $q \in Q$, for instance based on relational graph kernels. We replace the hard indicator function $I(\exists\sigma : s \models \sigma(q))$ in Eq. (5.2) by the kernel function, resulting in the more general kernel-based density

$$P_{k,Q}(s) \;\propto\; \textstyle\sum_{q\in Q} c_{\mathcal{E}}(q)\; k(s,q)\;. \tag{5.8}$$

If one sets $Q = \{s \,|\, s \in \mathcal{S}\}$ to the set of states, the density in Eq. (5.8) measures the distance to all observed predecessor states multiplied by experience counts. In the example of Table 5.1, if we use relational graph kernels, the isomorphism of the graph representations of s_1 and s_2 leads to a large kernel estimate $k(s_1, s_2)$, while the different graph structure of s_3 causes $k(s_1, s_3)$ to be small, therefore $P_{k,Q}(s_2) > P_{k,Q}(s_3)$.

Entropy: The above approaches are based on counts over the set of experiences $\mathcal{E}$. As a simple extension, we propose to consider the variability within $\mathcal{E}$, that is, the entropy of the experiences. Consider two different series of experiences $\mathcal{E}_1$ and $\mathcal{E}_2$ both of size n. Imagine that $\mathcal{E}_1$ consists of n times the same experience, while the experiences in $\mathcal{E}_2$ differ. For instance, $\mathcal{E}_1$ might list repeatedly grabbing the same object in the same context, while $\mathcal{E}_2$ might list grabbing different objects in varying conditions. $\mathcal{E}_2$ has a higher entropy. While the counts $c_{\mathcal{E}_1}(q)$ and $c_{\mathcal{E}_2}(q)$ as defined in Eq. (5.3) are the same, still one might be tempted to say that $\mathcal{E}_2$ confirms q better as the query q succeeded in more heterogeneous situations, supporting the claim of generalization. We formalize this by redefining the counts in Eq. (5.3) using a distance estimate $d(s, s') \geq 0$ for two states s and s' as

$$\begin{aligned} c^E_{\mathcal{E}}(q) \;&=\; \sum_{(s_t,a_t,s_{t+1})\in\mathcal{E}} \delta(s_t, \mathcal{E}^{(q,t)})\; I(\exists\sigma : s_t \models \sigma(q)) \;, \qquad (5.9)\\ &\text{with} \quad \mathcal{E}^{(q,t)} = \{(s_d, a_d, s_{d+1}) \in \mathcal{E} \mid \exists\sigma : s_d \models \sigma(q) \wedge d < t\}\\ &\text{and} \quad \delta(s, \mathcal{E}) \propto \min_{(s_e,a_e,s'_e)\in\mathcal{E}} d(s, s_e) \;. \end{aligned}$$

The count $c^E_{\mathcal{E}}(q)$ weights each experience based on its difference to prior experiences $\delta(s_t, \mathcal{E}^{(q,t)})$ (while the original $c_{\mathcal{E}}(q)$ assigns all experiences the same weight irrespective of other experiences). To compute state distances, a kernel as above might be employed, $d(s, s') \propto 1/k(s, s')$. In our evaluation, we investigate the usage of an entropy-based density estimator using a simple distance estimate based on least general unifiers (Ramon, 2002). For illustration, consider again the states in Table 5.1. Assume two series of experiences $\mathcal{E}_1 = \{s_1, s_2\}$ and $\mathcal{E}_2 = \{s_1, s_3\}$ and the query $q = on(X, Y) \wedge ball(X)$ (that there is a ball on-top of some other object). All states s_1, s_2 and s_3 cover this query, therefore the standard counts for $\mathcal{E}_1$ and $\mathcal{E}_2$ according to Eq. (5.3) are the same, $c_{\mathcal{E}_1}(q) = c_{\mathcal{E}_2}(q)$. As s_1 and s_2 share the same structure, however, the entropy of $\mathcal{E}_1$ is smaller than that of $\mathcal{E}_2$. Thus, $\mathcal{E}_2$ provides more heterogeneous evidence for q and therefore $c^E_{\mathcal{E}_1}(q) < c^E_{\mathcal{E}_2}(q)$.

5.4 Relational Exploration Framework

The density estimation approaches discussed above open a large variety of possibilities for concrete exploration strategies. In the following, we derive a relational model-based reinforcement learning framework in which these strategies can be applied. This framework is presented in Algorithm 4. It lifts E^3 to relational exploration and distinguishes explicitly between exploration and exploitation phases.

At each time-step, relational E^3 performs the following general steps:

1. It adapts the relational transition model $\mathcal{M}$ with the set of experiences $\mathcal{E}$.

2. Based on $\mathcal{E}$ and $\mathcal{M}$, it estimates the densities of known states and actions, $P(s)$ and $P(s, a)$. For instance, $\mathcal{M}$ might be used to provide formulas and contexts for specific relational density estimates.

3. These densities are used for decision-making for action a_t with the same phase-ordering as in E^3. If the current state s_t is known, exploitation is tried. If this fails (the planner does not find a policy with a sufficiently high value), planned exploration to unknown states is undertaken. If the current state s_t is not known, direct exploration is performed.

Algorithm 4 Relational Exploration

```
Input: Start state $s_0$, reward function $R$, confidence threshold $\xi$
Set of experiences $\mathcal{E} = \emptyset$
for t=0,1,2 ... do
    Update transition model $\mathcal{M}$ according to $\mathcal{E}$
    Estimate $P(s)$ and $P(s,a)$ from $\mathcal{E}$ and $\mathcal{M}$    ▷ Relational representation enables generalization
    if $P(s_t) > \xi$ then    ▷ If state is known ... → uses relational generalization
        Plan in space of known states    ▷ Try to exploit → uses relational generalization
        if resulting plan has value above some threshold then    ▷ Exploitation successful
            $a_t$ = first action of plan
        else    ▷ Planned exploration → uses relational generalization
            Plan in modified state space where unknown states get maximum reward
            $a_t$ = first action of plan
        end if
    else    ▷ State is unknown
        $a_t$ = action with smallest $P(s,a)$    ▷ Direct exploration → uses relational generalization
    end if
    Execute $a_t$
    Observe new state $s_{t+1}$
    Update set of experiences $\mathcal{E} \leftarrow \mathcal{E} \cup \{(s_t, a_t, s_{t+1})\}$
end for
```

4. Finally, the action a_t is executed, the resulting state s_{t+1} observed and added to the experiences $\mathcal{E}$, and the process repeated.

The relational density estimation of states and actions, $P(s)$ and $P(s,a)$, and the resulting generalization over states, actions and objects play a crucial role at several places in the algorithm, namely

- to decide whether the current state is considered known or novel;
- to determine the set of known states where to try to exploit;
- to determine the set of both known and novel states in planned exploration to decide for target states and plan-through states;
- and to determine which action to execute in direct exploration where the least known action with the lowest $P(s,a)$ is chosen; this combines E^3 (choosing the action with the fewest "visits") with relational generalization (defining "visits" by means of state abstraction).

An important parameter in relational E^3 is ξ, the threshold to decide whether states and actions are known or unknown. The original E^3 algorithm for enumerated state and action spaces provides a theoretical derivation for this threshold, which is required to prove the polynomial sample complexity of E^3. Unfortunately, this derivation is closely tied to the enumerated representation of states and actions, not straightforward to apply in practice and will lead to overly large thresholds (see Guestrin et al. (2002)). As the intriguing work of Walsh (2010) shows, it is nonetheless possible to provide bounds in this spirit for specific relational MDP learning problems. Our focus, however, is on general relational exploration strategies without making assumptions about the transition models $\mathcal{M}$ and the algorithms to learn them. Therefore, we set ξ empirically in our experiments.

5.5 A Complete Relational Model-Based Reinforcement Learner

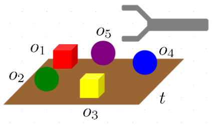
The relational E^3 framework presented in the last section is independent of the concrete choices for the representation of the transition model $\mathcal{M}$, the planning algorithm and the relational density estimation type. Here, we propose a concrete instance of this relational E^3 framework which we will use in our evaluation. We call our algorithm REX (short for relational explorer). To our knowledge, REX is the first empirically evaluated relational model-based reinforcement learning algorithm with explicit explore-exploit mechanism which learns and exploits full-fledged transition models $\mathcal{M}$. REX employs NID rules as model $\mathcal{M}$, plans with the PRADA algorithm and uses the contexts $\{\phi_r\}$ of learned NID rules r to estimate the densities $P_\Phi(s)$ and $P_\Phi(s, a)$. These contexts are learned from experience and provide compact descriptions of situation classes. Thus, a state is known if all actions in this state are known. An action is known if there is sufficient confidence in its covering rule. The confidence of a rule depends on the number of experiences in $\mathcal{E}$ it explains, as described above. As our evaluation will show, this density estimation method is simple, but empirically effective. We are certain that more elaborate and efficient exploration strategies can be derived from the principles of the previous sections in the future.

In general, it is unclear how to efficiently build and exclusively use an MDP of known relational states. Therefore, in exploitation and planned exploration we also plan through partially unknown states. However, the planner PRADA, and thus REX, only takes the known actions in a state for planning into account. It achieves this by only considering actions with unique covering rules in a given state. In planned exploration, PRADA returns plans to states whose actions are to a large extent unknown. Thus, REX plans for highly unknown states then.

5.5.1 Illustrative Example

In Table 5.2, we present an example of an agent using relational E^3 based on probabilistic relational rules, that is REX, in a robot manipulation domain. In the beginning, the robot is given a relational vocabulary to describe the world on a symbolic level. It has the ability to convert its perceptions into the corresponding symbolic representation. Furthermore, it possesses two different types of motor primitives for grabbing objects and putting them on other objects. It can trigger these motor primitives by the symbolic actions $grab(X)$ and $puton(X)$. These actions are always executed and their effects depend on the context. Thus, in the example scenario with objects $\mathcal{O} = \{o_1, o_2, o_3, o_4, o_5, t\}$ the action space of the robot consists of the actions $A = \{\ puton(t), puton(o_1), puton(o_2),$ $puton(o_3)$, $puton(o_4)$, $puton(o_5)$, $grab(o_1)$, $grab(o_2)$, $grab(o_3)$, $grab(o_4)$, $grab(o_5)\ \}$. See Sec. 4.6 for a corresponding real-world robotic setup.

Table 5.2: *Example of Relational E^3.* A robot manipulates objects scattered on a table by means of motor primitives triggered by symbolic actions $puton(\cdot)$ and $grab(\cdot)$. The robot is told to build a tower. It starts with zero knowledge ($\mathcal{E} = \emptyset$) and learns from experience how its actions change the state of the world. The arrows at $t = 2$ and $t = 9$ indicate that the manipulated objects have fallen off balls.

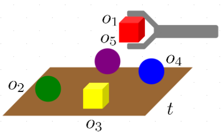


$\boldsymbol{t = 0}$

No rule learned yet.

Direct exploration

Next action: $a_0 = grab(o_1)$



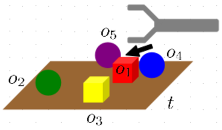
$\boldsymbol{t = 1}$

Learned rule r_1

$$grab(X): \; on(X,Y) \rightarrow \begin{cases} 1.0: & inhand(X),\ \neg on(X,Y) \\ 0.0: & \text{noise} \end{cases}$$

Direct exploration

Next action: $a_1 = puton(o_4)$



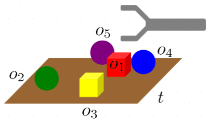
$\boldsymbol{t = 2}$

Learned rule r_2

$$puton(X): \; inhand(Y),\ table(Z) \rightarrow \begin{cases} 1.0: & on(Y,Z),\ \neg inhand(Y) \\ 0.0: & \text{noise} \end{cases}$$

Direct exploration

Next action: $a_2 = puton(o_3)$



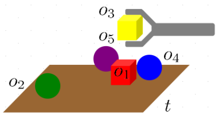
$\boldsymbol{t = 3}$

Learned rule r_3

$$puton(X): \; inhandNil() \rightarrow \begin{cases} 1.0: & - \\ 0.0: & \text{noise} \end{cases}$$

Direct exploration

Next action: $a_3 = grab(o_3)$



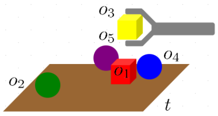
$\boldsymbol{t = 4}$

Confirmed rule r_1

$$grab(X): \; on(X,Y) \rightarrow \begin{cases} 1.0: & inhand(X),\ \neg on(X,Y) \\ 0.0: & \text{noise} \end{cases}$$

Direct exploration

Next action: $a_4 = grab(o_3)$



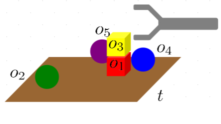
$\boldsymbol{t = 5}$

Learned rule r_4

$$grab(X): \; inhand(X) \rightarrow \begin{cases} 1.0: & - \\ 0.0: & \text{noise} \end{cases}$$

Direct exploration

Next action: $a_5 = puton(o_1)$



$\boldsymbol{t = 6}$

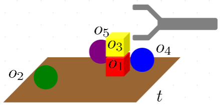
Learned rules r_5 and r_6

$$puton(X): \; \neg ball(X),\ inhand(Y) \rightarrow \begin{cases} 1.0: & on(Y,X),\ \neg inhand(Y) \\ 0.0: & \text{noise} \end{cases}$$

$$puton(X): ball(X),\ inhand(Y),\ table(Z) \rightarrow \begin{cases} 1.0: & on(Y,Z),\ \neg inhand(Y) \\ 0.0: & \text{noise} \end{cases}$$

Direct exploration

Next action: $a_6 = puton(o_2)$



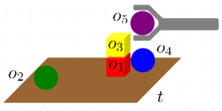
$\boldsymbol{t = 7}$

Confirmed rule r_3

$$puton(X): \; inhandNil() \rightarrow \begin{cases} 1.0: & - \\ 0.0: & \text{noise} \end{cases}$$

Planned exploration

Next action: $a_7 = grab(o_5)$



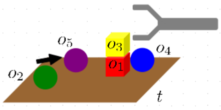
$\boldsymbol{t = 8}$

Confirmed rule r_1

$$grab(X): \; on(X,Y) \rightarrow \begin{cases} 1.0: & inhand(X),\ \neg on(X,Y) \\ 0.0: & \text{noise} \end{cases}$$

Planned exploration

Next action: $a_8 = puton(o_2)$



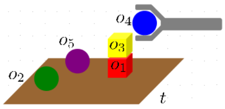
$\boldsymbol{t = 9}$

Confirmed rule r_6

$$puton(X): ball(X),\ inhand(Y),\ table(Z) \rightarrow \begin{cases} 1.0: & on(Y,Z),\ \neg inhand(Y) \\ 0.0: & \text{noise} \end{cases}$$

Planned exploration

Next action: $a_9 = grab(o_4)$



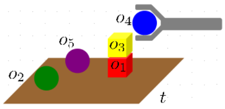
$\boldsymbol{t = 10}$

Confirmed rule r_1

$$grab(X): \; on(X,Y) \rightarrow \begin{cases} 1.0: & inhand(X),\ \neg on(X,Y) \\ 0.0: & \text{noise} \end{cases}$$

Planned exploration

Next action: $a_{10} = grab(o_4)$



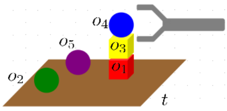
$\boldsymbol{t = 11}$

Confirmed rule r_4

$$grab(X): \; inhand(X) \rightarrow \begin{cases} 1.0: & - \\ 0.0: & \text{noise} \end{cases}$$

Planned exploration

Next action: $a_{10} = puton(o_3)$



$\boldsymbol{t = 12}$

Confirmed rule r_5

$$puton(X): \; \neg ball(X),\ inhand(Y) \rightarrow \begin{cases} 1.0: & on(Y,X),\ \neg inhand(Y) \\ 0.0: & \text{noise} \end{cases}$$

Done.

The robot is told to build a tower with the objects on the table. The robot does not know, however, how its actions change the state of the world. It has to learn this from experience and use its insights to achieve its goal, a prototypical model-based reinforcement learning situation. The robot will apply relational exploration to learn as much as possible about the transition dynamics of its world in as few time as possible. It uses NID rules to learn and represent a transition model $\mathcal{M}$. Based on the contexts of these NID rules $Q = \{\phi_r\}_r$ and its experiences $\mathcal{E}$, it estimates the density of known states and actions. A state is considered known if all actions in this state are known. An action is considered to be known if there is sufficient confidence in its unique covering rule. Here, we assume the robot is confident about a rule if this rule explains at least two of its experiences.

At $\boldsymbol{t = 0}$, the robot starts with zero knowledge about the transition dynamics. As s_0 is unknown, it performs a direct exploration action. All actions are equally unknown so it chooses randomly to grab cube o_1 and learns rule r_1 from the experience (s_0, a_0, s_1). At $\boldsymbol{t = 1}$, although all actions in s_1 and thus s_1 itself are unknown, the robot is less uncertain about the $grab(\cdot)$ actions for the objects lying on the table: these are covered by the rule r_1 which explains its single experience. It is most uncertain about grabbing the inhand object (which is not covered by r_1) and all $puton(\cdot)$ actions which are not covered by any rule. The robot chooses randomly among these most unknown actions for $puton(o_4)$. As o_4 is a ball, the inhand-held cube o_1 falls from o_4 on the table t, resulting in state s_2. The robot generalizes this experience in form of the rule r_2 that $puton(X)$ will lead to putting the inhand-object on the table. This false generalization is uncertain, however, as it is only covered by one experience.

At $\boldsymbol{t = 2}$, the robot is most uncertain about the $puton(\cdot)$ actions which are not covered by any of its learned rules (the previously learned rule r_2 requires $inhand(Y)$ in its context which is not fulfilled in s_2). Therefore, the robot chooses randomly among them and performs $puton(o_3)$. It observes no effects and learns the rule r_3, predicting $puton$ actions in contexts where nothing is held inhand. At $\boldsymbol{t = 3}$, the robot is equally uncertain about all actions (all actions are covered by rules which explain exactly one experience). It chooses randomly $grab(o_3)$. The experience (s_3, a_3, s_4) confirms the rule r_1 about which it is certain by now as r_1 explains two experiences. At $\boldsymbol{t = 4}$, $grab(o_3)$ is the only action which is not covered by any rule and therefore performed. The resulting state s_5 is not different from s_4, resulting in the learned rule r_4.

At $\boldsymbol{t = 5}$, the most uncertain actions, $grab(o_3)$ and all $puton$-actions, are covered by rules with confidence one (that is, explaining one experience). The robot chooses randomly $puton(o_1)$. This provides an insightful experience: by comparing the experiences (s_1, a_1, s_2) and (s_5, a_5, s_6), the robot "understands" that rule r_2 is a false generalization and instead learns r_5 for putting on cubes and the table and r_6 for putting on balls. At $\boldsymbol{t = 6}$, all $grab$-actions are known due to the confidence in rule r_1, in other words the robot can predict their effects with confidence. In contrast, it is uncertain about all $puton$-actions as rule r_3 explains only one experience, so it randomly chooses $puton(o_2)$. The resulting experience confirms r_3 about which it is certain then.

At $\boldsymbol{t = 7}$, the robot can predict the effects of all actions in the current state with certainty. Therefore, it tries to exploit its knowledge about known states to plan for its goal, namely to build a tower. The rule r_5 (modeling $puton$-actions for cubes) required for tower building, however, is still uncertain and thus the according actions and states are unknown and cannot be considered in planning. Therefore, exploitation fails and the robot performs planned exploration instead: it plans for unknown states in which its rules r_4, r_5 and r_6 can be tested. Such states are reached by $grab$-actions and the robot chooses randomly to perform $grab(o_5)$. At $\boldsymbol{t = 8}$, the planned exploration begun at the last time-step allows to confirm several rules by performing one of the unknown $puton$ actions or the likewise unknown $grab(o_5)$. The robot chooses randomly $puton(o_2)$. The resulting state s_9 confirms r_6 about which it is certain by now. At $\boldsymbol{t = 9}$, like in s_7 the robot performs planned exploration and grabs a random object, namely $grab(o_4)$.

At $\boldsymbol{t = 10}$, the unknown actions are $grab(o_4)$, $puton(o_1)$, $puton(o_3)$ and $puton(t)$ whose covering rules explain only one experience. The robot chooses randomly $grab(o_4)$ whose outcome in s_{11} confirms the rule r_4. At $\boldsymbol{t = 11}$, from the remaining three unknown actions, it chooses randomly $puton(o_3)$, confirming the rule r_5. At $\boldsymbol{t = 12}$, the highest possible tower has been built. Hence, the robot cannot exploit its knowledge to build an even higher tower. Similarly, planned exploration for unknown states fails and the robot concludes that it is done.

In this example, there have been no exploitation steps as in the last time-step achieved by direct and planned exploration, the highest reward has already been achieved. If there were more cubes, however, the robot could successfully exploit its knowledge and achieve even higher reward states from s_{12}.

To keep this illustrative example short, we simplified in some respects: First, the robot makes most often the correct generalizations, even if competing false generalizations have the same statistical evidence. For instance, from experience (s_2, a_2, s_3) the robot could also generalize that $puton(X)$ does not have effects if X is a cube (instead of if its hand is empty). A second simplification in our example is our neglect of noise in the actions of the robot. Note however that our algorithms account for stochastic actions and our experimental evaluation is performed in intrinsically noisy domains. The third simplification is that we determined the "random" choices of the robot to be informative actions. For instance, if the robot had chosen $puton(o_1)$ in s_{11}, it might have had to revise its rule r_5 to incorporate $clear(X)$ and to come up with a rule for putting on objects which are not clear, leading to a more accurate model requiring more exploration steps.

5.6 Evaluation

The intention in our evaluation is to compare propositional and relational techniques for exploring relational worlds. More precisely, we investigate the following questions:

- $Q1$: Can relational knowledge improve exploration performance?
- $Q2$: Do the relational explorers scale better with the number of domain objects than the propositional ones?

- $Q3$: Does relational E^3 exploration perform superior to random exploration with relational models?
- $Q4$: Can relational explorers transfer knowledge to new situations and objects?
- $Q5$: Can relational explorers transfer knowledge to new tasks?

To answer these questions, we compare four different methods inspired by E^3 based on propositional or relational transition models:

- **flat** E^3 exploration based on an enumerated state and action space
- **factored** E^3 exploration based on a factored propositional state and action space
- **relational** E^3 exploration based on a relational state and action space
- **relational ϵ-greedy** based on a relational state and action space (trying to exploit with probability $1 - \epsilon$ and randomly exploring otherwise)

Relational ϵ-greedy is not a "simple" baseline, but a rather powerful method learning abstract relational transition models and using them for exploitation (thus, thereby using the same set of known states as relational E^3). In contrast to relational E^3, it performs a random action for exploration. Relational ϵ-greedy often profits from its optimism to almost always try to exploit—in contrast to E^3, it is not forced to completely know a state before it can start exploitation. In our experiments, we set $\epsilon = 0.1$ which performed best in preliminary tests.

We use NID rules to learn transition models $\mathcal{M}$ for *all* representation types for two reasons: *(i)* there is an effective learning algorithm for NID rules, and *(ii)* we can express transition models on all representation levels with NID rules which allows for a consistent and comparable evaluation. Relational E^3 and relational ϵ-greedy learn abstract NID rules as described in Sec. 3.1.3. Factored E^3 learns propositional (i.e., ground) NID rules using a slightly modified learning algorithm; thus, ground literals imitate the propositional state attributes. Flat E^3 uses pseudo propositional NID rules where the rule context describes a complete ground relational state; hence, a rule is only applicable in a specific state. Exemplary rules for all representations are shown in Table 5.3.

We use Rex (Sec. 5.5) as a concrete algorithm for relational E^3 and propositional variants of Rex for the propositional E^3 frameworks. Thus, we learn (propositional or abstract) NID rules after each new observation from scratch using the algorithm of Pasula et al. (2007) (appropriately modified for the propositional representations) and employ PRADA (Sec. 4.2) for exploitation or planned exploration. The reward function is not learned, but provided to the agent. PRADA plans in the ground relational representation and therefore can deal with rules on all abstraction levels. To estimate the densities of known states and actions, Rex and similarly its propositional counterparts use the contexts of rules $Q = \{\phi_r\}_r$ (an extension will be discussed later). We define the densities and the threshold ξ for knowing states and actions such that an action is known in a state if its covering rule explains two experiences in $\mathcal{E}$. While this number is

Table 5.3: *Illustration of NID rules on different representation levels.* (a) The abstract rule uses variables to generalize over objects. (b) The factored propositional rule imitates propositional state attributes by using only ground atoms. (c) The flat rule specifies a complete state in its context. Note that the propositional rules (b) and (c) do not need to represent the typing predicates such as $cube(\cdot)$ as they do not abstract from object identities.

(a) Abstract NID rule

$$
\begin{array}{rl}
grab(X): & on(X,Y),\ ball(X),\ cube(Y),\ table(Z) \\
& \rightarrow \left\{ \begin{array}{lcl} 0.7 & : & inhand(X),\ \neg on(X,Y) \\ 0.2 & : & on(X,Z),\ \neg on(X,Y) \\ 0.1 & : & \text{noise} \end{array} \right.
\end{array}
$$

(b) Factored propositional NID rule

$$
\begin{array}{rl}
grab(d): & on(d,b) \\
& \rightarrow \left\{ \begin{array}{lcl} 0.7 & : & inhand(d),\ \neg on(d,b) \\ 0.2 & : & on(d,t),\ \neg on(d,b) \\ 0.1 & : & \text{noise} \end{array} \right.
\end{array}
$$

(c) Flat "NID rule"

$$
\begin{array}{rl}
grab(d): & on(a,t), on(b,t), on(c,a), on(d,b), \neg on(a,b), \neg on(a,c) \ldots, \neg inhand(a), \ldots \\
& \rightarrow \left\{ \begin{array}{lcl} 0.7 & : & inhand(d),\ \neg on(d,b) \\ 0.2 & : & on(d,t),\ \neg on(d,b) \\ 0.1 & : & \text{noise} \end{array} \right.
\end{array}
$$

not large, it enables the agent to explore the environments of our experiments within a reasonable number of actions (< 100). We have implemented REX and its propositional counterparts, the learning algorithm for NID rules and the planning algorithm PRADA in C++[2].

Our first test domain is our intrinsically noisy robot manipulation domain (Sec. 1.1.1). Our second test domains, called "IPPC" in the following, are domains taken from the international planning competition in 2008 (IPPC, 2008). While these domains are defined in the probabilistic planning domain definition language (PPDDL), the transition dynamics of many domains can be represented by NID rules (see Sec. 4.2 and Appendix B). In our experiments, we show that these representations can be learned using the algorithm of Pasula et al. (2007).

[2] The website `http://userpage.fu-berlin.de/tlang/explore/` provides our code of PRADA, of the learning algorithm of NID rules and of the robot manipulation simulator as well as videos of exemplary exploration rounds in the robot manipulation domain.

We perform three series of experiments to answer our evaluation questions where we pursue the same, similar or different tasks over multiple rounds:

- **Series 1 – Robot Manipulation domain**: comparison of relational vs. non-relational exploration approaches
  - Experiment 1: Unchanging worlds of cubes and balls (Fig. 5.2, p. 131)
  - Experiment 2: Unchanging worlds with boxes (Fig. 5.3, p. 132)
  - Experiment 3: Generalization to new worlds with boxes (Fig. 5.4, p. 133)
- **Series 2 – IPPC domains**: comparison of relational vs. non-relational exploration approaches
  - Experiment 4: Exploding Blocksworld (Fig. 5.5, p. 134)
  - Experiment 5: Triangle-tireworld (Fig. 5.6, p. 136)
  - Experiment 6: Search-and-rescue (Fig. 5.7, p. 137)
- **Series 3 – Robot Manipulation domain**: comparison of different relational exploration approaches
  - Experiment 7: Advanced task in unchanging worlds with boxes (Fig. 5.8, p. 138)
  - Experiment 8: Generalization to new tasks (Fig. 5.9, p. 139)

In all experiments *the robot starts from zero knowledge ($\mathcal{E} = \emptyset$) in the first round* and carries over experiences to the next rounds. In each round, we execute a maximum number of actions (50 in the simple, 100 in the difficult scenarios). If the task is still not solved by then, the round fails.

We report the success rates and the action numbers to which failed runs contribute with the maximum number. Both are direct measures of the goal-directedness and acting performance of an autonomous agent. We emphasize that likewise these measures also evaluate the learning performance of the agent: the goal-directed performance depends on the learned rules and the active exploration on the learned density estimates. Thus, high success rates and low action numbers indicate a good performance in learning rules and densities—"good" in the sense of enabling goal-directed behavior.

5.6.1 Series 1 – Robot Manipulation Domain

In this first series of experiments, we compare the flat, factored and relational E^3 approaches as well as relational ϵ-greedy in successively more complex problems.

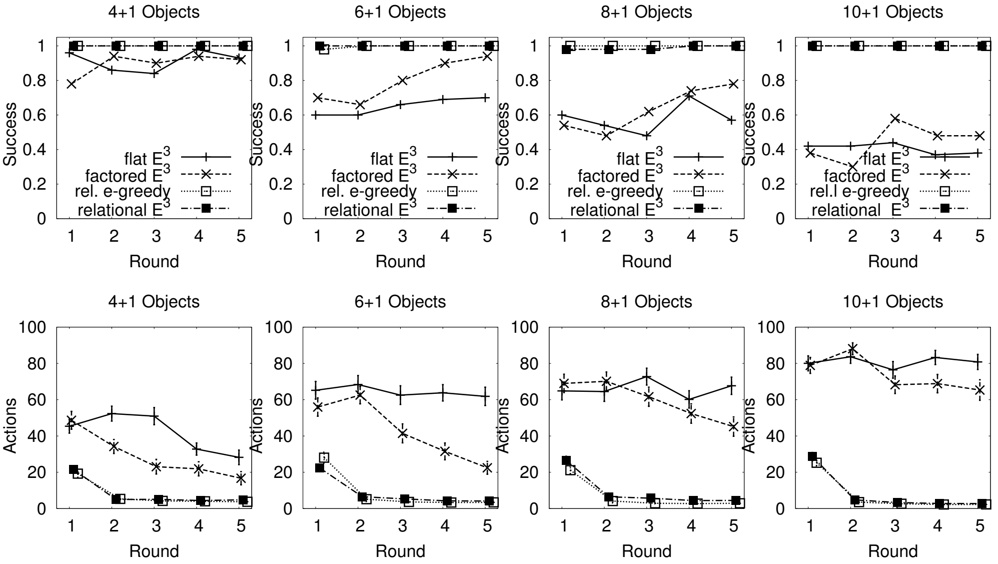


Figure 5.2: *Experiment 1 (Robot Manipulation domain): Unchanging Worlds of Cubes and Balls.* A run consists of 5 subsequent rounds with the same start situations and goal objects. The robot starts with no knowledge in the first round. The success rate and the mean estimators of the action numbers with standard deviations over 50 runs are shown (5 start situations, 10 seeds).

Experiment 1: Unchanging Worlds of Cubes and Balls

The goal in each round is to pile two specific objects, $on(o_1, o_2)$. To collect statistics we investigate worlds of varying object numbers (movable objects and table) and for each object number, we create five worlds with different objects. For each such world, we perform 10 independent runs with different random seeds. Each run consists of 5 rounds with the same goal instance and the same start situation. The results presented in Fig. 5.2 show that already in the first round the relational explorers solve the task with significantly higher success rates and require up to 3-4 times fewer actions than the propositional explorers. In subsequent rounds, the relational methods use previous experiences much better, solving those in almost minimal time. This indicates the good accuracy and generality of their learned rule-sets and densities. In contrast, the action numbers of the propositional explorers fall only slowly. The non-relational methods face problems with an increasing number of objects, while the relational methods perform well with all numbers of objects. There are no significant performance differences between relational E^3 and relational ϵ-greedy in this simple scenario.

Experiment 2: Unchanging Worlds with Boxes

We keep the task and the experimental setup as before, but in addition the worlds contain boxes, resulting in a more complex transition process. In particular, some goal objects are put in boxes in the beginning, necessitating more intense exploration to learn how to

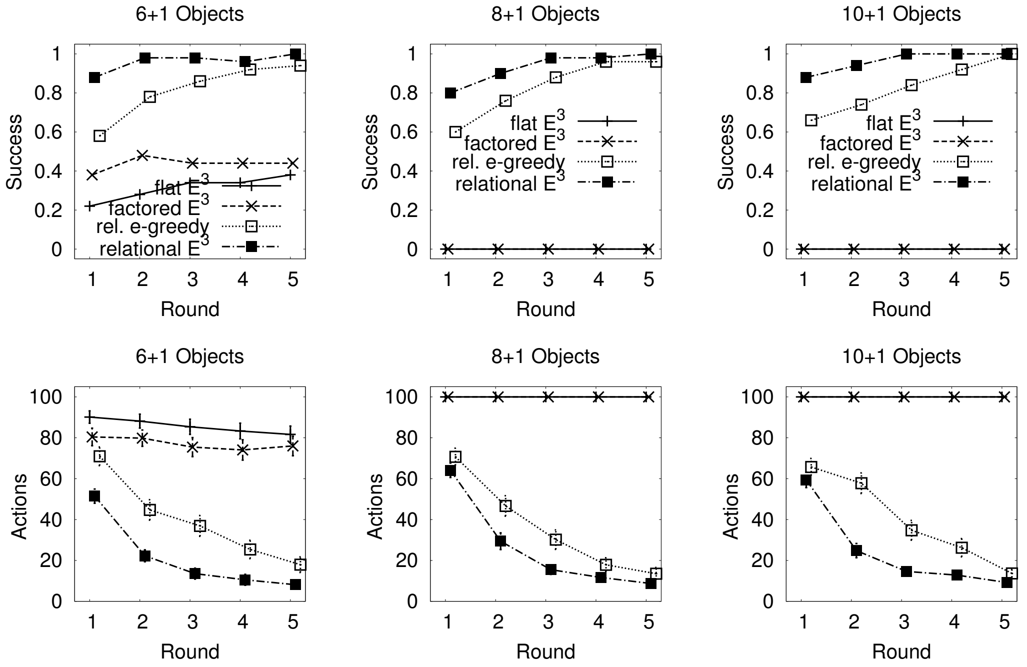


Figure 5.3: *Experiment 2 (Robot Manipulation domain): Unchanging Worlds with Boxes.* A run consists of 5 subsequent rounds with the same start situations and goal objects. The robot starts with no knowledge in the first round. The success rate and the mean estimators of the action numbers with standard deviations over 50 runs are shown (5 start situations, 10 seeds).

deal with boxes. Fig. 5.3 shows that again the relational explorers have superior success rates, require significantly fewer actions and reuse their learned knowledge effectively in subsequent rounds. The propositional explorers are overburdened with larger numbers of objects—their learned densities of known states and actions fail to recognize similar situations. In contrast, the relational explorers scale well with increasing numbers of objects. Relational E^3 is superior to relational ϵ-greedy. This shows that its learned state and action densities enable a directed active exploration.

Experiment 3: Generalization to New Worlds

In this series of experiments, the objects, their total numbers and the specific goal instances are different in each round (worlds of 7, 9 and 11 objects). We create 10 problem sequences (each with 10 rounds) and perform 10 runs for each sequence with different random seeds. As Fig. 5.4 shows the performance of the relational explorers is good from the beginning. Its learned densities and rules enable a stable performance of E^3 at a near-optimal level after already 2 rounds, while relational ϵ-greedy requires 4 rounds. This experiment shows that the relational explorers can transfer their learned knowledge to new situations and objects. In contrast, the propositional explorers cannot transfer their knowledge to different worlds due to the limits of their learned rules and densities and

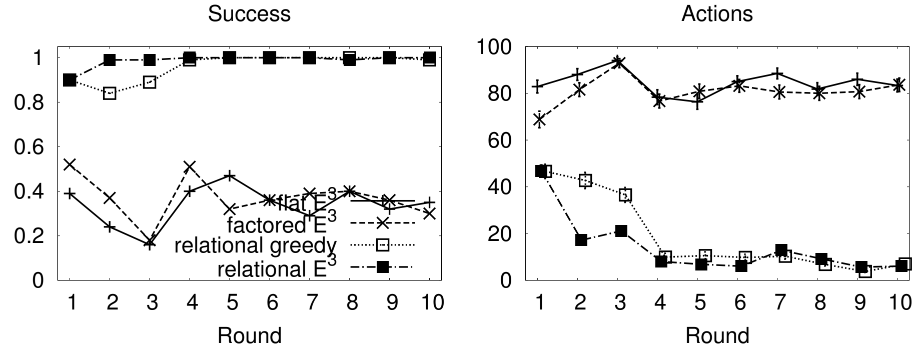


Figure 5.4: *Experiment 3 (Robot Manipulation domain): Generalization to New Worlds.* A run consists of a problem sequence of 10 subsequent rounds with different objects, numbers of objects (6 - 10 cubes/balls/boxes + table) and start situations in each round. The robot starts with no knowledge in the first round. The success rate and the mean estimators of the action numbers with standard deviations over 100 runs are shown (10 sequences, 10 seeds).

thus neither their success rates nor their action numbers improve in subsequent rounds.

Summary

All three experiments of this first evaluation series indicate that the usage of relational knowledge improves learning rules and densities of known states and actions and thus exploration (question $Q1$). From Experiments 1 and 2, we conclude that the usage of relational explorers scales better with the number of objects than for propositional explorers (question $Q2$). The slightly more challenging Experiments 2 and 3 show that the principled relational E^3 exploration outperforms random exploration with relational models (question $Q3$): the learned state and action densities enable an informative active exploration. Experiment 3 provides an answer to question $Q4$: relational explorers can transfer their knowledge to new situations and objects—while propositional explorers fail.

5.6.2 Series 2 – IPPC

In the second series of experiments, we compare the non-relational with the relational exploration strategies in domains of the international planning competition. We choose three domains whose transition processes can be represented by NID rules as we have shown before (Sec. 4.2 and Appendix B). We convert these domains manually into sets of NID rules which we use as the true underlying world dynamics. The reinforcement learning agent tries to estimate these rules from its experiences. In each domain, we present the results on selected problem instances. To collect statistics, we perform 50 runs on the same problem instance with different random seeds. Each run consists of 10 subsequent rounds. In these IPPC domains, most actions do not have effects in a given

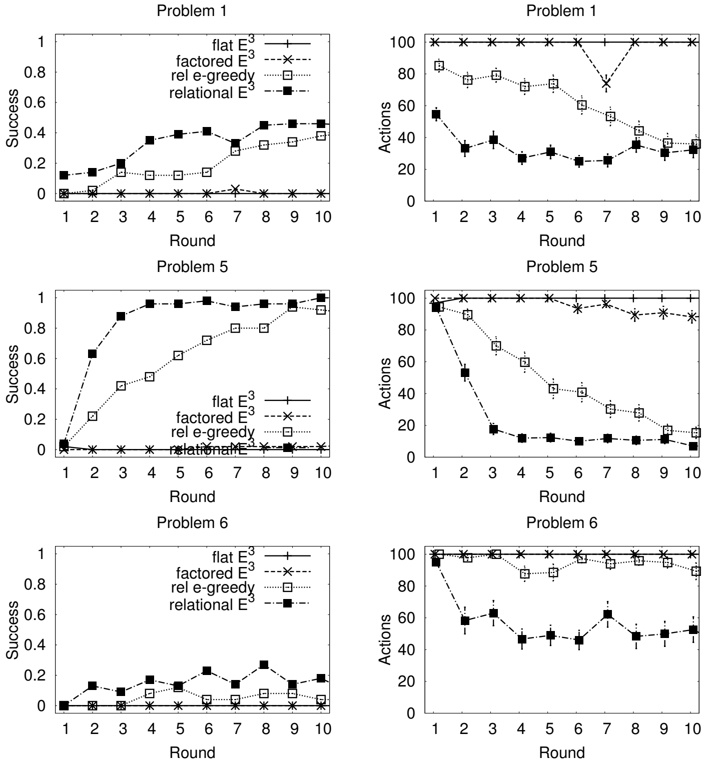


Figure 5.5: *Experiment 4 (IPPC): Exploding Blocksworld.* A run consists of 10 subsequent rounds with the same start situation and goal objects. The agent starts with no knowledge in the first round. The success rate and the mean estimators of the action numbers with standard deviations over 50 runs based on different random seeds are shown.

state. This is in contrast to the intrinsically noisy robot manipulation domain where actions almost always have an effect. For instance, in the latter it is always possible to try to grab an object, be it clear, at the bottom of a pile or in a box. In contrast, the PPDDL action operators of the IPPC domains specify restrictive action preconditions. For this reason, we introduce a further restriction for direct exploration in *all* investigated approaches: all actions which have not shown an effect since the last state change are forbidden until the next state change.

Experiment 4: Exploding Blocksworld

The results displayed in Fig. 5.5 show that the propositional explorers almost always fail completely. Their performance is perished in particular by the fact that most actions do not have effects in a given state. This is hazardous if one cannot generalize one's experiences over objects, resulting in barely useful density estimates of known states and actions: the propositional explorers spend too much time in each state exploring noneffective actions. In contrast, the relational explorers often succeed in solving the tasks, due to the superior generality of their learned densities. Their performance varies

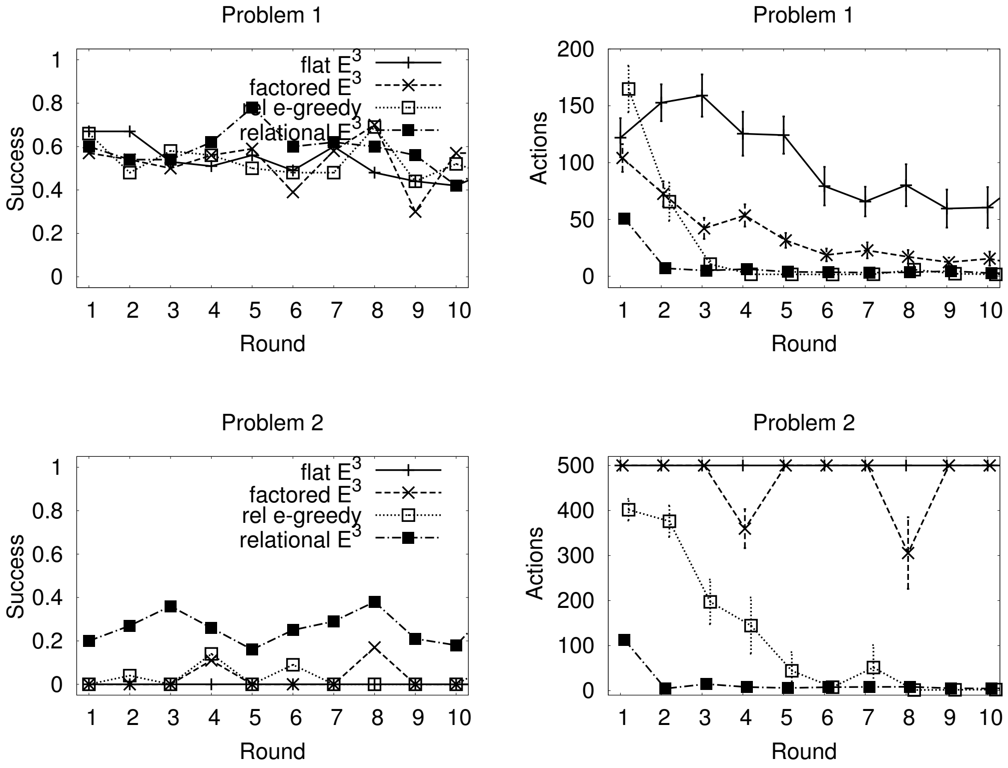


Figure 5.6: *Experiment 5 (IPPC): Triangle Tireworld.* A run consists of 10 subsequent rounds with the same start situation and goal specification. The agent starts with no knowledge in the first round. The success rate and the mean estimators of the action numbers with standard deviations over 50 runs based on different random seeds are shown.

with the problem instance difficulty and increases with the number of rounds in a run. Relational E^3 clearly outperforms relational ϵ-greedy in both the success rate as well as the number of required actions, indicating the usefulness of the learned densities of known states and actions for active exploration.

Experiment 5: Triangle Tireworld

The results presented in Fig. 5.6 show that in the first problem instance the success rate of all methods is comparable. The difficulty of this domain lies in dead-lock situations (where the agent has a flat tire, but no spare tire is available). Even for the relational explorers, the causes for landing in such dead-locks are not easy to recognize with confidence given the limited number of available relevant experiences. However, relational E^3 and after some rounds relational ϵ-greedy require only a fraction of the actions of the propositional explorers. In the second problem instance, the usefulness of expressive learned densities of known states and actions for active exploration is particularly significant: only relational E^3 is able to solve a stable fraction of the runs, even when the maximum number of actions in a round is set to 500.

Experiment 6: Search and Rescue

In this domain, the agent can collect intermediate and final rewards. We present the total rewards with the success rates and the action numbers in Fig. 5.7. Overall, relational E^3 performs clearly best w.r.t. all measures and scales with an increasing numbers of objects (which increase with the problem instances). While flat E^3 performs worst, here factored E^3 most often outperforms relational ϵ-greedy. In this scenario, a principled exploration strategy based on learned densities shows clear benefits over random exploration.

Summary

The results in the IPPC domains confirm our findings of the first series of experiments in the robot manipulation domain. All experiments here show that the usage of relational knowledge for learning expressive transition models and densities of known states and actions improves exploration (question $Q1$). The relational explorers scale better with the complexity of the problems in form of the number of objects (question $Q2$). Furthermore, all experiments, in particular Experiment 6, indicate that the principled E^3 exploration, driven by the learned densities, leads to significant performance gains over random exploration (question $Q3$).

5.6.3 Series 3 – Robot Manipulation Domain

After having extensively investigated the difference in performance of relational and non-relational explorers, in our final series of experiments we examine the relational explorers in more advanced problem settings. Here, we also investigate a variant of Rex using entropy-based counts for relational density estimation (cf. Sec. 5.3). We denote this variant by "relational E^3 entropy" in the following.

Experiment 7: Advanced Task in Unchanging Worlds with Boxes

The goal in each round is to "clear" the table. Movable objects (balls and cubes) of different colors are scattered over the table. To clear up such a movable object, it needs to be put into a box of the same color. To collect statistics, we investigate worlds of varying object numbers and for each object number, we perform 10 independent runs with different random seeds. Each run consists of 5 rounds with the same start situation (and thus with the same goal instance). In starting situations, balls and cubes may form piles, lie on top of closed boxes or be contained in boxes of a different color. Fig. 5.8 presents our results. The E^3 methods clearly outperform relational ϵ-greedy. Furthermore, relational E^3 entropy has some small, but significant advantages over the original relational E^3. This hints at the potential of improving performance by more sophisticated techniques to learn relational densities.

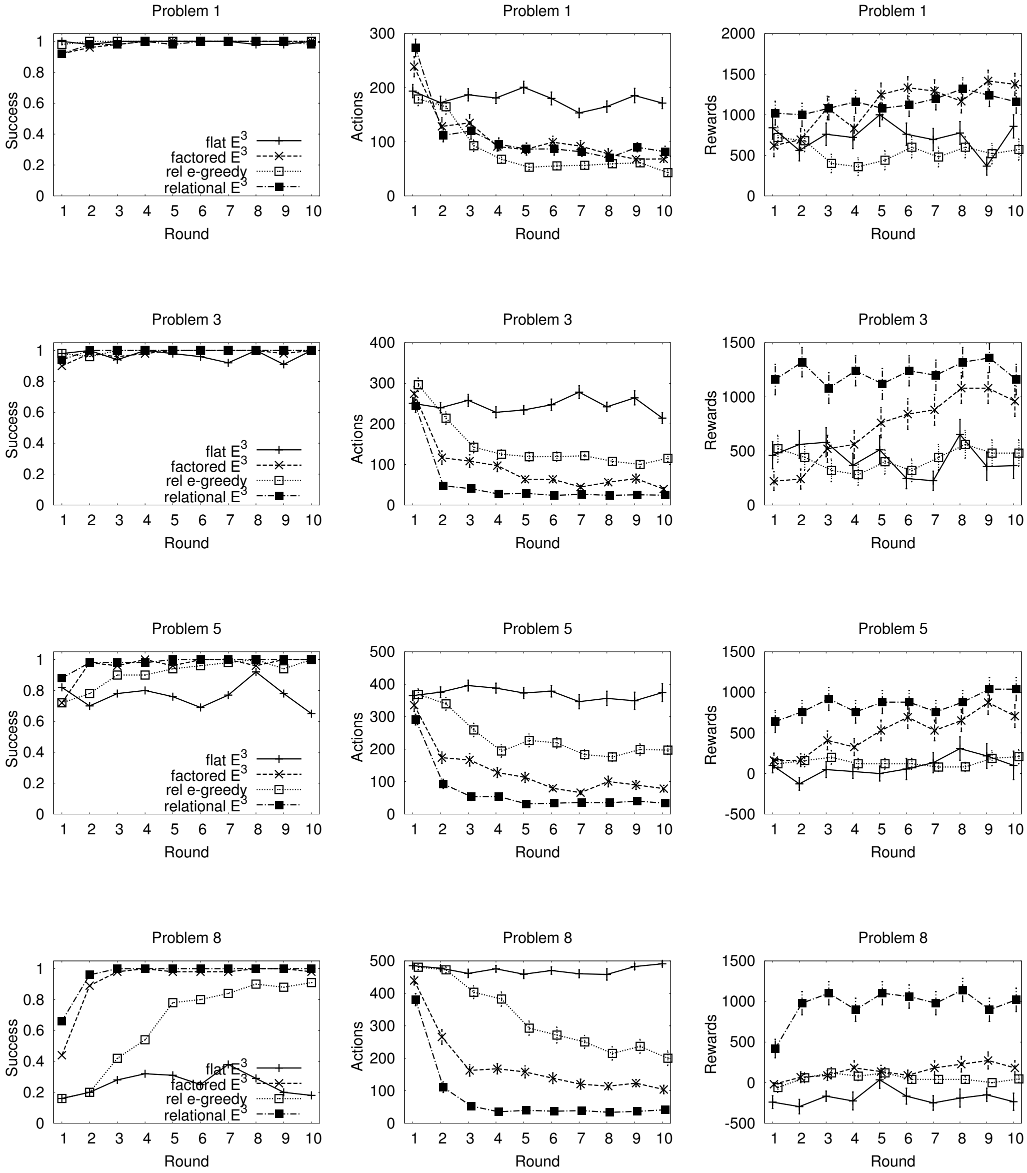


Figure 5.7: *Experiment 6 (IPPC): Search and Rescue.* A run consists of 10 subsequent rounds with the same start situation and goal specification. The agent starts with no knowledge in the first round. The success rate and the mean estimators of the action numbers and collected rewards with standard deviations over 50 runs based on different random seeds are shown. The reward for performing no actions is 0.

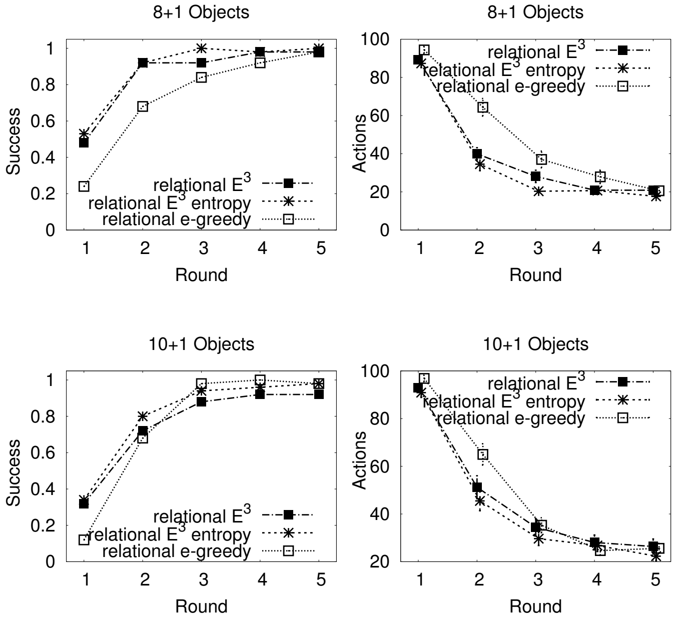
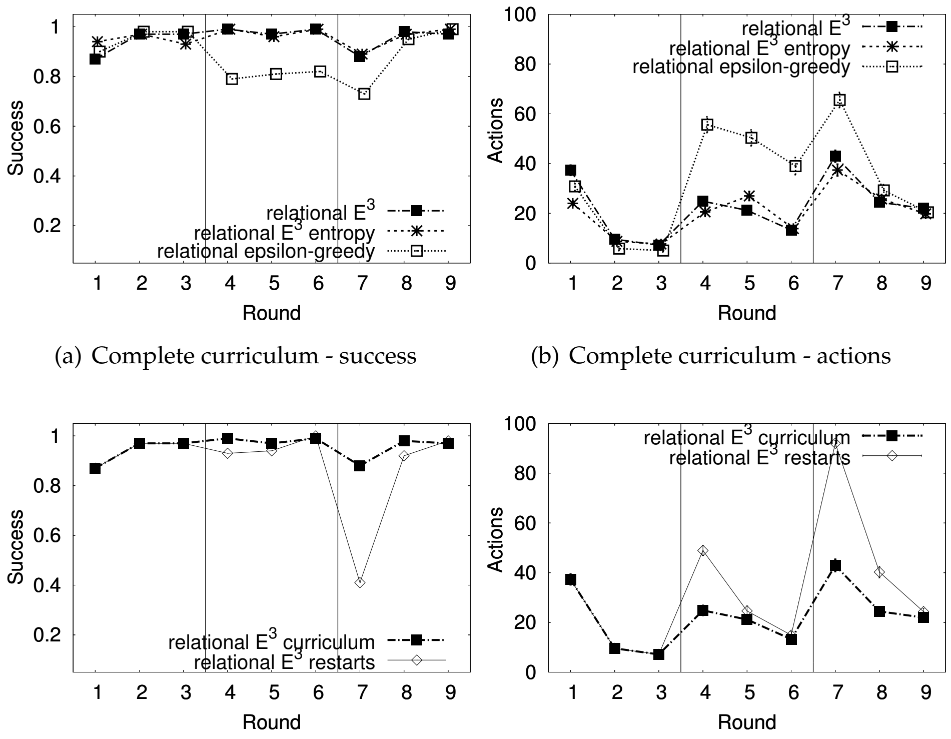


Figure 5.8: *Experiment 7 (Robot Manipulation domain): Advanced Task in Unchanging Worlds with Boxes.* The goal is to put colored balls and cubes into boxes of the same color. A run consists of 5 subsequent rounds with the same start situation (and thus with the same goal). The robot starts with zero knowledge in the first round. The success rate and the mean estimators of action numbers with standard deviations over 50 runs are shown (5 start situations, 10 seeds).

Experiment 8: Generalization to New Tasks

In our final experiment, we perform three tasks of increasing difficulty in succession: piling two specific objects in simple worlds with cubes and balls (as in Exp. 1), in worlds extended by boxes (as in Exp. 2 and 3) and clearing up the desktop by putting all movable objects into boxes of the same color where the required objects may be partially contained in wrong boxes in the beginning (as in Exp. 7). Each task is performed for three rounds in different worlds with different goal objects. The results presented in Fig. 5.9 confirm the previous results of Exp. 3: the relational explorers are able to generalize over different worlds for a fixed task. As seen in Exp. 3, propositional explorers fail to do this as they cannot generalize over objects. Beyond that, the relational explorers are able to transfer the learned knowledge from simple to difficult tasks in the sense of curriculum learning (Bengio et al., 2009). This is shown in Fig. 5.9 (c-d) where the results for relational E^3 are compared to restarting the learning procedure at the beginning of each new task (that is, in rounds 4 and 7) (the corresponding graphs for relational ϵ-greedy and relational E^3 entropy are similar). In the first rounds of each task (that is, in rounds 1, 4, and 7), relational E^3 entropy requires significantly less actions than the original relational E^3 which again underlines that it is promising to investigate other relational



(a) Complete curriculum - success

(b) Complete curriculum - actions

(c) relational E^3- success

(d) relational E^3- actions

Figure 5.9: *Experiment 8 (Robot Manipulation domain): Generalization to New Tasks.* A run consists of a problem sequence of 9 subsequent rounds with different objects, numbers of objects (6 - 10 cubes/balls/boxes + table) and start situations in each round. The tasks are changed between round 3 and 4 and round 6 and 7 to more difficult tasks. The robot starts with no knowledge in the first round. The success rate and the mean estimators of the action numbers with standard deviations over 100 runs are shown (10 sequences, 10 seeds).

density estimation techniques. Both E^3 methods profit from their learned densities for active exploration and clearly outperform relational ϵ-greedy.

Summary

Our results in this final series of experiments indicate that relational E^3 methods using learned relational densities for active exploration can solve advanced tasks and outperform relational explorers with random exploration (question $Q3$). Furthermore, the relational explorers are not only able to transfer learned knowledge to new situations and objects (question $Q4$), but also to new tasks (question $Q5$). Furthermore, our results show that we can improve on relational E^3 already by slightly modifying the relational density estimation. This points at the potential of future work on more sophisticated relational density estimation techniques for active relational exploration.

Putting everything together, the experimental results clearly show that all questions $Q1$-$Q5$ can be answered affirmatively.

5.7 Discussion

Efficient exploration in relational worlds is an interesting problem that is fundamental to many real-life decision-theoretic planning problems, but has only received little attention so far. We have approached this problem by proposing a relational exploration strategy that borrows ideas from efficient techniques for propositional representations. The key step in going from propositional to relational representations is a new definition of the concept of the novelty of states and actions. We have introduced a relational density estimation framework to derive smoothed estimates of the empirical density of relational data. This is a difficult, but promising problem since it requires learning from positive data only. We have presented different relational density estimation techniques and shown how to apply them to drive exploration in relational domains.

We have proposed the algorithm Rex, a practical solution in our relational E^3 framework, which integrates our relational planner PRADA and probabilistic relational rules. Rex is the first relational model-based reinforcement learner with explicit explore-exploit mechanism which learns complete transition models and solves difficult tasks in complex environments with many objects. Our experimental results show a significant improvement over established results for solving difficult, highly stochastic planning tasks in a 3D simulated complex robot manipulation domain and in domains of the international planning competition. Our results demonstrate that relational exploration, driven by relational density estimation, does not only improve the exploration performance, but also enables the transfer of learned knowledge to new situations and objects and even in a curriculum learning setting where different tasks have to be solved one after the other.

5.7.1 Future Work

There are several interesting avenues for future work. One is to investigate incremental learning of transition models. For instance, how can probabilistic relational rules be learned in an incremental fashion? The resulting algorithms might in turn provide new relational exploration strategies. Another avenue is to explore the connection between relational exploration and transfer learning. Also, one should start to explore statistical relational reasoning and learning techniques for the relational density estimation problem implicit in exploring relational worlds. Interesting candidates are relational variants of probability estimation (Neville et al., 2003), regression and cluster trees (Blockeel and de Raedt, 1998; Blockeel et al., 1998) as well as boosted relational dependency networks (Neville and Jensen, 2007; Natarajan et al., 2010) and their extension to the online learning setting.

We believe our conceptual discussion opens the door to a large variety of possible exploration strategies—our specific choices have only served as a first proof of concept.

For instance, it might be more effective to learn two separate densities for the known and for the novel states. It is interesting to investigate whether general formal guarantees on the sampling complexity can be derived from our exploration strategies in the sense of the work of Walsh (2010). As it is hopeless to explore the whole state of a non-trivial world, exploring the relevance of objects or object classes for the task at hand is a further central challenge for future research. Finally, examining our approach in other problem scenarios is appealing, for instance, applying it to large-scale applications such as the web or to geometrical reasoning of robots.

Chapter 6

Conclusions

Autonomous agents need to pursue many changing tasks in the real world. A household robot for instance wants to prepare a meal, repair a broken lamp and train its chess skills. Natural environments like households are referred to as stochastic relational worlds due to two main characteristics: the large number of contained objects which the agent can manipulate; and the uncertainty about the effects of its actions. Goal-directed behavior in stochastic relational worlds involves many challenging aspects. These can be observed in the behavior of the most intelligent "agents" in the world: both humans and animals learn from their experience how they can manipulate the world by means of their actions. They maintain their acquired knowledge in abstracting and generalizing models, instead of merely storing their individual experiences. When given a new task, humans and animals come up with appropriate plans of actions by cognitive processing based on such models, for instance by internal simulation. If they cannot find appropriate actions, they try to enhance their knowledge by actively exploring their environment in an efficient manner to find out about action possibilities they are currently not aware of.

In this thesis, we have introduced computational principles and algorithms for these fundamental aspects of intelligent human and animal behavior. We have proposed algorithms for reasoning and planning in ground relational domains using learned probabilistic relational models of the environment which generalize over objects and situations. Our approaches include methods for forward and for backward reasoning and are based on different techniques such as inference in graphical models and look-ahead trees. We have presented a framework for focusing on relevant objects in planning. We have introduced one of the first approaches to efficient exploration in relational domains: we have shown how to generalize over situations and objects when estimating the interestingness of states and actions for exploration. We have demonstrated the suitability of our relational planning techniques for realistic scenarios by an application on a real-world robot. We have provided more insight concerning learning probabilistic relational rules as well as their relation to other relational models.

Combining our proposed techniques with existing methods in an integrated framework allows to build agents for autonomous goal-directed behavior. Without doubt our work is only among the first steps into the direction of creating fully autonomous agents for the real world. Nonetheless, we hope to have convinced the reader that studying goal-directed behavior in a statistical relational artificial intelligence context is a promising and exciting approach to build intelligent agents—and to understand principles and concepts underlying intelligence in general. To substantiate the latter claim, in the following gedankenexperiment we take a look at how our methods could model the goal-directed behavior of chimpanzees.

A Computational Model for the Goal-Directed Behavior of Chimpanzees

In the introduction, we saw how the chimpanzee Grande found out how to get a banana hanging from the ceiling by building a tower of boxes (Fig. 6.1). Her behavior made the strong impression to involve significant cognitive processing. We described several capabilities and prerequisites which Grande needed to have mastered to perform this goal-directed behavior (p. 2). Along these lines, we discuss now how the methods proposed in this thesis in combination with existing techniques could be used to create an agent which reproduces Grande's behavior. We do not want to imply that chimpanzees really employ these computational principles; nor do we want to argue to have "solved" goal-directed behavior of chimpanzees. Significantly more work some of which we indicate below is required to apply symbolic reasoning methods to full-fledged real-world scenarios. Rather, the following considerations are meant to provide an intuition of the great potential of statistical relational AI techniques and to motivate further research.

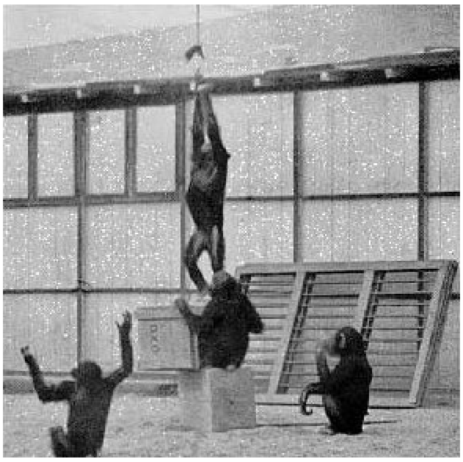


Figure 6.1: Statistical relational AI provides computational principles which can be used to model goal-directed behavior of chimpanzees. (Source: Köhler, 1917)

- Agent-Grande (the agent trying to reproduce Grande's behavior) could use a relational representation to describe the environment in terms of **objects and their properties and relationships**. For instance, Agent-Grande could use a symbol b to represent the banana and $box1$ to represent a specific box and predicates and functions to express properties and relationships of objects, such as $banana(b)$, $box(box1)$, $clear(box1)$, $on(box1, box2)$ and $height(box1)$, and symbolic actions such as $grab(b)$, $puton(box1)$ and $climb(box2)$ to abstract from motor primitives.

- As a **model of the effects of its actions**, Agent-Grande could use a compact probabilistic relational transition model which generalizes over situations and objects. For instance, it could maintain the following probabilistic relational rule to express that it can put an inhand-held box on top of another box which succeeds with high probability, but the box can also land on the ground instead:

$$\begin{array}{rl} \mathit{puton}(X): & \mathit{box}(X),\ \mathit{clear}(X),\ \mathit{box}(Y),\ \mathit{inhand}(Y) \\ \rightarrow & \begin{cases} 0.9 & : \ \mathit{on}(Y,X),\ \mathit{clear}(Y),\ \neg \mathit{clear}(X),\ \neg \mathit{inhand}(Y) \\ 0.1 & : \ \mathit{onGround}(Y),\ \mathit{clear}(Y),\ \neg \mathit{inhand}(Y) \end{cases} \end{array}$$

- Agent-Grande could **learn** such a rule from experience by means of the algorithm of Pasula et al. (2007) used in our experiments. For instance, the above rule could have been extracted from the following experience (among others):

$$\begin{array}{rl} \mathit{puton}(d)\text{:} & \mathit{box}(a),\ \mathit{box}(c),\ \mathit{box}(d),\ \mathit{onGround}(a),\ \mathit{on}(d,a),\ \underline{\mathit{inhand}(c)},\ \underline{\mathit{clear}(d)}, \\ \rightarrow & \mathit{box}(a),\ \mathit{box}(c),\ \mathit{box}(d),\ \mathit{onGround}(a),\ \mathit{on}(d,a),\ \underline{\mathit{on}(c,d)},\ \underline{\mathit{clear}(c)} \end{array}$$

- Agent-Grande could use our relational exploration framework and in particular our algorithm REX to efficiently **explore** the unknown objects and actions in the environment. For example, if it had never experienced a $\mathit{puton}(X)$ action with a rock, that is a situation where $\mathit{rock}(X)$ holds, it might want to perform this action to understand what to do with rocks (thereby generalizing over situations and specific rocks).

- All the frameworks and algorithms used in this thesis handle **uncertainty**. For instance, the rule from above specifies the possibility that a box falls on the ground when trying to put it on top of another box. Our algorithms for planning such as PRADA account for the uncertainty of action outcomes. Similarly, our relational exploration framework based on relational density estimation is a probabilistic approach.

- For **reasoning and planning**, Agent-Grande could employ any of our planning algorithms. For instance, PRADA could use learned rules to come up with an appropriate sequence of $\mathit{grab}(\cdot)$ and $\mathit{puton}(\cdot)$ actions to build a tower, followed by a $\mathit{climb}(\cdot)$ action resulting in similar values for $\mathit{height}(\mathit{grande})$ and $\mathit{height}(b)$ (b is the banana), so that finally $\mathit{grab}(b)$ can be performed successfully.
  Our planning algorithms reason in the ground relational domain, that is, on the level of concrete objects. If there are very many objects, Agent-Grande can focus on the relevant objects according to our relevance grounding framework. For instance, it could take only the three closest boxes and the banana into account while ignoring all other boxes, the cage door and the hat of the keeper.

6.1 Future Work

In the previous chapters, we have provided many concrete ideas for future work in the context of our proposed methods. Here, we discuss more generally research directions for the exciting challenge of real-world autonomous goal-directed behavior.

Object-Driven Goal-Directed Behavior

In our view, focusing on specific objects is one of the most promising structural assumptions and priors to develop methods for goal-directed behavior in the real world. In Chapter 4, we have described a framework for focusing on relevant objects when planning in ground relational domains. The crucial question is whether we can learn object relevance models, potentially based on priors such as the spatial proximity to goal objects. The impact of priors over objects might be even greater on exploration. Current work in machine learning and reinforcement learning, including ours in Chapter 5, defines novelty and interestingness with respect to states. In real-world environments with their complex state descriptions and large state spaces, it seems to be more efficient to formalize novelty and interestingness in terms of objects. This fundamental change in perspective comes arguably much closer to human behavior and has the potential to greatly increase the goal-directedness in exploration.

Expressive Transition Models

Transition models of the environment are at the heart of goal-directed behavior in stochastic relational worlds and need to permit the development of efficient methods for learning, planning and exploration. The rule representation and learning algorithm by Pasula et al. (2007), drawing from ideas in inductive logic programming, forms an appealing approach, but extensions in different directions are required for real-world applicability, such as incremental learning algorithms for online adaptation and parallel rules modeling different aspects of the environment. In our view, one of the most important directions is multi-time-step abstraction of actions. For instance, presumably humans cannot plan more than ten time-steps forward due to their limited cognitive resources; hence, they need to come up with abstractions which integrate multiple lower-level time-steps. While the importance of abstraction is widely appreciated, no frameworks are currently available for autonomous agents in natural environments: existing frameworks, such as so-called cognitive architectures, the options-framework in reinforcement learning and traditional macro-operator approaches, cannot be learned from data, cannot cope with uncertainty or do not generalize over objects and situations. Nonetheless, simple, but effective techniques for action-sequence abstraction seem to be in reach without significant changes to our algorithms and representations.

Symbol Grounding

In this thesis, we have followed the fundamental assumption of AI and cognitive science that many aspects of higher-level intelligence can be achieved by symbolic reasoning. But where do these symbols come from and how are they related to the real world? This is the question of symbol grounding and has been debated for long in AI and philosophy. While this question is far from being answered, future work can investigate well-defined aspects of symbol grounding. For instance, one can adapt motion primitives to become more coherent with abstract symbolic actions: the learned "typical" outcome of symbolic actions may provide an objective function to modify the control of motor primitives.

Relational Representations on Non-symbolic Levels

We used relational representations to describe natural environments on a discrete symbolic high level in this thesis. Relations may likewise define continuous properties and couplings of objects. Hence, relational representations can also be used to describe many lower-level aspects of natural environments, such as geometric relations and physical links between objects, for instance between a drawer and a cupboard or between a door and its frame. The concepts and principles of relational exploration, planning and learning discussed in this thesis seem to be applicable on such non-symbolic relational representations, but the computational techniques need to be modified appropriately. For instance, using relational representations on non-symbolic levels might guide an autonomous robot to explore and detect the handles of doors and drawers in its environment which it can manipulate.

Appendix A

Proofs

A.1 Proof of Proposition 4.2.1

Proposition 4.2.1 (p. 59) *The set of action sequences PRADA samples with non-zero probability is a super-set of the ones of SST and UCT.*

Proof: Let $\mathbf{a}^{0:T-1}$ be an action sequence that was sampled by SST (or UCT). Thus, there exists a state sequence $\mathbf{s}^{0:T}$ and a rule sequence $\mathbf{r}^{0:T-1}$ such that in every state s^t ($t < T$), action a^t has a unique covering rule r^t that predicts the successor state s^{t+1} with probability $p^t > 0$. For, if $p^t = 0$, then s^{t+1} would never be sampled by SST (or UCT).

We have to show that $\forall t, 0 \leq t < T : P(s^t \mid \mathbf{a}^{0:t-1}, s^0) > 0$. If this is the case then $P^t_{sample}(a^t) > 0$ as a^t has the unique covering rule r^t in s^t and a^t will eventually be sampled. $P(s^0) = 1 > 0$ is obvious. Now assume $P(s^t \mid \mathbf{a}^{0:t-1}, s^0) > 0$. If we execute a^t, we will get $P(s^{t+1} \mid \mathbf{a}^{0:t}, s^0) \geq p^t P(s^t \mid \mathbf{a}^{0:t-1}, s^0) > 0$. The posterior $P(s^{t+1} \mid \mathbf{a}^{0:t}, s^0)$ can be greater (first inequality) due to persistence or to previous states having non-zero probability that also lead to s^{t+1} given a^t.

The set of action sequences PRADA samples is larger than that of SST (or UCT) as SST (or UCT) refuses to model the noise outcomes of rules. Assume an action a and state s to be the only state where a has a unique covering rule. If an episode to s can only be simulated by means of rule predictions with the noise outcome, this action will never be sampled by SST (or UCT) (as the required states are never sampled). In contrast, PRADA also models the effects of the noise outcome by giving very low probability to all possible successor states. □

A.2 Proof of Lemma 4.4.1

Lemma 4.4.1 (p. 94) *When conditioning on a subset* $\mathbf{o}$ *of relevant objects, the following probabilities in the reduced model* $\Gamma_{\mathbf{o}}$ *are the same as in the full model* Γ*:*

- *State sequences:* $P(\mathbf{s} \mid \mathbf{o}, \mathbf{a}; \Gamma) = P(\mathbf{s} \mid \mathbf{a}; \Gamma_{\mathbf{o}})$

- *Rewards:* $P(R\,|\,\mathbf{o},\mathbf{a};\Gamma) = P(R\,|\,\mathbf{a};\Gamma_{\mathbf{o}})$
- *Action sequences:* $P(\mathbf{a}\,|\,\mathbf{o},R;\Gamma) = P(\mathbf{a}\,|\,R;\Gamma_{\mathbf{o}})$

Proof If $\mathbf{o} \in \Omega(\mathbf{s},\mathbf{a})$, we have:

$$\begin{aligned} P(\mathbf{s}\,|\,\mathbf{o},\mathbf{a};\Gamma) &= \prod_{t=0}^{T-1} P(s_{t+1}\,|\,\mathbf{o},s_t,a_t;\Gamma) \\ &= \prod_{t=0}^{T-1} P(s_{t+1}\,|\,s_t,a_t;\Gamma_{\mathbf{o}}) = P(\mathbf{s}\,|\,\mathbf{a};\Gamma_{\mathbf{o}})\,. \end{aligned}$$

If $\mathbf{o} \notin \Omega(\mathbf{s},\mathbf{a})$, we have $P(\mathbf{s}\,|\,\mathbf{o},\mathbf{a};\Gamma) = 0$. Similarly, s cannot be predicted in $\Gamma_{\mathbf{o}}$ as only the irrelevant object set $\mathbf{o}$ is available, so we get $P(\mathbf{s}\,|\,\mathbf{a};\Gamma_{\mathbf{o}}) = 0$. Furthermore, we have:

$$\begin{aligned} P(R\,|\,\mathbf{o},\mathbf{a};\Gamma) &= \sum_{\mathbf{s}} P(R,\mathbf{s}\,|\,\mathbf{o},\mathbf{a};\Gamma) = \sum_{\mathbf{s}} P(R\,|\,\mathbf{s},\mathbf{o},\mathbf{a};\Gamma)P(\mathbf{s}\,|\,\mathbf{o},\mathbf{a};\Gamma) \\ &= \sum_{\mathbf{s}} P(R\,|\,\mathbf{s},\mathbf{a};\Gamma)\,P(\mathbf{s}\,|\,\mathbf{a};\Gamma_{\mathbf{o}})\,I(\mathbf{o}\in\Omega(\mathbf{s},\mathbf{a})) \\ &= \sum_{\mathbf{s}:\,\mathbf{o}\in\Omega(\mathbf{s},\mathbf{a})} P(R\,|\,\mathbf{s},\mathbf{a};\Gamma)P(\mathbf{s}\,|\,\mathbf{a};\Gamma_{\mathbf{o}}) = P(R\,|\,\mathbf{a};\Gamma_{\mathbf{o}})\ . \end{aligned}$$

Finally, we have:

$$\begin{aligned} P(\mathbf{a}\,|\,\mathbf{o},R;\Gamma) &\propto P(R\,|\,\mathbf{o},\mathbf{a};\Gamma)\;P(\mathbf{a}\,|\,\mathbf{o};\Gamma) \\ &= P(R\,|\,\mathbf{a};\Gamma_{\mathbf{o}})\;P(\mathbf{a};\Gamma_{\mathbf{o}}) \;\propto\; P(\mathbf{a}\,|\,R;\Gamma_{\mathbf{o}})\ . \quad \Box \end{aligned}$$

Appendix B

Relation between NID rules and PPDDL

We use noisy indeterministic deictic (NID) rules (Pasula et al., 2007) as a relational model of the transition dynamics of probabilistic actions. Besides allowing for negative literals in the preconditions, NID rules extend probabilistic STRIPS operators (Kushmerick et al., 1995; Blum and Langford, 1999) by two special constructs, namely deictic references and noise outcomes, which are crucial for learning compact rule-sets. An alternative language to specify probabilistic relational planning problems used by the International Probabilistic Planning Competitions (IPPC, 2008) is the *probabilistic planning domain definition language* (PPDDL) (Younes and Littman, 2004). PPDDL is a probabilistic extension of a subset of PDDL (Ghallab et al., 1998) which was derived from the deterministic *action description language* (ADL) (Pednault, 1989). ADL, in turn, introduced universal and conditional effects and negative precondition literals into the (deterministic) STRIPS representation (Fikes and Nilsson, 1971). Thus, PPDDL allows for the usage of syntactic constructs which are beyond the expressive power of NID rules; however, many PPDDL descriptions can be converted into NID rules.

Before taking a closer look at how to convert PPDDL and NID rule representations into each other, we clarify what is meant by "action" in each of the formalisms, giving an intuition of the line of thinking when using either of these. We understand by "abstract action" an abstract action predicate, e.g. $pickup(X)$. Intuitively, this defines a certain type of action. The stochastic state transitions according to an abstract action can be specified by both abstract NID rules as well as abstract PPDDL action operators (also called schemata). Typically, several different abstract NID rules model the same abstract action, specifying state transitions in different contexts. In contrast, usually only one abstract PPDDL action operator is used to model an abstract action: context-dependent effects are modeled by means of conditional and universal effects.

To make predictions in a specific situation for a concrete action (a ground action predicate such as $pickup(greenCube)$), the procedure within the NID rule framework is to ground the set of abstract NID rules with respect to this state-action pair. If there is exactly one covering ground rule, it is chosen for prediction. If there is no such rule or if

Table B.1: Example for converting a PPDDL action operator into NID rules. The *putDown*-operator of the IPPC benchmark domain *Exploding Blocksworld* (a) contains a conditional effect which can be accounted for by two NID rules which either exclude (b) or include (c) this condition in their context.

(a)

```
( : action putDown
  : parameters (?b − block)
  : precondition (and (holding ?b) (noDestroyedTable))
  : effect (and (emptyhand) (onTable ?b) (not (holding ?b))
    (probabilistic 2/5 (when (noDetonated ?b) (and (not (noDestroyedTable)) (not (noDetonated?b))))))
)
```

(b)

$$\begin{aligned} &putDown(X): \quad block(X),\ holding(X),\ noDestroyedTable(),\ \neg noDetonated(X) \\ &\quad \rightarrow \Big\{\ 1.0 \ : \ emptyhand(X),\ onTable(X),\ \neg holding(X) \end{aligned}$$

(c)

$$\begin{aligned} &putDown(X): \quad block(X),\ holding(X),\ noDestroyedTable(),\ noDetonated(X) \\ &\quad \rightarrow \begin{cases} 0.6 & : \ emptyhand(X),\ onTable(X),\ \neg holding(X) \\ 0.4 & : \ emptyhand(X),\ onTable(X),\ \neg holding(X),\ \neg noDestroyedTable(),\ \neg noDetonated(X) \end{cases} \end{aligned}$$

there is more than one (the contexts of NID rules do not have to be mutually exclusive), one chooses the noisy default rule, essentially saying that one does not know what will happen (other strategies are conceivable, but not pursued here). In contrast, as there is usually exactly one operator per abstract action in PPDDL domains, there is no need of the concept of operator uniqueness and to distinguish between ground actions and operators.

Converting PPDDL to NID rules

In the following, we discuss how to convert PPDDL features into a NID rule representation. While it may be impossible to convert a PPDDL action operator into a single NID rule, one may often translate it into a *set* of rules with at most a polynomial increase in the size of representation. Table B.1 provides an example of a converted PPDDL action operator of the IPPC domain *Exploding Blocksworld*. As NID rules support many, but not all of the features a sophisticated domain description language such as PPDDL provides, using rules will not lead to compact representations in all possible domains. Our experiments, however, show that the dynamics of many interesting planning domains can be specified compactly. Furthermore, additional expressive power in rule contexts can be gained by using derived predicates and functions (defined by formulas over other predicates and functions) which allow to bring in various kinds of logical formulas such as quantification.

Conditional Effects A conditional effect in a PPDDL operator takes the form *when C then E*. It can be accounted for by two NID rules: the first rule adds *C* to its context and

E to its outcomes, while the second adds $\neg C$ to its context and ignores E.

Universal Effects PPDDL allows to define universal effects. These specify effects for all objects that meet some preconditions. An example is the $reboot$ action of the *SysAdmin* domain of the IPPC 2008 competition: it specifies that every computer other than the one rebooted can independently go down with probability 0.2 if it is connected to a computer that is already down. This *cannot* be expressed in a NID rule framework. While we can refer to objects other than the action arguments via deictic references, we require these deictic references to be unique. For the $reboot$ action, we would need a unique way to refer to each other computer which cannot be achieved without significant modifications (for example, such as enumerating the other computers via separate predicates).

Disjunctive Preconditions and Quantification PPDDL operators allow for disjunctive preconditions, including implications. For instance, the *Search-and-rescue* domain of the IPPC 2008 competition defines an action operator $goto(X)$ with the precondition $(X \neq base) \rightarrow humanAlive()$. A disjunction $A \vee B$ ($\equiv \neg A \rightarrow B$) can be accounted for by either using two NID rules, with the first rule having A in the context and the second rule having $\neg A \wedge B$. Alternatively, one may introduce a derived predicate $C \equiv A \vee B$. In general, the "trick" of derived predicates allows to overcome syntactical limitations of NID rules and bring in various kinds of logical formulas such as quantifications. As discussed by Pasula et al. (2007), derived predicates are an important prerequisite to being able to learn compact and accurate rules.

Types Terms may be typed in PPDDL, e.g. $driveTo(C - city)$. Typing of objects and variables in predicates and functions can be achieved in NID rules by the usage of typing predicates within the context, e.g. using an additional predicate $city(C)$.

State Transition Rewards In PPDDL, one can encode Markovian rewards associated with state transitions (including action costs as negative rewards) using fluents and update rules in action effects. One can achieve this in NID rules by associating rewards with the outcomes of rules.

Converting NID rules to PPDDL

We show in the following that the way NID rules are used in the planning algorithms presented in Chapter 4 (SST, UCT and PRADA) at *planning* time can be handled in PPDDL via at most a polynomial blowup in representational size. The basic building blocks of a NID rule, that is, the context as well as the outcomes, transfer one-to-one to PPDDL action operators. The deictic references, the uniqueness requirement of covering rules and the noise outcome need special attention.

Deictic References Deictic references in NID rules allow to refer to objects which are not action arguments. In PPDDL, one can refer to such objects by means of universal conditional effects. There is an important restriction, however: a deictic reference needs to pick out a single unique object in order to apply. If it picks out none or many, the rule fails to apply. There are two ways to ensure this uniqueness requirement within PPDDL. First, if allowing quantified preconditions, an explicit uniqueness precondition for each deictic reference D can be introduced. Using universal quantification, it constrains all objects satisfying the preconditions Φ_D of D to be identical: $\forall D_1, D_2 : \Phi_D(D_1, *) \wedge \Phi_D(D_2, *) \rightarrow D_1 = D_2$, where $*$ are some other variables. Alternatively, uniqueness of deictic references can be achieved with a careful planning problem specification by hand, which however cannot be guaranteed when learning rules.

Uniqueness of covering rules The contexts of NID rules do not have to be mutually exclusive. When we want to use a rule for prediction (as in planning), we need to ensure that it uniquely covers the given state-action pair. The procedural evaluation process for NID rules can be encoded declaratively in PPDDL using modified conditions which explicitly negate the contexts of competing rules. For instance, if there are three NID rules with potentially overlapping contexts A, B, and C (propositional for simplicity), the PPDDL action operator may define four conditions: $c_1 = \{A \wedge \neg B \wedge \neg C\}$, $c_2 = \{\neg A \wedge B \wedge \neg C\}$, $c_3 = \{\neg A \wedge \neg B \wedge C\}$, $c_4 = \{(\neg A \wedge \neg B \wedge \neg C) \vee (A \wedge B) \vee (A \wedge C) \vee (B \wedge C)\}$. Conditions c_1, c_2 and c_3 test for uniqueness of the corresponding NID rules and subsume their outcomes. Condition c_4 tests for non-uniqueness (either no covering rule or multiple covering rules) and models potential changes as noise, analogous to the situations in a NID rule context in which the noisy default rule would be used.

Noise outcome The noise outcome of a NID rule subsumes seldom or utterly complex outcomes. It relaxes the frame assumption: even not explicitly stated things may change with a certain probability. This comes at the price of the difficulty to ensure a well-defined successor state distribution $P(s' \mid s, a)$. In contrast, PPDDL needs to explicitly specify everything that might change. This may be an important reason why it is difficult to come up with an effective learning algorithm for PPDDL.

While in principle PPDDL does not provide for a noise outcome the way our approaches account for it in *planning* can be encoded in PPDDL. We either treat the noise outcome as having no effects (in SST and UCT; basically a noop operator then) which is trivially translated to PPDDL; or we consider the probability of each state attribute to change independently (in PRADA) which can be encoded in PPDDL with independent universal probabilistic effects.

The noise outcome allows to always make predictions for actions: if there are no or multiple covering rules, we may use the (albeit not very informative) prediction of the default rule. Such cases can be dealt with in PPDDL action operators using explicit conditions as described in the previous paragraph.

Appendix C

Theoretical Considerations concerning PRADA

In this appendix chapter, we provide some theoretical results and intuitions concerning our algorithm PRADA (Lang and Toussaint, 2009b, 2010b) presented in Chapter 4.

C.1 PRADA Finds the Optimal Solution with Exact Inference

We present two simple lemmata in the following which are probably intuitively obvious to the reader and affirm our understanding of PRADA. Let T denote the fixed planning horizon and $\mathbf{a}^* = \operatorname*{argmax}_{\mathbf{a}} P(u^T \mid \mathbf{a})$ the optimal plan.

Lemma C.1.1 *If PRADA uses exact instead of approximate inference to calculate the posteriors $P(\mathbf{s}^T \mid \mathbf{a})$ and $P(u^T \mid \mathbf{a})$ and provided the prior $P(\mathbf{a})$ has support for the optimal sequence, that is $P(\mathbf{a}^*) > 0$, then for any δ with $1 > \delta > 0$ there is a number N of action-sequence samples resulting in set $\hat{A}$ such that the best $\mathbf{a}' = \operatorname*{argmax}_{\mathbf{a} \in \hat{A}} P(u^T \mid \mathbf{a})$ is optimal with probability at least δ, that is, the probability that $\mathbf{a}^* = \mathbf{a}'$ is bigger than δ.*

Proof First of all note that the optimal solution for a given horizon T can trivially be found by looking at all action sequences. This is possible as the search space of plans is finite: the actions are discrete and their number is finite, and thus the number of action sequences of length T is also finite.

Instead of a systematic search, we prove the lemma for sampling with prior $P(\mathbf{a})$: we have to show that the probability to sample any sequence goes to 1 if $N \to \infty$. We look at the worst case that the optimal sequence has minimal probability $\gamma := P(\mathbf{a}^*) = \min_{\mathbf{a}} P(\mathbf{a})$ to be sampled. Then, the probability to have found the optimal sequence after N samples is $1 - (1 - \gamma)^N$. To bound this probability with $1 - (1 - \gamma)^N > \delta$, we require $N > \frac{\log(1-\delta)}{\log(1-\gamma)}$ samples. □

Thus, we almost surely sample $\mathbf{a}^*$ if $N \to \infty$. If an action-sequence $\mathbf{a}$ has a non-zero prior $P(\mathbf{a})$ with PRADA's sampling distribution under exact inference, then it also has a non-zero prior under approximate inference based on a factored frontier. (The FF inference approximates posterior joints by products of marginals and thus such joints are only 0 when some marginals are 0; but if such a marginal is 0 under exact inference, then the exact posterior joint must also be 0.) Thus, also with approximate FF-inference, PRADA will almost surely sample the optimal sequence $\mathbf{a}^*$; because of potential approximation errors, however, there is no guarantee that this sequence is assigned the highest estimated value $P(u \,|\, \mathbf{a}^*)$.

Furthermore, PRADA approximates the posterior $P(\mathbf{a} \,|\, u)$ over actions, taking into account the prior over actions and the likelihood to achieve reward $P(u \,|\, \mathbf{a})$. In the following, we use $\delta_{\mathbf{a}_i,\mathbf{a}}$ which evaluates to 1 if $\mathbf{a}_i = \mathbf{a}$ and to 0 otherwise.

Lemma C.1.2 *Under the assumption that PRADA uses exact inference, the Monte-Carlo approximation* $\hat{P}_N(\mathbf{a} \,|\, u) = \sum_{i=0}^{N} P(u \,|\, \mathbf{a}_i)\, \delta_{\mathbf{a}_i,\mathbf{a}}$ *based on* N *sampled action-sequences* $\mathbf{a}_i$ *converges to the exact posterior* $P(\mathbf{a} \,|\, u)$*.*

Proof We show that PRADA approximates $P(\mathbf{a} \,|\, u)$ by means of importance sampling. We rewrite

$$P(\mathbf{a} \,|\, u) = \frac{P(u \,|\, \mathbf{a}) P(\mathbf{a})}{P(u)} \propto P(u \,|\, \mathbf{a}) P(\mathbf{a}) \; .$$

We use $P(\mathbf{a})$ as proposal distribution to sample action-sequences $\mathbf{a}_i$ which has support for all $\mathbf{a}$ with $P(\mathbf{a} \,|\, u) > 0$. We define $w_i := P(u \,|\, \mathbf{a}_i)$ as importance weights. Following the lines of Andrieu et al. (2003), if $\mathbf{a}$ was continuous, then for any arbitrary smooth function f by the law of large numbers we would have:

$$\hat{I}_N(f) = \frac{1}{\sum_i w_i} \sum_{i=1}^{N} f(\mathbf{a}_i)\, w_i \rightarrow_{N\to\infty}^{almost\,surely} I(f) = \int f(\mathbf{a}) P(\mathbf{a} \,|\, u) d\mathbf{a} \; .$$

Applying this result to discrete $\mathbf{a}$, we get

$$\hat{P}_N(\mathbf{a} \,|\, u) = \frac{1}{\sum_i w_i} \sum_{i=1}^{N} \delta_{\mathbf{a}_i,\mathbf{a}}\, w_i \rightarrow_{N\to\infty}^{almost\,surely} P(\mathbf{a} \,|\, u) \; .$$

□

C.2 Sufficient Conditions for Exact Inference

We describe a set of conditions under which the inference of PRADA based on a factored frontier leads to exact posteriors. While these conditions clearly do not hold in most realistic scenarios, these considerations are instructive to get a better intuition of PRADA's behavior.

PRADA uses a factored frontier to compute the posteriors $P(\mathbf{s}^T \mid \mathbf{a})$ and $P(u^T \mid \mathbf{a})$. Its inference becomes exact if the processes of the individual state attributes remain decoupled, that is, if the attributes are independent conditioning on the action sequence $\mathbf{a}$. (In our derivation of PRADA's inference in Chapter 4, we used further minor approximations apart from the factored frontier; we ignore these here, but we note that one can easily renounce on them under the assumptions of the lemma.)

Lemma C.2.1 *If the exact initial belief is fully factored and if for each action it holds that it is modeled by a set of NID rules which (i) do not contain deictic references, (ii) all involve only the same single state attribute in both their contexts and outcomes and (iii) in all states there is a unique covering rule (thus, we never use the noisy default rule), then PRADA's inference of $P(\mathbf{s}^T \mid \mathbf{a})$ and $P(u^T \mid \mathbf{a})$ based on a factored frontier in the corresponding PRADA-DBN is exact.*

In the planning problems of this thesis, we set the initial state marginals deterministic according to the start state; then, the initial belief $P(\mathbf{s}^0)$ is fully factored. Under the conditions of the lemma, sensible rule-sets for modeling an action consist of at most two rules: either two rules specifying the contexts where the attribute does and does not hold, or a single rule with an empty context. An exemplary rule involving state attribute $b(\cdot,\cdot)$ might take the form

$$action(X,Y): \quad b(X,Y) \quad \rightarrow \quad \left\{ \begin{array}{lll} p & : & \neg b(X,Y) \\ 1.-p & : & - \end{array} \right. .$$

Proof We prove the lemma by induction. In the beginning at $t = 0$, the exact belief is fully factored, that is, $P(\mathbf{s}^0) = \prod_l P(s_l^0)$. Now assume the belief $P(\mathbf{s}^t \mid \mathbf{a}^{0:t-1})$ is exact. We want to infer the belief $P(\mathbf{s}^{t+1} \mid \mathbf{a}^{0:t})$ taking the action a^t into account. All rules modeling a^t involve the same state attribute s_i, and hence a^t manipulates only s_i. The posterior over covering rules and their contexts depends only on s_i^t. Therefore, the calculation of the posteriors over the contexts according to Eqs. (4.13) and (4.17) and in turn over rules according to Eq. (4.12) involves only the marginal over s_i^t and is thus exact. In turn, Eq. (4.9) calculates the exact marginal $P(s_i^{t+1} \mid \mathbf{a}^{0:t})$. All other state attributes s_k $(k \neq i)$ are set to persist. Hence, the posterior over any s_l^{t+1} is calculated independently of any other attribute, that is, $\forall k, l, k \neq l : P(s_l^{t+1} \mid s_k^{t+1}, \mathbf{a}^{0:t}) = P(s_l^{t+1} \mid \mathbf{a}^{0:t})$. Therefore, the exact marginals provide us exact beliefs $P(\mathbf{s}^{t+1} \mid \mathbf{a}^{0:t}) = \prod_l P(s_l^{t+1} \mid \mathbf{a}^{0:t})$. □

C.3 PRADA Assumes Rewards are Probable

The sampling distribution $P_{sample}(a)$ of an action-sampling planning algorithm incorporates a bias which makes the algorithm appropriate to specific planning scenarios. Let's look at PRADA's sampling distribution at an arbitrary, but fixed time-step t. Thus far, we have sampled $\mathbf{a}^{0:t-1}$. PRADA samples an action a^t according to the distribution (see p. 59)

$$P^t_{sample}(a) \propto \sum_{r \in \Gamma(a)} P\left(\phi^t_r = 1, \bigwedge_{r' \in \Gamma(a) \setminus \{r\}} \phi^t_{r'} = 0 \mid \mathbf{a}^{0:t-1}\right). \tag{C.1}$$

On the level of states, this sampling distribution chooses with equal probability among the actions that have a unique covering rule in a state. As PRADA maintains beliefs over states, its sampling probability of an action a depends on the belief over those states where a has a unique covering rule. Hence, PRADA's sampling strategy takes the previous action samples $\mathbf{a}^{0:t-1}$ into account. The basic assumption underlying PRADA's sampling strategy is the following:

> **Assumption of PRADA's action-sampling strategy**: *The reward is probable when executing the optimal action-sequence* $\mathbf{a}^*$*. That is,* $P(u \mid \mathbf{a}^*)$ *is large.*

Thus, the more likely reward is for $\mathbf{a}^*$ (the higher $P(u \mid \mathbf{a}^*)$ is), the higher is the probability of PRADA to sample $\mathbf{a}^*$. We illustrate this in toy scenarios in the following.

Analytical Toy Example

In our first simple toy scenario, there are two actions a_1 and a_2 modeled by the rules

$$\begin{array}{llll} a_1 : & - \rightarrow & \left\{ \begin{array}{lcl} \alpha & : & b_1 \\ 1. - \alpha & : & - \end{array} \right. & \text{and} \\ a_2 : & b_1 \rightarrow & \left\{ \begin{array}{lcl} 1.0 & : & b_2 \end{array} \right. & . \end{array}$$

While we can always execute a_1 (its rule has an empty context), the applicability of a_2 depends on the state attribute b_1. We start in the empty state $\mathbf{s}_0 = \{\}$, our goal is $\tau = \{b_2\}$ and our planning horizon is $T = 2$. The optimal sequence is $\mathbf{a}^* = (a_1, a_2)$ and has reward likelihood $P(u \mid \mathbf{a}^*) = \alpha$. Let's look at PRADA's sampling distributions $P^t_{sample}(a)$ at $t = 0$ and $t = 1$:

$P^t_{sample}(a)$	$t = 0$	$t = 1$
a_1	1	$\frac{1}{1+\alpha}$
a_2	0	$\frac{\alpha}{1+\alpha}$

At $t = 0$, the only action with a (unique) covering rule is a_1. Thus, a_1 is always sampled. This leads to two possible successor states $\mathbf{s}'_1 = \{b_1\}$ and $\mathbf{s}''_1 = \{\}$. While a_1 is applicable in both, a_2 is only applicable in $\mathbf{s}'_1$. Therefore, $P^{t=1}_{sample}(a_2)$ depends on the belief over $\mathbf{s}'_1$ which is $P(\mathbf{s}'_1 \mid a_1) = \alpha$. The sampling probability of the optimal sequence $\mathbf{a}^*$ is $P_{sample}(\mathbf{a}^*) = \frac{\alpha}{1+\alpha}$, and the expected number of action sequences required to sample $\mathbf{a}^*$ is $\frac{1+\alpha}{\alpha}$. We see that the higher α is, the more likely are both, sampling $\mathbf{a}^*$ as well as the reward. In contrast, sampling with a uniform distribution over actions is independent of the stochasticity α and the reward likelihood. Here are some expected numbers of required samples until $\mathbf{a}^*$ is sampled for the first time:

α	$P(u \mid \mathbf{a}^*)$	PRADA	uniform
0.1	0.1	11	4
0.5	0.5	3	4
0.9	0.9	2.1	4

Simulated Toy Example

We also performed a simulation in a relational toy scenario. There are three unary actions modeled by the following rules:

$$\begin{array}{llcl}
act1(X): & - & \rightarrow & \begin{cases} \alpha & : \; b_1(X) \\ 1-\alpha & : \; - \end{cases} \\
act2(X): & b_1(X) & \rightarrow & \begin{cases} \alpha & : \; b_2(X) \\ 1-\alpha & : \; - \end{cases} \\
act3(X): & b_2(X) & \rightarrow & \begin{cases} 1 & : \; b_3(X) \\ 0 & : \; - \end{cases}
\end{array}$$

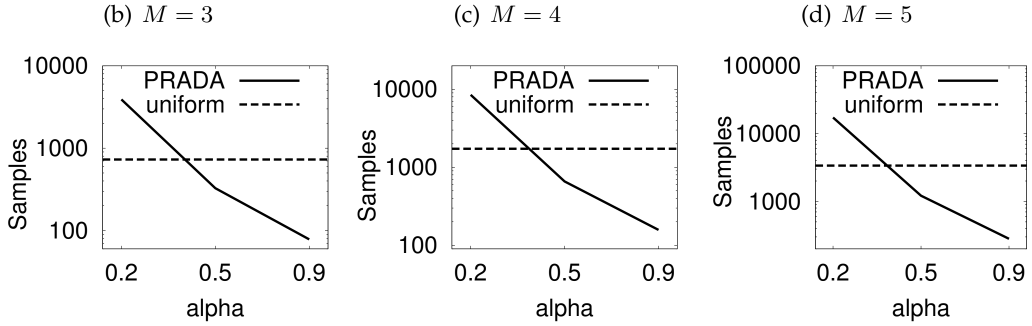
We start in the empty state $s_0 = \{\}$, there are M objects $\mathcal{O} = \{o_1, \ldots, o_M\}$, the goal is $\tau = \{b_3(o_1)\}$ and the planning horizon is $T = 3$. The optimal plan is $\mathbf{a}^* = (act1(o_1), act2(o_1), act3(o_1))$. The total number of ground actions is $3M$ and the search space of plans contains $(3M)^3$ different plans. The probability to sample $\mathbf{a}^*$ with a *uniform* distribution over actions is $\frac{1}{(3M)^3}$ and hence the expected number of samples until the optimal sequence is found is $(3M)^3$.

We investigate the planning performance of PRADA with respect to different settings of the stochasticity α and the number of objects M. To collect statistics, we performed 1000 runs for each combination of α and M. The results shown in Table C.1 indicate how PRADA's performance is related to the reward likelihood $P(u \,|\, \mathbf{a}^*)$: the more likely the reward is, the less samples are required by PRADA. In contrast, the uniform distribution is unrelated to the reward likelihood. Thus, PRADA's sampling strategy is much more efficient in scenarios with large reward likelihoods (in our case, when $\alpha = 0.5$ or $\alpha = 0.9$) and less efficient in scenarios with small reward likelihood ($\alpha = 0.2$).

Table C.1: *Results for evaluating PRADA's sampling strategy in a toy scenario.* PRADA's sampling distribution is significantly more efficient than uniform sampling when the reward likelihood $P(u \mid \mathbf{a}^*)$ is large. M denotes the number of objects and α determines the stochasticity of the planning problem. (a) The figures for PRADA are the mean estimators of the average required number of samples until $\mathbf{a}^*$ is sampled for the first time based on 1000 runs. The figures for uniform sampling are the exact expectations of the required number of samples. (b)-(d) Visualization of the figures in (a).

(a)

		$P(u \mid \mathbf{a}^*)$	PRADA	uniform
	$\alpha = 0.2$	0.04	3906.7 ± 3.9	729
$M = 3$	$\alpha = 0.5$	0.25	326.8 ± 0.3	729
	$\alpha = 0.9$	0.81	78.2 ± 0.1	729
	$\alpha = 0.2$	0.04	8488.3 ± 8.5	1728
$M = 4$	$\alpha = 0.5$	0.25	657.9 ± 0.6	1728
	$\alpha = 0.9$	0.81	157.5 ± 0.2	1728
	$\alpha = 0.2$	0.04	17293.2 ± 13.	3375
$M = 5$	$\alpha = 0.5$	0.25	1215.1 ± 1.2	3375
	$\alpha = 0.9$	0.81	281.1 ± 0.3	3375


(b) $M = 3$ (c) $M = 4$ (d) $M = 5$

Literature

J. Anderson. *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum, 1993.

C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine Learning Journal*, 50(1-2):5–43, 2003.

A. Baddeley. The episodic buffer: a new component of working memory? *Trends in Cognitive Sciences*, 4(11):417–423, 1999.

E. B. Baum. *What is thought*. MIT Press, Cambridge, Massachusetts, 2004.

H. Bay, T. Tuytelaars, and L. Van Gool. Surf: speeded up robust features. *Computer Vision and Image Understanding (CVIU)*, 110(3):346–359, 2008.

R. Bellman. *Dynamic programming*. Princeton, NJ: Princeton University Press, 1957.

Y. Bengio, J. Louradour, R. Collobert, and J. Weston. Curriculum learning. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 41–48, 2009.

S. S. Benson. *Learning action models for reactive autonomous agents*. PhD thesis, Stanford University, 1996.

D. Bertsekas and J. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, Belmont, MA, 1996.

M. Bilgic, L. Mihalkova, and L. Getoor. Active learning for networked data. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2010.

H. Blockeel and L. de Raedt. Top-down induction of first order local decision trees. *Artificial Intelligence Journal*, 101:185–297, 1998.

H. Blockeel, L. de Raedt, and J. Ramon. Top-down induction of clustering trees. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 55–63, 1998.

A. Blum and J. Langford. Probabilistic planning in the graphplan framework. In *Proc. of the Fifth European Conference on Planning (ECP)*, pages 319–332, 1999.

M. M. Botvinick and J. An. Goal-directed decision making in prefrontal cortex: a computational framework. In *Proc. of the Conf. on Neural Information Processing Systems (NIPS)*, pages 169–176, 2009.

C. Boutilier, T. Dean, and S. Hanks. Decision-theoretic planning: structural assumptions and computational leverage. *Journal of Artificial Intelligence Research (JAIR)*, 11:1–94, 1999.

C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order MDPs. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 690–700, 2001.

R. J. Brachman and H. J. Levesque. *Knowledge Representation and Reasoning*. Morgan Kaufmann, 2004.

R. I. Brafman and M. Tennenholtz. R-max - a general polynomial time algorithm for near-optimal reinforcement learning. *Journal of Machine Learning Research (JMLR)*, 3: 213–231, 2002.

M. Briers, A. Doucet, and S. Maskell. Smoothing algorithms for state-space models. *To appear in Annals of the Institute of Statistical Mathematics*, 2009.

R. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.

O. Buffet and D. Aberdeen. The factored policy-gradient planner. *Artificial Intelligence Journal*, 173(5-6):722–747, 2009.

T. Bylander. The computational complexity of propositional STRIPS planning. *Artificial Intelligence Journal*, 69(1-2):165–204, 1994.

H. Christensen. From internet to robotics – a roadmap for US robotics, May 2009. `http://www.us-robotics.us/reports/CCC%20Report.pdf`.

A. Cocora, K. Kersting, C. Plagemann, W. Burgard, and L. de Raedt. Learning relational navigation policies. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, Beijing, China, 2006.

D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research (JAIR)*, 4(1):129–145, 1996.

G. Cooper. A method for using belief networks as influence diagrams. In *Proc. of the Fourth Workshop on Uncertainty in Artificial Intelligence*, pages 55–63, 1988.

T. Croonenborghs, J. Ramon, H. Blockeel, and M. Bruynooghe. Online learning and exploiting relational models in reinforcement learning. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 726–731, 2007.

J. Cussens. Using prior probabilities and density estimation for relational classification. In *Proc. of the Int. Conf. on Inductive Logic Programming (ILP)*, pages 106–115, 1998.

J. Cussens. Loglinear models for first-order probabilistic reasoning. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 126–133, 1999.

L. de Raedt. Logical settings for concept-learning. *Artificial Intelligence Journal*, 95(1): 187–201, 1997.

L. de Raedt. *Logical and Relational Learning*. Springer, 2008.

L. de Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors. *Probabilistic Inductive Logic Programming*, volume 4911 of *Lecture Notes in Computer Science*. Springer, 2008.

R. Dearden and C. Boutilier. Abstraction and approximate decision-theoretic planning. *Artificial Intelligence Journal*, 89(1-2):219–283, 1997.

A. Dempster, N. Laird, and D. Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

F. Dignum and R. Conte. Goal-generation and agent autonomy. *Autonomos Agents Language, Theory and Architecture*, 1997.

C. Diuk, A. Cohen, and M. Littman. An object-oriented representation for efficient reinforcement learning. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2008.

C. Domshlak and J. Hoffmann. Probabilistic planning via heuristic forward search and weighted model counting. *Journal of Artificial Intelligence Research (JAIR)*, 30:565–620, 2007.

K. Doya, S. Ishii, A. Pouget, and R. Rao, editors. *Bayesian Brain: Probabilistic Approaches to Neural Coding*. MIT Press, 2007.

K. Driessens and S. Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning Journal*, 57(3):271–304, 2004.

K. Driessens, J. Ramon, and T. Gärtner. Graph kernels and Gaussian processes for relational reinforcement learning. *Machine Learning Journal*, 64(1-3):91–119, 2006.

S. Džeroski, L. de Raedt, and K. Driessens. Relational reinforcement learning. *Machine Learning Journal*, 43:7–52, 2001.

A. Epshteyn, A. Vogel, and G. DeJong. Active reinforcement learning. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 296–303, 2008.

V. Fedorov. *Theory of optimal experiments*. New York, 1972.

A. Fern, S. Yoon, and R. Givan. Approximate policy iteration with a policy language bias: solving relational Markov decision processes. *Journal of Artificial Intelligence Research (JAIR)*, 25(1):75–118, 2006.

R. Fikes and N. Nilsson. STRIPS: a new approach to the application of theorem proving to problem solving. *Artificial Intelligence Journal*, 2:189–208, 1971.

N. H. Gardiol and L. P. Kaelbling. Envelope-based planning in relational MDPs. In *Proc. of the Conf. on Neural Information Processing Systems (NIPS)*, 2003.

N. H. Gardiol and L. P. Kaelbling. Action-space partitioning for planning. In *Proc. of the Nat. Conf. on Artificial Intelligence (AAAI)*, pages 980–986, 2007.

N. H. Gardiol and L. P. Kaelbling. Adaptive envelope MDPs for relational equivalence-based planning. Technical Report MIT-CSAIL-TR-2008-050, MIT CS & AI Lab, Cambridge, MA, USA, 2008.

S. Gelly and D. Silver. Combining online and offline knowledge in UCT. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 273–280, 2007.

L. Getoor and B. Taskar, editors. *Introduction to statistical relational learning*. MIT Press, 2007.

L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Džeroski and N. Lavrac, editors, *Relational Data Mining*. Springer-Verlag, 2001.

M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - the planning domain definition language. Technical report, Yale University, 1998.

Y. Gil. Learning by experimentation: Incremental refinement of incomplete planning domains. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 87–95, 1994.

N. Gray, C. Istvan, J. Latshaw, R. Avery, and M. Krupanski. After the eulogy. Victory records, 2000.

C. Gretton and S. Thiébaux. Exploiting first-order rgeression in inductive policy selection. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 217–225, 2004.

R. Grush. Conscious thought as simulation of behaviour and perception. *Behaviorial and brain sciences*, 27:377–442, 2004.

C. Guestrin, R. Patrascu, and D. Schuurmans. Algorithm-directed exploration for model-based reinforcement learning in factored MDPs. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 235–242, 2002.

C. Guestrin, D. Koller, R. Parr, and S. Venkataraman. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 19:399–468, 2003.

F. Halbritter and P. Geibel. Learning models of relational MDPs using graph kernels. In *Proc. of the Mexican Conf. on AI (MICAI)*, pages 409–419, 2007.

M. Helmert. Complexity for standard benchmark domains in planning. *Artificial Intelligence Journal*, 143:219–262, 2003.

G. Hesslow. Conscious thought as simulation of behaviour and perception. *Trends in Cognitive Science*, 6(6):242–247, 2002.

M. Hoffman, H. Kueck, A. Doucet, and N. de Freitas. New inference strategies for solving Markov decision processes using reversible jump MCMC. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2009.

J. Hoffmann and B. Nebel. The FF planning system: fast plan generation through heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 14:253–302, 2001.

S. Hölldobler, E. Karabaev, and O. Skvortsova. FluCaP: a heuristic search planner for first-order MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 27:419–439, 2006.

R. Howard. *Dynamic programming and Markov processes*. MIT Press, Cambridge, Massachusetts, 1960.

IPPC. Sixth International Planning Competition, Uncertainty Part, 2008. URL `http://ippc-2008.loria.fr/wiki/index.php/Main_Page`.

M. Jaeger. Relational Bayesian networks. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 266–273, 1997.

F. Jensen. *An introduction to Bayesian networks*. Springer Verlag, New York, 1996.

M. I. Jordan, editor. *Learning in graphical models*. MIT Press, Cambridge, MA, USA, 1999.

S. Joshi, K. Kersting, and R. Khardon. Generalized first-order decision diagrams for first-order MDPs. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 1916–1921, 2009.

S. Joshi, K. Kersting, and R. Khardon. Self-taught decision theoretic planning with first order decision diagrams. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, 2010.

L. P. Kaelbling, M. Littman, and A. Moore. Reinforcement learning: a survey. *Journal of Artificial Intelligence Research (JAIR)*, 4:237–285, 1996.

L. P. Kaelbling, T. Oates, N. H. Gardiol, and S. Finney. Learning in worlds with objects, 2001. Working Notes of the AAAI Stanford Spring Symposium on Learning Grounded Representations.

E. Karabaev and O. Skvortsova. A heuristic search algorithm for solving first-order MDPs. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 292–299, 2005.

D. Katz and O. Brock. Extracting planar kinematic models using interactive perception. In *In Unifying Perspectives In Computational and Robot Vision*, volume 8 of *Lecture Notes in Electrical Engineering*, pages 11–23. Springer Verlag, May 2008.

D. Katz and O. Brock. A factorization approach to manipulation in unstructured environments. In *14th International Symposium of Robotics Research*, pages 1–16. Springer Verlag, 2009.

M. Kearns and D. Koller. Efficient reinforcement learning in factored MDPs. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 740–747, 1999.

M. Kearns and S. Singh. Near-optimal reinforcement learning in polynomial time. *Machine Learning Journal*, 49(2-3):209–232, 2002.

M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large Markov decision processes. *Machine Learning Journal*, 49(2-3):193–208, 2002.

C. Kemp, N. D. Goodman, and J. B. Tenenbaum. Learning and using relational theories. In *Proc. of the Conf. on Neural Information Processing Systems (NIPS)*, 2008.

K. Kersting and L. de Raedt. Basic principles of learning Bayesian logic programs. In L. de Raedt, P. Frasconi, K. Kersting, and S. Muggleton, editors, *Probabilistic Inductive Logic Programming: Theory and Applications*, volume 4911 of *LNAI*, pages 193–226. Springer, 2008.

K. Kersting and K. Driessens. Non–parametric policy gradients: A unified treatment of propositional and relational domains. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2008.

K. Kersting, M. van Otterlo, and L. de Raedt. Bellman goes relational. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 465–472, 2004.

R. Kindermann and J. L. Snell. *Markov random fields and their applications*. American Mathematical Society, 1980.

L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. In *Proc. of the European Conf. on Machine Learning (ECML)*, 2006.

W. Köhler. *Intelligenzprüfungen an Menschenaffen*. Springer, Berlin (3rd edition, 1973), 1917. English version: Wolgang Köhler (1925): *The Mentality of Apes.* Harcourt & Brace, New York.

S. Kok and P. Domingos. Learning Markov logic network structure via hypergraph lifting. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2009.

S. Kok and P. Domingos. Learning Markov logic networks using structural motifs. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2010.

N. Kushmerick, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence Journal*, 78(1-2):239–286, 1995.

U. Kuter, D. S. Nau, E. Reisner, and R. P. Goldman. Using classical planners to solve nondeterministic planning problems. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 190–197, 2008.

T. Lang and M. Toussaint. Relevance grounding for planning in relational domains. In *Proc. of the European Conf. on Machine Learning (ECML)*, 2009a.

T. Lang and M. Toussaint. Approximate inference for planning in stochastic relational worlds. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2009b.

T. Lang and M. Toussaint. Probabilistic backward and forward reasoning in stochastic relational worlds. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2010a.

T. Lang and M. Toussaint. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research (JAIR)*, 39:1–49, 2010b.

T. Lang, M. Toussaint, and K. Kersting. Exploration in relational worlds. In *Proc. of the European Conf. on Machine Learning (ECML)*, 2010.

T. Lang, M. Toussaint, and K. Kersting. Relational exploration. Submitted, 2011.

I. Little and S. Thiébaux. Probabilistic planning vs replanning. In *ICAPS-Workshop International Planning Competition: Past, Present and Future*, 2007.

M. Littman. Probabilistic propositional planning: Representations and complexity. In *Proc. of the Nat. Conf. on Artificial Intelligence (AAAI)*, pages 748–754, 1997.

M. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research (JAIR)*, 9:1–36, 1997.

M. Lungarella, G. Metta, R. Pfeifer, and G. Sandini. Developmental robotics: a survey. *Connection Science*, 15(4):151–190, 2003.

B. Massey. *Directions in planning: understanding the flow of time in planning*. PhD thesis, University of Oregon, 1999.

J. McCarthy. Situations, actions and causal laws. Technical report, Stanford University, Palo Alto, CA, USA, 1963.

L. Mihalkova and R. Mooney. Bottom-up learning of Markov logic network structure. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2007.

B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov. Blog: Probabilistic models with unknown objects. In *Proc. of the Int. Conf. on Artificial Intelligence (IJCAI)*, pages 1352–1359, 2005.

G. Miller. The magical number seven plus or minus two: some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1956.

S. Muggleton. Stochastic logic programs. In L. de Raedt, editor, *Advances in Inductive Logic Programming*, pages 254–264. IOS Press, 1996.

S. Muggleton. Learning from positive data. In *Selected Papers from the 6th International Workshop on Inductive Logic Programming*, pages 358–376, London, UK, 1997. Springer-Verlag.

K. P. Murphy. *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, UC Berkeley, 2002.

K. P. Murphy and Y. Weiss. The factored frontier algorithm for approximate inference in DBNs. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 378–385, 2001.

S. Natarajan, T. Khot, K. Kersting, B. Gutmann, and J. Shavlik. Boosting relational dependency networks. In *Proc. of the Int. Conf. on Inductive Logic Programming (ILP)*, 2010.

D. S. Nau. Current trends in automated planning. *AI Magazine*, 28(4):43–58, 2007.

J. Neville and D. Jensen. Relational dependency networks. *Journal of Machine Learning Research (JMLR)*, 8:653–692, 2007.

J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-03)*, 2003.

A. Newell and H. A. Simon. Computer science as empirical inquiry: symbols and search. *Commun. ACM*, 19(3):113–126, 1976.

L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, 171:147–177, 1997.

S.-H. Nienhuys-Cheng and R. de Wolf, editors. *Foundations of inductive logic programming*, volume 1228 of *Lecture Notes in Computer Science*. Springer, 1997.

Y. Niv, J. D., and D. P. A normative perspective on motivation. *Trends in Cognitive Sciences (TICS)*, 10:375–381, 2006.

T. Ott and R. Stoop. The neurodynamics of belief propagation on binary markov random fields. In *Proc. of the Conf. on Neural Information Processing Systems (NIPS)*, pages 1057–1064, 2007.

P. Oudeyer, F. Kaplan, and V. Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE Transactions on Evolutionary Computation*, 11(2):265–286, 2007.

H. M. Pasula, L. S. Zettlemoyer, and L. P. Kaelbling. Learning symbolic models of stochastic domains. *Journal of Artificial Intelligence Research (JAIR)*, 29:309–352, 2007.

J. Pearl. *Probabilistic reasoning in intelligent systems*. Morgan Kaufmann, San Mateo, CA, 1988.

E. P. D. Pednault. ADL: Exploring the middle ground between STRIPS and the situation calculus. In *KR*, pages 324–332, 1989.

J. Peters and S. Schaal. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 4:682–697, 2008.

A. Pfeffer. *Probabilistic reasoning for complex systems*. PhD thesis, Computer Science Department, Stanford University, USA, December 2000.

H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proc. of the Nat. Conf. on Artificial Intelligence (AAAI)*, 2007.

P. Poupart, N. Vlassis, J. Hoey, and K. Regan. An analytic solution to discrete Bayesian reinforcement learning. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 697–704, 2006.

M. L. Puterman. *Markov decision processes*. Wiley, New York, 1994.

L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 31, pages 257–286, 1989.

J. Ramon. *Clustering and instance based learning in first order logic*. PhD thesis, Department of Computer Science, K.U.Leuven, Leuven, Belgium, 2002.

J. Ramon, K. Driessens, and T. Croonenborghs. Transfer learning in reinforcement learning problems through partial policy recycling. In *Proc. of the European Conf. on Machine Learning (ECML)*, pages 699–707, 2007.

M. Richardson and P. Domingos. Markov logic networks. *Machine Learning Journal*, 62: 107–136, 2006.

J. Rintanen. Regression for classical and nondeterministic planning. In *Proc. of the European Conf. on Artificial Intelligence (ECAI)*, pages 568–572, 2008.

D. S. Ruchkin, J. Grafman, K. Cameron, and R. S. Berndt. Working memory retention systems: a state of activated long-term memory. *Behavioral and Brain Sciences*, 26:709–777, 2003.

S. Russell and P. Norvig. *Artificial intelligence: a modern approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

S. Sanner. Simultaneous learning of structure and value in relational reinforcement learning. In *Proc. of the ICML-05 Workshop on "Rich Representations for Relational Reinforcement Learning"*, 2005.

S. Sanner. Online feature discovery in relational reinforcement learning. In *Proc. of the ICML-06 Workshop on "Open Problems in Statistical Relational Learning"*, 2006.

S. Sanner and C. Boutilier. Approximate solution techniques for factored first-order MDPs. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 288–295, 2007.

S. Sanner and C. Boutilier. Practical solution techniques for first-order MDPs. *Artificial Intelligence Journal*, 173(5-6):748–788, 2009.

J. Schmidhuber. Curious model-building control systems. In *Proc. of Int. Joint Conf. on Neural Networks*, volume 2, pages 1458–1463, 1991a.

J. Schmidhuber. Adaptive confidence and adaptive curiosity. Technical Report FKI-149-91, Technical University Munich, 1991b.

J. Slaney and S. Thiébaux. Blocks world revisited. *Artificial Intelligence Journal*, 125(1-2): 119–153, 2001.

V. Solo. Smoothing estimation of stochastic processes: Two-filter formulas. *IEEE Transactions on Automatic Control*, 27(2):473–476, 1982.

C. Stachniss, G. Grisetti, and W. Burgard. Information gain-based exploration using rao-blackwellized particle filters. In *Proc. of the Int. Conf. on Robotics: Science and Systems*, pages 65–72, 2005.

J. Sturm, C. Plagemann, and W. Burgard. Body schema learning for robotic manipulators from visual self-perception. *Special Issue of the Journal of Physiology*, 2009.

R. S. Sutton and A. G. Barto. *Reinforcement learning: an introduction*. The MIT Press, March 1998.

I. Szita and A. Lörincz. The many faces of optimism: a unifying approach. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2008.

B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 485–492, 2002.

F. Teichteil-Konigsbuch, U. Kuter, and G. Infantes. Aggregation for generating policies in MDPs. In *Proc. of Int. Conf. on Autonomous Agents and Multiagent Systems*, 2010.

J. B. Tenenbaum, C. Kemp, T. L. Griffiths, and N. D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011.

S. Thrun. The role of exploration in learning control. In *Handbook for Intelligent Control: Neural, Fuzzy and Adaptive Approaches*. 1992.

M. Tomasello, M. Carpenter, J. Call, T. Behne, and H. Moll. Understanding and sharing intentions: The origins of cultural cognition. *Behavioral and Brain Sciences*, 28(5):675–691, 2005.

M. Toussaint. Robot trajectory optimization using approximate inference. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 2009a.

M. Toussaint. Probabilistic inference as a model of planned behavior. *Künstliche Intelligenz (German Artificial Intelligence Journal)*, 3, 2009b.

M. Toussaint. A Bayesian view on motor control and planning. In O. Sigaud and J. Peters, editors, *From motor to interaction learning in robots*. Springer, 2010.

M. Toussaint and C. Goerick. Probabilistic inference for structured planning in robotics. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, pages 3068–3073, 2007.

M. Toussaint and A. Storkey. Probabilistic inference for solving discrete and continuous state Markov decision processes. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, pages 945–952, 2006.

M. Toussaint, L. Charlin, and P. Poupart. Hierarchical POMDP controller optimization by likelihood maximization. In *Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, pages 562–570, 2008.

M. Toussaint, N. Plath, T. Lang, and N. Jetchev. Integrated motor control, planning, grasping and high-level reasoning in a blocks world using probabilistic inference. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2010.

M. van Otterlo. *The logic of adaptive behavior*. IOS Press, Amsterdam, 2009.

T. J. Walsh. *Efficient learning of relational models for sequential decision making*. PhD thesis, Rutgers, The State University of New Jersey, New Brunswick, NJ, 2010.

C. Wang, S. Joshi, and R. Khardon. First order decision diagrams for relational MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 31:431–472, 2008.

X. Wang. Learning by observation and practice: An incremental approach for planning operator acquisition. In *Proc. of the Int. Conf. on Machine Learning (ICML)*, 1995.

C. Watkins and P. Dayan. Q-learning. *Machine Learning Journal*, (8):279–292, 1992.

D. S. Weld. Recent advances in AI planning. *Artificial Intelligence Journal*, 20(2):93–123, 1999.

J. Weng. Developmental robotics: theory and experiments. *International Journal of Humanoid Robotics*, 1, 2004.

J. Weng, J. McClelland, A. Pentland, O. Sporns, I. Stockman, M. Sur, and E. Thelen. Artificial intelligence: autonomous mental development by robots and animals. *Science*, 291(5504):599, 2001.

P. Werbos. Advanced forecasting methods for global crisis warning and models of intelligence. *General Systems Yearbook*, 22:25–38, 1977.

D. Windridge and J. Kittler. Perception-action learning as an epistemologically-consistent model for self-updating cognitive representation. *Brain Inspired Cognitive Systems (special issue), Advances in Experimental Medicine and Biology*, 657, 2010.

J.-H. Wu, R. Kalyanam, and R. Givan. Stochastic enforced hill-climbing. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 396–403, 2008.

Z. Xu, K. Kersting, and T. Joachims. Fast active exploration for link–based preference learning using Gaussian processes. In *Proc. of the European Conf. on Machine Learning (ECML)*, 2010.

S. W. Yoon, A. Fern, and R. Givan. FF-Replan: a baseline for probabilistic planning. In *Proc. of the Int. Conf. on Automated Planning and Scheduling (ICAPS)*, pages 352–359, 2007.

S. W. Yoon, A. Fern, R. Givan, and S. Kambhampati. Probabilistic planning via determinization in hindsight. In *Proc. of the Nat. Conf. on Artificial Intelligence (AAAI)*, pages 1010–1016, 2008.

H. L. Younes and M. L. Littman. PPDDL 1.0: An extension to PDDL for expressing planning domains with probabilistic effects. Technical report, Carnegie Mellon University, 2004.

Zusammenfassung

Planen und Erkundung in Stochastischen Relationalen Welten

Zielgerichtetes Verhalten ist ein eindrucksvolles Zeugnis der Intelligenz von Menschen und Tieren. Diese Doktorarbeit untersucht formale Prinzipien für solches Verhalten in so genannten stochastischen relationalen Welten, welche sich durch zwei Haupteigenschaften auszeichnen: sie enthalten eine Vielzahl an Gegenständen, deren Eigenschaften und gegenseitige Beziehungen verändert werden können; und die Wirkung von Aktionen kann in ihnen nur mit Unsicherheit abgeschätzt werden. Viele natürliche Haushalts- und Arbeitsumgebungen fallen unter diese Kategorie. Wir verfolgen die Idee der Statistischen Relationalen Künstlichen Intelligenz und verknüpfen aussagekräftige relationale Wissensrepräsentationen mit einem probabilistischen Ansatz. Dadurch können wir Wissen über die Wirkungsweise von Aktionen in kompakten Modellen repräsentieren und aus wenigen Erfahrungen so verallgemeinernd erlernen, dass es sich auch auf bisher noch unerforschte Gegenstände erstreckt. Wir führen verschiedene Methoden zur Planung von Aktionsketten in konkreten relationalen Welten ein, die solche Modelle auf gegebene Gegenstände anwenden. Unsere Ansätze fußen großenteils auf dem Informationsverarbeitungsprinzip der probabilistischen Inferenz in graphischen Modellen. Wir entwerfen eine Theorie, die aufzeigt, wie allein relevante Objekte beim Planen berücksichtigt werden können. Wir erweitern existierende Theorien zur Erkundung grundlegend im Hinblick auf relationale Repräsentationen. Dies führt zu einer neuartigen Form explorativen Verhaltens, in der bewusst Gegenstände erkundet werden, auf die bisher erlerntes Wissen nicht verallgemeinert werden kann. Die Kombination unserer neuen Methoden mit existierenden Techniken ermöglicht den Entwurf künstlicher autonomer Agenten, die zielgerichtetes Verhalten in stochastischen relationalen Welten beherrschen.

Vita
Tobias Johannes Lang

CV not available in the online version due to privacy reasons.

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Dissertation selbstständig auf der Grundlage der in der Arbeit angegebenen Hilfsmittel und Hilfen verfasst habe.

Tobias Johannes Lang
Berlin, den 14. Februar 2011