KU LEUVEN

ARENBERG DOCTORAL SCHOOL
Faculty of Engineering Science

Hybrid Probabilistic Logic Programming

Davide Nitti

Supervisors:
Prof. dr. Luc De Raedt
Prof. dr. ir. Tinne De Laet

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science: Computer Science

August 2016

Hybrid Probabilistic Logic Programming

Davide NITTI

Examination committee:
Prof. dr. ir. Jean Berlamont, chair
Prof. dr. Luc De Raedt, supervisor
Prof. dr. ir. Tinne De Laet, supervisor
Prof. dr. ir. Hendrik Blockeel
Prof. dr. ir. Herman Bruyninckx
dr. Vaishak Belle
Prof. dr. Gerhard Lakemeyer
(Aachen University of Technology)
Prof. dr. Scott Sanner
(University of Toronto)

Dissertation presented in partial fulfillment of the requirements for the degree of Doctor in Engineering Science: Computer Science

August 2016

Abstract

An important issue in artificial intelligence and many other fields is modeling the domain of interest. Given a model it is possible to perform inference to answer questions of interest, or make decisions to maximize a given utility. An active research topic concerns declarative languages for modeling and learning a wide range of applications. In particular, probabilistic logic programming combines first-order-logic with probability theory to model uncertainty. However, the majority of such languages do not support continuous random variables, or their support for continuous variables is limited. In this thesis we address this issue extending probabilistic logic programming techniques to deal with hybrid relational domains, involving both discrete and continuous random variables. We first propose a new inference algorithm for the language of Distributional Clauses, that supports zero-probability evidence, including algebraic constraints for which most frameworks fail. Secondly, we extend the algorithm for filtering in temporal domains. Finally, we propose a planner to solve Markov Decision Processes described with Distributional Clauses. The proposed algorithms are tested in several synthetic and real-world problems showing that they are competitive with respect to the state of the art. In particular, we showed how the framework can be used to exploit relational and continuous information jointly to improve state estimation in robotics and vision applications.

Beknopte samenvatting

Een centraal probleem binnen artficiële intelligentie en vele andere onderzoeksvelden is de modelering van het beschouwde domein. Als het model gegeven is kan men met inferentie vragen beantwoorden of beslissingen maken die de opbrengst maximaliseren. Er is heel wat recent en actief onderzoek naar declaratieve talen voor het modeleren en leren binnen een breed scala van toepassingen. Probabilistisch logisch programmeren combineert eerste-orde logica met waarschijnlijkheidstheorie voor het modeleren van onzekerheid. De meerderheid van de bestaande talen ondersteunt geen continue variabelen of hun ondersteuning voor continue variabelen is erg beperkt. Dit doctoraat breidt probabilistisch logisch programmeren uit zodat het geschikt is voor hybride relationele domeinen die zowel discrete als continue variabelen beschouwen. Eerst en vooral introduceert dit doctoraat een inferentiealgoritme voor de taal van Distributional Clauses dat zero-probability waarnemingen (zero-probability evidence) ondersteunt. Ten tweede breidt dit doctoraat het algoritme uit voor filtering in dynamische domeinen die evolueren over de tijd. Ten slotte stelt dit doctoraat een planner voor om Markov Decision Processes op te lossen met Distributional Clauses. De ontwikkelde algoritmes zijn getest in verschillende artificiële en echte problemen. De testen tonen aan dat de algoritmes competitief zijn met de meest recente alternatieven. In het bijzonder toont het doctoraat aan dat het algoritme correct inferentie uitvoert met zero-probability waarnemingen met inbegrip van algebraïsche beperkingen waarvoor de meeste bestaande alternatieven falen. Bovendien toont het doctoraat hoe het ontwikkelde raamwerk bruikbaar is om relationele en continue informatie gezamenlijk te gebruiken voor toestandsschatting in toepassingen uit de robotica en visie.

Acknowledgments

Pursuing this Ph.D. has been a challenging but rewarding journey. This goal would have not be possible without the help and support of many people. I would like to thank all of them here for their contribution.

First of all, I would like to thank my supervisors Luc De Raedt and Tinne De Laet. They encouraged and supported me through these years. They helped me to think in a scientific way and to look in the right direction. At the same time they gave me the freedom to explore and find my own answers.

Secondly, I want to kindly thank all the members of my jury, Herman Bruyninckx, Hendrik Blockeel, Vaishak Belle, Scott Sanner, and Gerhard Lakemeyer for reading this text, and for the useful comments and remarks which helped me to improve it. I would also like to thank Prof. Jean Berlamont for kindly chairing my defence.

In these years, I collaborated and discussed with many people that together with my supervisors where important to write papers on which this thesis is based, and so I would want to thank here all of my co-authors and people that provided me valuable feedback throughout my Ph.D. studies: Luc, Tinne, Vaishak, Maurice, Bogdan, Mathias, Bernd, Ingo, Irma, Guy, Angelika, Jesse, Laura, McElory, and Plinio. Next, I would want to thank all the people with whom I had the pleasure of sharing an office during my time at KU Leuven: Bogdan, Ingo, Guy, Laura, Mathias, Francesco, and Behrouz, for many interesting discussions and the exchange of research ideas. In particular, I would like to thank Bogdan with whom I shared the passion for tea, and Francesco and Behrouz for the exciting discussions and the countless coffee breaks.

During these years in Leuven, I was happy to meet friends from all over the world and spend enjoyable days. I am particularly grateful to Buddha for the great time spent in PhD parties and other activities, Giuseppe for the nice coffee breaks, Saida for her support and many others.

I gratefully acknowledge the financial support received for the work performed during this Ph.D. thesis from the IWT (agentschap voor Innovatie door Wetenschap en Technologie), and by the First-MM project (Flexible Skill Acquisition and Intuitive Robot Tasking for Mobile Manipulation in the Real World) funded by the European Community's 7th Framework Programme, grant agreement First-MM-248258.

Finally, I would like to thank my parents for the unconditioned support in all these years, and for encouraging me to follow my passions and my dreams.

Davide Nitti
Leuven, August 2016

Abbreviations

AI	artificial intelligence
DBN	Dynamic Bayesian Network
DC	distributional clauses
DCPF	distributional clauses particle filter
DDC	dynamic distributional clauses
FOL	First-Order Logic
HMM	Hidden Markov Model
i.i.d.	independent and identically distributed
iff	if and only if
IOHMM	Input Output Hidden Markov Model
LW	likelihood weighting
MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
MDP	Markov Decision Process
PF	Particle Filter
STD	standard deviation

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of artificial intelligence (AI) concerns building machines that can solve problems that require some form of 'intelligence'. The first step in developing such machines is to formally represent the domain of interest and the problem to solve.

Instead of writing an algorithm that solves a specific task in a specific domain, it is more valuable to design a language that can represent a wide variety of domains and to build general inference mechanisms that are valid for any well-defined domain written in such language. This paradigm is called **declarative**, because it separates the definition of the problem from the way it will be solved. For this reason an important area of research in AI regards knowledge representation, that concerns how to formally represent information of the domain of interest and how to process it. One of the languages that can be used for knowledge representation is first-order logic (FOL). FOL allows to represent objects, their properties and the relations between them. The key feature of FOL is the ability to represent abstract knowledge with compact formulas. In addition, there are several inference mechanisms that allow to infer new knowledge from the current known information.

A variant of FOL is **logic programming** [Nilsson and Małiszyński, 1995; Apt, 1997; Lloyd, 1987], that expresses knowledge in terms of facts and rules (implication formulas). Logic programming is less expressive than FOL, but it allows to provide optimized inference algorithms and programming languages.

Given the noisy nature of real-world data, it is important to represent the uncertainty of the domain of interest. An active area of research is probabilistic logic programming, that combines probability theory with logic programming. A

related area is **statistical relational learning** (SRL) [Getoor and Taskar, 2007; De Raedt, 2008; De Raedt et al., 2008], which combines logical representations, probabilistic reasoning, and machine learning.

The expressivity of the language and the effectiveness of the inference algorithm are the key features of such probabilistic logic language. State-of-the-art probabilistic languages based on logic programming provide high expressivity and general purpose inference algorithms. Those approaches have been successful in many application areas ranging from natural language processing to bioinformatics. However, many probabilistic logic languages do not support continuous random variables, or their support for such variables is limited.

A more broad area is **probabilistic programming** that combines probability theory with programming paradigms (e.g., procedural or functional). Such languages support continuous random variables, even though they do not follow a declarative and logical perspective. Nonetheless, virtually all state-of-the-art probabilistic languages that do support continuous random variables do not properly handle zero-probability evidence in complex domains such as the Indian GPA problem, initially proposed by Stuart Russell and discussed in [Perov et al.].

A probabilistic logic language that supports both discrete and continuous random variables can be useful in many domains such as robotics and vision. Indeed, despite the progress in those fields, the majority of probabilistic models used, such as Bayesian networks, cannot easily represent relational information, that is, objects, properties, as well as the relations that hold between them. In contrast, logical (relational) representations allow to encode more general models and to integrate abstract background knowledge about the world. In addition, a hybrid relational language can help to bridge the gap between logical high-level reasoning with low-level sensory data processing.

Consider for example a robot that assists humans in an object manipulation task. In particular, assume that a human puts a hammer and a screwdriver inside a small box that is placed inside a bigger box; at some point the human asks to the robot to get the hammer. To solve this task, the robot needs to keep track of the objects and reason about them. For example, to estimate the position of an object when it is occluded, the robot has to assume that an object inside a box remains inside under certain conditions and that its position follows the position of the box. These qualitative rules are also valid recursively for boxes inside other boxes. Such knowledge can help the robot to keep track of the position of the hammer when it is occluded because it is inside the box, and thus get the hammer as required. Another human might enter in the room and ask the robot where is the screwdriver. The robot can provide this information converting its internal relational knowledge $\texttt{screwdriver}(\texttt{id1}), \texttt{inside}(\texttt{id1}, \texttt{id2}), \texttt{inside}(\texttt{id2}, \texttt{id3})$,

`box(id2)`, `box(id3)`, `big(id3)`, `small(id2)` in a natural language sentence, e.g., "the screwdriver is inside the small box that is inside the big box". If we assume that inside is a transitive relation, the robot can infer `inside(id1, id3)`, and thus reply "the screwdriver is inside the big box".

This task is relatively easy if the knowledge is already available in logical format. In practice the task is much harder, because the objects need to be detected by a camera, and their continuous positions are noisy and not always available. To solve the task these continuous information need to be filtered and converted in a relational (logical) knowledge. However, standard approaches use simple probabilistic models during the filtering process. It is much more powerful to integrate continuous and relational information in the same framework. This integration can help estimating the objects position not just using a simple probabilistic model, but exploiting relational high-level knowledge. For example, an object inside a box follows its position, as described before.

With this goal in mind, in this thesis we extend probabilistic logic programming techniques to deal with hybrid relational domains, involving both discrete and continuous random variables in static and dynamic domains. The resulting language and solvers have been used in several domains, including a tracking scenario similar to the one explained above, and simple planning tasks.

1.1 Contributions

This thesis extends probabilistic logic programming techniques to deal with hybrid relational domains for inference, learning and planning tasks.

The **first contribution** is a new inference algorithm for Distributional Clauses (DC) [Gutmann et al., 2011], a recent extension of Sato's distribution semantics [Sato, 1995] for dealing with continuous variables. The new inference algorithm is more efficient and, more importantly, it provides the correct results for some inference problems in which most probabilistic programming languages fail. For example, it is possible to answer queries with zero-probability evidence in non-trivial domains.

The **second contribution** is the extension of the DC framework to cope with time. For the resulting Dynamic Distributional Clauses (DDC), we develop a particle filter (DCPF) that exploits the static inference algorithm for filtering, and integrates online learning algorithms. Particle filters [Doucet et al., 2000b] are widely applied in domains such as probabilistic robotics [Thrun et al., 2005], and we adapt them here for use in hybrid relational domains, in which each state of the environment is represented as an interpretation, that is, a set of ground

facts that defines a possible world. The thesis analyses the conditions required to perform filtering avoiding inference backward in time (backinstantiation). Moreover, we prove theoretical correctness for DCPF and study its relation with Rao-Blackwellized particle filters. DCPF has been applied in several tracking scenarios, which shows its applicability in online real-world settings.

The **third contribution** is applying the DDC language to describe Markov Decision Processes (MDPs) and proposing a new algorithm for planning in a wide range of domains. The basic planner, called HYPE, exploits the model and importance sampling to find a good policy. The extension with abstraction exploits independence assumptions encoded in the model to perform a sample-based abstraction that improves the performance.

1.2 Thesis Roadmap

Chapter 2 presents background information on probability theory, graphical models, Monte Carlo methods, logic programming and planning.

Chapter 3 introduces Distributional Clauses and describes a new inference method based on backward reasoning and importance sampling. The chapter consists of research previously published in the following paper:

- D. Nitti, T. De Laet, L. De Raedt. *Probabilistic logic programming for hybrid relational domains*, in Machine Learning, volume 103, pages 307–449, Springer (2016).

Chapter 4 extends DC for dynamic domains, and provides a particle filter (DCPF) for the resulting language (DDC). DCPF has been applied in several tracking scenarios and syntactic domains. Moreover, standard online learning algorithms have been adapted and integrated in the framework.

The chapter consists of research previously published in the following papers:

- D. Nitti, T. De Laet, L. De Raedt. *Probabilistic logic programming for hybrid relational domains*, in Machine Learning, volume 103, pages 307–449, Springer (2016).
- D. Nitti, T. De Laet, L. De Raedt: *A particle filter for hybrid relational domains.* In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2013), pages 2764–2771 (2013)

- D. Nitti, T. De Laet, L. De Raedt: *Relational object tracking and learning.* In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2014), pages 935–942 (2014)
- D. Nitti, T. De Laet, L. De Raedt: *Distributional clauses particle filter.* In: Machine Learning and Knowledge Discovery in Databases, volume 8726 of Lecture Notes in Computer Science, pages 504–507. Springer Berlin Heidelberg (2014)
- D. Nitti, G. Chliveros, L. De Raedt, M. Pateraki, M. Hourdakis, P. Trahanias: *Application of dynamic distributional clauses for multi-hypothesis initialization in model-based object tracking.* In 9th International Conference on Computer Vision Theory and Applications (VISAPP 2014) volume 2, pages 256–261 (2014)

Chapter 5 extends the DC framework for MDPs and proposes a new planning algorithm based on importance sampling.

The chapter consists of research previously published in the following papers:

- D. Nitti, V. Belle, L. De Raedt: *Planning in discrete and continuous Markov decision processes by probabilistic programming.* In: Proceedings of the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2015), volume 9285 of *Lecture Notes in Computer Science*, pages 327–342. Springer International Publishing (2015). **Best Student Paper - Machine Learning Journal Award.**
- D. Nitti, V. Belle, T. De Laet, L. De Raedt: *Sample-based abstraction for hybrid relational MDPs.* In: European Workshop on Reinforcement Learning (EWRL 2015)
- D. Nitti, V. Belle, T. De Laet, L. De Raedt: *Planning in Hybrid Relational MDPs.* Under review at Machine Learning, Springer

Chapter 6 concludes and discusses directions for future work.

The following papers performed during my PhD research has not been included in this thesis:

- D. Nitti, I. Ravkic, J. Davis, L. de Raedt: *Learning the Structure of Dynamic Hybrid Relational Models.* Accepted at the 22nd European Conference on Artificial Intelligence (ECAI 2016).
- B. Moldovan, P. Moreno, D. Nitti, J. Santos-Victor, L. De Raedt. *Using Relational Affordances for Multiple-Action Two-Arm Manipulation Tasks.* Under review at Robotics and Autonomous Systems Journal

The first paper regards learning the structure of DDC clauses from object manipulation data collected by a robot arm. The goal was to learn the effects (i.e. the state transition model) of several actions performed by a robot arm on cubic objects. The second paper has a goal similar to the first, but it uses a different learner and it is applied to two-arm manipulation tasks.

Chapter 2

Background

This chapter introduces basic concepts used in the thesis. We will first start with probability theory (Section 2.1), probabilistic graphical models such as Bayesian Networks (Section 2.2), and inference methods like Monte Carlo (Section 2.3). We then continue with first-order logic concepts (Section 2.4) and probabilistic languages (Section 2.5). We will end with probabilistic planning concepts (Section 2.6).

2.1 Probability Theory

Probability theory is a well-founded mathematical framework to model uncertainty. We will briefly overview some of the concepts that are used in this thesis. For a more extensive introduction we refer to [Jaynes, 2003; Kadane, 2011].

In probability theory we consider a set of atomic outcomes Ω, called the sample space. For example, $\Omega = \{1, 2, 3, 4, 5, 6\}$ is the sample space of rolling a die. An event is a subset of Ω, e.g., $e = \{1, 3, 5\}$ is the event of obtaining an odd number after rolling a die. We indicate with $\mathcal{F} \subset 2^{\Omega}$ a set of subset of outcomes. A probability function associates a number in $[0, 1]$ to each event, informally this number indicates how likely the event is to happen. For example, if we assume equally probable events, $P(\{1\}) = 1/6$, and $P(\{1, 2, 3\}) = 1/2$.

Definition 2.1. *A measurable space is a pair $(\Omega, \mathcal{F})$, where Ω is a nonempty set and $\mathcal{F} \subset 2^{\Omega}$ is a σ-algebra, i.e. a (nonempty) set of subsets of Ω such that $\Omega \in \mathcal{F}$, and every set A in $\mathcal{F}$ has its complement in $\mathcal{F}$:*

$A \in \mathcal{F} \Rightarrow (\Omega \setminus A) \in \mathcal{F}$, and the union of any countable sequence of events $A_i \in \mathcal{F}$ is in $\mathcal{F}$: $(\forall i A_i \in \mathcal{F}) \Rightarrow \bigcup_i A_i \in \mathcal{F}$.

Definition 2.2. *A probability space is a triple $(\Omega, \mathcal{F}, P)$, where $(\Omega, \mathcal{F})$ is a measurable space, and $P : \mathcal{F} \to [0, 1]$ is a function that assigns to each event a real number in $[0, 1]$, called probability. A probability space requires that:*

- *$P : \mathcal{F} \to \mathbb{R}$ is a measure on $\mathcal{F}$, i.e. $\forall A \in \mathcal{F} : P(A) \geq P(\emptyset) = 0$, and $P(\bigcup_i E_i) = \sum_i P(E_i)$ for any countable sequence of disjoint events $E_i \in \mathcal{F}$.*
- *$P(\Omega) = 1$*

Definition 2.3. *Let $(\Omega, \mathcal{F})$ and (S, Σ) be measurable spaces, a random variable X is a measurable function $X : \Omega \to S$, that is a function such that $\forall A \in \Sigma : X^{-1}(A) \in \mathcal{F}$, where $X^{-1}(A)$ is the preimage of A under X.*

Informally, a random variable X is a mapping that associates a value $s \in S$ to each outcome $o \in \Omega$, where S is often the set of real numbers $\mathbb{R}$. We denote with $P(X = x)$ the probability that random variable X has value x. Abusing notation we use $P(x)$ to define the distribution of X, i.e. a function that returns $P(X = x)$ for a given x.

Example 2.1. *Consider drawing two balls (with replacement) from an urn that contains blue and red balls. The sample space is $\Omega = \{blueblue, bluered, redblue, redred\}$. We can define the random variables $ball_1$ and $ball_2$ that represent the color respectively of the first and second drawn ball. Assuming the same number of red and blue balls we have $P(ball_1 = blue) = P(ball_1 = red) = P(ball_2 = blue) = P(ball_2 = red) = 0.5$.*

Definition 2.4. *The joint distribution $P(X_1 = x_1, ..., X_n = x_n)$ of n random variables X_i is the probability of the conjunction $\bigwedge_i X_i = x_i$.*

For example, $P(X = x, Y = x)$ or simply $P(x, y)$ is the probability that $X = x$ and $Y = y$. From now on, all the concepts will be defined in terms of random variables.

Definition 2.5. *The cumulative distribution function of a random variable X with values in a measurable space $(\mathbb{R}, \mathcal{B})$ is $F_X(x) = P(X \leq x)$.*

Definition 2.6. *The probability density function of a random variable X with values in a measurable space $(\mathbb{R}, \mathcal{B})$ is a function $p_X(x)$ such that $P(X \in A) = \int_A p_X(x)dx$ for any measurable set $A \in \mathcal{B}$.*

Informally, we can define probability density function as $p_X(x) = \frac{d}{dx}F_X(x)$. It is easy to show that $P(X \in [a,b]) = F_X(b) - F_X(a) = \int_a^b p_X(x)dx$ and $F_X(x) = \int_{-\infty}^{x} p_X(t)dt$.

The cumulative and density functions can be generalized for n random variables. For example, the joint cumulative distribution for random variables X, Y is $F_{XY}(x,y) = P(X \leq x, Y \leq x)$ and the joint density $f_{XY}(x,y)$ is such that $P(X \in A, Y \in B) = \int_A \int_B f_{XY}(x,y)dxdy$.

The density function does not always exist. Consider a random variable X, where $P(X = 1) = 0.6$ and $P(X = 2) = 0.4$. The cumulative $F_X(x)$ is discontinuous in 1 and 2, thus the derivative is not defined in those points. We can solve this issue using the Dirac delta function $\delta(x)$: a generalized function that is zero everywhere except at zero (in which it tends to infinity), moreover $\int_{-\infty}^{\infty} \delta(x) = 1$. Since the unit step function $H(x)$ (i.e. $H(x) = 1$ for $x \geq 0$, and $H(x) = 0$ otherwise) is the cumulative of a random variable X such that $P(X = 0) = 1$ and $H(x) = \int_{-\infty}^{x} \delta(t)dt$, $\delta(x)$ can be considered the density of such random variable. We can generalize the concept and define $\delta_v(x) = \delta(x - v)$ as the density of a random variable X' such that $P(X' = v) = 1$.

Note that any random variable X that has a discrete domain has a density that is a sum of Dirac delta functions.

Example 2.2. *Consider a random variable X, where $P(X = 1) = 0.6$ and $P(X = 2) = 0.4$, the density is $p_X(x) = 0.6\delta_1(x) + 0.4\delta_2(x) = 0.6\delta(x-1) + 0.4\delta(x-2)$.*

In this thesis we will use this notation to describe discrete distributions such as empirical distributions that are obtained from samples.

Imagine that we have a probability distribution P and we already know that an event B is true. We can define a new distribution P', called conditional probability, that reflects this information.

Definition 2.7. *The conditional probability of event A, conditional on event B, is defined if $P(B) > 0$ by: $P(A|B) = P(A \cap B)/P(B)$*

From definitions 2.2 and 2.7 we can derive several properties, such as:

$$P(A \cap B) = P(A|B)P(B), \tag{2.1}$$

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}, \tag{2.2}$$

$$P(A \cup B) = P(A) + P(B) - P(A \cap B). \tag{2.3}$$

Definition 2.7 and the above properties are trivially extended to random variables. For example, $P(X = x|Y = y) = P(X = x \cap Y = y)/P(Y = y)$, where we often call $Y = y$ evidence. Conditional probabilities are properly defined when the event we are conditioning on has a positive probability, i.e. $P(Y = y) > 0$. Unfortunately, when Y is a continuous random variable, $\forall y : P(Y = y) = 0$ in many practical cases, such as Gaussian distributions. Indeed, in robotics and vision it is common to have as evidence a zero-probability event (i.e. $P(Y = y) = 0$). For this reason it is necessarily to define $P(X = x|Y = y)$ when $P(Y = y) = 0$. The conditional distribution has to satisfy the equation $P(X = x \cap Y = y) = P(X = x|Y = y)P(Y = y)$, but this is satisfied for any $P(X = x|Y = y)$ when $P(Y = y) = 0$. Indeed, $P(Y = y) = 0 \Rightarrow P(X = x \cap Y = y) = 0$. This (informally) shows that $P(X = x|Y = y)$ does not have a unique definition when $P(Y = y) = 0$.

In this thesis we define the conditional probability for zero-probability evidence as follows [Kadane, 2011]:

$$P(q|E = v) = \lim_{dv \to 0} \frac{P(q, E \in [v - dv/2, v + dv/2])}{P(E \in [v - dv/2, v + dv/2])}. \tag{2.4}$$

In other words we consider a small interval around v and limit such interval to zero. The cumulative conditional probability is the following (assuming two random variables X and E):

$$\begin{aligned} P(X \leq x|E = v) &= \lim_{dv \to 0} \frac{P(X \leq x, E \in [v - dv/2, v + dv/2])}{P(E \in [v - dv/2, v + dv/2])} \\ &= \lim_{dv \to 0} \frac{\int_{\infty}^{x} p(X = \tilde{x}, E = v) d\tilde{x} dv}{p(E = v) dv} \\ &= \frac{\int_{\infty}^{x} p(X = \tilde{x}, E = v) d\tilde{x}}{p(E = v)}, \end{aligned} \tag{2.5}$$

The conditional density is thus $p(X = x|E = v) = \frac{p(X=x,E=v)}{p(E=v)}$. This definition does not solve all the zero-probability evidence issues. For example, the limit does not always exists.

The ambiguity of conditional distributions for zero-probability evidence cases is known as the Borel-Kolmogorov paradox [Kolmogorov, 1956; Gyenis et al., 2016; Kadane, 2011], where it has been shown that a reformulation of the problem, e.g., a change of variables, can produce a different result. Today this issue is not considered as a paradox in the mathematical sense [Gyenis et al., 2016], but only a counterintuitive ambiguity. Nonetheless, how the ambiguity should be removed in practical problems and how the assumptions made influence the results is an open question. This issue will be important in Chapter 3.

Definition 2.8 (independence)**.** *Two random variables A and B are independent, indicated by $A \perp B$, if and only if*

$$\forall a, b : P(A = a, B = b) = P(A = a)P(B = b)$$

Two random variables A and B are conditionally independent given C, indicated by $A \perp B|C$, if and only if

$$\forall a, b, c : P(A = a, B = b|C = c) = P(A = a|C = c)P(B = b|C = c)$$

From this definition it follows that two independent random variables A and B satisfy $P(A|B) = P(A)$. Similarly, conditional independent variables A and B given C ($A \perp B|C$) satisfy $P(A|B, C) = P(A|C)$.

2.2 Probabilistic Graphical Models

It is often convenient to represent a probability distribution using a graph. This allows to visualize the distribution, its independence properties, and build algorithms based on such structure. We will introduce Bayesian networks [Pearl, 1988], one of the most popular graphical models, and their dynamic extensions. There exist other graphical models, such as Markov random field based on undirected graphs and factor graphs. We will not discuss those further, but we refer to [Koller and Friedman, 2009] for more details.

2.2.1 Bayesian networks

Definition 2.9. *A Bayesian network consists in a directed acyclic graph $BN = (V, E)$, where the vertices $x_i \in V$ represent random variables and the edges $(x_i, x_j) \in E$ encode dependences. The distribution associated with a Bayesian network BN factorizes as follows:*

$$p(x_1, x_2, ..., x_n) = \prod_i p(x_i|PA(x_i)),$$

where $PA(x_i)$ is the set of parents of x_i, that is the set of all variables that have an edge towards x_i.

The function $p(.)$ indicates a probability or a density distribution. Throughout the thesis we will often use $p(.)$ in both cases when the context is clear.

Definition 2.10. *A variable A is an ancestor of B and B is a descentant of A in a Bayesian network, if and only if there is a directed path from A to B. We indicate with $ANC(X)$ the set of ancestors of X, and $DESC(X)$ the set of descentants of X.*

A Bayesian network BN encodes conditional independence assumptions, for example it is easy to show that $p(x_i|PA(x_i), ANC(x_i)) = p(x_i|PA(x_i))$. A Bayesian network graph BN does not define a unique distribution, but only conditional independence assumptions. To define a distribution it is necessarily to specify the distributions $p(x_i|PA(x_i))$. In standard Bayesian networks, the random variables are discrete, thus the distributions can be defined with a table that specifies the probability for each value of $x_i, PA(x_i)$ (conditional probability table).

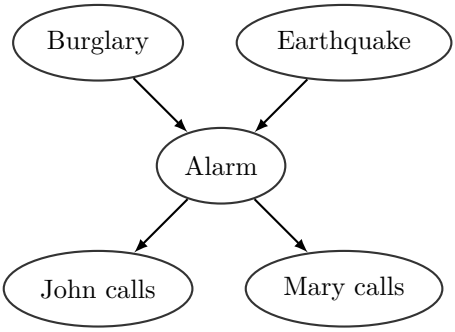


Figure 2.1: Alarm Bayesian Network from J. Pearl.

Example 2.3. *Consider the well-known alarm network proposed by J. Pearl [Pearl, 1988; Russell and Norvig, 2009] of figure 2.1. The joint distribution factorize as follows:*

$$p(Bulglary, Earthquake, Alarm, Johncalls, Marycalls) =$$

$$p(Bulglary)p(Earthquake)p(Alarm|Bulglary, Earthquake)\cdot$$

$$p(Johncalls|Alarm)p(Marycalls|Alarm)$$

An important concept in Bayesian Networks is d-separation, that allows to determine conditional independence between random variables.

Definition 2.11. *An undirected path P is d-separated [Murphy, 2012] by a set of nodes (random variables) C iff at least one of the following conditions hold:*

- *P contains a chain,* $s \to m \to t$ *or* $s \leftarrow m \leftarrow t$*, where* $m \in C$
- *P contains a fork,* $s \leftarrow m \to t$*, where* $m \in C$
- *P contains a collider* $s \to m \leftarrow t$*, where* $m \notin C$ *and* $\forall x \in DESC(m)$: $x \notin C$.

Definition 2.12. *Two sets of random variables A and B are d-separated [Murphy, 2012] given a set C (evidence) if and only if each undirected path from every node* $a \in A$ *to every node* $b \in B$ *is d-separated by C.*

D-separation is equivalent to conditional independence, i.e.:

$$A \perp B|C \Leftrightarrow \text{ A is d-separated from B given C.}$$

2.2.2 Temporal models

Bayesian networks can be extended to model temporal data (e.g., time series). Temporal data can be modeled using stochastic processes, that define a distribution over functions. In particular, discrete-time stochastic processes define the distribution of sequences of random variables, i.e. $p(x_t, x_{t+1}, ..., x_{t+n})$ for any t and n. The most simple discrete-time probabilistic model for temporal data is the Markov chain, where the probability of the next variable depends only on the current variable (Markov assumption):

$$p(x_{t+1}|x_0, x_1, ..., x_t) = p(x_{t+1}|x_t).$$

It is often assumed that $p(x_{t+1}|x_t)$ is the same for each time step t (stationary Markov chain).

In partially observable environments, the state x_t might not be (directly) observable. A Hidden Markov Model (HMM) captures this situation, where the states x_t in the Markov chain are not observed, but z_t variables are observed. A HMM defines the joint distribution $p(x_0, x_1, z_1, ..., z_T, x_T) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t)p(z_{t+1}|x_{t+1})$. As a convention, the state x_0 does not have any observation z_0.

A HMM is described by $p(x_{t+1}|x_t)$ called state transition model, and $p(z_{t+1}|x_{t+1})$ called observation model. Note that a Markov chain and a HMM define a distribution for a sequence of arbitrary length, replicating the state transition and observation models.

In several applications it is convenient to add an input u_t (action) that influences the next states x_{t+1} and the observations z_{t+1}. The resulting model is called

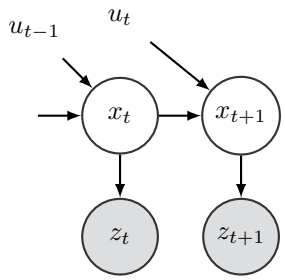


Figure 2.2: IOHMM assuming $p(z_{t+1}|x_{t+1}, u_t) = p(z_{t+1}|x_{t+1})$.

input-output HMM (IOHMM):

$$p(x_0, x_1, z_1, ...z_T, x_T|u_0, ..., u_T) = p(x_0) \prod_{t=0}^{T-1} p(x_{t+1}|x_t, u_t)p(z_{t+1}|x_{t+1}, u_t).$$

Throughout this thesis we assume that: $p(z_{t+1}|x_{t+1}, u_t) = p(z_{t+1}|x_{t+1})$.

2.2.3 Dynamic Bayesian networks

Dynamic Bayesian Networks (DBN) are an extension of HMMs to model dynamic systems. The difference with HMMs is that the states and observations are not single random variables but a set of correlated random variables.

A DBN consists of a Bayesian network that defines the state distribution at time zero: $p(x_0)$, and a two-slice temporal Bayes network (2TBN) that defines the conditional distribution $p(z_{t+1}, x_{t+1}|x_t, u_t) = p(x_{t+1}|x_t, u_t)p(z_{t+1}|x_{t+1}, u_t)$.

For clarity we made an explicit distinction between the hidden state x_t, the evidence or observations z_t, and the action u_t (input) as in an IOHMMs. However, in a general DBN, the variables observed are not necessarily the same in each step, thus the distinction between z_t and x_t is not strictly required.

2.3 Inference

Given a distribution $p(a, b, c, d)$, performing inference consists in computing functions of the distribution. For example, we might be interested in the

marginal probability $p(a = 1)$, or conditional probabilities such as $p(a = 1|b = 0)$. Inference algorithms for graphical models exploit the structure of the model (e.g., independence assumptions) to speed-up the computation. The most common algorithms are belief propagation and the junction tree algorithm [Koller and Friedman, 2009]. However, in complex distributions involving continuous and discrete random variables, exact inference is often intractable. To address this issue several approximate inference algorithms have been proposed, such as Monte-Carlo methods and variational inference. In this thesis we focus on Monte-Carlo methods [Robert and Casella, 2004; Lemieux, 2009].

2.3.1 Naive Monte-Carlo

Monte-Carlo Methods approximate an integral with a weighted sum by means of sampling. In detail, an integral $\int_a^b h(x)dx$ can be reformulated as the expectation of a function $f(x)$ with respect to a distribution $p(x)$ such that $h(x) = f(x)p(x)$:

$$\int_a^b h(x)dx = \int_a^b f(x)p(x)dx = E_{p(x)}[f(x)].$$

Given N independent and identically distributed (i.i.d.) random variables x_i drawn from distribution $p(x)$, we approximate the integral as follows:

$$\int_a^b h(x)dx = \int_a^b f(x)p(x)dx = E_{p(x)}[f(x)] \approx \frac{1}{N}\sum_{i=0}^{N} f(x_i).$$

Probabilistic inference can be formulated as computing such integral for a function $f(x)$ of interest and a given distribution $p(x)$. For example, the probability that a query (i.e. an arbitrary formula) $q(x)$ is true is:

$$p(q) = E_{p(x)}[\mathbb{1}(q(x))] = \int_x \mathbb{1}(q(x))p(x)dx \approx \frac{1}{N}\sum_{i=0}^{N} \mathbb{1}(q(x_i)), \tag{2.6}$$

where $\mathbb{1}$ is the indicator function that returns 1 when the argument is true, and 0 otherwise. Intuitively, the probability of a query is approximated as the ratio of times in which the query is true in the samples. In this thesis we focus on computing the probability of a query, however the discussion is valid for a generic function $f(x)$.

Since the Monte-Carlo estimator of $p(q)$ denoted by $\tilde{p}(q)$ is a sum of random variables, it is itself a random variable. The law of large numbers and the central limit theorem state that, given certain conditions, $\tilde{p}(q)$ converges to $p(q)$ for $N \to \infty$, and its distribution converges to a Gaussian. For a finite

N, the expectation of the estimator is $E[\tilde{p}(q)] = p(q)$, i.e. the estimator is unbiased, and the variance is $Var[\tilde{p}(q)] = Var_{p(x)}[\mathbb{1}(q(x))]/N$, and more generally $Var_{p(x)}[f(x)]/N$ for the estimator of $E_{p(x)}[f(x)]$. Since $\tilde{p}(q)$ is approximately a Gaussian, it is easy to compute confidence intervals.

2.3.2 Importance sampling

It is often not possible (or not convenient) to sample from the distribution $p(x)$ (called target). Importance sampling addresses this issue of estimating $E_{p(x)}[f(x)]$ using samples generated from another distribution $g(x)$ (called proposal):

$$E_{p(x)}[f(x)] = \int_x f(x) \underbrace{\frac{p(x)}{g(x)}}_{w(x)} g(x)dx \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i)w^{(i)} = \tilde{\mu}. \tag{2.7}$$

The importance sampling estimator $\tilde{\mu}$ is unbiased and converges to $E_{p(x)}[f(x)]$ provided that $\forall x : |f(x)|p(x) > 0 \Rightarrow g(x) > 0$ or equivalently: $\forall x : g(x) = 0 \Rightarrow f(x)p(x) = 0$ [Robert and Casella, 2004]. The variance of the importance sampling estimator is $Var[\tilde{\mu}] = Var_{q(x)}[f(x)w(x)]/N$. The optimal proposal distribution is $g(x) \propto |f(x)|p(x)$, i.e. $g(x) = \frac{|f(x)|p(x)}{\int_x |f(x)|p(x)dx}$. Such proposal is optimal in the sense that minimizes the variance of the estimator. If $f(x)p(x)$ has always the same sign, one sample is sufficient to estimate $E_{p(x)}[f(x)]$ exactly, in other words the variance of the estimator is zero. The optimal proposal distribution is generally not known because it requires $\int_x |f(x)|p(x)dx$. However, adaptive importance sampling can be used to learn such distribution using previous samples.

So far we have considered the expectation of a generic function $E_{p(x)}[f(x)]$ and a specific case where $f(x) = \mathbb{1}(q(x))$ with $q(x)$ a query of interest. In many applications we are interested in computing the probability of a query given evidence $p(q|e)$. This can be estimated using formula (2.6) or (2.7) twice, once to estimate $p(q, e)$ and another time to estimate $p(e)$, then $p(q|e)$ is approximated as the ratio of the two quantities. As an alternative, formula (2.6) or (2.7) can be adapted to use the same samples to estimate the two quantities. For importance sampling we have:

$$p(q|e) = \frac{p(q,e)}{p(e)} = \frac{\int_x \mathbb{1}(q(x))\mathbb{1}(e(x))\frac{p(x)}{g(x)}g(x)dx}{\int_x \mathbb{1}(e(x))\frac{p(x)}{g(x)}g(x)dx} \approx \frac{\sum_{i=1}^{N}\mathbb{1}(q(x_i))\mathbb{1}(e(x_i))w^{(i)}}{\sum_{i=1}^{N}\mathbb{1}(e(x_i))w^{(i)}}. \tag{2.8}$$

In this case $g(x)$ has to satisfy the condition: $\mathbb{1}(e(x))p(x) > 0 \Rightarrow g(x) > 0$.

2.3.3 Sampling from a Bayesian Network

The easiest way to sample from a Bayesian Network, is **ancestral sampling** that starts sampling from root nodes (random variables without parents) and then samples children nodes until all the variables are samples. Each random variable is sampled from $p(x_i|PA(x_i))$. In the presence of evidence, it is convenient to reject the sample as soon as it is inconsistent with the evidence. This is called **logic sampling** [Henrion, 1986]. However, this algorithm is still inefficient and it can be improved using **likelihood weighting** (LW) [Fung and Chang, 1989]. LW is a type of importance sampling that forces variables to be consistent with the evidence $E = \{e_1, e_2, ..., e_n\}$ by using the proposal distribution $g(x_0, ..., x_N) = \prod_{x_i \notin E} p(x_i|PA(x_i)) \prod_{x_i \in E} \delta_{e_i}(x_i)$. In other words, non-evidence variables $x_i \in E$ are sampled from a proposal equal to the target distribution $p(x_i|PA(x_i))$, while evidence variables are forced to be consistent with the observed values e_i, i.e. the proposal distribution is a Dirac delta $\delta_{e_i}(x_i)$.

It has been shown that LW reduces the variance of the estimator with respect to the naive Monte-Carlo estimator [Fung and Chang, 1989].

2.3.4 Markov Chain Monte Carlo methods

Markov Chain Monte Carlo (MCMC) [Andrieu et al., 2003] is one of the most popular Monte-Carlo methods in the last decades . It allows to perform approximate inference in high-dimensional and unnormalized distributions.

MCMC exploits Markov chain theory to generate samples from the target distribution $g(x)$. This method is also applicable when $g(x)$ is known only up to a normalization constant: $g(x) \propto f(x)$. The basic idea is to define a transition (jumping or kernel) distribution $p(x_{t+1}|x_t)$ such that the resulting Markov chain has the target $g(x)$ as stationary distribution. Thus, if we sample sequentially from $p(x_{t+1}|x_t)$, under certain conditions, the distribution $p(x_t)$ converges to the target $g(x)$ for $t \to \infty$.

MCMC generates correlated samples and requires a burn-in period to make $p(x_t)$ converge to $g(x)$. The first issue is mitigated using only one sample every k, for the Monte-Carlo estimation. The second issue is handled ignoring the first n samples, with n sufficiently big.

The most popular MCMC methods are Gibbs sampling [Geman and Geman, 1984] and the Metropolis-Hastings (MH) algorithm [Hastings, 1970; Metropolis et al., 1953].

Gibbs sampling In Gibbs sampling, each n-dimensional sample $x = \{x(1), x(2), ..., x(n)\}$ is generated sampling in turn each component $x(i)$ given the latest values of the others $x \setminus \{x(i)\}$ according to the target distribution $g(x)$. For example, for a two-dimensional state $x = \{a, b\}$ and a target distribution $g(a, b)$, a_t is sampled from $g(a|b_{t-1})$ and b_t is sampled from $g(b|a_t)$. Several variations of Gibbs sampling have been proposed. For example, Collapsed Gibbs Sampling that performs Gibbs sampling on a subset of components, marginalizing the rest. This can reduce the variance of the estimator. Another example is Blocked Gibbs Sampling that samples group of variables together.

Metropolis-Hastings In the Metropolis-Hastings (MH) algorithm, the $t+1$-th sample is generated from a proposal distribution $q(x_{t+1}|x_t)$, and it is accepted with probability $min\big(\frac{g(x_{t+1})q(x_t|x_{t+1})}{g(x_t)q(x_{t+1}|x_t)}, 1\big)$. The acceptance rate of MH might be low if the proposal distributions are not properly designed. In addition, MH might remain in a high-probability isolated region without (or rarely) traversing the rest of the space. This happens when the proposal distribution makes local moves. There are several solutions to these issues, such as Hamiltonian Monte Carlo (HMC).

Hamiltonian Monte Carlo The Hamiltonian Monte Carlo (or Hybrid Monte Carlo) [Neal, 2010] scheme can be used when the state is continuous and the gradient of the (unnormalized) log-target distribution is computable. HMC adds to the state x an auxiliary variable r, that can be interpreted as the momentum of a particle with position x. Starting from a point (x, r), an approximate Hamiltonian dynamics is applied and the resulting point (x', r') is accepted or rejected as in the MH algorithm.

The Hamiltonian dynamics is based on differential equations and it keeps the joint distribution $p(x, r)$ constant over time. Thus, the probability of a point (x', r') to a be accepted is ideally one (lower than one in practice). At the same time the state space can be better traversed.

Reversible jump (trans-dimensional) MCMC The sample space of the distribution $p(x)$ can have outcomes x with a different number of variables. For example, an urn where the number of balls is itself a random variable. Once the number of balls is sampled, a ball is sampled uniformly. For each ball we can associate one or more random variables (e.g., ID, color and size). Thus, the number of variables is not fixed. Sampling in spaces of differing dimensionality is easy if we sample variables sequentially as in ancestral sampling. However, applying MCMC schemes in such spaces is not trivial. Indeed, the proposal has to jump between spaces of different dimensions. This generates theoretical and practical difficulties, for example, in the computation of the MH acceptance ratio, we compare densities defined in different dimensionality spaces, which is meaningless in general [Murphy, 2012].

A possible solution is the reversible jump MCMC [Green, 1995], that augments the low dimensional space with extra random variables so that the two spaces have a common measure. However, this sampling scheme remains complex in practice.

Another solution that avoids reversible jumps is MCMC applied to computational traces [Goodman et al., 2008; Wingate et al., 2011]. A computational trace is a graph that represents the execution of probabilistic program. This method can be considered the state-of-the art for approximate inference for probabilistic programming. The main issue of such approach is the definition of the proposal that has to perform jumps to consistent states.

2.3.5 Inference in temporal models

In temporal models such as HMMs or DBNs there are specific types of inferences:

- filtering: estimating $p(x_t|z_{1:t}, u_{1:t})$;
- smoothing: estimating $p(x_t|z_{1:T}, u_{1:T})$ with $T > t$;
- likelihood of a sequence: estimating $p(z_{1:T}|u_{1:T})$;
- most probable hidden path: estimating $argmax_{x_{0:t}} p(x_{0:T}|z_{1:T}, u_{1:T})$.

In particular, filtering is concerned with estimating the belief, that is, the probability density function $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$. The Bayes filter computes recursively the belief at time $t + 1$, starting from the belief at t, the last observation z_{t+1}, and the last action performed u_{t+1} through

$$bel(x_{t+1}) = \eta p(z_{t+1}|x_{t+1}) \int_{x_t} p(x_{t+1}|x_t, u_{t+1}) bel(x_t) dx_t,$$

where η is a normalization constant. The above integral is only tractable for specific combinations of distributions $bel(x_t)$, $p(x_{t+1}|x_t, u_{t+1})$, and $p(z_{t+1}|x_{t+1})$ (e.g., the Kalman filter [Kalman, 1960] for linear Gaussian models). Therefore one has to resort to approximations, such as Monte-Carlo techniques.

Particle filter

Filtering inference can be approximated using sequential applications of importance sampling. The resulting algorithm is called sequential Monte-Carlo or Particle Filter [Doucet et al., 2000b]. The key idea of particle filtering is to represent the belief by a set of weighted samples (often called particles). Given N weighted samples $\{(x_t^{(i)}, w_t^{(i)})\}$ distributed as $bel(x_t)$, a new observation z_{t+1}, and a new action u_{t+1}, the particle filter generates a new weighted sample set that approximates $bel(x_{t+1})$.

The Particle Filtering (PF) algorithm proceeds in three steps:

(a) **Prediction step**: sample a new set of samples $x_{t+1}^{(i)}$, $i = 1, \ldots, N$, from a proposal distribution $g(x_{t+1}|x_t^{(i)}, z_{t+1}, u_{t+1})$.

(b) **Weighting step**: assign to each sample $x_{t+1}^{(i)}$ the weight:

$$w_{t+1}^{(i)} = w_t^{(i)} \frac{p(z_{t+1}|x_{t+1}^{(i)}) p(x_{t+1}^{(i)}|x_t^{(i)}, u_{t+1})}{g(x_{t+1}^{(i)}|x_t^{(i)}, z_{t+1}, u_{t+1})}.$$

(c) **Resampling**: if the variance of the sample weights exceeds a certain threshold, resample with replacement, from the sample set, with probability proportional to $w_{t+1}^{(i)}$ and set the weights to 1.

A common simplification is the *bootstrap filter*, where the proposal distribution is the state transition model $g(x_{t+1}|x_t^{(i)}, z_{t+1}, u_{t+1}) = p(x_{t+1}|x_t^{(i)}, u_{t+1})$ and the weight simplifies to $w_{t+1}^{(i)} = w_t^{(i)} p(z_{t+1}|x_{t+1}^{(i)})$.

2.4 First-Order Logic

Probabilistic graphical models are useful to represents random variables and their dependencies, however they define each random variable independently. These models are called propositional, because they cannot encode abstract knowledge that can be valid for a countably infinite number of facts (or random

variables). First-order logic (FOL) addresses this issue and provides a higher expressivity with respect to 'propositional' representations.

We will now define FOL concepts and then focus on Logic Programming. See [Nilsson and Małiszyński, 1995; Apt, 1997; Lloyd, 1987] for an extensive introduction.

Definition 2.13. *A first-order theory consists of an alphabet, a first-order language, a set of axioms, and a set of inference rules [Lloyd, 1987]*

Definition 2.14. *An alphabet consists of constants (i.e. objects), variables (i.e. generic objects), function symbols, predicate symbols (i.e. relations), connectives, and quantifiers.*

Constants represent objects of the domain, for example the strings 'one' and '1' are two constants that refer to same 'object' (the number 1). Variables represent a generic object, as will be clear later. In this thesis, variable names start with an uppercase letter and constants start with a lowercase letter. Function symbols refer to a function as in mathematics. For example, the string '+' refers to the well-known function sum. Predicate symbols refer to relations. For example, the string '<' refers to the lower-than relation of arity 2.

Definition 2.15. *A term is a constant, a variable or an n-ary function f applied to a tuple of terms $t_1, t_2, ..., t_n$, i.e. $f(t_1, t_2, ..., t_n)$.*

Definition 2.16. *An atomic formula (atom) is a n-ary predicate p applied to a list of terms.*

Example 2.4. *Consider the statement 'object* 1 *is inside object* 2*'. This fact can be represented with the atomic formula* $\texttt{inside}(1, 2)$*, where* `inside` *is a predicate, sometimes called relation, and* $1, 2$ *are constant symbols that refer to objects.*

Definition 2.17. *A literal is an atomic formula or a negated atomic formula.*

More complex formulas can be obtained by combining atomic formulas with connectives such as conjunction and disjunction. For example, $\texttt{inside}(1, 2), \texttt{inside}(2, 3)$ is a conjunction of two atomic formulas. A formula can contain quantifiers such as forall and exists ($\forall, \exists$). For example, $\exists \texttt{X} : \texttt{inside}(1, \texttt{X})$ means that there exists an object `x` such that $\texttt{inside}(1, \texttt{x})$. The term `X` is a (logical) variable.

Definition 2.18. *The first-order language given by an alphabet consists of the set of all formulas constructed from the symbols of the alphabet. [Lloyd, 1987]*

We will now introduce concepts that are often used in logic programming, a subset of FOL.

Definition 2.19. *A (definite)* clause*, in logic programming, is a first-order formula of the form* $\mathtt{h} \leftarrow \mathtt{b_1}, ..., \mathtt{b_n}.$*, equivalent to* $\forall A_1, A_2, ..., A_n : h \vee not(b_1) \vee ... \vee not(b_n)$*. Where* $\mathtt{h}$ *is an atom called head (atom),* $\mathtt{b_1}, ..., \mathtt{b_n}.$ *is a list of atoms called body, and* $A_1, A_2, ...A_n$ *is the list of variables that appear in the head or body.*

Example 2.5. *The clause*

$$\mathtt{inside(A, B)} \leftarrow \mathtt{inside(A, C), inside(C, B)}$$

states that for all $\mathtt{A, B}$ *and* $\mathtt{C}$*,* $\mathtt{A}$ *is inside* $\mathtt{B}$ *if* $\mathtt{A}$ *is inside* $\mathtt{C}$ *and* $\mathtt{C}$ *is inside* $\mathtt{B}$ *(transitivity property).* $\mathtt{A, B}$ *and* $\mathtt{C}$ *are logical variables.*

A clause generally contains non-ground atoms, that is, atoms with logical variables (e.g., $\mathtt{inside(A, B)}$). A clause with logical variables is assumed to be preceded by universal quantifiers for each logical variable, e.g., in the above clause: $\forall \mathtt{A}, \forall \mathtt{B}, \forall \mathtt{C}$.

Definition 2.20. *A substitution* θ *is a set of* $V_i = t_i$ *pairs where* V_i *are variables and* t_i *terms.*

A substitution θ applied to a formula replaces the variables V_i with terms t_i. For example, for $\theta = \{\mathtt{A} = 1, \mathtt{B} = 2, \mathtt{C} = 3\}$ the above clause becomes:

$$\mathtt{inside}(1, 2) \leftarrow \mathtt{inside}(1, 3), \mathtt{inside}(3, 1)$$

and states that if $\mathtt{inside}(1, 3)$ and $\mathtt{inside}(3, 1)$ are true, then $\mathtt{inside}(1, 2)$ is true.

Definition 2.21. *A unifier* θ *for* $\mathtt{A, B}$ *is a substitution that makes* $\mathtt{A}\theta = \mathtt{B}\theta$*. We indicate with* $\theta = \mathtt{mgu(A, B)}$ *the most general unifier, i.e. a unifier* θ *such that for each unifier* α *of* $\mathtt{A, B}$*, there exists a substitution* γ *such that* $\alpha = \theta\gamma$*.*

The formulas $\mathtt{A, B}$ are unifiable if there exists a unifier for $\mathtt{A, B}$.

Example 2.6. *The two atoms* $\mathtt{inside(A, 1), inside(B, C)}$ *are unifiable. A unifier is* $\alpha = \{\mathtt{A} = 2, \mathtt{B} = 2, \mathtt{C} = 1\}$*. The most general unifier is* $\theta = \{\mathtt{A} = \mathtt{B}, \mathtt{C} = 1\}$*. Note that* $\alpha = \theta\gamma$ *with* $\gamma = \{\mathtt{B} = 2\}$*.*

So far we defined objects and formulas symbolically, however it is necessary to associate those with actual elements and relation of the domain of interest. This association is called interpretation.

Definition 2.22. *An interpretation of a first-order language* L *consists of a set* D *called domain, for each constant in* L *an assignment of an element in* D*, for each n-ary function symbol in* L *a mapping* $D^n \rightarrow D$*, for each n-ary predicate symbol in* L *a mapping* $D^n \rightarrow \{true, false\}$ *(or equivalently a relation on* D*, i.e. a subset of* D^n*).*

2.4.1 Logic Programming

In this section we will describe concepts related to the FOL variant called Logic programming.

In logic programming a program $\mathbb{P}$ is a set of clauses and facts (grounded atoms) that represent what is assumed to be true (axioms). Given a program, it is possible to derive which formulas are true using inference rules.

Example 2.7. *Consider the following program:*

$$\texttt{box}(1). \tag{2.9}$$

$$\texttt{container}(2). \tag{2.10}$$

$$\texttt{cup}(3). \tag{2.11}$$

$$\texttt{spoon}(4). \tag{2.12}$$

$$\texttt{container}(\texttt{X}) \leftarrow \texttt{box}(\texttt{X}). \tag{2.13}$$

$$\texttt{container}(\texttt{X}) \leftarrow \texttt{cup}(\texttt{X}). \tag{2.14}$$

This contains facts about the type of objects and two clauses that define the concept container.

In logic programming Herbrand interpretations are generally used.

Definition 2.23. *A Herbrand interpretation of a first-order language L is an interpretation where the domain D is the set of all possible ground terms that can be formed from L (called Herbrand universe), and each constant and function is assigned with himself.*

Definition 2.24. *The Herbrand base B_L of a first-order language L is the set of all possible ground atoms which can be formed from predicates in L.*

A Herbrand interpretation can be defined specifying which atoms in B_L are true. Thus, a Herbrand interpretation is often defined as a subset of the Herbrand base B_L, i.e. the set of all formulas assumed to be true.

Example 2.8. *Consider a domain with two objects $D = \{1, 2\}$ and a first-order language L with a predicate* `inside` *of arity 2 and no function symbols. The Herbrand base is $B_L = \{\texttt{inside}(1,1), \texttt{inside}(1,2), \texttt{inside}(2,1), \texttt{inside}(2,2)\}$. A Herbrand interpretation $I = \{\texttt{inside}(1,2)\}$ represents a world where the atomic formula $\texttt{inside}(1,2)$ is true and any other atomic formula is false.*

There are two main types of inference used in logic programming: forward chaining and backward chaining. Forward chaining starts from known facts and derives new facts. This is repeated iteratively. Backward chaining starts from a query of interest and tries to find a proof. When we are interested in determining whether a query is true or false, backward chaining is more efficient because it focuses on the relevant part of the program. In logic programming forward chaining can be performed applying the $T_P(I)$ operator.

Definition 2.25. *Let $\mathbb{P}$ be a definite program and I a Herbrand interpretation. The $T_P(I)$ operator is defined as follows:*

$$T_P(I) = \{h\theta | h \leftarrow b_1, \ldots, b_n \in \mathbb{P}, \{b_1\theta, \ldots, b_n\theta\} \subseteq I, ground(h\theta)\}.$$

That is if the body of a rule is true in I for a substitution θ, the (ground) head $h\theta$ is in $T_P(I)$. Given a program it is possible to derive all possible true ground atoms using the T_P operator a number of times recursively starting from $I = \emptyset$, until a fixpoint is reached (i.e., $T_P(I) = I$). The interpretation obtained is called Least Herbrand Model and contains all the atoms that can be derived from $\mathbb{P}$.

For the program in Example 2.7 we have

$$T_P(\emptyset) = I_0 = \{\texttt{box(1)}, \texttt{container(2)}, \texttt{cup(3)}, \texttt{spoon(4)}\}$$

$$T_P(I_0) = I_1 = \{\texttt{box(1)}, \texttt{container(2)}, \texttt{cup(3)}, \texttt{spoon(4)}, \texttt{container(3)}, \texttt{container(1)}\}.$$

Backward chaining can be performed using resolution. In particular SLD-resolution [Nilsson and Małiszyński, 1995; Apt, 1997; Lloyd, 1987] is used in Prolog [Nilsson and Małiszyński, 1995], the most common logic programming system. SLD-resolution is an inference procedure to prove a query q, that focuses the proof on the relevant part of the program $\mathbb{P}$. SLD-resolution is a refutation process, the negated query is added to the program, and resolution is applied until a contradiction is reached (empty goal) or no further resolution steps are possible. If the empty goal is reached then the query is proved. If it is impossible to reach the empty goal, the query is assumed false under the closed-world assumption.

SLD-resolution starts from a goal equal to the negated query: $\leftarrow q_1, q_2, ..., q_n$. Then a generic step goes as follow. Given the current goal $\leftarrow q_1, q_2, \ldots, q_n$ and a rule $head \leftarrow body \in \mathbb{P}$ such that $\theta = mgu(q_1, head)$, then the new goal becomes $\leftarrow (body, q_2, \ldots, q_n)\theta$. There may be more than one rule that satisfies the mentioned conditions, resulting in the SLD-tree. In Prolog the tree is

traversed using depth-first search with backtracking. Moreover, Prolog applies SLD-resolution of the leftmost atom in the current goal.

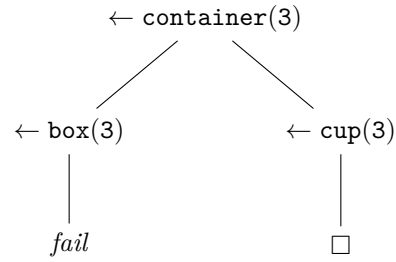
Example 2.9. *Consider the logic program in Example 2.7. The query* `container(3)` *can be proved as follows using SLD-resolution:*

$$\leftarrow container(3)$$

$$\leftarrow cup(3)$$

$$\leftarrow \square$$

Such sequence is called refutation or proof. The SLD-tree for the same query is the following:



A (program) clause can contain negated atoms in the body.

Definition 2.26. *A program* clause*, in logic programming, is a first-order formula of the form* $\mathtt{h} \leftarrow \mathtt{b_1}, ..., \mathtt{b_n}$*. Where* `h` *is an atom called head (atom), and* $\mathtt{b_1}, ..., \mathtt{b_n}$ *is a list of literals (atoms or negated atoms) called body.*

Normal programs are programs that contains program clauses. In logic programming a negated atom follows the **negation as failure** semantics. In this semantics, a literal `not(q)` is true whenever the query `q` fails and false otherwise. Inference in normal programs can be performed using SLDNF. This is a variant of SLD resolution, where whenever a negated atom `not(q)` is encountered, a SLD resolution is applied to query `q`. If the query `q` is proved then `not(q)` fails, otherwise if the query `q` fails, then `not(q)` succeeds.

2.5 Probabilistic Languages

First-order languages such as logic programming can represent abstract knowledge compactly, but do not handle uncertainty. To address this limitation,

several extensions has been proposed to combine probability theory with FOL, such as ProbLog [Kimmig et al., 2008], BLOG [Milch et al., 2005a], PRISM [Sato and Kameya, 1997], and Distributional Clauses (DC) [Gutmann et al., 2011]. Those frameworks go under the fields of probabilistic logic programming and *statistical relational learning* (SRL) [Getoor and Taskar, 2007; De Raedt, 2008; De Raedt et al., 2008], which combine logical representations, probabilistic reasoning, and machine learning. However, there are other probabilistic languages such as Anglican [Wood et al., 2014] and Church [Goodman et al., 2008] that belong to the wide area of *probabilistic programming* that combines probability theory with programming paradigms (e.g., procedural or functional).

In this thesis we extend Distributional Clauses based on distribution semantics defined by [Sato, 1995]. The distribution semantics provides a well-founded semantics that combines logic programming concepts with probability theory. In this semantics a program DB consists of a set of facts F and a set of clauses BK. Instead of considering facts deterministically true or false, the distribution semantics assigns a probability distribution P_F over subset of facts $F' \subset F$. Then a unique distribution $P_{DB}(I)$ over interpretations I is derived from P_F. In detail, $P_{DB}(I) = P_F(F')$ if I is the least Herbrand model of DB extending F', $P_{DB}(I) = 0$ otherwise. In other words the distribution semantics defines a distribution over possible worlds (interpretations).

Example 2.10. *Consider the logic program in Example 2.7, where the facts* $F = \{\texttt{box(1)}, \texttt{container(2)}, \texttt{cup(3)}, \texttt{spoon(4)}\}$ *are probabilistic. Let us assume that the joint distribution of such facts is the following:*

$$P_F(F_1 = \{\texttt{container(2)}\}) = 0.1, \tag{2.15}$$

$$P_F(F_2 = \{\texttt{box(1)}\}) = 0.2, \tag{2.16}$$

$$P_F(F_3 = \{\texttt{box(1)}, \texttt{container(2)}\}) = 0.7 \tag{2.17}$$

and zero in all the remaining cases. Note that the facts not in F_j *are assumed to be false, e.g.,* $F_1 = \{\texttt{container(2)}\}$ *means that* $\texttt{container(2)}$ *is true and the remaining facts* $\texttt{box(1)}, \texttt{cup(3)}, \texttt{spoon(4)}$ *are false. Thus, in this example* $\texttt{cup(3)}, \texttt{spoon(4)}$ *are always false. We can derive the least Herbrand models for each case and obtain three possible interpretations with non-zero probability, with* $P_{DB}(I_j) = P_F(F_j)$*:*

$$P_{DB}(I_1 = \{container(2)\}) = 0.1, \tag{2.18}$$

$$P_{DB}(I_2 = \{box(1), container(1)\}) = 0.2, \tag{2.19}$$

$$P_{DB}(I_3 = \{box(1), container(1), container(2)\}) = 0.7 \tag{2.20}$$

Any other interpretation has probability zero.

2.6 Planning

Classical planning consists in finding a sequence of actions that reach a goal state with minimal cost. This definition refers to deterministic and goal-oriented tasks, and it can be generalized to probabilistic environments and for tasks that do not necessarily have goal states. In general, the goal of probabilistic planning is acting in an environment such that a given cost function is minimized (or a reward function is maximized) in average. The most common frameworks for probabilistic planning are Markov Decision Processes (MDPs) for fully observable problems, and Partially Observable Markov Decision Processes (POMDPs) for partially observable problems. We will introduce first standard MDPs, then describe relational MDPs and finally describe some languages used to express planning problems.

2.6.1 Markov Decision Processes

In an MDP [Sutton and Barto, 1998; Wiering and van Otterlo, 2012], a putative agent is assumed to interact with its environment, described using a set S of *states*, a set A of *actions* that the agent can perform, a *transition function* $p : S \times A \times S \rightarrow [0, 1]$, and a *reward function* $R : S \times A \rightarrow \mathbb{R}$. That is, when in state s_t and on doing a, the probability of reaching s_{t+1} is given by $p(s_{t+1}|s_t, a)$, for which the agent receives the reward $R(s_t, a)$. The agent is taken to operate over a finite or infinite number of time steps $t = 0, 1, \ldots, T$, with the goal of maximizing the expected reward: $\mathbb{E}[\sum_{t=0}^{T} \gamma^t R(s_t, a_t)] = \mathbb{E}[G(E)]$, where $\gamma \in [0, 1]$ is a discount factor, $E =< s_0, a_0, s_1, a_1, ..., s_T, a_T >$ is the state and action sequence called episode and $G(E) = \sum_{t=0}^{T} \gamma^t R(s_t, a_t)$ is the total discounted reward of E. This is achieved by computing a (deterministic) policy $\pi : S \times D \rightarrow A$ that determines the agent's action at state s and remaining steps d (horizon). For compactness we write $\pi(s_t)$ instead of $\pi_d(s_t)$, indeed the horizon is unambiguously defined by $d = T - t$. The expected reward starting from state s_t and following a policy π is called the *value function* (V-function):

$$V_d^\pi(s_t) = \mathbb{E}[G(E_t)|s_t, \pi] = \mathbb{E}\left[\sum_{k=t}^{t+d} \gamma^{k-t} R(s_k, a_k) \mid s_t, \pi\right]. \tag{2.21}$$

Where $E_t =< s_t, a_t, s_{t+1}, a_{t+1}, ..., s_T, a_T >$ is the subset of E from time t. The V-function can be defined recursively (Bellman equation):

$$V_d^\pi(s_t) = R(s_t, \pi(s_t)) + \gamma \int_{s_{t+1}} p(s_{t+1}|s_t, \pi(s_t)) V_{d-1}^\pi(s_{t+1}) ds_{t+1}. \tag{2.22}$$

Furthermore, the expected reward starting from state s_t while executing action a_t and following a policy π is called the *action-value function* (Q-function):

$$Q_d^\pi(s_t, a_t) = \mathbb{E}[G(E_t)|s_t, a_t, \pi] = \mathbb{E}\left[\sum_{k=t}^{t+d} \gamma^{k-t} R(s_k, a_k) \mid s_t, a_t, \pi\right]. \quad (2.23)$$

Since $T = t + d$, in the following formulas we will use T for compactness. An *optimal policy* π^* is a policy that maximizes the V-function for all states. The optimal policy satisfies the Bellman optimality equation:

$$V_d^*(s_t) = max_a\Big(R(s_t, a_t) + \gamma \int_{s_{t+1}} p(s_{t+1}|s_t, a_t) V_{d-1}^*(s_{t+1}) ds_{t+1}\Big). \quad (2.24)$$

This formula is used to solve the MDP in value-iteration methods. In detail, starting from $V_0^*(s_t) = 0$ formula (2.24) is applied d times for finite horizon MDPs or until convergence for infinite horizon MDPs. The optimal policy is obtained by

$$\pi^*(s_t) = argmax_a\Big(R(s_t, a_t) + \gamma \int_{s_{t+1}} p(s_{t+1}|s_t, a_t) V_{d-1}^*(s_{t+1}) ds_{t+1}\Big). \quad (2.25)$$

Another method to solve the MDP is policy iteration. It consists of policy evaluation and policy improvement. The algorithm starts with an arbitrary policy, it computes $V_d^\pi(s_t)$ applying (2.22) (policy evaluation) and updates the policy (policy improvement) applying (2.25). This process is repeated until convergence. Policy iteration is generally used in infinite horizon MDPs.

All the above formulas can be used for infinite horizon MDPs, where $d \to \infty$, as long as $\gamma < 1$. Indeed, the discount factor γ is needed to keep the sum of infinite terms finite. Note that for $d \to \infty$, $V_d^\pi(s_t) = V_{d-1}^\pi(s_t)$ and the policy does not depend on the horizon, since for any t, d is always infinite.

In large or complex domains the integrals in (2.22) (2.24) cannot be computed exactly. A common solution is approximating such integrals with Monte-Carlo methods. Generally, sample-based planners combine Monte-Carlo methods with policy iteration to solve an MDP and find a (near) optimal policy. Such planners simulate (by sampling) interaction with the environment in episodes $E^m = <s_0^m, a_0^m, s_1^m, a_1^m, ..., s_T^m, a_T^m>$, following some policy π. Each episode is a trajectory of T time steps, and we let s_t^m denote the state visited at time t during episode m. (So, after M episodes, $M \times T$ states would be explored). After or during episode generation, the sample-based planner updates $Q_d(s_t^m, a_t^m)$ estimations for each t according to a backup rule, for example, averaging the total rewards obtained starting from (s_t^m, a_t^m) till the end. The policy is improved using a strategy that trades-off exploitation and exploration. For example, the

ϵ-greedy strategy selects the action $argmax_a Q_d(s,a)$ with probability $1-\epsilon$, otherwise selects $\pi(a|s) \sim uniform(actions)$ with $0 \leq \epsilon \leq 1$. In this case the policy used to sample the episodes is not deterministic; we indicate with $\pi(a_t|s_t)$ the probability to select action a_t in state s_t under the policy π. Under certain conditions, after a sufficiently large number of episodes, the policy converges to a (near) optimal policy, and the planner can execute the greedy policy $argmax_a Q_d(s_t,a)$.

2.6.2 Relational MDPs

In first-order (relational) MDPs [Mausam and Kolobov, 2012; Wiering and van Otterlo, 2012], the state is described as a set of objects, object attributes, and relationships between them. In particular, in relational MDPs based on logic programming, a state is a Herbrand interpretation and the actions are described as facts. The state transition model and the reward function are compactly defined in terms of probabilistic rules exploiting first-order logic.

For example, let us consider the blocksworld domain, where there are objects on a table or on top of other objects. In this domain we can move objects with no objects on top (clean), and the action succeeds with a certain probability or leaves the state unchanged. A possible state in the blocksworld is $\texttt{on}(\texttt{1},\texttt{2}), \texttt{clean}(\texttt{1}), \texttt{on}(\texttt{1},\texttt{table})$, i.e. `1` is on `2`, `1` is clean (i.e. it does not have objects on top) and `1` is on the table. All other facts (e.g., $\texttt{on}(\texttt{2},\texttt{1})$) are assumed to be false. In this domain we can say that if $\texttt{on}(\texttt{A},\texttt{C}), \texttt{clean}(\texttt{B})$ holds then action $\texttt{move}(\texttt{A},\texttt{B})$ will add $\texttt{on}(\texttt{A},\texttt{B})$ with probability 0.9 to the state and remove $\texttt{on}(\texttt{A},\texttt{C}), \texttt{clean}(\texttt{B})$, otherwise with probability 0.1 the state will remain unchanged.

In addition, it is often convenient to define when an action is applicable in a given state. This can be specified again in terms of rules (clauses). The conditions that make an action applicable are often called *preconditions*.

A relational MDP can be solved using the Bellman equation applied to abstract states with logical regression, instead of single states individually. This method is called Symbolic Dynamic Programming (SDP) [Boutilier et al., 2001, 2000], and it has been successfully used to solve big MDPs efficiently [Kersting et al., 2004; Wang et al., 2008; Joshi et al., 2010; Hölldobler et al., 2006]. In SDP approaches, the V-function is represented in compact structures. For relational MDPs the V-function is represented using first-order decision diagrams (FODD [Wang et al., 2008]) or rules as in REBEL [Kersting et al., 2004]. For example, for the blocksworld the V-function we can be described as follows [Kersting

et al., 2004]:

$$10 \leftarrow \texttt{on}(\texttt{a}, \texttt{b}), \texttt{a} \neq \texttt{b}. \tag{2.26}$$

$$8.1 \leftarrow \texttt{clear}(\texttt{a}), \texttt{clear}(\texttt{b}), \texttt{on}(\texttt{a}, \texttt{X}), \texttt{a} \neq \texttt{b}, \texttt{a} \neq \texttt{X}, \texttt{b} \neq \texttt{X}. \tag{2.27}$$

[...]

Any of the countably infinite interpretations has a V-function derivable from those rules. This shows the advantage of exploiting the domain and a compact representation to solve the MDP. Similar principles have been applied in (propositional) continuous and hybrid domains [Sanner et al., 2011; Zamani et al., 2012].

Despite the effectiveness of such approaches, they make restrictive assumptions (e.g., deterministic transition model for continuous variables) to keep exact inference tractable. For more general domains approximations are needed, for example sample-based methods or confidence intervals [Zamani et al., 2013]. Another issue of SDP is keeping the structures that represent the V-function compact. Nonetheless, some solutions are available in the literature, such as pruning or real-time SDP [Vianna et al., 2015].

2.6.3 Languages for planning

There are several languages that have been proposed to describe planning problems. We will briefly describe the main features of those languages. STRIPS (Stanford Research Institute Problem Solver) [Fikes and Nilsson, 1971] is a well-known language that inspired more recent languages. A more recent language is PDDL [Mcdermott et al., 1998] (Planning Domain Definition Language) inspired by STRIPS, created with the purpose of standardizing planning languages.

In STRIPS a planning task consists of an initial state, the set of goal states, and a set of actions. For each action we need to define preconditions (what must be true before the action is performed) and postconditions (the effects of an action). STRIPS adopts logic-programming concepts. The initial state is described as a set of facts (atomic formulas). For example: $\texttt{on}(1, 2), \texttt{on}(2, \texttt{table}), \texttt{clear}(1)$ describe the initial state in the blocksworld example, where object `1` is on `2`, `2` is on the table, and `1` is clear (no objects on top it). Any other fact is assumed to be false (closed-world assumption). The goal state is represented as a conjunction of literals (not necessarily ground). For example, the goal $\texttt{on}(\texttt{X}, 1), \texttt{clear}(2)$ represents all the states where an arbitrary object is on top of `1` and `2` is clear.

An action is defined as a predicate, the preconditions are defined as a conjunction of literals, and the postconditions are described in addlist (what needs to be added) and deletelist (what needs to be removed), both conjunction of literals. For example, the action move object `X` from `Y` to `Z` in the blocksworld is defined as: `move(X,Y,Z)`, prec: `on(X,Y)`, `clear(X)`, `clear(Z)`, addlist: `on(X,Z)`, deletelist: `on(X,Y)`, `clear(Z)`.

PDDL is similar to STRIPS with additional features, that will not be discussed here. Both PDDL and STRIPS can describe only deterministic planning problems. For probabilistic planning problems, there exists several extensions, such as PPDDL (Probabilistic PDDL), and the recent RDDL [Sanner] (Relational Dynamic influence Diagram Language). RDDL describes the domain as a relational Dynamic Bayesian Network, it supports arbitrary reward functions and other advanced features (e.g., concurrent actions). Thus, it can be used to model a wide class of MDPs.

The described languages can only formally describe a planning problem, but they do not provide a method to solve the task. A solver (planner) is required to interpret the task description and provide a meaningful plan (policy).

Chapter 3

Distributional Clauses

In this chapter we introduce distributional clauses (DC) [Gutmann et al., 2011], an extension of distribution semantics [Sato, 1995], and we propose a new inference algorithm based on importance sampling (Section 3.2). The algorithm is able to perform inference in complex hybrid domains that most related frameworks cannot handle. This chapter consists of research previously published in the journal paper [Nitti et al., 2016].

3.1 Distributional Clauses

Formally, a *distributional clause* is a formula of the form $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_n}$, where the $\mathtt{b_i}$ are literals and $\sim$ is a binary predicate written in infix notation. The intended meaning of a distributional clause is that each ground instance of the clause $(\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_n})\theta$ defines the random variable $\mathtt{h}\theta$ with distribution $\mathcal{D}\theta$ whenever all the $\mathtt{b_i}\theta$ hold, where θ is a substitution. In other words, a distributional clause can be seen as a powerful template to define conditional probabilities: $p(\mathtt{h}\theta|(\mathtt{b_1}, \ldots, \mathtt{b_n})\theta) = \mathcal{D}\theta$. The term $\mathcal{D}$ can be nonground, i.e., values, probabilities, or distribution parameters can be related to conditions in the body. Furthermore, given a random variable r, the term $\simeq(r)$ constructed from the reserved functor $\simeq/1$ represents the value of r. Abusing notation, for brevity, we shall sometimes write $r \sim\!= v$ instead of $\simeq(r) = v$, which is true iff the value of the random variable r unifies with v.

Example 3.1. *Consider the following clauses:*

$$\mathtt{n} \sim \mathtt{poisson}(6). \tag{3.1}$$

$$\mathtt{pos}(\mathtt{P}) \sim \mathtt{uniform}(0, \mathtt{M}) \leftarrow \mathtt{n} \sim\!= \mathtt{N}, \mathtt{between}(1, \mathtt{N}, \mathtt{P}), \mathtt{M\ is\ 10} * \mathtt{N}. \tag{3.2}$$

$$\mathtt{left}(\mathtt{A}, \mathtt{B}) \leftarrow \simeq(\mathtt{pos}(\mathtt{A})) <\!\simeq(\mathtt{pos}(\mathtt{B})). \tag{3.3}$$

Where '$\mathtt{is}$' *is the equality operator. Capitalized terms such as* $\mathtt{P}, \mathtt{A}$, *and* $\mathtt{B}$ *are logical variables, which can be substituted with any constant. Clause* (3.1) *states that the number of people* $\mathtt{n}$ *is governed by a Poisson distribution with mean 6. Clause* (3.2) *models the position* $\mathtt{pos}(\mathtt{P})$ *as a continuous random variable uniformly distributed from 0 to* $\mathtt{M} = 10\mathtt{N}$ *(that is, 10 times the number of people), for each person identifier* $\mathtt{P}$ *such that* $1 \leq \mathtt{P} \leq \mathtt{N}$*, where* $\mathtt{P}$ *is integer and* $\mathtt{N}$ *is unified with the number of people* $\mathtt{n}$*. For example, if the value of* $\mathtt{n}$ *is 2, there will be 2 independent random variables* $\mathtt{pos}(1)$ *and* $\mathtt{pos}(2)$ *with distribution* $\mathtt{uniform}(0, 20)$*. Finally, clause* (3.3) *defines the binary relation* $\mathtt{left}$*, comparing people positions. Note that the atom* $\mathtt{left}(\mathtt{a}, \mathtt{b})$ *is defined using a deterministic clause, but it is a random variable as it depends on other random variables.*

DC supports continuous distributions (under reasonable conditions) and naturally copes with an unknown number of objects [Gutmann et al., 2011]. In addition to distributional clauses, we shall also employ deterministic clauses as in Prolog. We shall often talk about clauses when the context is clear.

A distributional program $\mathbb{P}$ is a set of distributional and/or deterministic clauses that defines a distribution $p(x)$ over possible worlds x. The probability $p(q)$ of a query q can be estimated using Monte-Carlo methods, that is, possible worlds are sampled from $p(x)$, and $p(q)$ is approximated as the ratio of samples in which the query q is true.

The procedure used to generate possible worlds defines the semantics and a basic inference algorithm. A possible world is generated starting from the empty partial world $x = \emptyset$; then for each distributional clause $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b}_1, \ldots, \mathtt{b}_n$, whenever the body $\{\mathtt{b}_1\theta, \ldots, \mathtt{b}_n\theta\}$ is true in the set x for the substitution θ, a value v for the random variable $\mathtt{h}\theta$ is sampled from the distribution $\mathcal{D}\theta$ and $\mathtt{h}\theta = v$ is added to the new partial world $\hat{x}$. This is also performed for deterministic clauses, adding ground atoms to $\hat{x}$ whenever the body is true. This process is then recursively repeated until a fixpoint is reached, that is, until no more variables can be sampled and added to the world. The final world is called complete or full, while the intermediate worlds are called partial. The process is based on sampling, thus 'world' is often replaced with sample or particle. Notice that a possible world may contain a countably infinite number of random variables (and atoms).

Example 3.2. *Given the DC program* $\mathbb{P}$ *defined by* (3.1), (3.2), *and* (3.3), $\{\mathtt{n} = 2, \mathtt{pos}(1) = 3.1, \mathtt{pos}(2) = 4.5, \mathtt{left}(1,2)\}$ *is a possible (complete) world. This world is sampled in the following order:* $\emptyset \rightarrow \{\mathtt{n} = 2\} \rightarrow \{\mathtt{n} = 2, \mathtt{pos}(1) = 3.1, \mathtt{pos}(2) = 4.5\} \rightarrow \{\mathtt{n} = 2, \mathtt{pos}(1) = 3.1, \mathtt{pos}(2) = 4.5, \mathtt{left}(1,2)\}$.

ST_P operator Gutmann et al. [2011] formally describe this generative process using the ST_P operator, a stochastic version of the T_P operator. This operator is applied on partial worlds (or interpretations); these contain ground atoms (as in standard logic programming), and for each random variable $\mathtt{r}$ defined in the partial worlds, there will be an atom of the form $\mathtt{r} \sim \mathcal{D}$, and an equality $\mathtt{r} = v$ in a separated table, where v is the value sampled from the distribution $\mathcal{D}$. Throughout the thesis we do not always write the $\mathtt{r} \sim \mathcal{D}$ explicitly. To generate a possible world, one starts from the empty partial world $I = \emptyset$, and applies $I \leftarrow ST_P(I)$ until a fixpoint is reached ($ST_P(I) = I$).

Validity conditions To define a proper probability distribution $p(x)$, a DC program $\mathbb{P}$ needs to satisfy the validity conditions described in [Gutmann et al., 2011]. For each predicate h or random variable we assign a rank (natural number) that defines an order. A DC program is valid if:

1. For each ground $\mathtt{h}\theta$, $\mathtt{h}\theta \sim \mathcal{D}\theta$ has to be unique in the least fixpoint, i.e., there is one distribution defined for each random variable.
2. The program has to be stratified, that is, there exists a rank assignment such that for each distributional clause $\mathtt{h} \sim \mathcal{D} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_n}. : rank(\mathtt{h} \sim \mathcal{D}) > rank(\mathtt{b_i})$, while each definite clause $\mathtt{h} \leftarrow \mathtt{b_1}, \ldots, \mathtt{b_n}. : rank(\mathtt{h}) \geq rank(\mathtt{b_i})$.
3. All ground probabilistic facts are Lebesgue-measurable.
4. Each atom in the least fixpoint can be derived from a finite number of probabilistic facts (finite support condition [Sato, 1995]).

Throughout the thesis the partial worlds will be written as x^P, while $x \supseteq x^P$ indicates a complete world consistent with x^P, and $x^a = x \setminus x^P$ represents the remaining part of the world.

Negation We extended DC to allow for negated literals in the body of distributional clauses. To accommodate negation, we need to consider the case that a random variable is not defined in a full world x. Any comparison involving a non-defined random variable will fail; therefore, its negation will

succeed. In addition, a (deterministic) ground atom `a` is considered false in x, if $\mathtt{a} \notin x$ (standard closed-world assumption).

A distributional program $\mathbb{P}$ to be valid needs to be stratified, thus no specific requirements are needed to support negation. The semantics is defined along the same lines as the perfect model M_P [Przymusinski, 1988], that is, the ST_P operator is applied at each rank from lowest to highest rank. The result is a world x sampled from the distribution $p(x)$ defined by the program $\mathbb{P}$.

Consider a DC program $\mathbb{P}$, a literal l and a (partial) world $x^{P(i)}$ obtained by applying the ST_P operator till the least fixpoint for $rank(var(l))$ is reached (or exceeded). If l is a classic atomic formula, l is true in $x^{P(i)}$ iff $l \in x^{P(i)}$, and $not(l)$ holds in $x^{P(i)}$ iff $l \notin x^{P(i)}$. This follows the closed world assumption: a literal is false if it is not in the least fixpoint. If l is a comparison operator involving a random variable r: $l = (r \sim\!= val)$, then $not(r \sim\!= val)$ holds in $x^{P(i)}$ whenever $r \sim\!= val$ is false in $x^{P(i)}$ or the random variable r is not defined in $x^{P(i)}$, that is, when $r \notin var(x^{P(i)})$. Note that $x^{P(i)}$ is the least fixpoint for $rank(r)$ (or higher), thus $r \notin var(x^{P(i)})$ implies that r is not defined for each world $m \supseteq x^{P(i)}$ consistent with $x^{P(i)}$.

To determine $not(l)$, $not(r \sim\!= val)$ or just the existence of a variable r it is not required to explicitly apply the ST_P operator. In logic programming, negation as failure is used to prove negated formulas: if the query q fails then $not(q)$ is true and vice versa. A common inference procedure is SLDNF, that is SLD resolution with negation as failure.

Consider the following examples:

$$\mathtt{n} \sim \mathtt{poisson}(6). \tag{3.4}$$

$$\mathtt{color}(\mathtt{X}) \sim \mathtt{uniform}([\mathtt{red}, \mathtt{blue}, \mathtt{black}]) \leftarrow \mathtt{n} \sim\!= \mathtt{N}, \mathtt{between}(1, \mathtt{N}, \mathtt{X}). \tag{3.5}$$

$$\mathtt{notred} \leftarrow \mathtt{not}(\mathtt{color}(2) \sim\!= \mathtt{red}). \tag{3.6}$$

$$\mathtt{nothing\_red} \leftarrow \mathtt{not}(\mathtt{color}(\mathtt{X}) \sim\!= \mathtt{red}). \tag{3.7}$$

`notred` is true in those worlds where the value of the random variable `color(2)` is not red. In some worlds, the variable `color(2)` is not defined, for example when $n = 1$, in such cases `notred` still succeeds. Similarly, the atom `nothing_red` is true in a world x iff the query $\mathtt{color}(\mathtt{X}) \sim\!= \mathtt{red}$ fails, that is, iff every variable `color(X)` defined in x is not red (for `X` between 1 and n). Therefore, `nothing_red` is also true in worlds with no objects ($\mathtt{n} = 0$).

3.2 Static Inference for Distributional Clauses

Sampling full worlds is generally inefficient or may not even terminate as possible worlds can be infinitely large. Therefore, Gutmann et al. [2011] use magic sets [Bancilhon et al., 1986] to generate only those facts that are relevant for answering the query. Magic sets are a well-known logic programming technique for forward reasoning. In this thesis, we propose a more efficient sampling algorithm based on backward reasoning and likelihood weighting.

3.2.1 Importance Sampling

Given a program $\mathbb{P}$, the probability of the query q is estimated by applying importance sampling to partial samples $x^{P(i)}$, with $i = 1, \ldots, N$ where N is the number of samples. In importance sampling the proposal probability $g(x)$ used to generate samples is not necessarily the target probability $p(x)$.

The Monte-Carlo approximation is the following (where $\mathbb{1}$ is the indicator function[1]):

$$p(q) = E_{p(x)}[\mathbb{1}(x \models q)] = \int_x \mathbb{1}(x \models q) p(x) dx = \int_{x^P} \int_{x^a} \mathbb{1}(x \models q) p(x^a | x^P) p(x^P) dx^a dx^P$$

$$= \int_{x^P} \underbrace{\int_{x^a} \mathbb{1}(x \models q) p(x^a | x^P) dx^a}_{p(q|x^P)} p(x^P) dx^P = \int_{x^P} p(q|x^P) p(x^P) dx^P$$

$$= \int_{x^P} \underbrace{p(q|x^P) \frac{p(x^P)}{g(x^P)}}_{w_q} g(x^P) dx^P \approx \frac{1}{N} \sum_{i=1}^{N} w_q^{(i)}, \tag{3.8}$$

where the weight is $w_q^{(i)} = p(q|x^{P(i)}) \frac{p(x^{P(i)})}{g(x^{P(i)})}$. Formula (3.8) uses a fixed split $x = x^P \cup x^a$. Following [Milch, 2006] we extend this idea to exploit context-specific independencies: we can have a different split in different samples.

There are two reasons to sample partial worlds instead of complete ones. First, the sampling process is faster and terminates (under some conditions) even when the complete world is a countably infinite set. Second, the estimator variance is generally lower with respect to a naive Monte-Carlo estimator that samples complete worlds. Indeed, sampling some variables $x^{P(i)}$ and computing $p(q|x^{P(i)})$ analytically is an instance of the conditional Monte-Carlo method

[1] The indicator function is 1 when the argument is true, zero otherwise.

[Lemieux, 2009] (sometimes called Rao-Blackwellization), which has better performance with respect to naive Monte-Carlo.

The split of x has to guarantee that the probability $p(q|x^{P(i)})$ is analytically computable. Let $var(q)$ denote the set of all random variables in q. If $var(q) \subseteq var(x^{P(i)})$, all the variables in q are instantiated in $x^{P(i)}$, thus q can be determined deterministically: $p(q|x^{P(i)}) = \mathbb{1}(x^{P(i)} \models q)$. In some cases it is possible to compute $p(q|x^{P(i)})$ without sampling variables in q.

Example 3.3. *Given the clauses* (3.1), (3.2), (3.3),

$$p(q|x^{P(i)}) = p(\texttt{pos(1)} > \texttt{5} \mid \{\texttt{n} = \texttt{3}, \texttt{pos(1)} = \texttt{3}\}) = \texttt{0},$$

while the marginalized variables $\texttt{pos(2)}, \texttt{pos(3)}, \texttt{left(X, Y)}$ *are irrelevant. We could even do better, for example* $p(q|x^{P(i)}) = p(\texttt{pos(1)} > \texttt{5}|\{\texttt{n} = \texttt{3}\})$ *is analytically computable without sampling* $var(q) = \{\texttt{pos(1)}\}$.

In general, it is impossible to sample only $var(q)$ as the DC program does not directly define the distribution of these variables. For example, to sample `pos(1)` defined by (3.2) we first need to sample `n` defined by (3.1); thus we need to follow the generative sampling process (ST_P operator) until the variables of interest are sampled. Backward reasoning (or the magic set transformation) can help to focus the sampling.

The probability of a query given evidence $p(q|e)$ can be estimated using formula (3.8) twice, once to estimate $p(q, e)$ and another time to estimate $p(e)$, then $p(q|e)$ is approximated as the ratio of the two quantities. As an alternative, formula (3.8) can be adapted to use the same partial samples to estimate the two quantities. In detail, each sample is split in $x = x_e^P \cup x_q^P \cup x^a$, such that x_e^P is sufficient to compute $p(e|x_e^P)$ and $x_q^P \cup x_e^P$ is sufficient to compute $p(q|e, x_q^P, x_e^P)$. Formula (3.8) is applied to estimate $p(e)$, then the same partial samples $x_e^{P(i)}$

are expanded to estimate $p(q,e)$:

$$p(q,e) = \int_{x_e^P}\int_{x_q^P} p(q,e|x_q^P,x_e^P)p(x_q^P,x_e^P)dx_q^P dx_e^P$$

$$= \int_{x_e^P}\int_{x_q^P} p(q|e,x_q^P,x_e^P)p(e|x_e^P)p(x_q^P|x_e^P)p(x_e^P)dx_q^P dx_e^P$$

$$= \int_{x_e^P}\int_{x_q^P} p(q|e,x_q^P,x_e^P)p(x_q^P|x_e^P)dx_q^P p(e|x_e^P)p(x_e^P)dx_e^P$$

$$= \int_{x_e^P}\int_{x_q^P} p(q|e,x_q^P,x_e^P)\frac{p(x_q^P|x_e^P)}{g(x_q^P|x_e^P)}g(x_q^P|x_e^P)dx_q^P p(e|x_e^P)\frac{p(x_e^P)}{g(x_e^P)}g(x_e^P)dx_e^P$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\underbrace{p(q|e,x_q^{P(i)},x_e^{P(i)})\frac{p(x_q^{P(i)}|x_e^{P(i)})}{g(x_q^{P(i)}|x_e^{P(i)})}}_{w_q^{(i)}}\underbrace{p(e|x_e^{P(i)})\frac{p(x_e^{P(i)})}{g(x_e^{P(i)})}}_{w_e^{(i)}} = \frac{1}{N}\sum_{i=1}^{N} w_q^{(i)}w_e^{(i)}$$

$$p(q|e) = \frac{p(q,e)}{p(e)} \approx \frac{\sum_{i=1}^{N} w_q^{(i)}w_e^{(i)}}{\sum_{i=1}^{N} w_e^{(i)}}. \tag{3.9}$$

Where the proposal g has the same factorization of the target distribution: $g(x_q^P,x_e^P) = g(x_q^P|x_e^P)g(x_e^P)$.

3.2.2 Sampling partial possible worlds

We now present our approach to sampling possible worlds and computing $p(q)$ following Equation (3.8) and $p(q|e)$ following (3.9). Central is the algorithm with signature

EvalSampleQuery(q : query, $x^{P(i)}$: partial world) **returns** $(w_q^{(i)}, x_q^{P(i)})$

that starts from a given query q and a partial world $x^{P(i)}$ (which will be empty in case there is no evidence, cf. below), and generates an expanded partial world $x_q^{P(i)}$ together with its weight $w_q^{(i)}$ so that

1. $x_q^{P(i)} \supseteq x^{P(i)}$, i.e., $x_q^{P(i)}$ is an expansion of $x^{P(i)}$,
2. $var(q) \subseteq var(x_q^{P(i)})$, which ensures that we can evaluate q in $x_q^{P(i)}$ and therefore $p(q|x_q^{P(i)})$,

3. $w_q^{(i)} = p(q|x_q^{P(i)}) \frac{p(x_q^{P(i)}|x^{P(i)})}{g(x_q^{P(i)}|x^{P(i)})}$.

$p(q)$ is estimated by calling $(w_q^{(i)}, x_q^{P(i)}) \leftarrow$ EvalSampleQuery$(q, \emptyset)$ N times and applying (3.8). The probability $p(q|e)$ is estimated by calling, for each sample, $(w_e^{(i)}, x_e^{P(i)}) \leftarrow$ EvalSampleQuery$(e, \emptyset)$ for the evidence, and $(w_q^{(i)}, x_q^{P(i)}) \leftarrow$ EvalSampleQuery$(q, x_e^{P(i)})$ for the query given $x_e^{P(i)}$, and then applying (3.9).

The key question is thus how to sample one such partial world $x_q^{P(i)}$ for a generic call of EvalSampleQuery$(q, x^{P(i)})$. To realize this, we combine likelihood weighting (LW) [Fung and Chang, 1989; Koller and Friedman, 2009] with a variant of SLD-resolution in the EvalSampleQuery algorithm that we describe below.

Adapting SLD-resolution. EvalSampleQuery employs an extension of SLD-resolution to determine which random variables to sample, until the query q can be evaluated. However, unlike traditional SLD-resolution, it keeps track of a number of global variables:

1. the weight $w_q^{(i)}$, initialized to 1,
2. the initial query iq, initialized to q, and
3. the partial sample $x^{P(i)}$.

Starting from a goal $G = q$, EvalSampleQuery applies inference rules until the goal G is empty (i.e., q has been proven) or no more rules can be applied (q fails). If the query succeeds, the algorithm returns the final weight and the expanded sample $(w_q^{(i)}, x_q^{P(i)})$. If the query fails it returns $(w_q^{(i)} = 0, x_q^{P(i)})$.

Definition 3.1. *Given two formulas f and g, a DC program $\mathbb{P}$ and a partial sample $x^{P(i)}$, we say that f and g are equivalent with respect to $\mathbb{P}$ and $x^{P(i)}$, denoted by $\mathbb{P}, x^{P(i)} \models f \Leftrightarrow g$, iff f and g have the same truth value in all possible interpretations I generated (sampled) by $\mathbb{P}$, starting from $x^{P(i)}$. Analogously, $\mathbb{P}, x^{P(i)} \models f \Rightarrow g$ iff $f \Rightarrow g$ is true in all possible interpretations I generated (sampled) by $\mathbb{P}$, starting from $x^{P(i)}$.*

Given a goal G and the global variables $w_q^{(i)}, iq, x^{P(i)}$, applying a rule produces a new goal G' and modifies the global variables:

1. G' is the new goal obtained from G using a kind of SLD-resolution step;

2. if a new variable r is sampled with value v,

   - set $w_q^{(i)} \leftarrow w_q^{(i)} \frac{p(r=v|x^{P(i)})}{g(r=v|x^{P(i)})}$ (based on LW) and
   - $x^{P(i)} \leftarrow x^{P(i)} \cup \{r = v\}$.

   In addition, if $r \sim= Val \in iq$ and $(r \sim= v, iq) \Leftrightarrow iq\theta$ with r grounded and $\theta = \{Val = v\}$ then:

   - $iq \leftarrow iq\theta$

3. if a new atom h is proved, set $x^{P(i)} \leftarrow x^{P(i)} \cup \{h\}$.

The inference rules applied by EvalSampleQuery resemble SLD resolution applied to a query q: they are applied with a backtracking strategy, negation as failure to prove negated literals (as in SLDNF) and tabling to improve performance. However, important differences are required to handle the stochastic nature of sampling and to exploit LW whenever possible. Note that the partial sample $x^{P(i)}$ can only grow during the application of the inference rules, i.e., backtracking does not remove sampled values from the partial sample. This is necessary to guarantee the generation of a sample from the defined proposal distribution g.

The algorithm needs the initial query to apply LW. Since the inference rules change the current goal to prove, we distinguish the current goal G from the initial query $iq = q$; during sampling iq can be simplified (e.g., applying substitutions) as long as $\mathbb{P}, x^{P(i)} \models iq \Leftrightarrow q$, i.e. iq is logically equivalent to q given $x^{P(i)}$ and the DC program $\mathbb{P}$. For those reasons $x^{P(i)}, w_q^{(i)}$, and iq are global variables.

Let us assume that the query $q = (q_1, q_2, \ldots, q_n)$ contains only equality comparisons for random variables, e.g., $r \sim= v$. Any other comparison operator $r \odot v$ can be converted in $r \sim= Val, Val \odot v$, where Val is a logical variable and $Val \odot v$ is ground during evaluation. For example, $\simeq(n) > 5$ becomes $n \sim= N, N > 5$. The inference rules are the following:

1a. If $[\exists h \in x^{P(i)} : \theta = mgu(q_1, h)]$ OR $[builtin(q_1), \exists \theta : q_1\theta]$ then:

$$(q_1, q_2, \ldots, q_n) \vdash (q_2, \ldots, q_n)\theta$$

i.e., if $q_1\theta$ is true in $x^{P(i)}$ for a substitution θ, remove q_1 from the current goal and apply the substitution θ to the current goal. q_1 can also be a built-in predicate such as $1 < 4$ that is trivially proved.

1b. If $\exists\theta$ s.t. $h \leftarrow body \in \mathbb{P}, \theta = mgu(q_1, h)$ then:

$$(q_1, q_2, \ldots, q_n) \vdash (body, add(h), q_2, \ldots, q_n)\theta$$

i.e., if q_1 unifies with the head of a deterministic clause, then add the body of the clause and $add(h)$ to the current goal, and apply substitution θ. The special predicate $add(h)$ indicates that h must be added to $x^{P(i)}$ after the body has been proven.

2a. If $\exists\theta$ s.t. $h = v \in x^{P(i)}, \theta = mgu(q_1, h \sim= v)$:

$$(q_1, q_2, \ldots, q_n) \vdash (q_2, \ldots, q_n)\theta$$

i.e., if $q_1\theta$ compares a sampled random variable h to a value and $q_1\theta$ is true in $x^{P(i)}$, then remove q_1 from the current goal and apply substitution θ.

2b. If $\exists\theta$ s.t. $h \sim \mathcal{D} \leftarrow body \in \mathbb{P}, \theta = mgu(q_1, h \sim= Val), h \notin var(x^{P(i)})$:

$$(q_1, q_2, \ldots, q_n) \vdash (body, sample(h, \mathcal{D}), q_1, q_2, \ldots, q_n)\theta$$

i.e., if q_1 compares a (not yet sampled) variable h that unifies with the head of a DC clause, then add the body of the clause and $sample(h, \mathcal{D})$ to the current goal and apply the substitution θ; $sample(h, \mathcal{D})$ is a special predicate that indicates that we need to sample h from $\mathcal{D}$ and add $h = val$ to $x^{P(i)}$ after the body has been proven. If $\mathcal{D}$ refers to random variables, they have to be added to the current goal.

3a. If $(h \sim= v) \in iq, ground(h \sim= v), h \notin var(x^{P(i)}), [\mathbb{P}, x^{P(i)} \models h \neq v \Rightarrow \neg iq]$:

$$(sample(h, \mathcal{D}), q_2, \ldots, q_n) \vdash \ (q_2, \ldots, q_n)$$

$$w_q^{(i)} \leftarrow w_q^{(i)} \cdot likelihood_{\mathcal{D}}(h = v)$$

$$x^{P(i)} \leftarrow x^{P(i)} \cup \{h = v\}$$

i.e., if $sample(h, \mathcal{D})$ is in the current goal, $h \sim= v$ is ground in iq, and $h \neq v$ makes iq false (always true if iq is a conjunction of literals), and h is not sampled in $x^{P(i)}$, then add $h = v$ to $x^{P(i)}$, weight accordingly (LW), and remove $sample(h, \mathcal{D})$ from the current goal.

3b. If $h \notin var(x^{P(i)}), ground(h)$, and rule 3a is not applicable:

$$(sample(h, \mathcal{D}), q_2, \ldots, q_n) \vdash \ (q_2, \ldots, q_n)$$

$$x^{P(i)} \leftarrow x^{P(i)} \cup \{h = v\}$$

if $h \sim= Val \in iq, ((h \sim= v, iq) \Leftrightarrow iq\gamma)$ then $iq \leftarrow iq\gamma$

with v sampled from $\mathcal{D}$, and $\gamma = \{Val = v\}$. That is, if $sample(h, \mathcal{D})$ is in the current goal, and rule 3a is not applicable, then sample h, add it to $x^{P(i)}$, and remove $sample(h, \mathcal{D})$ from the current goal. Finally, apply the substitution γ to iq iff $iq\gamma$ is equivalent to iq with $h \sim= v$ (always true if iq is a conjunction of literals).

3c. If $ground(h)$:

$$(add(h), q_2, \ldots, q_n) \vdash (q_2, \ldots, q_n)$$

$$x^{P(i)} \leftarrow x^{P(i)} \cup \{h\}$$

i.e., if $add(h)$ is in the current goal and h is ground, then add h to $x^{P(i)}$ and remove $add(h)$ from the current goal.

EvalSampleQuery performs lazy instantiation exploiting context-specific independencies: only the random variables needed to answer the query are sampled, the values of the remaining random variables are irrelevant to determine the true value of q for that specific partial instantiation.

Theorem 3.1. *For $N \to \infty$ samples generated using* EvalSampleQuery*, the estimation $\hat{p}(q)$ of $p(q)$ obtained using* (3.8) *converges with probability 1 to $p(q)$ as long as the program $\mathbb{P}$ is valid.*

Proof. It is sufficient to prove that EvalSampleQuery satisfies the importance sampling requirement for which convergence guarantees are available [Robert and Casella, 2004], that is $\forall x : p(q|x)p(x) > 0 \Rightarrow g(x) > 0$ or equivalently: $\forall x : g(x) = 0 \Rightarrow p(q|x)p(x) = 0$. The algorithm samples random variables h using the target distribution when LW is not applied (rule 3b): $g(h|x^{P(i)}) = p(h|x^{P(i)})$. LW is applied with proposal $g(h = val|x^{P(i)}) = 1$ for grounded equalities in the initial query $(h \sim= val) \in iq$ (rule 3a). Therefore, $g(h \neq val, x^{P(i)}) = 0$ but also $p(q|h \neq val, x^{P(i)})p(h \neq val, x^{P(i)}) = 0$, because the query q fails for $h \neq val$. Indeed, $\mathbb{P}, x^{P(i)} \models (h \neq val \Rightarrow \neg iq)$ as required in rule 3a, and it is easy to show that $\mathbb{P}, x^{P(i)} \models iq \Leftrightarrow q$, therefore $\mathbb{P}, x^{P(i)} \models h \neq val \Rightarrow \neg q$. The requirement is thus satisfied ($\forall x : g(x) = 0 \Rightarrow p(q|x)p(x) = 0$). □

The described inference rules extend SLD-resolution to properly handle the stochasticity of random variables and likelihood weighting. Thus, the termination conditions of EvalSampleQuery are similar to those of SLD-resolution. SLD-resolution is a complete procedure, and thus if a query is true it has a SLD refutation (proof). However, standard implementations used in Prolog always select the leftmost atom in a goal together with a depth-first search rule. In such cases termination is not guaranteed. Similar considerations are made for SLDNF resolution that handle negation as failure.

Theorem 3.1 is extendable for conditional probabilities $p(q|e) = p(q,e)/p(e)$, as long as $p(e) > 0$. The remainder of this section will consider $p(e) = 0$.

Zero probability evidence. Special considerations need to be made for queries with zero probability evidence. For example, when the evidence is $h \sim= val$ with h a continuous random variable defined with a density distribution. Such conditional distributions are not unambiguously defined, and a reformulation of the problem, e.g., a change of variables, can produce a different result (Borel-Kolmogorov paradox [Kolmogorov, 1956; Gyenis et al., 2016; Kadane, 2011]). To avoid these issues we need to make some assumptions. In this section we make an explicit distinction between probabilities P, and densities p: $\frac{d}{dx}P(X \leq x) = p(x)$. Following Kadane [Kadane, 2011], we define the conditional probability for zero probability evidence as follows:

$$\begin{aligned} P(q|e=v) &= \lim_{dv\to 0} \frac{P(q, e \in [v-dv/2, v+dv/2])}{P(e \in [v-dv/2, v+dv/2])} \\ &= \lim_{dv\to 0} \frac{\int_x \mathbb{1}(x \models q) p(x, e=v) dx dv}{p(e=v)dv} \\ &= \frac{\int_x \mathbb{1}(x \models q) p(x, e=v) dx}{p(e=v)}. \end{aligned} \tag{3.10}$$

The conditional density is thus $p(x|e=v) = \frac{p(x,e=v)}{p(e=v)}$. Definition (3.10) is extendable to more complex distributions that involve multiple variables and mixtures of discrete and continuous variables. However, if the evidence is a disjunction, the limit might not exists, e.g.,:

$$\begin{aligned} P(q|e=a \vee e=b) &= \lim_{\substack{da\to 0 \\ db\to 0}} \frac{P(q, (e\in[a-da/2, a+da/2] \vee e\in[b-db/2, b+db/2]))}{P(e\in[a-da/2, a+da/2] \vee e\in[b-db/2, b+db/2])} \\ &= \lim_{\substack{da\to 0 \\ db\to 0}} \frac{P(q, e\in[a-da/2, a+da/2]) + P(q, e\in[b-db/2, b+db/2]))}{P(e\in[a-da/2, a+da/2]) + P(e\in[b-db/2, b+db/2])} \\ &= \lim_{\substack{da\to 0 \\ db\to 0}} \frac{\int_x \mathbb{1}(x \models q) p(x, e=a) dx da + \int_x \mathbb{1}(x \models q) p(x, e=b) dx db}{p(e=a)da + p(e=b)db}. \end{aligned}$$

To avoid this issue, we assume $da = db$. In this case we obtain

$$P(q|e=a \vee e=b) = \frac{\int_x \mathbb{1}(x \models q) p(x, e=a) dx + \int_x \mathbb{1}(x \models q) p(x, e=b) dx}{p(e=a) + p(e=b)}.$$

For example, the probability of nationality given a height of $180cm$ or $160cm$ is defined as $P(nationality|height \in [180 - da/2, 180 + da/2]\ OR\ height \in$

$[160 - db/2, 160 + db/2]$) for $da \to 0, db \to 0$. The limit depends on how da and db relate to each other. Setting $da = db$ seems a reasonable assumption when the intervals are comparable quantities, but other assumptions are possible.

To apply importance sampling to definition (3.10), it is sufficient to estimate $P(e \in [v-dv/2, v+dv/2])$ and $P(q, e \in [v-dv/2, v+dv/2])$ for $dv \to 0$. Knowing that $dr \to 0 \Rightarrow P(h \in [r - dr/2, r + dr/2]) \to p(h = r)dr$, every time we apply LW to a continuous variable, $h = r$ is intended as $h \in [r - dr/2, r + dr/2]$ with $dr \to 0$, thus the incremental weight in rule 3a is $p(h = r)dr$.

Formula (3.9) needs the sum of importance weights. This has to be carefully computed when there is a mix of densities and probability masses [Owen, 2013, Chapter 9.8]. Imagine that there is a sample weight $w_1 = P(x)$ obtained assigning a discrete variable to a value, and a second sample weight $w_2 = p(y)dy$ obtained assigning a continuous variable to a value, or more precisely to a range $[y - dy/2, y + dy/2]$, with $dy \to 0$. The weight w_1 **trumps** w_2, because the latter goes to zero. Indeed, the second sample has a weight infinitely smaller than the first, and thus it is ignored in the weight sum: $w_1 + w_2 = P(x) + p(y)dy = P(x)$ (for $dy \to 0$). Analogously, a weight $w_a = p(x_1, \ldots, x_n)dx_1, \ldots, dx_n$ that is the product of n (one-dimensional) densities trumps a weight $w_b = p(x'_1, \ldots, x'_n, x'_{n+1}, \ldots, x'_{n+m})dx_1, \ldots, dx_{n+m}$ that is the product of n densities of the same variables and $m > 0$ other densities (i.e. $w_a + w_b = w_a$). If all the weights are n-dimensional densities (of the same variables), then the quantities are comparable and are trivially summed. However, if the weights refer to different variables, we need an assumption to ensure the existence of the limit, e.g., $\forall i : dx_i = dx$, thus $dx^{n+m}/dx^n \to 0$, making again n-dimensional densities trump $(n + m)$-dimensional densities. For $m = 0$ the densities are trivially summed, e.g., $w_1 = p(a, b, c)dadbdc$ and $w_2 = p(f, g, e)dfdgde$, assuming $dadbdc = dfdgde = dx^3$, we obtain $w_1 + w_2 = (p(a, b, c) + p(f, g, e))dx^3$. Finally, the ratio of weights sums in (3.9) is computed assuming again $\forall i : dx_i = dx$, obtaining $P(q|e) \approx \lim_{dx \to 0} (k_n dx^v)/(k_d dx^l)$. If $v > l$ then $P(q|e) = 0$, otherwise for $v = l$ we have $P(q|e) \approx k_n/k_d$. Those distinctions are automatically performed in EvalSampleQuery. For zero probability evidence we do not have convergence results for every DC program, query, and evidence, nonetheless the inference algorithm produces the correct results in many domains, as shown in the next section and in the experiments.

3.2.3 Examples

We now illustrate EvalSampleQuery and the cases when LW can be applied with the following example.

Example 3.4.

$$\texttt{n} \sim \texttt{uniform}([1,2,3,4,5,6,7,8,9,10]). \tag{3.11}$$

$$\texttt{color}(\texttt{X}) \sim \texttt{uniform}([\texttt{grey},\texttt{blue},\texttt{black}]) \leftarrow \texttt{material}(\texttt{X}) \sim= \texttt{metal}. \tag{3.12}$$

$$\texttt{color}(\texttt{X}) \sim \texttt{uniform}([\texttt{black},\texttt{brown}]) \leftarrow \texttt{material}(\texttt{X}) \sim= \texttt{wood}. \tag{3.13}$$

$$\texttt{material}(\texttt{X}) \sim \texttt{finite}([0.3{:}\texttt{wood}, 0.7{:}\texttt{metal}]) \leftarrow \texttt{n} \sim= \texttt{N}, \texttt{between}(1,\texttt{N},\texttt{X}). \tag{3.14}$$

$$\texttt{drawn}(\texttt{Y}) \sim \texttt{uniform}(\texttt{L}) \leftarrow \texttt{n} \sim= \texttt{N}, \texttt{findall}(\texttt{X}, \texttt{between}(1,\texttt{N},\texttt{X}), \texttt{L}). \tag{3.15}$$

$$\texttt{size}(\texttt{X}) \sim \texttt{beta}(2,3) \leftarrow \texttt{material}(\texttt{X}) \sim= \texttt{metal}. \tag{3.16}$$

$$\texttt{size}(\texttt{X}) \sim \texttt{beta}(4,2) \leftarrow \texttt{material}(\texttt{X}) \sim= \texttt{wood}. \tag{3.17}$$

We have an urn, where the number of balls n *is a random variable and each ball* X *has a color, material, and size with a known distribution. The i-th ball drawn with replacement from the urn is named* drawn(i)*. The special predicate* findall(A, B, L) *finds all* A *that makes* B *true and puts them in a list* L*. In* (3.15) L *is the list of integers from* 1 *to* N*, where* N *is unified with the number of balls.*

Let us consider the query p(color(2) $\sim=$ black), the derivation is the following (omitting $iq = q$):

$1: (\texttt{color}(2) \sim= \texttt{black}); w_q^{(i)} = 1; x^{P(i)} = \emptyset$
↓2b on (3.12) :
$2: (\texttt{material}(2) \sim= \texttt{metal}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim= \texttt{black}); w_q^{(i)} = 1; x^{P(i)} = \emptyset$
↓2b on (3.14) :
$3: (\texttt{n} \sim= \texttt{N}, \texttt{between}(1, \texttt{N}, 2), \texttt{sample}(\texttt{material}(2), \mathcal{D}_{\texttt{material}(2)}), \texttt{material}(2) \sim= \texttt{metal},$
$\texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim= \texttt{black}); w_q^{(i)} = 1;\ x^{P(i)} = \emptyset$
↓2b on (3.11) :
$4: (\texttt{sample}(\texttt{n}, \mathcal{D}_{\texttt{n}}), \texttt{n} \sim= \texttt{N}, \texttt{between}(1, \texttt{N}, 2), \texttt{sample}(\texttt{material}(2), \mathcal{D}_{\texttt{material}(2)}),$
$\texttt{material}(2) \sim= \texttt{metal}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim= \texttt{black}); w_q^{(i)} = 1; x^{P(i)} = \emptyset$
↓3b :
$5: (\texttt{n} \sim= \texttt{N}, \texttt{between}(1, \texttt{N}, 2), \texttt{sample}(\texttt{material}(2), \mathcal{D}_{\texttt{material}(2)}), \texttt{material}(2) \sim= \texttt{metal},$
$\texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim= \texttt{black}); w_q^{(i)} = 1;\ x^{P(i)} = \{\texttt{n} = 3\}$
↓2a followed by 1a
$6: (\texttt{sample}(\texttt{material}(2), \mathcal{D}_{\texttt{material}(2)}), \texttt{material}(2) \sim= \texttt{metal}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)})$
$\texttt{color}(2) \sim= \texttt{black}); w_q^{(i)} = 1;\ x^{P(i)} = \{\texttt{n} = 3\}$
↓3b :
$7: (\texttt{material}(2) \sim= \texttt{metal}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim= \texttt{black})$
$w_q^{(i)} = 1;\ x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}\}$
8 : fail, backtracking to 1

↓2b on (3.13) :

$9: (\texttt{material}(2) \sim= \texttt{wood}, \texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim= \texttt{black})$
$w_q^{(i)} = 1;\ x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}\}$
↓ 2a :
$10: (\texttt{sample}(\texttt{color}(2), \mathcal{D}_{\texttt{color}(2)}), \texttt{color}(2) \sim= \texttt{black}); w = 1; x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}\}$
↓ 3a :
$11: (\texttt{color}(2) \sim= \texttt{black}); w_q^{(i)} = 1/2;\ x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}, \texttt{color}(2) = \texttt{black}\}$
↓ 1a :
$12: \square; w_q^{(i)} = 1/2;\ x^{P(i)} = \{\texttt{n} = 3, \texttt{material}(2) = \texttt{wood}, \texttt{color}(2) = \texttt{black}\}$

The algorithm starts checking whether a rule is applicable to the current goal initialized with the query. For example, rule 2a fails because $\texttt{color}(2)$ is not in the sample $x^{P(i)}$. Rule 2b can be applied to clause (3.12), obtaining tuple 2. At this point $\texttt{material}(2)$ needs to be evaluated, it is not sampled and it unifies with the head of clause (3.14), thus applying rule 2b we obtain tuple 3. Now '$\texttt{n}$' is required, thus it is sampled with value e.g, 3 (rules 2b on (3.11) and 3b), for which '$\texttt{n} \sim= \texttt{N}, \texttt{between}(1, \texttt{N}, 2)$' succeeds for $\texttt{N} = 3$ (rule 2a and 1a). At tuple 6 the body of (3.14) has been proven; therefore, $\texttt{material}(2)$ is sampled with a value e.g., $\texttt{wood}$ (rule 3b). Now in tuple 7, the formula $\texttt{material}(2) \sim= \texttt{metal}$ fails because the sampled value is $\texttt{wood}$ (and thus the body of clause (3.12) fails); the algorithm backtracks to tuple 1 and applies rule 2b on clause (3.13) obtaining tuple 9. This time $\texttt{material}(2) \sim= \texttt{wood}$ is true and can be removed from the current goal (rule 2a). At this point $\texttt{color}(2)$ needs to be sampled (tuple 10). LW is applied because $\texttt{color}(2)$ is in the initial query $iq = q$, thus

$\texttt{color}(2) = \texttt{black}$ is added to the sample with weight 1/2 (rule 3a). The query in this sample is true with final weight 1/2.

Query expansion

Instead of asking for $\texttt{color}(2) \sim= \texttt{black}$, let us add $\texttt{a} \leftarrow \texttt{color}(2) \sim= \texttt{black}$ to the DC program and ask $p(\texttt{a})$; the query does not change. However, the described rules do not apply LW because $iq = \texttt{a}$, and $\texttt{a}$ is deterministic. It is clear that LW should be applicable also in this case. To solve this issue it is sufficient to expand iq, replacing each literal in iq by its definition; e.g., $iq = \texttt{a}$ becomes $iq = (\texttt{color}(2) \sim= \texttt{black})$. At this point the algorithm is able to apply LW as before. We can go even further, replacing $iq = (\texttt{color}(2) \sim= \texttt{black})$ with $iq = (\texttt{color}(2) \sim= \texttt{black}, (\texttt{material}(2) \sim= \texttt{metal OR material}(2) \sim= \texttt{wood}))$, where the disjunction of the bodies that define $\texttt{color}(2)$ has been added. Note that for random variables we need to keep the literal in iq after the expansion (e.g., $\texttt{color}(2) \sim= \texttt{black}$). Indeed, the disjunction of the bodies guarantees only that the random variable exists, not that it takes the value black. This procedure is a type of partial evaluation [Lloyd and Shepherdson, 1991] adapted for probabilistic DC programs. There are several ways to unfold (expand) a query, one possible way is the following. Before applying the inference rules, the initial query iq is set to the disjunction of all proofs of q without sampling: $iq = (proof_1\ OR\ proof_2\ OR\ \ldots\ OR\ proof_n)$. Each proof is determined using SLD-resolution (a number of unfolding operations); since random variables are not sampled, if the truth value of a literal cannot be determined because it is non-ground (e.g., $\texttt{N} > 0$, or $\texttt{between}(\texttt{A}, \texttt{B}, \texttt{C})$) it is left unchanged in the proof. Starting from $proof = q$, a proof can be found as follows:

e1 replace each deterministic atom $h \in proof$ with $body\theta$ if $head \leftarrow body \in \mathbb{P}$ and $\theta = mgu(h, head)$, the process is repeated recursively for $body\theta$;

e2 if the truth value of $h \in proof$ cannot be determined it is left unchanged;

e3 for each $h \sim= value \in proof$ add $body\theta$ if $head \sim \mathcal{D} \leftarrow body \in \mathbb{P}$ and $\theta = mgu(h, head)$, the process is repeated recursively for $body\theta$.

A depth limit is necessary for recursive clauses. The obtained formula can be simplified, e.g., $(a, b)\ OR\ (a, d)$ becomes $a, (b\ OR\ d)$; this is useful to know which variables can be forced to be true (in the example 'a').

After the iq expansion the sampling algorithm can start using the same inference rules, that cover the case in which iq contains disjunctions. As described in rule 3a, we can apply LW setting $h = val$, only when $\mathbb{P}, x^{P(i)} \models h \neq val \Rightarrow \neg iq$.

For example, if $iq = ((\texttt{h} \sim= \texttt{val OR a}), \texttt{b})$, LW cannot be applied for $\texttt{h} \sim= \texttt{val}$. However, if `a` becomes false, iq simplifies to $(\texttt{h} \sim= \texttt{val}, \texttt{b})$, and LW can be applied setting $\texttt{h} = \texttt{val}$, because $\texttt{h} \neq \texttt{val}$ makes iq false. To determine whether $\mathbb{P}, x^{P(i)} \models h \neq val \Rightarrow \neg iq$ holds, it is convenient to simplify iq whenever a random variable is sampled. For example, after a random variable `h` has been sampled with value `v`, $\texttt{h} \sim= \texttt{val}$ is replaced with its truth value (true or false). Furthermore, (`true OR a`) is simplified with `true`, (`false OR a`) with `a`, and so on. In the current implementation we support the propagation of simple algebraic constraints, for example, if $\texttt{Y is 2}*\texttt{X}+\texttt{1}$ has to be true (i.e. $\texttt{Y} = \texttt{2X}+\texttt{1}$) and `Y` is substituted with a number, `X` is substituted with the number $(\texttt{Y}-\texttt{1})/\texttt{2}$. If a random variable has to be equal to `X`, the proposal is chosen to satisfy the constraint as discussed above. More complex constraint propagation methods can be integrated following the same principles.

The expansion of iq and the simplification guarantee that $\mathbb{P}, x^{P(i)} \models iq \Leftrightarrow q$ as required by Theorem 3.1. The iq expansion allows to exploit LW in a broader set of cases. Indeed, it is basically a form of partial evaluation adapted for DCs and being able to exploit similar optimizations, e.g., using constraint propagation where the constraints that make the query true are propagated with the iq expansion, and updated according to the sampled variables.

Constraint propagation for algebraic constraints

Consider a set of random variables $\mathbf{X} = X_1, X_2, ...X_n$ with an order defined by the DC program $\mathbb{P}$, and an algebraic constraint $f(\mathbf{X})$ as evidence. Let us also assume that the random variables are continuous and thus real numbers.

If $f(\mathbf{X})$ is satisfied in a region $A \subset \mathbb{R}^n$, with $P(A) > 0$, sampling from $\mathbf{X}|f(\mathbf{X})$ is not a problem. Rejection sampling is still possible and LW with query expansion might be used to reduce the rejection rate. For example, when we need to sample X_i, we can ideally sample it from the optimal proposal $P(X_i = x_i|f(\mathbf{X}), X_{i-1}, ..., X_1) \propto P(X_i = x_i, f(\mathbf{X})|X_{i-1}, ..., X_1)$. If this is not possible, suboptimal solutions are still feasible, e.g., if we know a region B_i such that $P(X_i \in B_i, f(\mathbf{X})|X_{i-1}, ..., X_1) = 0$ we can sample X_i from a proposal $g(X_i)$ such that $g(B_i) = 0$. This reduces the rejection rate. For example for $f(X, Y) : X > Y$, $g(X|Y, f(X, Y))$ can be chosen to satisfy such constraint. This kind of constraints are not propagated in the current implementation, but they are relatively easy to integrate.

The more interesting case is when the constraint is a zero-probability event (but still possible), i.e. $P(f(\mathbf{X})) = 0$, with a density $p(\mathbf{X}^*)$ positive in at least a root $\mathbf{X}^*$ of $f(\mathbf{X})$. We discussed how the query $h = v$, where h is a continuous

random variable and v a value, is intended as $h = v \pm dh/2$ with $dh \to 0$. For compactness we write $h = v \pm dh/2$ to indicate $h \in [v - dh/2, v + dh/2]$.

More complex constraints are interpreted as follows. If $f(\mathbf{X})$ has only one root $x_1^*, x_2^*, ..., x_n^*$, then $f(\mathbf{X})$ is intended as $X_1 = x_1^* \pm dx_1/2, X_2 = x_2^* \pm dx_2/2, ..., X_n = x_n^* \pm dx_n/2$. In this case EvalSampleQuery sets each variable to $X_i = x_i^*$ with a weight $p(X_i = x_i^* | X_{i-1}, ..., X_1) dX_i$. Formally, the proposal distribution for variable X_i is

$$g(X_i | X_{i-1}, ..., X_1) = \begin{cases} 1 \text{ for } X_i = x_i^* \pm dX_i/2 \\ 0 \text{ otherwise} \end{cases}$$

This can be extended to any constraint with a finite number of roots. For example, when $f(\mathbf{X})$ has two roots $x_1^*, x_2^*, ..., x_n^*$ and $x_1', x_2', ..., x_n'$, $f(\mathbf{X})$ is intended as $(X_1 = x_1^* \pm dx_1/2, X_2 = x_2^* \pm dx_2/2, ..., X_n = x_n^* \pm dx_n/2)$ OR $(X_1 = x_1' \pm dx_1/2, X_2 = x_2' \pm dx_2/2, ..., X_n = x_n' \pm dx_n/2)$. In the current implementation query expansion with multiple roots is not supported, but this can be easily implemented.

The constraint $f(\mathbf{X})$ can be satisfied in a region $A \subset \mathbb{R}^n$, with $P(A) = 0$. For example, $Y - 2X - 1 = 0$ is satisfied in an infinite number of points that make a line. One variable, let us say X, can be sampled ignoring the constraint, and the other variable Y can be imposed to the value that satisfies the constraint. In general, let us assume that $X_i | X_{i-1}, ..., X_1, f(\mathbf{X})$ has only one solution $X_i = x_i^*$, then the described algorithm and query expansion implicitly assumes that $f(\mathbf{X}) | X_{i-1}, ..., X_1$ is satisfiable only for $X_i = x_i^* \pm dx_i/2$ with $dx_i \to 0$ (and eventually imposing other constraints for the remaining variables $X_{i+1}, ..., X_n$). Since the target distribution is positive only when the constraint (evidence) is satisfied, EvalSampleQuery assigns $X_i = x_i^*$ (or more formally $X_i = x_i^* \pm dx_i/2$ with $dx_i \to 0$) with weight $p(X_i = x_i^* | X_{i-1}, ..., X_1) dx_i$.

Let $\mathbf{X_{-i}} = X_1, ..., X_{i-1}, X_{i+1}, ..., X_n$, the probability of the constraint $f(\mathbf{X})$ is

$$P(f(\mathbf{X})) = \int_{\mathbf{X_{-i}}} P(f(\mathbf{X}) | \mathbf{X_{-i}}) p(\mathbf{X_{-i}}) d\mathbf{X_{-i}} =$$

$$= \int_{\mathbf{X_{-i}}} \delta X_i p(X_i = x_i^* | \mathbf{X_{-i}}) p(\mathbf{X_{-i}}) d\mathbf{X_{-i}}. \tag{3.18}$$

The formula assumes that $X_i = x_i^*$ is the only solution to $f(\mathbf{X}) | \mathbf{X_{-i}}$, and $P(f(\mathbf{X}) | \mathbf{X_{-i}}) = P(X_i = x_i^* \pm \delta X_i/2 | \mathbf{X_{-i}}) = p(X_i = x_i^* | \mathbf{X_{-i}}) \delta X_i$ for $\delta X_i \to 0$.

The constraint probability (3.18) should be the same for every i. Unfortunately, this is not the case when $\forall i : \delta X_i = \delta X_j = \delta x$ as assumed in the previous

section to make the limit properly defined. This means that the order in which the variables are sampled influences the interpretation of the constraint, and thus the limit (assuming $\forall i, j : \delta X_i = \delta X_j = \delta x$). This issue is connected with the Borel-Kolmogorov paradox. If an apparently harmless change of variable modifies the conditional probability limit, it is not surprising that assuming a common $\delta x \to 0$ the interpretation (3.18) of the constraint changes with i. There is no right or wrong assumption, unless there is a clear definition of constraint and thus a clear limit to perform. There are ways to relate δX_i and δX_j to make the constraint probability (3.18) consistent for every i. However, this might be hard in complex hybrid domains. Fortunately, this can be avoided. If we fix the order in which the random variables are sampled, and thus the index i in (3.18) we maintain a single interpretation across the samples. Moreover, if all the samples have weights with the same infinitesimal intervals (e.g., the weights are of the form $w_k = v_k dX_a, dX_b$), such intervals simplify in the weights ratio limit, and thus it is not necessarily to consider how an interval dX_a relates with dX_b.

One might also sample with different orders across samples with the simple assumption $\forall i : \delta X_i = \delta X_j = \delta x$. In this case, the estimation of $P(q|f(\mathbf{X}))$ will be some sort of average across different interpretations of the constraint $f(X)$. This can be a practical solution when a clear interpretation of a constraint is not available. In the current implementation the sampling algorithm follows the variable order defined by the DC program, however when the random variables are independent the order is chosen arbitrarily (since the DC program does not provide an order in this case). Other solutions to define and sample from conditional probabilities given algebraic constraints include the use of geometric measure theory as in [Diaconis et al., 2013], or Dirac delta [Afshar et al., 2016]. Such methods could be integrated in the DC framework in the future.

Complex queries

Example 3.5. *A more complex query is* $\simeq(\texttt{color}(2)) = \simeq(\texttt{color}(1))$*, which is converted to* $\texttt{color}(2) \sim= \texttt{Y}, \texttt{color}(1) \sim= \texttt{Y}$ *as in this way each subgoal refers to a single random variable. In this case,* $\texttt{color}(2)$ *is sampled (rule 3b: LW is not used) for example to* `red` *(after sampling* `n` *and* `material`*). Assuming* $\texttt{n} \geq 2$ *the first subgoal* $\texttt{color}(2) \sim= \texttt{Y}$ *succeeds with substitution* $\gamma = \{\texttt{Y} = \texttt{red}\}$*, thus the initial query becomes* $iq = (\texttt{color}(2) \sim= \texttt{red}, \texttt{color}(1) \sim= \texttt{red})$*. The remaining subgoal will be* $\texttt{color}(1) \sim= \texttt{red}$ *for which LW is used (rule 3a). Indeed, the initial query iq becomes grounded and LW can be applied.*

The examples show that EvalSampleQuery exploits LW in complex queries (or evidence). This is also valid for zero-probability evidence, for which standard

MCMC and naive MC fail to provide an answer (all samples are rejected). For simple evidence or queries (e.g., $\texttt{size}(1) \sim= \texttt{v}$) classical LW is sufficient to solve the problem. For more complex evidence (e.g., $\simeq(\texttt{size}(1)) = \simeq(\texttt{size}(2))$) MCMC, naive MC or classical LW will fail to provide an answer. Many probabilistic languages cannot handle those queries (if we exclude explicit approximations such as discretization). In contrast, the proposed algorithm is able to provide a meaningful answer.

Example 3.6. *Let us consider the Indian GPA problem [Perov et al.] proposed by Stuart Russell. According to Perov, Paige and Wood [Perov et al.] "Stuart Russell [...] pointed out that most probabilistic programming systems [...] produce the wrong answer to this problem". The reason for this is that contemporary probabilistic programming languages do not adequately deal with mixtures of density and probability mass distributions. Indeed, such complex distributions raise issues as the discussed Borel-Kolmogorov paradox. In addition, many languages do not propagate deterministic constraints regarding continuous random variables. This might cause the rejection of all samples, unless noise is added to relax the constraint (evidence).*

The proposed inference algorithm for DC does provide the correct results for the Indian GPA problem. The DC program that defines the domain is the following:

$$\texttt{isdensityA} \sim \texttt{finite}([0.95\texttt{:true}, 0.05\texttt{:false}]).$$

$$\texttt{agpa} \sim \texttt{beta}(8,2) \leftarrow \texttt{isdensityA} \sim= \texttt{true}.$$

$$\texttt{americanGPA} \sim \texttt{finite}([0.85\texttt{:}4.0, 0.15\texttt{:}0.0]) \leftarrow \texttt{isdensityA} \sim= \texttt{false}.$$

$$\texttt{americanGPA} \sim \texttt{val}(\texttt{V}) \leftarrow \texttt{agpa} \sim= \texttt{A}, \texttt{V is A} * 4.0.$$

$$\texttt{isdensityI} \sim \texttt{finite}([0.99\texttt{:true}, 0.01\texttt{:false}]).$$

$$\texttt{igpa} \sim \texttt{beta}(5,5) \leftarrow \texttt{isdensityI} \sim= \texttt{true}.$$

$$\texttt{indianGPA} \sim \texttt{finite}([0.1\texttt{:}0.0, 0.9\texttt{:}10.0]) \leftarrow \texttt{isdensityI} \sim= \texttt{false}.$$

$$\texttt{indianGPA} \sim \texttt{val}(\texttt{V}) \leftarrow \texttt{igpa} \sim= \texttt{A}, \texttt{V is A} * 10.0.$$

$$\texttt{nation} \sim \texttt{finite}([0.25\texttt{:america}, 0.75\texttt{:india}]).$$

$$\texttt{studentGPA} \sim \texttt{val}(\texttt{A}) \leftarrow \texttt{nation} \sim= \texttt{america}, \texttt{americanGPA} \sim= \texttt{A}.$$

$$\texttt{studentGPA} \sim \texttt{val}(\texttt{I}) \leftarrow \texttt{nation} \sim= \texttt{india}, \texttt{indianGPA} \sim= \texttt{I}.$$

Where $\texttt{h} \sim \texttt{val}(\texttt{v})$ *means that* h *has value* v *with probability* 1. *Briefly, the student GPA has a mixed distribution that depends on the student nationality.*

An interesting query is $P(\texttt{nation} \sim= \texttt{america}|\texttt{studentGPA})$. *For example, for* $\texttt{studentGPA} = 4$ *such probability is* 1. *The proposed inference algorithm provides the correct result for this query. This is due to the proper estimation of limit* (3.10) *and the relative importance weights. Moreover, the iq expansion and the generalized LW avoid rejection-sampling issues with continuous evidence.*

Example 3.7. *The last case to discuss is a query that contains random variables that are nonground terms, e.g.,* $\texttt{color(X)} \sim= \texttt{black}$, *which is interpreted as* $\exists\texttt{X}\ \texttt{color(X)} \sim= \texttt{black}$. *In this case LW is not applied because the goal is nonground. Applying LW would produce wrong results because we would force the value* `black` *only for the first proof (e.g.,* $\texttt{color(1)} \sim= \texttt{black}$*), ignoring the other possible proofs (e.g.,* $\texttt{color(2)} \sim= \texttt{black}$*), and thus violating the importance sampling requirement. In some cases, query expansion can enumerate all possible grounded proofs, making LW applicable.*

LW can also be applied for the query $\simeq(\texttt{material}(\simeq(\texttt{drawn}(1)))) = \texttt{wood}$ *(the first drawn ball is made of wood) which is converted to* $(\texttt{drawn}(1) \sim= \texttt{X}, \texttt{material(X)} \sim= \texttt{wood})$. *Once* $\texttt{drawn}(1)$ *is sampled to a value* `v` *(without LW), the substitution* $\theta = \{\texttt{X} = \texttt{v}\}$ *is applied to the current goal and to iq. At this point LW can be applied to* $\texttt{material}(\texttt{v}) \sim= \texttt{wood}$ *because it is grounded in iq (rule 3a). In other words, for the partial world* $x^{P(i)} = \{\texttt{drawn}(1) = \texttt{v}\}$ *the only value of* $\texttt{material}(\texttt{v})$ *that makes the query true is* `wood` *for which LW is applicable. For this query one sample is sufficient to obtain the exact result.*

3.3 Experiments

This section answers the following questions:

(Q1) How fast does the EvalSampleQuery algorithm converge to the correct results?

(Q2) How does the DC perform with respect to a representative state-of-the-art probabilistic programming language?

Among the several state-of-the-art probabilistic languages, BLOG [Milch et al., 2005a] is a system that shares some similarities with DC, e.g. both use lazy instantiation and likelihood weighting. For this reason, we compared the performance with BLOG.

All algorithms were implemented in YAP Prolog and run on an Intel Core i7 3.3GHz for simulations and on a laptop Core i7 for real-world experiments. To measure the error between the predicted and the exact probability for a given

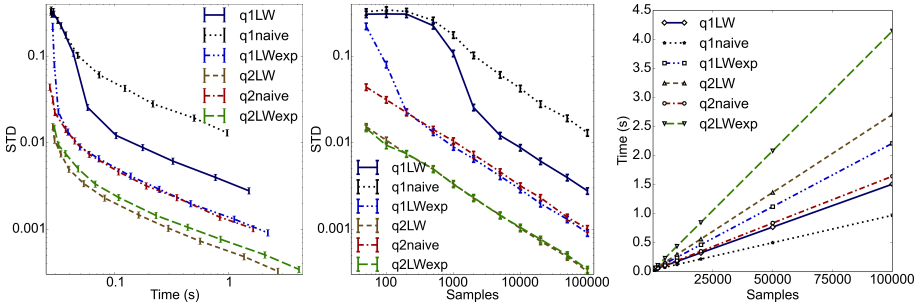
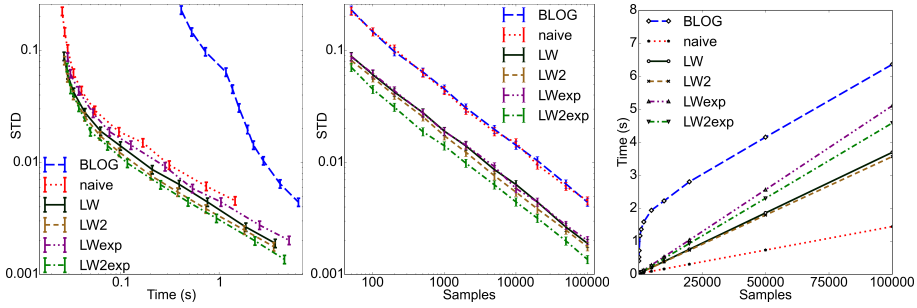


Figure 3.1: Results of EvalSampleQuery for static inference with LW and without LW (naive). For LW we show results with (LWexp) and without query expansion. The queries are $\texttt{q1} = (\texttt{drawn}(3) \sim= 10)$ with evidence $((\texttt{drawn}(1) \sim= 9, \texttt{drawn}(2) \sim= 9) \quad \texttt{OR} \quad (\texttt{drawn}(1) \sim= 10, \texttt{drawn}(2) \sim= 10))$ and $\texttt{q2} = (\texttt{drawn}(1) \sim= \texttt{X}, \texttt{drawn}(2) \sim= \texttt{X}, \texttt{color}(\texttt{X}) \sim= \texttt{black})$. The axes in (a) and (b) are in logarithmic scale. q1LWexp and q2naive partially overlap in (a) and (b); q2LWexp and q2LW overlap in (b).

query, we compute the empirical standard deviation (STD). The average used to compute STD is the exact probability when available or the empirical average otherwise. We report STD 99% confidence intervals. Notice that those intervals refer to the uncertainty of the STD estimation, not to the uncertainty of the probability. If the number of samples is not sufficient to give an answer (e.g., all samples are rejected), a value is randomly chosen from 0 to 1. The results are averaged over 500 runs. In all the experiments we measure the CPU time ("user time" in the Unix "time" command). This makes a fair comparison between DC (not parallelized in the current implementation) and BLOG that often uses more than one CPU at the time. Time includes initialization: around $0.3s$ for BLOG, $0.03s$ for DC.

We first describe experiments in static domains, then in dynamic domains (synthetic and real-world scenarios). In the **first experiment** we tested the performance of EvalSampleQuery for static inference with and without LW and query expansion (Q1) using example 3.4 in Section 3.2.3. Fig. 3.1 shows some results. The error (STD) converges to zero for all algorithms. EvalSampleQuery with LW (without query expansion) has a lower STD (Fig. 3.1b), but it is slower for the same number of samples (Fig. 3.1c). Nonetheless, the overhead is beneficial because for the same execution time the STD of LW is lower (Fig. 3.1a). Adding query expansion (Section 3.2.3) to LW has a computational cost. This is beneficial for query q1 (as defined in the caption of Fig. 3.1), allowing to exploit LW in disjunctions. Nonetheless,



(a) (b) (c)

Figure 3.2: Identity uncertainty domain used in [Milch et al., 2005b]. The axes in (a) and (b) are in logarithmic scale. LW and LWexp overlap in (b); BLOG and naive overlap in (b); LW and LW2 overlap in (c).

for query q2 the query expansion overhead is not compensated by an error reduction.

In the **second experiment** (Fig. 3.2) we considered an identity uncertainty domain[2] used in BLOG [Milch et al., 2005b]. The query considered is the probability that the second and third drawn balls are the same, given that the color of the drawn balls are respectively black, white, and white. We compared DC with BLOG. We consider several settings for DC: naive (EvalSampleQuery without LW), LW (EvalSampleQuery with LW, using formula (3.9) to compute $P(q|e)$), and LW2 where EvalSampleQuery estimates $P(q, e)$ and $P(e)$ independently using half of the available samples each. LW and LW2 are tested with and without query expansion (respectively LWexp and LW2exp). The results show that the error (STD), for the same number of samples or time, is lower for DC with LW (Fig. 3.2a-3.2b). In particular, the lowest error is obtained with LW2exp. Any DC setting is faster than BLOG (Fig. 3.2c). The latter has an unexpected logarithmic-like behavior for a small number of samples. In addition, it seems that BLOG does not use LW in this domain, indeed the error is comparable with naive Monte Carlo for the same number of samples (Fig. 3.2b). In contrast, as described in Section 3.2.3 EvalSampleQuery exploits LW in complex queries as equalities between random variables.

In the **third experiment** we considered continuous variables using Example 3.4 (Fig. 3.3). We queried the probability that the first drawn ball is made of wood, given that its size is 0.4. BLOG and naive MC failed to give an answer, while

[2]available at `https://github.com/BayesianLogic/blog/blob/master/example/balls/id-uncert-det.blog`

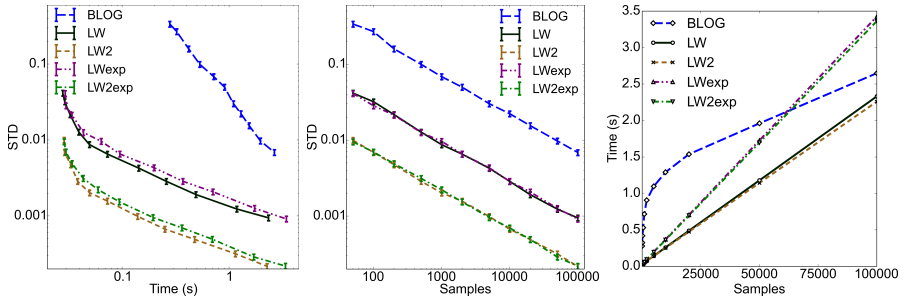


Figure 3.3: Experiments with continuous evidence. The query is the probability that the first drawn ball is made of wood, given that its size is 0.4. BLOG requires evidence discretization. LW and LWexp overlap in (b); LW2 and LW2exp overlap in (b); LWexp and LW2exp overlap in (c).

DC (LW) provides a probability. To compare with BLOG we had to consider an interval instead of a value ($[0.39, 0.41]$). Both DC and BLOG converge to 0.16, this confirms that DC works properly with continuous variables. Fig. 3.3 shows the STD and time performance. EvalSampleQuery exploits LW also in this case, while BLOG needs evidence discretization to give an answer and does not exploit LW in this case. For this reason BLOG performs poorly. In this case query expansion does not provide an improvement. Another tested query is the probability that two drawn balls are the same, given that they have the same size. This probability is one, because the size has a continuous distribution, thus the probability of having two different balls with the same size is infinitely smaller than the probability of sampling the same ball; for this reason the balls must be the same. Again, DC provides the correct result, while BLOG does not provide an answer.

In the **fourth experiment** we considered the Indian GPA problem (Example 3.6). Most probabilistic programming languages are not able to handle this domain. In contrast, DC is able to give the correct results. For instance, it provides $P(\texttt{nation} \sim= \texttt{america} \mid \texttt{studentGPA} \sim= 4) = 1 \pm 0$ and $P(\texttt{nation} \sim= \texttt{america} \mid \texttt{studentGPA} \sim= 3.9) = 0.193 \pm 0.0035$ as expected, respectively in $0.33s$ and $0.37s$ with 10000 samples. In this domain the query expansion is necessary.

From the above experiments we make the following comparison with BLOG. Given a query BLOG stacks variables that need to be sampled to answer the query, and uses LW to generate samples consistent with the evidence. This follows the lazy instantiation principle applied in EvalSampleQuery,

nonetheless there are the following differences. BLOG exploits LW only for simple evidence statements of the form `r = value`, thus it performs worse than DC with complex queries described in Section 3.2.3, which the experiments confirm. Furthermore, BLOG is not always able to give an answer for complex queries or evidence containing continuous variables as shown above. In contrast, DC gives meaningful answers and exploits LW in a much wide range of queries. Finally, BLOG seems to be less suited than DC for real-time inference, because it is generally slower with higher variance.

3.4 Related work

The proposed framework is related to other probabilistic logic languages such as IBAL [Pfeffer, 2001], PRISM [Sato and Kameya, 1997], and ProbLog [Kimmig et al., 2008], however, such languages can only describe discrete domains. The proposed inference algorithm is related to BLOG [Milch et al., 2005a] inference and to Monte-Carlo inference used in ProbLog [Kimmig et al., 2008], and to the original DC inference [Gutmann et al., 2011]. BLOG is based on LW and lazy instantiation as EvalSampleQuery. However, BLOG exploits LW only for simple evidence statements, thus it performs worse than DC with complex queries described in Section 3.2.3. Furthermore, many probabilistic languages that do handle continuous variables (e.g., Anglican [Wood et al., 2014], Church [Goodman et al., 2008] and BLOG) are not always able to give an answer for complex queries with evidence containing continuous variables as shown in the experiments (for BLOG). In those frameworks, a solution to such issue is an ad-hoc discretization (or adding noise) that is generally inefficient. In contrast, DC gives meaningful answer in those cases exploiting LW in a larger set of cases. Moreover, EvalSampleQuery exploits a type of constraint propagation to avoid rejections. Even though this idea is not new [Gogate and Dechter, 2011; Gutmann et al., 2011], it has been used in discrete domains, and not in complex hybrid domains with zero-probability evidence.

One of the few works that can handle complex constraints as evidence is [Afshar et al., 2016] that introduces a new random variable to represent the constraint with a Dirac delta distribution. Afshar et al. compute the distribution given the constraint with a collapsing mechanism (based on variable marginalization) that involves a change of variable. Such definition of (marginalized) conditional probability can be integrated in DC in the future. In addition, Afshar et al. showed how the marginalization can be performed analytically for Gibbs sampling in a broad class of domains. Our approach exploits constraint propagation and importance sampling to handle algebraic constraints. This makes our approach arguably easier to implement and to apply in a wide

range of domains, including those with an unknown number of objects, without computing integrals. Nonetheless, closed-form Gibbs sampling might have a better convergence rate on some class of domains; comparison of the approaches is a topic for future work.

Another way to define conditional probabilities with zero-probability evidence exploits geometric measure theory [Diaconis et al., 2013] and allows to sample from manifolds, i.e. from subspaces that can represent constraints. Such method can be integrated in DC in the future.

Many probabilistic programming languages (e.g., Church) adopt MCMC for inference. Such methods are applicable to DC, even though it is not easy to guarantee that the proposal generates samples consistent with the semantics of the DC program.

3.5 Conclusions

We proposed a flexible representation for hybrid relational domains and provided an efficient inference algorithm, which converges quickly to the correct results and has a limited inference cost per sample. Indeed, this framework exploits the relational representation and (context specific) independence assumptions to reduce the sample size (through partial worlds) and thus the inference cost.

Moreover, the proposed static algorithm EvalSampleQuery exploits LW in a wider range of cases with respect to systems such as BLOG, and supports complex queries with continuous variables, for which most related frameworks fail.

The proposed solution involves a practical solution for conditional probabilities with zero-probability evidence. Such solution is then applied in an importance sampling algorithm without additional ad-hoc discretizations (or noise) in the presence of zero-probability evidence, as required in other systems. Moreover, the proposed query expansion allows to perform constraint propagation and thus avoids rejections in a wide range of cases.

The inference algorithm EvalSampleQuery was empirically evaluated and applied in several experiments. The results show that EvalSampleQuery outperforms naive MC and BLOG.

EvalSampleQuery can be improved using a more advanced sampling schema. For example, adaptive importance sampling can help to cope with complex high-dimensional domains. Adaptive importance sampling learns the optimal proposal distribution using the previous samples. Under certain conditions the

proposal converges to the optimal distribution, with the estimator variance that converges to zero.

Chapter 4

Dynamic Distributional Clauses

In this chapter we extend Distributional Clauses for modeling dynamic domains (Section 4.1). Then we describe the DCPF framework to perform filtering and online learning with such language (Section 4.2). This framework has been tested in several synthetic and tracking scenarios as described in Section 4.4. This chapter consists of research previously published in the papers [Nitti et al., 2013, 2014, 2016].

4.1 Dynamic Distributional Clauses

Dynamic Distributional Clauses (DDC) is a dynamic extension of Distributional Clauses. In this framework we explicitly distinguish between the hidden state x_t, the evidence or observations z_t, and the action u_t (input), as in IOHMM. Therefore, in DDC, each predicate/variable is classified as state x, action u, or observation z, with a subscript that refers to time 0, for the initial step; time t for the current step, and $t+1$ for the next step. The definition of a discrete-time stochastic process follows the same idea of a Dynamic Bayesian Network (DBN). We need sets of clauses that define:

1. the prior distribution: $\mathtt{h}_0 \sim \mathcal{D} \leftarrow \mathtt{body}_0$,
2. the state transition model: $\mathtt{h}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{body}_{t:t+1}$ (the body involves variables at time t and eventually at time $t+1$),

3. the measurement model: $\mathtt{z}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{body}_{t+1}$, and
4. clauses that define a random variable at time t from other variables at the same time (intra-time dependence): $\mathtt{h}_t \sim \mathcal{D} \leftarrow \mathtt{body}_t$.

Obviously, deterministic clauses are allowed in the definition of the stochastic process. As these clauses are all essentially distributional clauses, the semantics remains unchanged.

Example 4.1. *Let us consider a dynamic model for the position of 2 objects: a ball and a box. For the sake of clarity we consider one-dimensional positions.*

$$\mathtt{pos(ID)}_0 \sim \mathtt{uniform}(0,1) \leftarrow \mathtt{between}(1,2,\mathtt{ID}). \tag{4.1}$$

$$\mathtt{pos(ID)}_{t+1} \sim \mathtt{gaussian}(\simeq(\mathtt{pos(ID)}_t), 0.1). \tag{4.2}$$

$$\mathtt{obsPos(ID)}_{t+1} \sim \mathtt{gaussian}(\simeq(\mathtt{pos(ID)}_{t+1}), 0.01). \tag{4.3}$$

The object positions have a uniform distribution at step 0 (4.1)*. The next position of each object is equal to the current position plus Gaussian noise* (4.2)*. In addition, the observation model* (4.3) *for each object is a Gaussian distribution centered in the actual object position. As in Example 3.1, we can also define the number of objects as a random variable.*

4.2 DCPF: A Particle Filter For Dynamic Distributional Clauses

If we consider the time step as an argument of the random variable, inference can be done as in the static case with no changes. However, the performance will degrade, while time and space complexity will grow linearly with the maximum time step considered in the query. This problem can be mitigated if we are interested in filtering. In this case we can use a particle filter as described in Section 2.3.5.

We now develop a (hybrid relational) particle filter for a set of dynamic distributional clauses that define the prior distribution, state transition model and observation model, (cf. Section 4.1). Throughout this development, we only consider the bootstrap filter for simplicity, but other proposal distributions are possible.

4.2.1 Filtering Algorithm

The basic relational particle filter applies the same steps as the classical particle filter sketched in Section 2.3.5 and employs the forward reasoning procedure for distributional clauses sketched in Section 3.1. Each sample $x_t^{(i)}$ will be a complete possible world at time t. Working with complete worlds is computationally expensive and may lead to bad performance. Therefore, we shall work with samples that are partial worlds as in the static case. The resulting framework, that we shall now introduce, is called the Distributional Clauses Particle Filter (DCPF).

Starting from a DDC program $\mathbb{P}$, weighted partial samples $\{(x_t^{P(i)}, w_t^{(i)})\}$, a new observation list $z_{t+1} = \{z_{t+1}^j = v_j\}$, and a new action u_{t+1}, the DCPF filtering algorithm performs the weighting and prediction steps from time t to time $t+1$ expanding the partial samples as shown in Fig. 4.1. The new set of samples is $\{(\hat{x}_{t+1}^{P(i)}, w_{t+1}^{(i)})\}$. The DCPF filtering algorithm is the following:

- **Step (1)**: expand the partial sample to compute $w_{t+1}^{(i)} = w_t^{(i)} p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$
- Resampling (if necessary)
- **Step (2)**: complete the prediction step (a)

Step (1) performs the weighting step (b) and implicitly (part of) the prediction step (a): it computes the weight $w_{t+1}^{(i)} = w_t^{(i)} p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$ calling EvalSampleQuery$(z_{t+1}, x_t^{P(i)})$. EvalSampleQuery will automatically sample relevant variables at time $t+1$ and t until $p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$ is computable. If there are no observations, Step (1) is skipped, and the weights remain unchanged. At this point each sample has a new weight $w_{t+1}^{(i)}$, and resampling can be performed. For the latter, we use systematic resampling [Douc and Cappé, 2005], but other methods are possible.

Step (2) performs the prediction step for variables that have not yet been sampled because they are not directly involved in the weighting step. The algorithm queries the head of any DC clause in the state transition model (intra-time clauses **excluded**), thus it evaluates the body recursively. Whenever the body is true for a substitution θ, the variable-distribution pair $r_{t+1}\theta \sim \mathcal{D}\theta$ is added to the sample. Avoiding sampling is beneficial for performance, as discussed in section 3.2.1. Step (2) is necessary to make sure that the partial sample at the previous time step t can be safely forgotten, as we shall discuss

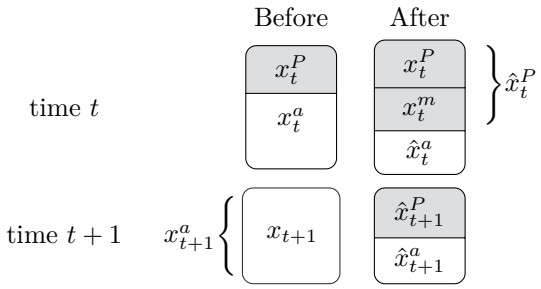


Figure 4.1: Sample partition, before (left) and after (right) the filtering algorithm. Initially x_{t+1} is not sampled, therefore $x_{t+1}^a = x_{t+1}$ and $x_{t+1}^P = \emptyset$. The inference algorithm samples variables $x_t^m \subseteq x_t^a$, $x_{t+1}^m \subseteq x_{t+1}^a$ and adds them respectively to x_t^P and x_{t+1}^P. Indeed, $\hat{x}_t^P = x_t^P \cup x_t^m$, $\hat{x}_t^a = x_t^a \setminus x_t^m$, $\hat{x}_{t+1}^P = x_{t+1}^P \cup x_{t+1}^m = x_{t+1}^m$, $\hat{x}_{t+1}^a = x_{t+1}^a \setminus x_{t+1}^m$.

in the next section. After Step (2) the set of partial samples $\hat{x}_{t+1}^{P(i)}$ with weights $w_{t+1}^{(i)}$ approximates the new belief $bel(x_{t+1})$.

In the classical particle filter resampling is the last step (c). In contrast, DCPF performs resampling before completing the prediction step (i.e., before Step (2)). This is loosely connected to auxiliary particle filters that perform resampling before the prediction step [Whiteley and Johansen, 2010]. Anticipating resampling is beneficial because it reduces the variance of the estimation. Intuitively, resampling before Step (2) makes the random variables sampled in Step (2) different between particles. In contrast, resampling after Step (2) generates exact copies of particles, which reduces the diversity among samples. For a formal discussion we refer to [Whiteley and Johansen, 2010].

To answer a query q_{t+1}, it suffices to call for each sample $(w_q^{(i)}, \hat{x}_{q_{t+1}}^{P(i)}) \leftarrow \textsc{EvalSampleQuery}(q, \hat{x}_{t+1}^{P(i)})$ and use formula (3.9) where $w_e^{(i)}$ is replaced by $w_{t+1}^{(i)}$. After querying, the partial samples $\hat{x}_{q_{t+1}}^{P(i)} \supseteq \hat{x}_{t+1}^{P(i)}$ are discarded, i.e., the partial samples remain $\hat{x}_{t+1}^{P(i)}$. This improves the performance, indeed querying does not expand $\hat{x}_{t+1}^{P(i)}$.

Example 4.2. *Consider an extension of Example 4.1, where the next position of the object after a tap action is the current position plus a displacement and plus Gaussian noise. The latter two parameters depend on its type and the material of the object below it. We consider a single axis for simplicity.*

$$\texttt{pos(ID)}_{t+1} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_t), 0.01) \leftarrow \texttt{not}(\texttt{tap(ID)}_{t+1}). \tag{4.4}$$

$$\texttt{pos(ID)}_{t+1} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_t) + 0.3, 0.04) \leftarrow$$
$$\texttt{type(ID, cube)}, \texttt{on(ID, B)}_t, \texttt{material(B, wood)}, \texttt{tap(ID)}_{t+1}. \tag{4.5}$$

$$\texttt{pos(ID)}_{t+1} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_t) + 0.2, 0.02) \leftarrow$$
$$\texttt{type(ID, cube)}, \texttt{on(ID, B)}_t, \texttt{material(B, fabric)}, \texttt{tap(ID)}_{t+1}. \tag{4.6}$$

$$\texttt{pos(ID)}_{t+1} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_t) + 1, 0.1) \leftarrow$$
$$\texttt{type(ID, ball)}, \texttt{tap(ID)}_{t+1}. \tag{4.7}$$

$$\texttt{obsPos(ID)}_{t+1} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_{t+1}), 0.01). \tag{4.8}$$

$$\texttt{type(1, cube)}. \quad \texttt{type(2, ball)}. \quad \texttt{type(3, table)}. \quad \texttt{material(3, wood)}. \tag{4.9}$$

We define $\texttt{on(A, B)}_t$ *from the z position of* A *and* B. A *is on* B *when* A *is above* B *and the distance is lower than a threshold. We omit the clause for brevity.*

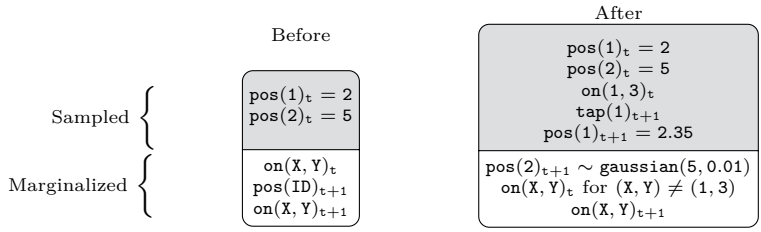


Figure 4.2: A partial sample for example 4.2, before (left) and after (right) the filtering algorithm.

To understand the filtering algorithm let us consider Step (1) for the observation $\texttt{obsPos(1)}_{t+1} \sim\!= 2.5$ *(there is no observation for object 2), and action* $\texttt{tap(1)}_{t+1}$. *Let us assume* $x_t^{P(i)} = \{\texttt{pos(1)}_t = 2, \texttt{pos(2)}_t = 5\}$. *The sample before and after the filtering step is shown in Fig.* (4.2). *The algorithm tries to prove* $\texttt{obsPos(1)}_{t+1} \sim\!= 2.5$. *Rule 2b applies for DC clause* (4.8) *that defines* $\texttt{obsPos(1)}_{t+1}$ *for* $\theta = \{\texttt{ID} = 1\}$. *The algorithm tries to prove the body and the variables in the distribution recursively, that is* $\texttt{pos(1)}_{t+1}$. *The latter is not in the sample and rule 2b applies for DC clause* (4.4) *with* $\theta = \{\texttt{ID} = 1\}$. *The proof fails, therefore it backtracks and applies rule 2b to* (4.5). *Its body is true assuming that* $\texttt{on(1, 3)}_t$ *succeeds (and added to the sample). Thus,* $\texttt{pos(1)}_{t+1}$

will be sampled from $\texttt{gaussian}(2.3, 0.04)$ *and added to the sample (rule 3b). Deterministic facts in the background knowledge, such as* $\texttt{type}(1, \texttt{cube})$*, are common to all samples; therefore, they are not added to the sample. At this point* $p(\texttt{obsPos}(1)_{\texttt{t+1}} \sim= 2.5|\hat{x}_{t+1}^{P(i)})$ *is given by* (4.8)*. This is equivalent to applying rule 3a that imposes the query to be true and updates the weight.*

Step (1) is complete, let us consider Step (2). The algorithm queries all the variables in the head of a clause in the state transition model, in this case $\texttt{pos}(\texttt{ID})_{\texttt{t+1}}$*. This is necessary to propagate the belief for variables not involved in the weighting process, such as* $\texttt{pos}(2)_{\texttt{t+1}}$*. The query* $\texttt{pos}(\texttt{ID})_{\texttt{t+1}} \sim= \texttt{Val}$ *succeeds for* $\texttt{ID} = 2$ *applying* (4.4)*, and* $\texttt{pos}(2)_{\texttt{t+1}} \sim \texttt{gaussian}(5, 0.01)$ *is added to the sample. The algorithm backtracks looking for alternative proofs of* q*, there are none, so the procedure ends. In the next step* $\texttt{pos}(2)_{\texttt{t+1}}$ *is required for* $\texttt{pos}(2)_{\texttt{t+2}}$*, so* $\texttt{pos}(2)_{\texttt{t+1}}$ *will be sampled from the distribution stored in the sample. Note that* $\texttt{on}(\texttt{A}, \texttt{B})_{\texttt{t}}$ *is evaluated selectively. Any other relation or random variable eventually defined in the program remains marginalized. For example, any relation that involves object* 2 *is not required (e.g.,* $\texttt{on}(2, \texttt{B})_{\texttt{t}}$*).*

4.2.2 Avoiding backinstantiation

We showed that lazy instantiation is beneficial to reduce the number of variables to sample and to improve the precision of the estimation in the static case. However, to evaluate a query at time t in dynamic models, the algorithm might need to instantiate variables at previous steps, sometimes even at time 0. We call this **backinstantiation**. This requires one to store the entire sampled trajectory $x_{1:t}^{P(i)}$, which may have a negative effect on performance. If we are interested in filtering, this is a waste of resources.

We shall now show that the described filtering algorithm performs lazy instantiation over time and avoids backinstantiation. We will first derive sufficient conditions for avoiding backinstantiation in DDC, and then prove that these conditions hold for the DCPF algorithm.

Rao-Blackwellization. Let us assume that the complete world at time t can be written as $x_t = x_t^P \cup x_t^a$ (Fig. 4.1). Let us consider the following factorization:

$$\begin{aligned} bel(x_t^a, x_{1:t}^P) &= p(x_t^a, x_{1:t}^P | z_{1:t}, u_{1:t}) \\ &= p(x_t^a | x_{1:t}^P, z_{1:t}, u_{1:t}) p(x_{1:t}^P | z_{1:t}, u_{1:t}) \\ &= p(x_t^a | x_{1:t}^P, z_{1:t}, u_{1:t}) bel(x_{1:t}^P) \\ &\approx \sum_{i=1}^{N} p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t}) w_t^{(i)} \delta_{x_{1:t}^{P(i)}}(x_{1:t}^P). \end{aligned}$$

In the particle filtering literature this is called Rao-Blackwellization [Doucet et al., 2000a], where $bel(x_{1:t}^P)$ is approximated as a set of weighted samples[1] $\sum_i w^{(i)} \delta_{x_{1:t}^{P(i)}}(x_{1:t}^P)$ ($\delta_v(x)$ is the Dirac delta function centered in v), while the posterior distribution of x_t^a is available in closed form.

Rao-Blackwellized particle filters (RBPF) described in the literature, adopt a fixed and manually defined split of $x_t = x_t^P \cup x_t^a$. In contrast, our approach exploits the language and its inference algorithm to perform a dynamic split that may differ accross samples, as described for the static case in Section 3.2.

Backinstantiation in the DCPF One contribution in the DCPF is that it integrates RBPF and logic programming to avoid backinstantiation over variables $r \in x_{1:t-1}$. For this reason we are interested in performing a filtering step determining the smallest partial samples that approximate the new belief $bel(x_{t+1})$ and are d-separated from the past. To avoid backinstantiation we require that $p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t})$ is a known distribution for each sample i or at most parametrized by $x_t^{P(i)}$: $p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t}) = f_t^{(i)}(x_t^a; x_t^{P(i)})$. Note that the latter equation does not make any independence assumptions: $f_t^{(i)}$ is a probability distribution that incorporates the dependency of previous states and observations and can be different in each sample.

Formally, starting from a partial sample $x_{1:t}^{P(i)}$ with weight $w_t^{(i)}$ sampled from $p(x_{1:t}^P | z_{1:t}, u_{1:t})$, a new observation z_{t+1}, a new action u_{t+1}, and the distribution $p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t})$, we look for the **smallest** partial sample $\hat{x}_{1:t+1}^{P(i)} = \{x_{1:t-1}^{P(i)}, \hat{x}_t^{P(i)}, \hat{x}_{t+1}^{P(i)}\}$ with $x_t^{P(i)} \subseteq \hat{x}_t^{P(i)}$, such that $\hat{x}_{1:t+1}^{P(i)}$ with weight $w_{t+1}^{(i)}$ is distributed as $p(\hat{x}_{1:t+1}^P | z_{1:t+1}, u_{1:t+1})$ and $p(x_{t+1}^a | \hat{x}_{1:t+1}^{P(i)}, z_{1:t+1}, u_{1:t+1})$ is

[1] The Monte-Carlo approximation replaces a distribution with an empirical distribution given by a set of (weighted) samples. If the distribution is continuous the empirical distributon is described as a sum of Dirac delta centered in the samples.

a probability distribution available in closed form. Even though the formulation considers the entire sequence $x_{1:t+1}$, to estimate $bel(x_{t+1})$ the previous samples $\{x_{1:t-1}^{P(i)}, \hat{x}_t^{P(i)}\}$ can be forgotten.

D-separation conditions. We will now describe sufficient conditions that guarantee d-separation and thus avoid backinstantiation; then we will show that these conditions hold for the DCPF filtering algorithm. The belief update is performed by adopting RBPF steps.

Starting from $x_t^{P(i)}, w_t^{(i)}, p(x_t^a | x_{1:t}^{P(i)}, z_{1:t}, u_{1:t}) = f_t^{(i)}(x_t^a; x_t^{P(i)})$ and a new observation z_{t+1}, let us expand $x_t^{P(i)}$ sampling random variables $r_t \in var(x_t^a)$ from $f_t^{(i)}(x_t^a; x_t^{P(i)})$, and $r_{t+1} \in var(x_{t+1})$ from the state transition model $p(r_{t+1}|x_t^{P(i)})$ defined by DDC clauses, until the expanded sample $\{\hat{x}_t^{P(i)}, \hat{x}_{t+1}^{P(i)}\}$ and the remaining $\hat{x}_t^a, \hat{x}_{t+1}^a$ satisfy the following **conditions**:

1. the partial interpretation $\hat{x}_{t+1}^{P(i)}$ does not depend on the marginalized variables x_t^a: $p(\hat{x}_{t+1}^{P(i)} | \hat{x}_t^{P(i)}, \hat{x}_t^a, u_{t+1}) = p(\hat{x}_{t+1}^{P(i)} | \hat{x}_t^{P(i)}, u_{t+1})$;
2. the weighting function does not depend on the marginalized variables: $p(z_{t+1}|x_{t+1}^{(i)}) = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$
3. $p(\hat{x}_{t+1}^a | \hat{x}_{1:t+1}^{P(i)}, z_{1:t+1}, u_{1:t+1}) = f_{t+1}^{(i)}(\hat{x}_{t+1}^a; \hat{x}_{t+1}^{P(i)})$ is available in closed form.

Condition 1 is a common simplifying assumption in RBPF [Doucet et al., 2000a], while condition 2 is not strictly required; however it simplifies the weighting and the computation of $f_{t+1}^{(i)}$. In some cases condition 2 can be removed, for example for discrete or linear gaussian models (using Kalman Filters). Condition 3 guarantees that the previous samples $\{x_{1:t-1}^{P(i)}, \hat{x}_t^{P(i)}\}$ can be forgotten to estimate $bel(x_\tau)$ for $\tau \geq t+1$.

Theorem 4.1. *: Under the d-separation conditions 1,2,3 the samples* $\{\hat{x}_{t+1}^{P(i)}, f_{t+1}^{(i)}(\hat{x}_{t+1}^a; \hat{x}_{t+1}^{P(i)})\}$ *with weights* $\{w_{t+1}^{(i)}\}$ *approximate* $bel(x_{t+1})$*, with*

$$w_{t+1}^{(i)} \propto p(z_{t+1}|\hat{x}_{t+1}^{P(i)}) w_t \text{ and}$$

$$f_{t+1}^{(i)}(\hat{x}_{t+1}^a; \hat{x}_{t+1}^{P(i)}) = p(\hat{x}_{t+1}^a | \hat{x}_{1:t+1}^{P(i)}, z_{1:t+1}, u_{1:t+1})$$

$$= \int_{\hat{x}_t^a} p(\hat{x}_{t+1}^a | \hat{x}_t^a, \hat{x}_t^{P(i)}, \hat{x}_{t+1}^{P(i)}) f_t^{(i)}(\hat{x}_t^a; \hat{x}_t^{P(i)}) d\hat{x}_t^a. \quad (4.10)$$

If $\hat{x}_{t+1}^a$ *does not depend on* $\hat{x}_t^a$*, then* $f_{t+1}^{(i)}(\hat{x}_{t+1}^a; \hat{x}_{t+1}^{P(i)}) = p(\hat{x}_{t+1}^a | \hat{x}_t^{P(i)}, \hat{x}_{t+1}^{P(i)})$.

Proof. The formulas are derived from RBPF (for the bootstrap filter) for which:

$$f_{t+1}^{(i)}(\hat{x}_{t+1}^{a};\hat{x}_{t+1}^{P(i)}) = \frac{1}{\eta} p(z_{t+1}|x_{t+1}^{(i)}) \int_{\hat{x}_t^a} p(\hat{x}_{t+1}^{a}|\hat{x}_t^{a},\hat{x}_t^{P(i)},\hat{x}_{t+1}^{P(i)}) f_t^{(i)}(\hat{x}_t^{a};\hat{x}_t^{P(i)}) d\hat{x}_t^{a}$$

$$w_{t+1}^{(i)} \propto \eta \cdot w_t^{(i)},$$

where $\eta = p(z_{t+1}|\hat{x}_{0:t+1}^{P(i)}, z_{1:t})$. Condition 2 makes $p(z_{t+1}|x_{t+1}^{(i)}) = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$ and $\eta = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$, simplifying the formulas. □

Step (1) in the filtering algorithm (Section 4.2.1) guarantees condition 1 and 2, while Step (2) guarantees condition 3, as described in the following theorems.

Theorem 4.2. *Given a valid DC program $\mathbb{P}$, a DC clause $\mathtt{r} \sim \mathcal{D} \leftarrow \mathtt{body}$ and a partial world $x^{P(i)}$, if there exists a grounding substitution θ such that $(\forall v : v \in var(x^{P(i)}) \Rightarrow rank(v) \leq rank(\mathtt{r}\theta))$ and $(x^{P(i)} \models \mathtt{body}\theta)$ then $p(r\theta|x^{P(i)}) = p(\mathtt{r}\theta|\mathtt{body}\theta) = \mathcal{D}\theta$.*

sketch. The proof can be obtained from the semantics of DC (see [Gutmann et al., 2011], or [Milch, 2006] for a general discussion). The result is similar to Bayesian networks for which each random variable is conditionally independent of its non-descendants given its parents. The same result is valid for context-specific independencies. □

Theorem 4.3. *Step (1) guarantees d-separation conditions 1 and 2*

Proof. We need to prove $p(\hat{x}_{t+1}^{P(i)}|\hat{x}_t^{P(i)},\hat{x}_t^{a},u_{t+1}) = p(\hat{x}_{t+1}^{P(i)}|\hat{x}_t^{P(i)},u_{t+1})$ and $p(z_{t+1}|x_{t+1}^{(i)}) = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$. The sampling algorithm can sample a random variable r_{t+1} only when the body of a clause that defines r_{t+1} is true in the partial sample, i.e., when there exists a substitution θ and a clause $\mathtt{h}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{body}_{t:t+1} \in \mathbb{P}$ such that $r_{t+1} = \mathtt{h}_{t+1}\theta$, $var(\mathtt{body}_{t:t+1}\theta) \subseteq var(\hat{x}_{t:t+1}^{P(i)})$ and $\hat{x}_{t:t+1}^{P(i)} \models \mathtt{body}_{t:t+1}\theta$. From Theorem 4.2, we know that $p(r_{t+1}|\mathtt{body}_{t:t+1}\theta,\hat{x}_t^{P(i)},\hat{x}_t^{a},u_{t+1}) = p(r_{t+1}|\mathtt{body}_{t:t+1}\theta)$ holds for each $r_{t+1} \in \hat{x}_{t+1}^{P(i)}$, this proves condition 1 since $var(\mathtt{body}_{t:t+1}\theta) \subseteq var(\hat{x}_{t:t+1}^{P(i)})$. Similarly, the sampling algorithm will sample variables that prove the evidence z_{t+1}; by applying Theorem 4.2 again we have $p(z_{t+1}|x_{t+1}^{(i)}) = p(z_{t+1}|\hat{x}_{t+1}^{P(i)})$. □

Theorem 4.4. *Step (2) guarantees d-separation condition 3.*

Proof. Condition 3 is satisfied iff the posterior distribution of the marginalized variables: $f_{t+1}^{(i)}(\hat{x}_{t+1}^{a};\hat{x}_{t+1}^{P(i)}) = \prod_{r_{t+1} \in \hat{x}_{t+1}^{a}} f_{t+1}^{(i)}(\mathtt{r}_{t+1}; Parents(\mathtt{r}_{t+1}))$ is computed for each sample. The probability distribution $f_{t+1}^{(i)}$ is represented using

DDC clauses or storing $\mathtt{r}_{t+1} \sim \mathcal{D}$ in the sample. There are two cases to discuss for each random variable $\mathtt{r}_{t+1} \in \hat{x}^a_{t+1}$:

- c1 $\mathtt{r}_{t+1}$ is defined with a grounded DC clause of the form $\mathtt{h}_{t+1}\theta \sim \mathcal{D}\theta \leftarrow \mathtt{body}_{t+1}\theta$ (derived from $\mathtt{h}_t \sim \mathcal{D} \leftarrow \mathtt{body}_t$), with $\mathtt{r}_{t+1} = \mathtt{h}_{t+1}\theta$. In this case no actions are required because in the next step $\mathtt{r}_{t+1} \to \mathtt{r}_t$ and $f^{(i)}_t(\mathtt{r}_t; Parents(\mathtt{r}_t))$ is defined by $\mathtt{h}_t\theta \sim \mathcal{D}\theta \leftarrow \mathtt{body}_t\theta$. The variable $\mathtt{r}_{t+1}$ may depend on random variables $\mathtt{d}_{t+1} \in \mathtt{body}_{t+1}\theta$ that are not yet sampled: $\mathtt{d}_{t+1} \in \hat{x}^a_{t+1}$. In this case, if $\mathtt{d}_{t+1}$ is defined by intra-time clauses of the form $\mathtt{d}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{dbody}_{t+1}$, the case c1 applies recursively. If $\mathtt{d}_{t+1}$ is defined by inter-time clauses: $\mathtt{d}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{dbody}_{t:t+1}$, the case c2 applies.
- c2 $\mathtt{r}_{t+1}$ is defined with a grounded DC clause of the form $\mathtt{h}_{t+1}\theta \sim \mathcal{D}\theta \leftarrow \mathtt{body}_{t:t+1}\theta$ (derived from the state transition model) with $\mathtt{r}_{t+1} = \mathtt{h}_{t+1}\theta$. In this case Step (2) queries $\mathtt{r}_{t+1}$, thus variables in $\mathtt{body}_{t:t+1}\theta$ will be eventually sampled if not in $\hat{x}^{P(i)}_{t:t+1}$. If $\mathtt{body}_{t:t+1}\theta$ is true in $\hat{x}^{P(i)}_{t:t+1}$, Step (2) adds $\mathtt{r}_{t+1} \sim \mathcal{D}\theta$ is to the sample. Indeed, from Theorem 4.1 we have $f^{(i)}_{t+1}(\mathtt{r}_{t+1}; Parents(\mathtt{r}_{t+1})) = p(\mathtt{r}_{t+1} | \hat{x}^{P(i)}_t, \hat{x}^{P(i)}_{t+1}) = \mathcal{D}\theta$.

□

The case c1 implies that Step (2) does not need to query variables defined by intra-time clauses $\mathtt{h}_{t+1} \sim \mathcal{D} \leftarrow \mathtt{body}_{t+1}$. Querying those variables would sample unnecessary random variables. For the remaining variables, Step (2) performs the prediction step, that is, determines the distribution of such variables (case c2). If lifted belief update or precomputed belief are used for a random variable r_{t+1}, the marginal distribution $f^{(i)}_{t+1}$ of such variable is defined by DDC clauses. In conclusion, c1 and c2 show that Step (2) guarantees that the distributions of the marginalized variables are defined in any situation.

EvalSampleQuery used in Step (1) and (2) will never need to sample variables at time $t-1$ or before, because the belief distribution of marginalized variables $r_t \in \hat{x}^a_t$ is $f^{(i)}_t(x^a_t; \hat{x}^{P(i)}_t)$, available in closed form and (eventually) parameterized by $\hat{x}^{P(i)}_t$, while $r_{t+1} \in \hat{x}^a_{t+1}$ are sampled from the state transition model. After Step (2) any $r_{t+1} \in \hat{x}^a_{t+1}$ is derivable from $\hat{x}^{P(i)}_{t+1}$ together with the DDC program. These conditions avoid backinstantiation during filtering or query evaluation, thus previous partial states $x^{P(i)}_{0:t}$ can be forgotten.

Step (2) avoids computing the integral (4.10). The integral is approximated with a single sample, or equivalently the partial sample is expanded until $\hat{x}^a_{t+1}$ does not depend on marginalized $\hat{x}^a_t$, for which $f^{(i)}_{t+1}(\hat{x}^a_{t+1}; \hat{x}^{P(i)}_{t+1}) =$

$p(\hat{x}^a_{t+1}|\hat{x}^{P(i)}_t, \hat{x}^{P(i)}_{t+1})$ is derivable from the DDC program. In detail, for each $r_{t+1}\theta \in \hat{x}^a_{t+1}$ Step (2) stores $r_{t+1}\theta \sim \mathcal{D}\theta$, where $\mathcal{D}\theta = p(r_{t+1}\theta|\hat{x}^{P(i)}_t, \hat{x}^{P(i)}_{t+1})$, e.g., $\texttt{pos(2)}_{\texttt{t+1}} \sim \texttt{Gaussian}(5, 0.01) = f^{(i)}_{t+1}(\texttt{pos(2)}_{\texttt{t+1}})$ as shown in Fig. 4.2. Such distribution is not parametrized and it is generally different in each sample. In contrast, $p(\texttt{size(A)}_\texttt{t}|x^{P(i)}_{1:t}, z_{1:t}, u_{1:t}) = \mathit{Gaussian}([\text{if } \texttt{type(A, ball)} \text{ then } \mu = 1; \text{ else } \mu = 2], 0.1) = f^{(i)}_t(\texttt{size(A)}_\texttt{t}; x^{P(i)}_t)$ is a distribution parametrized by other variables in $x^{P(i)}_t$. It is sufficient to store this parametric function once, using DDC clauses, instead of storing each distribution separately for each sample. Similarly, the DDC clause that defines $\texttt{on(A, B)}_\texttt{t}$ from object positions at time t is sufficient to represent all marginalized facts $\texttt{on(a, b)}_\texttt{t}$ not in $x^{P(i)}_t$. In general, intra-time DDC clauses can represent distributions of marginalized variables for an unspecified number of objects. Thus, Step (2) does not need to query variables defined by intra-time clauses, as proved in Theorem 4.4.

Step (2) could be improved applying (4.10) whenever possible. Moreover, if there is a set of variables that has the same prior and transition model, the belief update (4.10) can be performed once for the whole set. Whenever a variable is required, it will be sampled. This does not require to bounding the number of such sets of variables, and it can be considered as a simple form of lifted belief update. In some cases the belief $f^{(i)}_t(x^a_t; x^{P(i)}_t)$ can be directly specified for any time t, we call this precomputed belief. As lifted belief update, this is applicable to an unbounded set of variables, and avoids unnecessary sampling.

Theorem 4.5. *DCPF has a space complexity per step and sample bounded by the size of the largest partial state $x^{P(i)}_t$, together with $f^{(i)}_t(x^a_t; x^{P(i)}_t)$.*

Proof sketch. The filtering algorithm proposed for DCPF avoids backinstantiation, therefore the space complexity is bounded by the dimension of the state space at time t. A tighter bound is the size of the largest partial state. □

4.2.3 Comparison with Murphy's interface algorithm

Murphy [Murphy, 2002] proposed the interface algorithm for Dynamic Bayesian Networks to perform efficient exact filtering. It is based on the notion of interface: the set of variables that have children in the next time slide.

In DCPF, we can define the interface as the set of random variables that appears in the body of a clause of the state transition model. The interface is sufficient to d-separate the future from the past. Thus, Step (2) can perform the prediction step only for random variables in the interface. However, to query a non-interface variable h_t, the partial sample $x^{P(i)}_{t-1}$ is required in addition to

$x_t^{P(i)}$ (while $x_{0:t-2}^{P(i)}$ can be forgotten). This is because the prediction step is not performed for non-interface variables. Unfortunately, the interface is fixed and does not consider context-specific independencies, therefore the non-interface set might be empty or small in several domains.

For the described reasons the interface concept is less appealing for inference optimization in DCPF. Therefore, the interface is not exploited in the current implementation. Nonetheless, some domains might benefit of this improvement.

4.2.4 Limitations

We will now describe the limitations of the proposed algorithms.

Lazy instantiation is beneficial only when there are facts or random variables that are irrelevant during inference. For example, this is true when the model includes background knowledge that is not entirely required for a query. In fully-connected models or when the entire world is relevant for a query, lazy instantiation is useless. Nonetheless, the proposed method generalizes LW, thus it can be beneficial even in the described worse cases.

The above issues apply also to inference in dynamic models, but the latter raises additional issues for filtering in particular. One is the curse of dimensionality that produces poor results for high-dimensional state spaces, or equivalently it requires a huge number of samples to give acceptable results. There are some solutions in the literature (e.g., factorising the state space [Ng et al., 2002]). To make the particle filter more efficient, the optimal proposal distribution $p(x_{t+1}|x_t, z_{t+1})$ and the corresponding weight $p(z_{t+1}|x_t)$ can be used. Given a complex nonlinear transition model those distributions are not easy to compute analytically, therefore in DCPF we adopt suboptimal solutions.

4.3 Online Parameter Learning

So far we assumed that the model used to perform state estimation is known. In practice, it may be hard to determine or to tune the parameters manually, and therefore the question arises as to whether we can learn them. We will first review online parameter learning in classical particle filters, then we will show how to adapt those methods in DCPF.

4.3.1 Learning in Particle Filters

A simple solution to perform state estimation and parameter learning with particle filters consists of adding the static parameters θ to the state space: $\bar{x}_t = \{x_t, \theta\}$. The posterior distribution $p(\bar{x}_t|z_{1:t})$ is then described as a set of samples $\{x_t^{(i)}, \theta^{(i)}\}$. However, this solution produces poor results due to the following **degeneracy problem**. As the parameters are sampled in the first step and left unchanged (since they are static variables), after a few steps the parameter samples $\theta^{(i)}$ will degenerate to a single value due to resampling. This value will remain unchanged regardless of incoming new evidence. Limiting or removing resampling is not a good solution, because it will produce poor state estimation results. Better strategies have been proposed and are summarized in [Kantas et al., 2009]. We focus on two simple techniques with limited computational cost: artificial dynamics [Higuchi, 2001] and resample-move [Gilks and Berzuini, 2001]. Both methods introduce diversity among the samples to solve the described degeneracy problem.

The first method adds **artificial dynamics** to the parameter θ: $\theta_{t+1} = \theta_t + \epsilon_{t+1}$, where ϵ_{t+1} is artificial noise with a small and decreasing variance over time. This strategy has the advantage to be simple and fast, nonetheless it modifies the original problem and requires tuning [Kantas et al., 2009]. We will show that this technique is suitable for the scenarios considered in this thesis (for a limited number of parameters).

The second method is **resample-move** that adds a single MCMC step to rejuvenate the parameters in the samples. There are several variants of this technique, the most notable are Storvik's filter [Storvik, 2002] and Particle Learning [Carvalho et al., 2010a]. To understand these approaches, consider the following factorization of the joint distribution of interest:

$$\begin{aligned} p(x_{0:t}, \theta|z_{1:t}) &= \frac{p(x_{0:t}, \theta, z_t|z_{1:t-1})}{p(z_t|z_{1:t-1})} \\ &= p(\theta|x_{0:t-1}, z_{1:t-1}) p(x_{0:t-1}|z_{1:t-1}) \frac{p(z_t|x_t, \theta) p(x_t|x_{t-1}, \theta)}{p(z_t|z_{1:t-1})}. \end{aligned}$$

In addition to the standard propagation and weighting steps, both algorithms perform a Gibbs sampling step that samples a new parameter value $\theta_t^{(i)}$ from the distribution $p(\theta|x_{0:t}^{(i)}, z_{1:t}) = p(\theta|s_t^{(i)})$ where $s_t^{(i)}$ captures the sufficient statistics

of the distribution. $p(\theta|x_{0:t}, z_{1:t})$ is recursively updated as follows:

$$\begin{aligned} p(\theta|s_t) &= p(\theta|x_{0:t}, z_{1:t}) \\ &\propto p(z_t, x_t, \theta|x_{0:t-1}, z_{1:t-1}) \\ &= p(\theta|x_{0:t-1}, z_{1:t-1})p(z_t|x_t, \theta)p(x_t|x_{t-1}, \theta) \\ &= p(\theta|s_{t-1})p(z_t|x_t, \theta)p(x_t|x_{t-1}, \theta) \end{aligned} \tag{4.11}$$

This leads to a deterministic sufficient statistics update $s_t = S(s_{t-1}, x_t, x_{t-1}, z_t)$. **Storvik's filter** algorithm goes as follows:

- propagate: $x_t^{(i)} \sim g(x_t|x_{t-1}^{(i)}, \theta_{t-1}^{(i)}, z_t)$,
- resample samples with weights: $w_t^{(i)} = \frac{p(z_t|x_t^{(i)}, \theta_{t-1}^{(i)})p(x_t^{(i)}|x_{t-1}^{(i)}, \theta_{t-1}^{(i)})}{g(x_t^{(i)}|x_{t-1}^{(i)}, \theta_{t-1}^{(i)}, z_t)}$,
- propagate sufficient statistics: $s_t^{(i)} = S(s_{t-1}^{(i)}, x_t^{(i)}, x_{t-1}^{(i)}, z_t^{(i)})$, and
- sample $\theta_t^{(i)} \sim p(\theta|s_t^{(i)})$.

The Particle Learning [Carvalho et al., 2010a] is an optimization of Storvik's filter since it adopts the auxiliary particle filter [Pitt and Shephard, 1999] and an optimal proposal distribution g. Resample-move strategies do not change the problem as in the artificial dynamics, and have been proven to be successful for several classes of problems [Carvalho et al., 2010a,b; Lopes et al., 2010]. However, they suffer from the sufficient statistics degeneracy problem that can produce an increasing error in the parameter posterior distribution [Andrieu et al., 2005].

4.3.2 Online Parameter Learning for DCPF

We now propose an integration of the mentioned learning methods in DCPF. The main contribution is to adapt artificial dynamics and the Storvik's filter for DCPF and allow learning of a number of parameters defined at run-time. Indeed, the relational representation allows to define an unbounded set of parameters to learn, e.g., the size of each object `size(ID)`. The number of objects and thus parameters to learn is not necessarily known in advance.

Artificial dynamics in DCPF. To describe the integration of artificial dynamics in DCPF we consider an object tracking scenario called Learnsize (Section 4.4.3),

in which the parameters to learn are the sizes of all objects. We defined a uniform prior: $\texttt{size(ID)}_0 \sim \texttt{uniform}(0, 20)$. Since the number of objects is not defined in advance we can directly define the size distribution at time t for any $\texttt{size(ID)}_\texttt{t}$ not yet sampled: $\texttt{size(ID)}_\texttt{t} \sim \texttt{uniform}(0, 20)$ (i.e., $f_t^{(i)}(x_t^a; x_t^{P(i)})$ is directly defined for not sampled $\texttt{size(ID)}_\texttt{t}$). Whenever the size of an object $\texttt{x}$ (not yet sampled) is needed for inference, $\texttt{size(x)}_\texttt{t}$ is sampled for the above rule with no need to perform backinstantiation. While the transition model defines the artificial dynamics: $\texttt{size(ID)}_{\texttt{t}+1} \sim \texttt{gaussian}(\simeq(\texttt{size(ID)}_\texttt{t}), \bar{\sigma}^2/\texttt{T}^\texttt{x})$, where $\texttt{T}$ is the time step, $\texttt{X}$ is a fixed exponent (set to 1 in the experiments) and $\bar{\sigma}^2$ is a constant that represents the initial variance. Initially the variance is high, thus the particle filter can "explore" the parameter space, after some steps the variance decreases in the hope that the parameter converges to the real value.

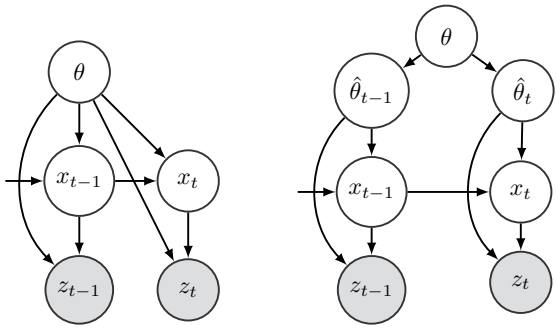


Figure 4.3: Left: HMM-like dynamic model parameterized by θ. Right: modified version used to apply a Storvik's filter variant in DCPF.

Storvik's filter in DCPF. Equation (4.11) shows how to update the parameter posterior and then the sufficient statistics for each sample. However, this formulation needs a conjugate prior for the parameter likelihood. For a complex distribution this may be hard. We developed a variant of Storvik's filter to overcome this problem. In detail, we add $\hat{\theta}_t$ to the state that represents the currently sampled "parameter" value, while θ is the parameter to estimate, e.g., the mean of $\hat{\theta}_t$ (Fig. 4.3 on the right). We also assume the state x_t and the observations depend only on $\hat{\theta}_t$: $p(x_t|x_{t-1}, \hat{\theta}_t, \theta) = p(x_t|x_{t-1}, \hat{\theta}_t)$ and

$p(z_t|x_t, \hat{\theta}_t, \theta) = p(z_t|x_t, \hat{\theta}_t)$. Thus, the posterior becomes:

$$p(x_{0:t}, \theta, \hat{\theta}_{0:t}|z_{1:t}) \propto p(z_t, x_{0:t}, \theta, \hat{\theta}_{0:t}|z_{1:t-1}) =$$

$$= p(z_t, x_t, \hat{\theta}_t, \theta, \hat{\theta}_{0:t-1}, x_{0:t-1}|z_{1:t-1}) =$$

$$= p(z_t|x_t, \hat{\theta}_t)p(x_t|x_{t-1}, \hat{\theta}_t)p(\hat{\theta}_t|\theta)p(\theta|\hat{\theta}_{0:t-1})p(\hat{\theta}_{0:t-1}, x_{0:t-1}|z_{1:t-1}).$$

Knowing that $p(\theta|\hat{\theta}_{0:t}, x_{0:t}, z_{1:t}) \propto p(x_{0:t}, \theta, \hat{\theta}_{0:t}|z_{1:t})$ we replace (4.11) with:

$$p(\theta|\hat{\theta}_{0:t}, x_{0:t}, z_{1:t}) \propto p(\hat{\theta}_t|\theta)p(\theta|\hat{\theta}_{0:t-1}) = p(\hat{\theta}_t|\theta)p(\theta|s_{t-1}).$$

Thus $p(\theta|\hat{\theta}_{0:t}, x_{0:t}, z_{1:t}) = p(\theta|\hat{\theta}_{0:t}) = p(\theta|s_t)$. At this point we can avoid sampling θ as required by the Storvik's filter, but sample $\hat{\theta}_t$ from the marginal distribution: $p(\hat{\theta}_t|s_{t-1}) = \int_\theta p(\hat{\theta}_t|\theta)p(\theta|s_{t-1})d\theta$.

For example, in the Learnsize scenario, for each object `ID` we have $\hat{\theta}_t = \texttt{cursize(ID)}_\texttt{t}$ and the parameter to learn is $\theta = \texttt{size(ID)}$. For each object $p(\hat{\theta}_t|\theta)$ is defined as $\texttt{cursize(ID)}_\texttt{t} \sim \texttt{Gaussian}(\simeq(\texttt{size(ID)}), \bar{\sigma}^2)$, where $\bar{\sigma}^2$ is a fixed variance. The conjugate prior of $\texttt{size(ID)}$ is a Gaussian with hyperparameters $\mu(\texttt{ID})_0, \sigma^2(\texttt{ID})_0$: $\texttt{size(ID)} \sim \texttt{Gaussian}(\mu(\texttt{ID})_0, \sigma^2(\texttt{ID})_0)$. As explained $\theta = \texttt{size(ID)}$ need not be sampled, indeed $\hat{\theta}_t = \texttt{cursize(ID)}_\texttt{t}$ is directly sampled from $p(\hat{\theta}_t|s_{t-1})$, i.e. $\texttt{cursize(ID)}_\texttt{t} \sim \texttt{Gaussian}(\mu(\texttt{ID})_{\texttt{t}-1}, \sigma^2(\texttt{ID})_{\texttt{t}-1} + \sigma^2)$. For each `ID` the posterior $p(\theta|s_t)$ is a Gaussian as the prior, and the sufficient statistics $s_t = \mu(\texttt{ID})_\texttt{t}, \sigma^2(\texttt{ID})_\texttt{t}$ are computed using Bayesian inference. The posterior distribution of the parameters can become peaked in few steps, causing again a degeneration problem. This issue is mitigated reducing the influence of the evidence during the Bayesian update.

4.4 Experiments

This section answers the following questions:

(Q1) How do the DCPF and the classical particle filter compare?

(Q2) How does the DCPF perform with respect to a representative state-of-the-art probabilistic programming language for dynamic domains?

(Q3) Is the DCPF suitable for real-world applications?

(Q4) How do the learning algorithms perform?

Among the several state-of-the-art probabilistic languages, BLOG [Milch et al., 2005a] is a system that shares some similarities with DC and DCPF. For this reason, we compared the performance with DBLOG [de Salvo Braz et al., 2008], an extension of BLOG for temporal domains.

All algorithms were implemented in YAP Prolog and run on an Intel Core i7 3.3GHz for simulations and on a laptop Core i7 for real-world experiments. To measure the error between the predicted and the exact probability for a given query, we compute the empirical standard deviation (STD). The average used to compute STD is the exact probability when available or the empirical average otherwise. We report STD 99% confidence intervals. Notice that those intervals refer to the uncertainty of the STD estimation, not to the uncertainty of the probability. If the number of samples is not sufficient to give an answer (e.g., all samples are rejected), a value is randomly chosen from 0 to 1. The results are averaged over 500 runs. In all the experiments we measure the CPU time ("user time" in the Unix "time" command). This makes a fair comparison between DCPF (not parallelized in the current implementation) and DBLOG that often uses more than one CPU at the time. Time includes initialization: around $0.3s$ for DBLOG, $0.03s$ for DCPF.

We first describe experiments in synthetic domains, then in real-world scenarios.

4.4.1 Synthetic dynamic domains

We now answer questions Q1 and Q2 comparing the classical particle filter, DCPF and DBLOG in dynamic domains. In all dynamic experiments we disabled the query expansion because it is not necessary.

In this section we used a probabilistic Wumpus world (inspired by [Russell and Norvig, 2009]). This is a discrete world with a two-dimensional grid of cells, that can be either free, a wall, or a pit. In one of the cells the horrible wumpus lives and each cell can contain gold. Each pit produces a breeze in the neighboring cells, and the wumpus produces a stench in the neighboring cells. The agent has to estimate the hidden state consisting of the wumpus' location, the state of each cell (free, wall or pit), as well as its own position in the maze. The agent has four stochastic 'move' actions: up, down, left, right, which change the position by 1 cell or lead to no change with a particular probability. Furthermore, the agent has noisy sensors to observe whether there is a breeze, a stench, or gold in the cell, and whether there are walls in the neighboring cells. We assume that the agent starts from position (0,0), therefore the cell (0,0) is free.

In the Wumpus domain we use a lifted belief update or precomputed

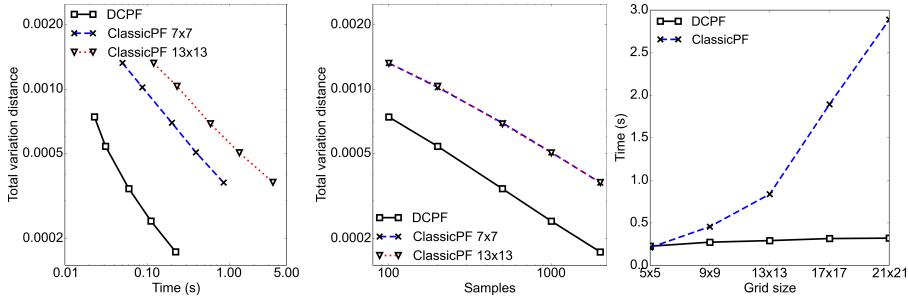
belief. For example, if the belief at time t for each cell not in the partial sample is $\texttt{maze}(\texttt{X},\texttt{Y})_\texttt{t} \sim \texttt{finite}([\texttt{0.6:free}, \texttt{0.4:wall}])$ and the state transition is $\texttt{maze}(\texttt{X},\texttt{Y})_{\texttt{t+1}} \sim \texttt{val}(\simeq(\texttt{maze}(\texttt{X},\texttt{Y})_\texttt{t}))$ (the next cell state is equal to the current cell state), the belief update can be done once for all the cells that have not been sampled yet. In this case the belief remains the same over time, thus, we can directly define the belief at time t without doing lifted belief update. If a cell $\texttt{maze}(\texttt{x},\texttt{y})_\texttt{t}$ is required it will be sampled, and the belief update for this cell will be performed by sampling. For non-sampled cells $\texttt{maze}(\texttt{X},\texttt{Y})_\texttt{t} \sim \texttt{finite}([\texttt{0.6:free}, \texttt{0.4:wall}])$ is used instead. Lifted belief update and precomputed belief do not require that the size of the grid is specified.

Classical Particle Filter (Q1). The classical particle filter samples the entire state every step with a forward reasoning procedure.

In the **first experiment** (Wumpus1) there are no pits and no wumpuses. The goal is to compute the joint distribution of 3 cells state ($\texttt{maze}(\texttt{0},\texttt{2})_\texttt{t}$, $\texttt{maze}(\texttt{1},\texttt{1})_\texttt{t}$, $\texttt{maze}(\texttt{2},\texttt{0})_\texttt{t}$) given noisy gold and cell observations. The experiment consists of 3 steps (to keep exact inference feasible) with one or two observations per step. In the classical particle filter, we need to limit the size of the grid in advance. In contrast, the DCPF estimates the borders of the maze itself. To measure the error between the predicted and the exact posteriors we use the total variation distance (i.e., the sum of absolute differences averaged over runs). Figures 4.4a and 4.4b show that both algorithms provide correct results, but our DCPF produces lower errors and is faster when compared to the classical particle filter for the same number of samples (Figure 4.4b). This is because DCPF reduces the number of tracked variables and therefore reduces both the variance in the sampling process and the execution time. As expected, the maze size of the classical particle filter affects the performance. The DCPF by contrast does not require a fixed grid size using a precomputed belief or lifted belief update. This makes DCPF more flexible and faster in comparison (Q1).

In the **second experiment** (Wumpus2) we used a wumpus world with one wumpus that produces stench sensed with a noisy sensor. We also model the agent's energy as a continuous variable that decreases over time with Gaussian noise. The probability to move is a function of the energy. We randomly generated worlds of different sizes (2 worlds per size, 5 runs per world), together with a sequence of 100 actions and observations (neighboring cells state, stench and energy). The sequence of actions and observations was used as the input to the particle filter (classic or DCPF) with 1000 samples. The model is too complex to compute exactly, therefore we focused on runtime evaluation. The results (Figure 4.4c) show that DCPF is faster than the classical particle filter

(Q1), because it reduces the sample size. For completeness, in the last step the average number of sampled variables in DCPF was 30.8, 40.6, 51.8, 53.1, 61.4 respectively for worlds 5x5, 9x9, 13x13, 17x17, 21x21.



(a) Wumpus1 (b) Wumpus1 (c) Wumpus2

Figure 4.4: Experiments. Total variation distance as a function of run-time (a) or the number of samples (b) for the probabilistic wumpus example. (c) Run-time for various grid sizes with 1000 samples. (a) and (b) are in logarithmic scale.

DBLOG (Q2). Because DBLOG cannot fully cope with continuous evidence as DCPF, we now focus on a discrete case for a further comparison.

In the **third experiment** (Wumpus3) we used a Wumpus domain similar to Wumpus1 to compare DCPF with DBLOG. We executed 10 steps and queried $q = (\texttt{maze}(2,1)_t \sim= \texttt{free}, \texttt{maze}(0,1)_t \sim= \texttt{free}, \texttt{maze}(1,0)_t \sim= \texttt{free}, \texttt{gold}(1,1)_t \sim= \texttt{true})$; the error and time performance are shown in Fig. 4.5. The results highlight that DCPF has a slightly lower error than DBLOG for the same number of samples (Fig. 4.5b). In addition DCPF is faster for a small number of samples (Fig. 4.5c). For a large number of samples DBLOG becomes faster, probably because of the unoptimized prolog implementation of DCPF.

The **fourth experiment** (Wumpus4) is similar to the previous one (Wumpus3) where the evidence contains statements such as the observed cell on the left is equal to the observed cell of the right. The results are shown in Fig. 4.6 and highlight that DCPF has a lower error for the same number of particles (Fig. 4.6b) because DCPF exploits LW with such complex evidence.

DBLOG does not currently implement Step (2) of our filtering algorithm[2]. This requires a workaround that consists in manually querying all the variables that

[2]According to the following bug report `https://github.com/BayesianLogic/blog/issues/330`

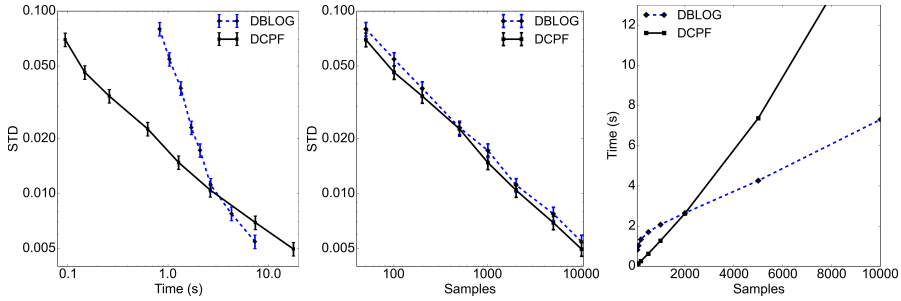


Figure 4.5: Experiments (Wumpus3). Time refers to 10 steps. The axes in (a) and (b) are in logarithmic scale. For STD there is a 99% confidence interval.

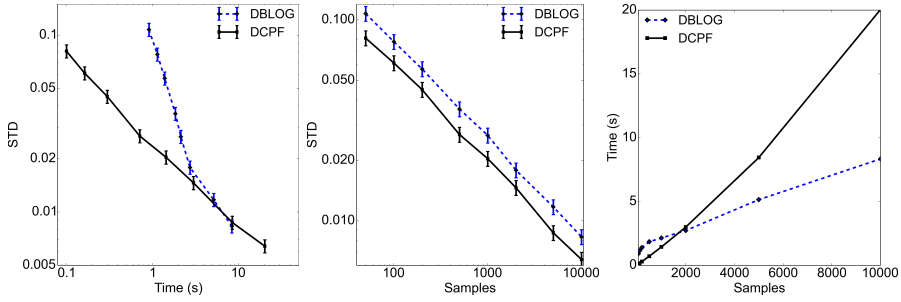


Figure 4.6: Wumpus experiments with complex evidence (Wumpus4). The axes in (a) and (b) are in logarithmic scale. Time refers to 10 steps. For STD there is an 99% confidence interval.

might be relevant for the given query, thus to fix the number of random variables (e.g., the size of the maze). In Wumpus3 and Wumpus4 this is avoided because the cell variables are static and they do not require belief updates. If the cell state changes over time, DBLOG needs to query all the the cells at each step. This makes DBLOG equivalent to a classical particle filter, where the size of the maze is fixed and sampled entirely. Thus, DBLOG becomes slow with high variance, while for DCPF we exploit lifted belief update. This is shown in the **fifth experiment** (Wumpus5, Fig. 4.7) where the cells state changes over time. It is problematic for DBLOG when the worlds may contain different numbers of random variables. It is not trivial to determine in advance which variables are relevant for a given query at time t. Indeed, every sample will have different variables, thus propagation has to be performed in a different way for each

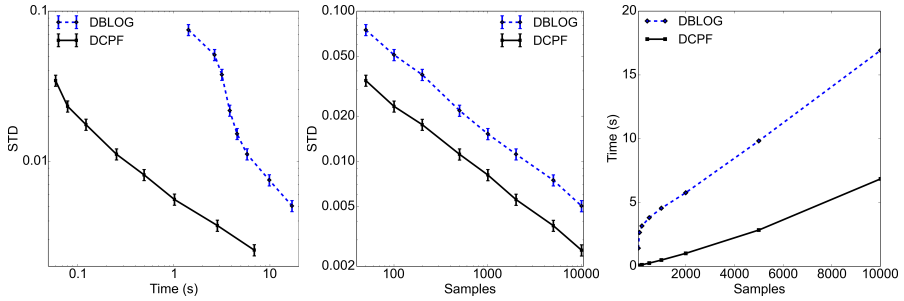


Figure 4.7: Wumpus experiments with changing cells (Wumpus5). Time refers to 4 steps. For DBLOG the rows and columns cells from $[-3, 3]$ has been queried at each step. The axes in (a) and (b) are in logarithmic scale.

sample. In contrast, Step (2) in DCPF samples variables that are sufficient to guarantee d-separation, and those variables can be different in different samples because of context-specific independences. This avoids backinstantiation with the possibility to use lifted belief updates and precomputed beliefs.

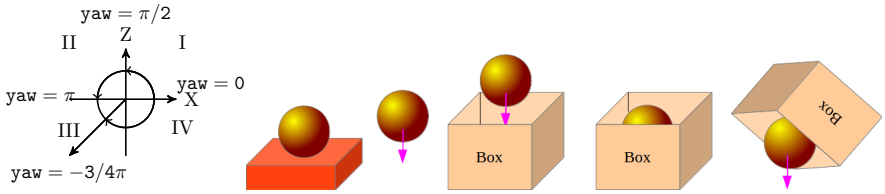
4.4.2 Real-world dynamic domains

The experiments shown so far are generated from synthetic data. We also ran experiments with real-world data (Q3). We considered two tracking scenarios called Packaging and Learnsize; the latter is used to evaluate parameter learning. The objects have markers for an easy detection with a camera (Fig. 4.11).

Packaging scenario

The goal of this scenario is to track objects moved by a human during a packaging activity with boxes (Fig. 4.9). The framework should be able to keep track of objects inside boxes. To solve this problem we defined a model in Dynamic Distributional Clauses where the **state** consists of the position, the velocity and orientation of all objects, plus the relations between them. The relations considered are left, right, near, on, and inside plus object properties such as color, type and size; whenever a new object is observed its pose is added to the state together with all the derived relations.

We modelled the following physical principles (Fig. 4.8b) in the **transition model**:



(a) (b)

Figure 4.8: (a) Yaw of an object. Yaw is positive in quadrants I and II. (b) Physical principles considered.

1. if an object is on top of another object, it cannot fall down;
2. if there are no objects under an object, the object will fall down until it collides with another object or the table;
3. an object may fall inside the box only if it is on the box in the previous step, is smaller than the box and the box is open-side up;
4. if an object is inside a box it remains in the box and its position follows that of the box as long as it is open-side up;
5. if the box is rotated upside down the objects inside will fall down with a certain probability.

As example consider the property 3: if an object $\mathtt{A}$ is not inside a box, is on top of a box $\mathtt{B}$ with rotation $\mathtt{yaw(B)}_t > 0$ (i.e. open-side up, Fig. 4.8a) and the object $\mathtt{A}$ is smaller than the box $\mathtt{B}$, then it falls inside $\mathtt{B}$ with probability 0.8 in the next step. This can be modelled by the following clause:

$$\mathtt{inside(A,B)}_{t+1} \sim \mathtt{finite([0.8:true,0.2:false])} \leftarrow \mathtt{not(inside(A,C)}_t \sim = \mathtt{true}),$$
$$\mathtt{on(A,B)}_t, \mathtt{type(B,box)}, \simeq(\mathtt{yaw(B)}_t) > 0, \mathtt{smaller(A,B)}. \quad (4.12)$$

To model the position and the velocity of objects in free fall we use the rule:

$$\mathtt{pos\_vel_z(A)}_{t+1} \sim \mathtt{gaussian}\left(\begin{bmatrix} \simeq(\mathtt{pos_z(A)}_t) + \Delta t \cdot \simeq(\mathtt{vel_z(A)}_t) - 0.5 g \Delta t^2 \\ \simeq(\mathtt{vel_z(A)}_t) - g\Delta t \end{bmatrix}, \Sigma\right)$$
$$\leftarrow \mathtt{not(inside(A,C)}_t \sim = \mathtt{true}), \mathtt{not(on(A,D)}_t), \mathtt{held(A)}_t \sim = \mathtt{false}. \quad (4.13)$$

It states that if the object $\mathtt{A}$ is neither 'on' nor 'inside' any object, and is not held, the object will fall with gravitational acceleration[3] g. For simplicity

[3]To avoid high velocities in the experiments, we use an acceleration lower than the actual gravitational acceleration.

we specify only the position and velocity for the coordinate z. The variable $\texttt{held(A)}_\texttt{t}$ indicates whether the object is held or not, let us assume the following distribution:

$$\texttt{held(A)}_\texttt{t} \sim \texttt{finite}([\texttt{0.6:true}, \texttt{0.4:false}]). \tag{4.14}$$

The measurement model is the product of Gaussian distributions around each object's position (thereby assuming i.i.d. measurements):

$$\texttt{obsPos(A)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{pos(A)}_\texttt{t}), \Sigma_{obs}). \tag{4.15}$$

Where $\texttt{pos(A)}_\texttt{t}$ is the subvector of $\texttt{pos\_vel}_\texttt{t}$ related to the x, y, z position, and Σ_{obs} is a fixed covariance matrix.

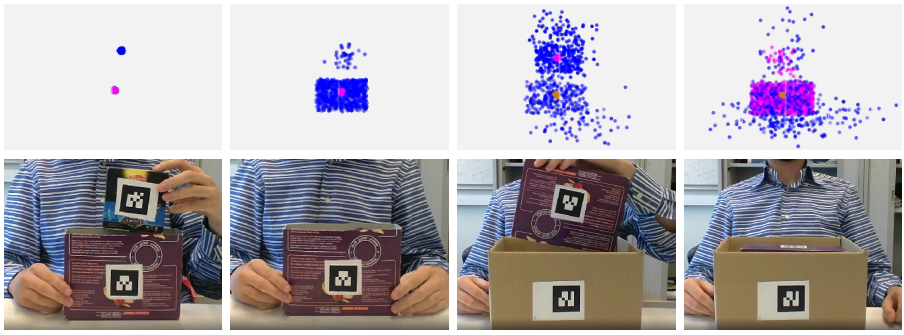
Furthermore, if A is inside B at time t, the relation holds at $t+1$ with probability close to one (clause omitted). The inside concept is transitive, therefore we defined a transitive inside relation $\texttt{tr\_inside(A,B)}_\texttt{t}$ from $\texttt{inside(A,B)}_\texttt{t}$:

$$\texttt{tr\_inside(A,B)}_\texttt{t} \leftarrow \texttt{inside(A,B)}_\texttt{t} \sim= \texttt{true}.$$

$$\texttt{tr\_inside(A,B)}_\texttt{t} \leftarrow \texttt{inside(A,C)}_\texttt{t} \sim= \texttt{true}, \texttt{tr\_inside(C,B)}_\texttt{t}.$$

In addition, we assume that the probability of observing an object inside a box is null. Therefore, to improve the performance we consider a proposal distribution for $\texttt{inside(A,B)}_\texttt{t}$ that depends on the observation of A.

For the packaging scenario we performed several test-cases. We assume the type, size and color are known for each object. We also encoded the static object 'table', therefore when an object is not held it will fall down until it collides with another object or the table. The first test-case consists in taking a box, putting an object inside the box, moving the box and then rotating the box upside down. The second test-case consists in putting an object inside a small box, putting the small box in a bigger box, then moving the bigger box and rotating it upside down. Finally, we put an object inside a small box and then rotate the box on top of another box, the object inside has to fall inside the other object. For this scenario we used 600 samples. The results (Fig. 4.9) showed that the model is stable, correctly tracks the objects, and successfully estimates the transitive relation inside (Q3). For example, whenever we put an object in a box the DCPF estimates that it is inside the box or still outside with a small probability. Similarly, when we rotate a box upside down the object that was inside falls outside and goes inside the box below. One issue that was encountered is when the objects are moved rapidly, in this case the filter keeps track of the visible objects but may lose the sample diversity of the invisible objects. To avoid this problem the variance in the state transition and observation model needs to be increased.



(a) cube on the small box (b) cube inside the small box (c) rotated small box on the big box (d) cube and box inside the big box

Figure 4.9: Packaging scenario experiments. The bottom images represent moments of the experiment, while the top images show the corresponding estimated objects' positions, where each colored point represents an object in a sample. The cube is represented in blue, the small box in fuchsia and the big box in beige.

In this scenario we can consider queries such as how many objects there are in the box with the respective probability for each object, or if there is a blue cube in the red box. The second query would be the conjunction: $\texttt{type(A,cube)},\texttt{color(A,blue)},\texttt{inside(A,B)}_{\texttt{t}} \sim\!=\texttt{true},\texttt{type(B,box)},\texttt{color(B,red)}$. The answer is the probability that the query is true. Alternatively we can list all objects pairs $(\texttt{A},\texttt{B})$ that satisfy the query with the respective probability.

The filter performance for this scenario depends on the number of objects, the framework spends around 0.37 ms, 0.6 ms, 0.87 ms and 1.08 ms per sample respectively for 1, 2, 3 and 4 objects, assuming all objects are visible. If some objects are not visible the performance is better. Figure 4.10 shows an example of execution time per step with 3 objects and 500 samples.

4.4.3 Learnsize scenario

To illustrate the described learning algorithms and their adaptation for DCPF we consider an object tracking scenario called Learnsize. In this scenario we have a table with several objects. The goal is to track the objects moved by a human (or a robot), and estimate online the size of the objects from their interaction. This problem involves static parameters, that is, the objects' size. This makes the problem difficult as explained in Section 4.3.1. The **state**

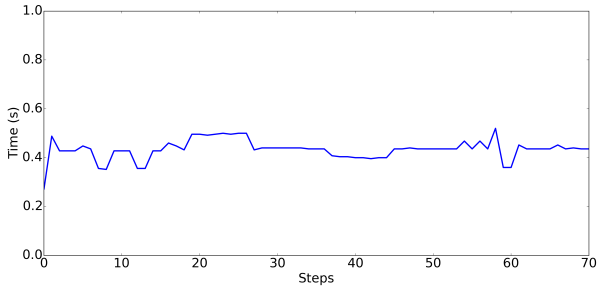


Figure 4.10: Inference time per step in the packaging scenario, with 3 objects and 500 samples.

Figure 4.11: Learnsize scenario. Sketch on the left: the objects are pushed away from each other when they overlap, applying a displacement. Picture on the center with 3 objects. The right figure represents the estimated objects' positions (yellow, orange and grey), and the estimated size (one point per sample) using artificial dynamics. The blue lines are the real size and the black lines the average estimated size. The distance is measured in meters.

variables to estimate are the object positions, the size for each object (i.e., diameter assuming round objects), and the object ID moved by a human or robot (if any). The **observation** is a set of noisy object positions, and there are no actions. Whenever the objects touch (or overlap) each other, each object is pushed away from the other one (Fig. 4.11). The actual objects never overlap, nonetheless this can happen in the samples. The overlap occurs when the distance between the center of the two objects is smaller than the sum of the objects' radiuses (average of the diameter). If there is an overlap we apply a displacement proportional to the absolute difference between the distance and the radiuses sum. Multiple displacements can be applied to the same object, in that case the total displacement is the sum of the simple components. However, whenever the object is held we assume the displacement is 0. The object size

distribution is defined in Section 4.3.2 according to the learning method.

In this scenario we need to estimate which object is held and moved by a human. To simplify the problem we assume the human can move at most one object at a time. Thus, we added the variable $\texttt{move}_\texttt{t}$ in the state that indicates the object ID held/moved or zero for none. We defined the following transition model, which considers more probable to remain in the same state:

$$\texttt{move}_{\texttt{t}+1} \sim \texttt{uniform}([\simeq(\texttt{move}_\texttt{t}), \simeq(\texttt{move}_\texttt{t}) | \texttt{L}]) \leftarrow \texttt{findall}(\texttt{ID}, \texttt{object}(\texttt{ID}) \sim = \texttt{V}, \texttt{L}).$$

Whenever $\texttt{move}_\texttt{t}$ has the value ID the transition model for object ID will have a noise variance double that of the other unmoved objects, e.g., for the axis x:

$$\texttt{pos}_\texttt{x}(\texttt{ID})_{\texttt{t}+1} \sim \texttt{gaussian}(\simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_\texttt{t}), \sigma^2) \leftarrow \simeq(\texttt{move}_\texttt{t}) = \texttt{ID}. \tag{4.16}$$

$$\texttt{pos}_\texttt{x}(\texttt{ID})_{\texttt{t}+1} \sim \texttt{gaussian}\Big(\simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_\texttt{t}) + \simeq(\texttt{totDisp}_\texttt{x}(\texttt{ID})_\texttt{t}), \frac{\sigma^2}{2}\Big) \leftarrow \simeq(\texttt{move}_\texttt{t}) \neq \texttt{ID}.$$

If the object is not held we take in account the eventual displacements caused collisions with other objects. Thus, $\texttt{totDisp}_\texttt{x}(\texttt{ID})_\texttt{t}$ is the sum of all displacements $\texttt{displacement}_\texttt{x}(\texttt{ID}, \texttt{C})_\texttt{t}$ applied to object ID along the x axis and caused by contact with object C.

Finally, to improve the performance we used a suboptimal proposal distributions. Let us assume that the state is $x_t = \{a_t, b_t\}$ and the observations depend only on b_{t+1}, then the weight can be written as:

$$w_{t+1}^{(i)} = w_t^{(i)} \frac{p(z_{t+1}|b_{t+1}^{(i)}) p(b_{t+1}^{(i)}|a_{t+1}^{(i)}, x_t^{(i)}) p(a_{t+1}^{(i)}|x_t^{(i)})}{g(x_{t+1}^{(i)}|x_t^{(i)}, z_{t+1})}.$$

A suboptimal proposal is $g(x_{t+1}|x_t^{(i)}, z_{t+1}) = p(b_{t+1}|a_{t+1}, x_t^{(i)}, z_{t+1}) p(a_{t+1}|x_t^{(i)})$, with weight $w_{t+1}^{(i)} = w_t^{(i)} p(z_{t+1}|a_{t+1}^{(i)}, x_t^{(i)})$. If $p(b_{t+1}|a_{t+1}, x_t)$ is a linear Gaussian transition model or discrete we can easily compute the above suboptimal proposal and relative weight after sampling a_{t+1}. In the scenarios, b_t is the set of the objects' positions, while the remaining states define the set a_t. For example, the proposal that replaces (4.16) is:

$$\begin{aligned} \texttt{pos}_\texttt{x}(\texttt{ID})_{\texttt{t}+1} \sim \texttt{gaussian}(\texttt{M}, \texttt{Var}) \leftarrow \\ \texttt{Var is } (\sigma^{-2} + \omega^{-2})^{-1}, \\ \texttt{M is Var} * (\simeq(\texttt{obsPos}_\texttt{x}(\texttt{ID})_\texttt{t})\omega^{-2} + \simeq(\texttt{pos}_\texttt{x}(\texttt{ID})_\texttt{t})\sigma^{-2}), \\ \simeq(\texttt{move}_\texttt{t}) = \texttt{ID}. \end{aligned} \tag{4.17}$$

Where ω^2 is the variance of the Gaussian measurement model.

We tested the two described learning algorithms in the Learnsize scenario: artificial dynamics and the proposed Storvik's filter variation. We performed 10 trials for each of the two algorithms for three objects. In each trial we randomly pulled and pushed one object at a time. The results of the experiments are summarized in Table 4.1 and were performed using 700 samples. In the experiments with three objects both learning strategies perform reasonably well (Q4). Artificial dynamics performs better and is faster than Storvik's filter

Table 4.1: Learnsize scenario results. Avg error is the absolute distance between the ground truth and the averaged estimation of the objects size (averaged over over objects and trials). 'Correct' is the total number of objects size estimated correctly, that is with an error below 1.5cm.

Algorithm	Correct	avg error (cm)	time per sample (ms)
Artificial dynamics	27/30	0.7	1.6
Storvik's filter variation	23/30	1.3	2.4

variation. An example with three objects using artificial dynamics is shown in Fig. 4.11. Learning becomes harder with a higher number of objects. For better performance offline methods are required. In addition, the performance are highly dependent on the pushes/pulls performed. Indeed, there are different objects sizes that can produce a similar behavior when moved. To make the algorithms converge to the ground truth all objects were put in contact with all the others during the experiments[4]. The videos of the described and other experiments are available at `https://dtai.cs.kuleuven.be/ml/systems/DC/`

4.5 Related Work

In this section we will review related frameworks and applications for dynamic inference.

[4]This is valid for 3 or more objects, with 2 objects it is not possible to learn the objects size from pushes. Indeed, any pair of objects with the same sum of sizes produces the same behaviour, because the distance between the objects during contact is the same.

4.5.1 Frameworks

DCPF is related to probabilistic programming languages such as BLOG, Church [Goodman et al., 2008], ProbLog [Kimmig et al., 2008], and Distributional Clauses [Gutmann et al., 2011]. While these languages are expressive enough to be used for modeling dynamic relational domains, these languages do not support explicitly filtering (BLOG excluded), which makes inference prohibitively slow or unreliable for dynamic models. Also worth mentioning is first-order logical filtering (e.g., see [Shirazi and Amir, 2011]), the logical deterministic counterpart of probabilistic filtering. This method can inspire further DCPF extensions, nonetheless the absence of a probabilistic framework and continuous distributions make them less suitable for the range of applications considered in this thesis.

There exist probabilistic programming approaches for temporal models. A variant of BLOG for filtering in dynamic domains (DBLOG [de Salvo Braz et al., 2008]) has been proposed, it instantiates the variables needed for inference as BLOG. However, as discussed in Section 4.4.1, DBLOG does not currently implement Step (2) of our filtering algorithm. This requires one to manually query all the variables that d-separate from the past or at least those that might be relevant for the given query. In contrast, DCPF automatically determines which variable to samples to guarantee d-separation, exploiting context specific independences. This avoids backinstantiation with the possibility to use lifted beliefs update and precomputed beliefs. Furthermore, all the considerations about LW in complex queries and continuous evidence are valid for the dynamic case.

Logical HMMs [Kersting et al., 2006] employ logical atoms as observations and states and hence, their expressivity is more limited. The lifted relational Kalman filter [Choi et al., 2011], performs efficient lifted exact inference for continuous dynamic domains, but it assumes linear Gaussian models. The relational particle filter of [Manfredotti et al., 2010] cannot handle partial samples. Finally, the approaches that are most similar to ours are those of [Zettlemoyer et al., 2007] and Probabilistic Relational Action Model (PRAM) [Hajishirzi and Amir, 2008]. The former employs first-order formulas to represent a set of states called hypothesis; these are similar to our partial worlds in that they represent a potentially infinite number of states. The key difference is that our approach explicitly defines random variables, (in)dependence assumptions, and their conditional distributions in relationship to other random variables, which allows us to efficiently compute the distribution of a random variable that needs to be sampled and added to the sample. In PRAM the filtering problem is converted into a deterministic first-order logic problem that can be solved using progression, regression and sampling. PRAM is mainly suited for relational domains, that are inherently discrete and binary. In addition, PRAM

performs regression of a formula from time t to time 0 that implies performance issues as previously discussed.

Furthermore, none of the frameworks of [Thon et al., 2011; Kersting et al., 2006; Zettlemoyer et al., 2007; Hajishirzi and Amir, 2008; Natarajan et al., 2008] supports continuous random variables (other than through discretization), therefore these techniques cannot deal with real-world applications in robotics. Discretization is not always a good solution, and it can dramatically increase the number of states, therefore it is unclear whether these algorithms would maintain good performance in such cases. Finally, their first-order logic representation allows discrete and fixed probabilities (for the transition and measurement model), instead DCPF provides a flexible language to represent continuous and discrete distributions that can be parameterized by other random variables or logical variables used in the body. This allows a compact model and faster inference.

Several improvements to classical particle filtering have been proposed, such as Rao-Blackwellization [Casella and Robert, 1996] and Factored Particle Filtering [Pfeffer et al., 2009]. The first method has been exploited in our approach, but further improvements are possible. Factored Particle Filters cluster the state space reducing the variance and improving the accuracy. These methods are complementary to our work and could be adapted in future work.

4.5.2 Applications

Some state estimation applications with a relational representation have been proposed. The relational particle filter of [Cattelani et al., 2012] uses relations such as 'walking together' in people tracking to improve prediction and the tracking process. They divide the state in two sets: object attributes and relations, making some assumptions to speed up inference. In their approach a relation can be true or false. In contrast, our approach does not make a real distinction between attributes and relations, indeed, each random variable has a relational representation, regardless of the distribution (binary, discrete or continuous). This allows parametrization and template definition for any kind of random variable. Furthermore, our language and inference algorithm are more general, keeping inference relatively fast. In addition, it is not clear if they can support partial states and integrate background knowledge while keeping good performance. A relational representation has been used in [Meyer-Delius et al., 2008] for situation characterization over time. However, this work is based on HMMs and uses only binary relations (true or false). Interesting works have been proposed [Tenorth and Beetz, 2009], [Beetz et al., 2012] for manipulation tasks exploiting a relational representation. Those works integrate

relational knowledge about the world to reason about the objects and perform complex tasks. For probabilistic inference and belief update they use MLNs (Markov Logic Networks). However, MLNs are arguably less efficient for filtering inference because they are undirected models that might require MCMC or the computation of the partition function at each step, even though recent optimizations have been proposed [Papai et al., 2012].

4.6 Conclusions

We proposed a flexible representation for hybrid relational domains and provided an efficient inference algorithm for filtering. This framework exploits the relational representation and (context specific) independence assumptions to reduce the sample size (through partial worlds) and the inference cost.

The proposed static algorithm EvalSampleQuery exploits LW in a wider range of cases with respect to systems such as BLOG, and supports complex queries with continuous variables, for which most related frameworks fail. These features are also valid for dynamic domains, where DCPF calls EvalSampleQuery during filtering. At the same time, DCPF avoids backinstantiation to bound the space complexity and reduce time performance variability. This makes DCPF particularly suited for online applications.

One of the advantages of a relational framework like DCPF is the flexibility and generality of the model with respect to a particular situation. Indeed, whenever a new object appears all the respective properties and relations with other objects are implicitly defined (but not necessary computed and added to the sample). In addition, the expressivity of the language helps to bridge the gap between robotics and the high-level symbolic representation used in artificial intelligence.

DCPF was empirically evaluated and applied in several synthetic and real-world scenarios. The results show that DCPF outperforms the classical particle filter and DBLOG for a small number of samples. The DCPF averaged error is lower than the DBLOG error for the same number of samples. Nonetheless, DBLOG seems to be faster for a large number of samples, which might be caused by implementation reasons.

The object tracking experiments show that DCPF is promising for robotics applications. The overall performance is acceptable, but could be improved to scale well with high-dimensional states. Indeed, each sample represents the entire state, therefore inference can be computationally intensive for a high

number of objects and relations. Nonetheless, DCPF exploits the structure of the model and partial samples to speed up inference and improve the performance.

Finally, both learning strategies tested in this framework perform reasonably well for a limited number of parameters. More sophisticated strategies and offline methods need to be investigated for a higher number of parameters.

Chapter 5

Planning

In this chapter we extend the proposed framework for probabilistic planning tasks in fully observable environments. This chapter consists of research previously published in the papers [Nitti et al., 2015b,a] and a submitted journal paper [Nitti et al.].

5.1 Introduction

The DDC language can be easily extended to define MDPs. Beside the state transition model, we need to define a reward function $R(s_t, a_t)$, the terminal states that indicate when the episode terminates, and the applicability of an action a_t is a state s_t as in PDDL/STRIPS.

Example 5.1. *Let us consider an object search scenario (objsearch) used in the experiments, in which a robot looks for a specific object in a shelf. Some of the objects are visible, others are occluded. The robot needs to decide which object to remove to find the object of interest. Every time the robot removes an object, the objects behind it become visible. This happens recursively, i.e., each new uncovered object might occlude other objects. The number and the types of occluded objects depend on the object covering them. For example, a box might cover several objects because it is big. This scenario involves an unknown number of objects and can be written as a partially observable MDP. However, it can be also described as a MDP in DDC where the state is the type of visible objects; in this case the state grows over time when new objects are observed or shrink when objects are removed without uncovering new objects. The probability*

of observing new objects is encoded in the state transition model, for example:

$$\texttt{type(X)}_{t+1} \sim \texttt{val(T)} \leftarrow \texttt{type(X)}_t \sim= \texttt{T},\ \texttt{not(removeObj(X)}_t\texttt{)}. \tag{5.1}$$

$$\texttt{numObjBehind(X)}_{t+1} \sim \texttt{poisson(1)} \leftarrow$$
$$\texttt{type(X)}_t \sim= \texttt{box}, \texttt{removeObj(X)}_t. \tag{5.2}$$

$$\texttt{type(ID)}_{t+1} \sim \texttt{finite([0.2 : glass, 0.3 : cup, 0.4 : box, 0.1 : can])} \leftarrow$$
$$\texttt{type(X)}_t \sim= \texttt{box}, \texttt{removeObj(X)}_t, \texttt{numObjBehind(X)}_{t+1} \sim= \texttt{N},$$
$$\texttt{getLastID(Last)}_t, \texttt{NewID is Last} + 1,$$
$$\texttt{EndNewID is NewID} + \texttt{N}, \texttt{between(NewID, EndNewID, ID)}. \tag{5.3}$$

Clause (5.1) *states that the type of each object remains unchanged when we do not perform a remove action. Otherwise, if we remove the object, its type is removed from the state at time* $t + 1$ *because it is not needed anymore. Clauses* (5.2) *and* (5.3) *define the number and the type of objects behind a box* `X`*, added to the state when we perform a remove action on* `X`*. Similar clauses are defined for other types. The predicate* $\texttt{getLastID(Last)}_t$ *returns the highest object* `ID` *in the state and is needed to make sure that any new object has a different* `ID`*.*

To complete the MDP specification we need to define a reward function $R(s_t, a_t)$ and the terminal states:

$$\texttt{stop}_t \leftarrow \texttt{type(X)}_t \sim= \texttt{can}.$$
$$\texttt{reward(20)}_t \leftarrow \texttt{stop}_t.$$
$$\texttt{reward(}-1\texttt{)}_t \leftarrow \texttt{not(stop}_t\texttt{)}.$$

That is, a state is terminal when we observe the object of interest (e.g., a can), for which a reward of 20 is obtained. The remaining states are nonterminal with reward -1.

To define action applicability we use a set of clauses of the form

$$\texttt{applicable(action}_t\texttt{)} \leftarrow \texttt{preconditions}_t.$$

For example, action $\texttt{removeObj(X)}_t$ is applicable for each object `X` in the state, that is when its type is defined with an arbitrary value `Type`:

$$\texttt{applicable(removeObj(X)}_t\texttt{)} \leftarrow \texttt{type(X)}_t \sim= \texttt{Type}.$$

In DDC a (complete) state contains facts as in standard relational MDPs and statements $\texttt{variable} \sim= \texttt{v}$ (the value of variable is v) for continuous or categorical random variables, e.g.: $\texttt{on}(1,2)_\texttt{t}, \texttt{clean}(1)_\texttt{t}, \texttt{on}(1,\texttt{table})_\texttt{t}$, $\texttt{energy}_\texttt{t} \sim= 1.3$. All the facts not in the state are assumed false and all variables not in the state are assumed not existent.

5.2 HYPE: Planning by Importance Sampling

In this section we introduce HYPE (= *hybrid episodic planner*), a planner for hybrid relational MDPs described in DDC. HYPE is a planner that adopts an *off-policy strategy* [Sutton and Barto, 1998] based on importance sampling and *derived* from the transition model. Related work is discussed more comprehensively in Section 5.4, but as we note later, sample-based planners typically only require a generative model (a way to generate samples) and do not exploit the model of the MDP (i.e., the actual probabilities) [Keller and Eyerich, 2012]. In our case, this knowledge leads to an effective planning algorithm that works in discrete, continuous, hybrid domains, and domains with an unknown number of objects under weak assumptions. Moreover, HYPE performs abstraction of sampled episodes. In this section we introduce HYPE without abstraction; the latter will be introduced in Section 5.3.

5.2.1 Basic algorithm

In a nutshell, HYPE samples episodes E^m and stores for each visited state s_t^m an estimation of the V-function (e.g., the total reward obtained from that state). The action selection follows an given strategy (e.g., ϵ-greedy), where the Q-function is estimated as the immediate reward plus the weighted average of the previously stored V-function points at time $t+1$. This is justified by means of importance sampling as explained later. The essential steps of our planning system HYPE are given in Algorithm 1.

The algorithm realizes the following key ideas:

- $\tilde{Q}$ and $\tilde{V}$ denote approximations of the Q and V-function respectively.
- Lines 8 select an action according to a given strategy.
- Lines 9-12 sample the next step and recursively the remaining episode of total length T, then stores the total discounted reward $G(E_t^m)$ starting from the current state s_t^m. This quantity can be interpreted as a sample of the

Algorithm 1 HYPE without abstraction

1: **function** SAMPLEEPISODE(d, s_t^m, m) ▷ Horizon d, state s_t^m in episode m
2: **if** $d = 0$ **then**
3: **return** 0
4: **end if**
5: **for** each applicable action a in s_t^m **do** ▷ Q-function estimation
6: $\tilde{Q}_d^m(s_t^m, a) \leftarrow R(s_t^m, a) + \gamma \frac{\sum_{i=0}^{m-1} w^i \tilde{V}_{d-1}^i(s_{t+1}^i)}{\sum_{i=0}^{m-1} w^i}$
7: **end for**
8: $a_t^m \leftarrow policy(\{\tilde{Q}_d^m(s_t^m, a)\})$ ▷ action policy, e.g., ϵ-greedy
9: sample $s_{t+1}^m \sim p(s_{t+1} \mid s_t^m, a_t^m)$ ▷ sample next state
10: $G(E_t^m) \leftarrow R(s_t^m, a_t^m) + \gamma \cdot$SAMPLEEPISODE $(d-1, s_{t+1}^m, m)$ ▷ recursive call
11: $\tilde{V}_d^m(s_t^m) \leftarrow G(E_t^m)$
12: store $(s_t^m, \tilde{V}_d^m(s_t^m), d)$
13: **return** $\tilde{V}_d^m(s_t^m)$ ▷ V-function estimation for s_t^m at horizon d
14: **end function**

expectation in formula (2.21), thus an estimator of the V-function. For this and other reasons explained later, $G(E_t^m)$ is stored as $\tilde{V}_d^m(s_t^m)$.

- Most significantly, line 6 approximates the Q-function using the *weighted average* of the stored $\tilde{V}_{d-1}^i(s_{t+1}^i)$ points:

$$\tilde{Q}_d^m(s_t^m, a) \leftarrow R(s_t^m, a) + \gamma \frac{\sum_{i=0}^{m-1} w^i \tilde{V}_{d-1}^i(s_{t+1}^i)}{\sum_{i=0}^{m-1} w^i}, \tag{5.4}$$

where w^i is a *weight function* for episode i at state s_{t+1}^i. The weight exploits the transition model and is defined as:

$$w^i = \frac{p(s_{t+1}^i \mid s_t^m, a)}{q(s_{t+1}^i)} \alpha^{(m-i)}. \tag{5.5}$$

Here, for evaluating an action a at the current state s_t, we let w^i be the ratio of the transition probability of reaching a stored state s_{t+1}^i and the probability used to sample s_{t+1}^i, denoted using q. Recent episodes are considered more significant than previous ones, and so α is a parameter for realizing this. We provide a detailed justification for line 6 below.

We note that line 6 requires us to go over a finite set of actions, and so in the presence of continuous action spaces (e.g., real-valued parameter for a move action), we can discretize the action space or sample from it. More sophisticated approaches are possible [Forbes and Andre, 2002; Smart and Kaelbling, 2000].

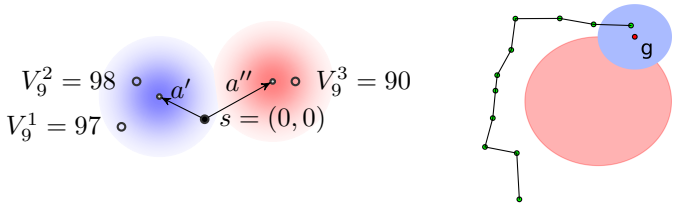


Figure 5.1: Left: weight computation for the objpush domain. Right: a sampled episode that reaches the goal (blue), and avoids the undesired region (red).

Example 5.2. *As a simple illustration, consider the following example called objpush. We have an object on a table and an arm that can push the object in a set of directions; the goal is to move the object close to a point g, avoiding an undesired region (Fig. 5.1). The state consists of the object position* (x, y)*, with push actions parameterized by the displacement* $(\texttt{DX}, \texttt{DY})$*. The state transition model is a Gaussian around the previous position plus the displacement:*

$$\texttt{pos(ID)}_{\texttt{t+1}} \sim \texttt{gaussian}(\simeq(\texttt{pos(ID)}_{\texttt{t}}) + (\texttt{DX}, \texttt{DY}), \texttt{cov}) \leftarrow \texttt{push}(\texttt{ID}, (\texttt{DX}, \texttt{DY})). \tag{5.6}$$

The clause is valid for any object ID*; nonetheless, for simplicity, we will consider a scenario with a single object. The terminal states and rewards in DDC are:*

$$\texttt{stop}_{\texttt{t}} \leftarrow \texttt{dist}(\simeq(\texttt{pos(A)}_{\texttt{t}}), (0.6, 1.0)) < 0.1.$$

$$\texttt{reward}(100)_{\texttt{t}} \leftarrow \texttt{stop}_{\texttt{t}}.$$

$$\texttt{reward}(-1)_{\texttt{t}} \leftarrow \texttt{not}(\texttt{stop}_{\texttt{t}}), \texttt{dist}(\simeq(\texttt{pos(A)}_{\texttt{t}}), (0.5, 0.8)) >= 0.2.$$

$$\texttt{reward}(-10)_{\texttt{t}} \leftarrow \texttt{not}(\texttt{stop}_{\texttt{t}}), \texttt{dist}(\simeq(\texttt{pos(A)}_{\texttt{t}}), (0.5, 0.8)) < 0.2.$$

That is, a state is terminal when there is an object close to the goal point $(0.6, 1.0)$ *(i.e., distance lower than* 0.1*), and so, a reward of* 100 *is obtained. The nonterminal states have reward* -10 *whether inside an undesired region centered in* $(0.5, 0.8)$ *with radius* 0.2*, and* $R(s_t, a_t) = -1$ *otherwise.*

Let us assume we previously sampled some episodes of length $T = 10$*, and we want to sample the* $m = 4$*-th episode starting from* $s_0 = (0, 0)$*. We compute* $\tilde{Q}_{10}^m((0, 0), a)$ *for each action* a *(line 6). Thus we compute the weights* w^i *using* (5.5) *for each stored sample* $\tilde{V}_9^i(s_1^i)$*. For example, Figure 5.1 shows the computation of* $\tilde{Q}_{10}^m((0, 0), a)$ *for action* $a' = (-0.4, 0.3)$ *and* $a'' = (0.9, 0.5)$*, where we have three previous samples* $i = \{1, 2, 3\}$ *at depth* 9*. A shadow*

represents the likelihood $p(s_1^i|s_0 = (0,0), a)$ *(left for* a' *and right for* a''*). The weight* w^i (5.5) *for each sample* s_1^i *is obtained by dividing this likelihood by* $q(s_1^i)$ *(with* $\alpha = 1$*). If* $q^i(s_1^i)$ *is uniform over the three samples, sample* $i = 2$ *with total reward* $\tilde{V}_9^2(s_1^2) = 98$ *will have higher weight than samples* $i = 1$ *and* $i = 3$*. The situation is reversed for* a''*. Note that we can estimate* $\tilde{Q}_d^m(s_t^m, a)$ *using episodes* i *that may never encounter* s_t^m, a_t *provided that* $p(s_{t+1}^i|s_t^m, a_t) > 0$*.*

5.2.2 Computing the (Approximate) Q-Function

The purpose of this section is to motivate our approximation to the Q-function using the weighted average of the V-function points in line 6. Let us begin by expanding the definition of the Q-function from (2.23) as follows:

$$Q_d^\pi(s_t, a_t) = R(s_t, a_t) + \gamma \int_{s_{t+1:T}, a_{t+1:T}} G(E_{t+1}) p(s_{t+1:T}, a_{t+1:T}|s_t, a_t, \pi) ds_{t+1:T}, a_{t+1:T}, \tag{5.7}$$

where $G(E_{t+1})$ is the total (discounted) reward from time $t+1$ to T: $G(E_{t+1}) = \sum_{k=t+1}^{T} \gamma^{k-t-1} R(s_k, a_k)$. Given that we sample trajectories from the target distribution $p(s_{t+1:T}, a_{t+1:T}|s_t, a_t, \pi)$, we obtain the following approximation to the Q-function equaling the true value in the sampling limit:

$$Q_d^\pi(s_t, a_t) \approx R(s_t, a_t) + \frac{1}{N} \gamma \sum_i G(E_{t+1}^i). \tag{5.8}$$

Policy evaluation can be performed sampling trajectories using another policy, this is called *off-policy* Monte-Carlo [Sutton and Barto, 1998]. For example, we can evaluate the greedy policy while the data is generated from a randomized one to enable exploration. This is generally performed using (normalized) *importance sampling* [Shelton, 2001b]. We let w^i be the ratio of the target and proposal distributions to restate the sampling limit as follows:

$$Q_d^\pi(s_t, a_t) \approx R(s_t, a_t) + \frac{1}{\sum w^i} \gamma \sum_i w^i G(E_{t+1}^i). \tag{5.9}$$

In standard off-policy Monte-Carlo the proposal distribution is of the form:

$$p(s_{t+1:T}, a_{t+1:T}|s_t, a_t, \pi') = \prod_{k=t}^{T-1} \pi'(a_{k+1}|s_{k+1}) p(s_{k+1}|s_k, a_k).$$

The target distribution has the same form, the only difference is that the policy is π instead of π'. In this case the weight becomes equal to the policy ratio because the transition model cancels out. This is desirable when the model is not

available, for example in model-free Reinforcement Learning. The question is whether the availability of the transition model can be used to improve off-policy methods. This thesis shows that the answer to that question is positive.

We will now describe the proposed solution. Instead of considering only trajectories that start from s_t, a_t as samples, we consider all sampled trajectories from time $t+1$ to T. Since we are ignoring steps before $t+1$, the proposal distribution for sample i is the marginal

$$p(s_{t+1:T}, a_{t+1:T}|s_0, \pi^i) = q^i(s_{t+1})\pi^i(a_{t+1}|s_{t+1}) \prod_{k=t+1}^{T-1} \pi^i(a_{k+1}|s_{k+1})p(s_{k+1}|s_k, a_k),$$

where q^i is the marginal probability $p(s_{t+1}|s_0, \pi^i)$. To compute $\tilde{Q}_d^m(s_t^m, a)$ we use (5.9), where the weight w^i (for $0 \leq i \leq m-1$) becomes the following:

$$\frac{p(s_{t+1}^i|s_t^m, a)\pi(a_{t+1}^i|s_{t+1}^i) \prod_{k=t+1}^{T-1} \pi(a_{k+1}^i|s_{k+1}^i)p(s_{k+1}^i|s_k^i, a_k^i)}{q^i(s_{t+1}^i)\pi^i(a_{t+1}^i|s_{t+1}^i) \prod_{k=t+1}^{T-1} \pi^i(a_{k+1}^i|s_{k+1}^i)p(s_{k+1}^i|s_k^i, a_k^i)}$$

$$= \frac{p(s_{t+1}^i|s_t^m, a)}{q^i(s_{t+1}^i)} \frac{\prod_{k=t}^{T-1} \pi(a_{k+1}^i|s_{k+1}^i)}{\prod_{k=t}^{T-1} \pi^i(a_{k+1}^i|s_{k+1}^i)} \tag{5.10}$$

$$\approx \frac{p(s_{t+1}^i|s_t^m, a)}{q^i(s_{t+1}^i)} \alpha^{(m-i)}. \tag{5.11}$$

Thus, we obtain line 6 in the algorithm given that $\tilde{V}_{d-1}^i(s_t^i) = G(E_{t+1}^i)$. In our algorithm the target (greedy) policy π is not explicitly defined, therefore the policy ratio is hard to compute. We replace the unknown policy ratio with a quantity proportional to $\alpha^{(m-i)}$ where $0 < \alpha \leq 1$; thus, formula (5.10) is replaced with (5.11). The quantity $\alpha^{(m-i)}$ becomes smaller for an increasing difference between the current episode index m and the i-th episode. Therefore, the recent episodes are weighted (on average) more than the previous ones, as in *recently-weighted average* applied in on-policy Monte-Carlo [Sutton and Barto, 1998]. This is justified because the policy is improved over time, thus recent episodes should have higher weight.

In general, $q^i(s_{t+1}^i)$ is not available in closed form; we adopt the following approximation:

$$q^i(s_{t+1}^i) = p(s_{t+1}^i|s_0, \pi_i) = \int_{s_t, a_t} p(s_{t+1}^i|s_t, a_t)p(s_t, a_t|s_0, \pi_i)$$

$$\approx \frac{1}{2D+1} \sum_{j=i-D}^{i+D} p(s_{t+1}^i|s_t^j, a_t^j). \tag{5.12}$$

Where we are assuming that $\pi^j \approx \pi^i$ for $i-D \leq j \leq i+D$, and the samples s_t^j, a_t^j refer to episode E^j. Each episode E^j is sampled from $p(s_{0:T}, a_{0:T}|s_0, \pi_j)$, thus samples (s_t^j, a_t^j) are distributed as $p(s_t, a_t|s_0, \pi_j)$ and are used in the estimation of the integral.

The likelihood $p(s_{t+1}^i|s_t^m, a)$ is required to compute the weight. This probability can be decomposed using the chain rule, e.g., for a state with 3 variables we have:

$$p(s_{t+1}^i|s_t^m, a) = p(v_3|v_2, v_1, s_t^m, a)p(v_2|v_1, s_t^m, a)p(v_1|s_t^m, a),$$

where $s_{t+1}^i = \{v_1, v_2, v_3\}$. In DDC this is performed evaluating the likelihood of each variable in v_i following the topological order defined in the DDC program. The target and the proposal distributions might be mixed distributions of discrete and continuous random variables; importance sampling can be applied in such distributions as discussed in Section 3.2.2.

To summarize, for each state s_t^m, $Q(s_t^m, a_t)$ is evaluated as the immediate reward plus the weighted average of stored $G(E_{t+1}^i)$ points. In addition, for each state s_t^m the total discounted reward $G(E_t^m)$ is stored. We would like to remark that we can estimate the Q-function also for states and actions that have never been visited, as shown in Example 5.2. This is possible without using function approximations (beyond importance sampling).

5.2.3 Extensions

Our derivation follows a Monte-Carlo perspective, where each stored point is the total discounted reward of a given trajectory: $\tilde{V}_d^m(s_t^m) \leftarrow G(E_t^m)$. However, following the Bellman equation, $\tilde{V}_d^m(s_t^m) \leftarrow max_a \tilde{Q}_d^m(s_t^m, a)$ can be stored instead. The Q-function estimation formula in line 6 is not affected; indeed we can repeat the same derivation using the Bellman equation and approximate it with importance sampling:

$$\begin{aligned} Q_d^\pi(s_t, a_t) &= R(s_t, a_t) + \gamma \int_{s_{t+1}} V_{d-1}^\pi(s_{t+1}) p(s_{t+1}|s_t, a_t) ds_{t+1} \\ &\approx R(s_t, a) + \gamma \sum \frac{w^i}{\sum w^i} \tilde{V}_{d-1}^i(s_{t+1}^i) = \tilde{Q}_d^m(s_t, a_t), \end{aligned} \tag{5.13}$$

with $w^i = \frac{p(s_{t+1}^i|s_t, a_t)}{q^i(s_{t+1}^i)}$ and s_{t+1}^i the state sampled in episode i for which we have an estimation of $\tilde{V}_{d-1}^i(s_{t+1}^i)$, while $q^i(s_{t+1}^i)$ is the probability with which s_{t+1}^i has been sampled. This derivation is valid for a fixed policy π; for a changing policy we can make similar considerations to the previous approach and add the

term $\alpha^{(m-i)}$. If we choose $\tilde{V}^i_{d-1}(s^i_{t+1}) \leftarrow G(E^i_{t+1})$, we obtain the same result as in (5.4) and (5.11) for the Monte-Carlo approach.

Instead of choosing between the two approaches we can use a linear combination, i.e., we replace line 11 with $\tilde{V}^m_d(s^m_t) \leftarrow \lambda G(E^m_t) + (1-\lambda) max_a \tilde{Q}^m_d(s^m_t, a)$. The analysis from earlier applies by letting $\lambda = 1$. However, for $\lambda = 0$, we obtain a local value iteration step, where the stored $\tilde{V}$ is obtained maximizing the estimated $\tilde{Q}$ values. Any intermediate value balances the two approaches (this is similar to, and inspired by, TD(λ) [Sutton and Barto, 1998]). Another strategy consists in storing the maximum of the two: $\tilde{V}^m_d(s^m_t) \leftarrow max(G(E^m_t), max_a \tilde{Q}^m_d(s^m_t, a))$. In other words, we alternate Monte-Carlo and Bellman backup according to which one has the highest value. This strategy works often well in practice; indeed it avoids a typical issue in (on-policy) Monte Carlo methods: bad policies or exploration lead to low rewards, averaged in the estimated Q/V-function.

5.2.4 Practical improvements

In this section we briefly discuss some practical improvements of HYPE. To evaluate the Q-function the algorithm needs to query all the stored examples, making the algorithm potentially slow. This issue can be mitigated with solutions used in instance-based learning, such as hashing and indexing. For example, in discrete domains we avoid multiple computations of the likelihood and the proposal distribution for samples of the same state. In addition, assuming policy improvement over time, only the N_{store} most recent episodes are kept, since older episodes are generally sampled with a worse policy.

HYPE's algorithm relies on importance sampling to estimate the Q-function, thus we should guarantee that $p > 0 \Rightarrow q > 0$, where p is the target and q is the proposal distribution. This is not always the case, like when we sample the first episode. Nonetheless we can have an indication of the estimation reliability. In our algorithm we use $\sum w^i$ with expectation equal to the number of samples: $\mathbb{E}[\sum w^i] = m$. If $\sum w^i < thres$ the samples available are considered insufficient to compute $Q^m_d(s^m_t, a)$, thus action a can be selected according to an exploration policy. It is also possible to add a fictitious weighted point in line 6, that represents the initial $Q^m_d(s^m_t, a)$ guess. This can be used to exploit heuristics during sampling.

A more problematic situation is when, for some action a_t in some state s_t, we always obtain null weights, that is, $p(s^i_{t+1}|s_t, a_t) = 0$ for each of the previous episodes i, no matter how many episodes are generated. This is the case when the state contains continuous variables and the state transition model is deterministic and probabilistic according to the chosen action. This issue is

solved by adding noise to the state transition model, e.g., Gaussian noise for continuous random variables. This effectively 'smoothes' the V-function. Indeed the Q-function is a weighted sum of V-function points, where the weights are proportional to a noisy version of the state transition likelihood.

5.3 Abstraction

By exploiting the (relational) model, we can improve the algorithm by using abstract states, because often, only some parts of the state determine the total reward. The idea is to generalize the specific states into abstract states by removing the irrelevant facts (for the outcome of the episode). This resembles symbolic methods to exactly solve MDPs in propositional and relational domains [Wiering and van Otterlo, 2012]. The main idea in symbolic methods is to apply the Bellman equation on abstract states, using logical regression (backward reasoning), as described in Section 2.6.2 and Section 5.4, symbolic methods are more challenging in hybrid relational MDPs. The main issues are the intractability of the integral in the Bellman equation (2.22), and the complexity of symbolic manipulation in complex hybrid relational domains.

Algorithm 2 HYPE with abstraction

function SAMPLEEPISODE(d, s_t^m, m) ▷ Horizon d, state s_t^m, episode m
  if $d = 0$ **then**
    return $(\emptyset, 0)$
  end if
  for each applicable action a in s_t^m **do** ▷ Q-function estimation
    $\tilde{Q}_d^m(s_t^m, a) \leftarrow R(s_t^m, a) + \gamma \frac{\sum_{i=0}^{m-1} w^i \tilde{V}_{d-1}^i(\hat{s}_{t+1}^i)}{\sum_{i=0}^{m-1} w^i}$
  end for
  $a_t^m \leftarrow policy(\{\tilde{Q}_d^m(s_t^m, a)\})$ ▷ action policy, e.g., ϵ-greedy
  sample $s_{t+1}^m \sim p(s_{t+1}|s_t^m, a_t^m)$
  $(\hat{s}_{t+1}^m, v) \leftarrow$SAMPLEEPISODE$(d-1, s_{t+1}^m, m)$
  $\hat{s}_t^m \leftarrow$ REGRESS$(\{R(s_t^m, a_t^m) \wedge \hat{s}_{t+1}^m\}, \{s_t^m, a_t^m, \hat{s}_{t+1}^m\})$ ▷ abstraction
  $G(E_t^m) \leftarrow R(s_t^m, a_t^m) + \gamma v$
  $\tilde{V}_d^m(s_t^m) \leftarrow G(E_t^m)$
  store $(\hat{s}_t^m, \tilde{V}_d^m(s_t^m), d)$
  return $(\hat{s}_t^m, \tilde{V}_d^m(s_t^m))$
end function

To overcome these difficulties, we propose to perform abstraction at the level of samples. The modified algorithm with abstraction is sketched in Algorithm 2. The main differences with Alg. 1 are:

- Q-function estimation from abstracted states (line 6)
- regression of the current state (line 11)
- the procedure returns the abstract state and its V-function, instead of the latter only (line 15). This is required for recursive regression.

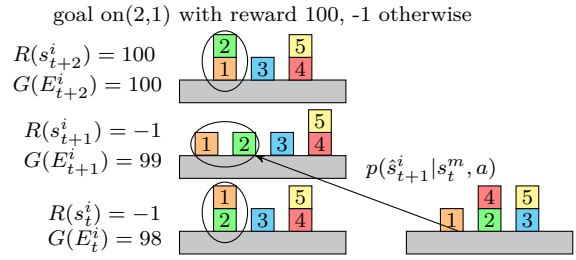


Figure 5.2: Blocksworld with abstraction. Current full state on the right, and a sampled episode on the left. The abstracted states are circled.

5.3.1 Basic principles of abstraction

Before describing abstraction formally, let us consider the blocksworld example to give an intuition. Fig. 5.2 shows a sampled episode from the first state (left down) to the last state (left up) that ends in the goal state $\texttt{on}(2,1)$. Informally, the relevant part of the episode is the set of facts that are responsible for reaching the goal, or more generally responsible for obtaining a given total reward. This relevant part is called the abstracted episode. Fig. 5.2 shows the abstract states (circled) that together define the abstract episode. Intuitively, objects $3, 4, 5$ and their relations are irrelevant to reach the goal $\texttt{on}(2,1)$, and thus do not belong to the abstracted episode.

The abstraction helps to exploit the previous episodes in more cases, speeding up the convergence. For example, Fig. 5.2 shows the computation of a weight w^i (using (5.5)) to compute the Q-function of the (full) state s^m_t depicted on the right, exploiting the abstract state $\hat{s}^i_{t+1}$ indicated by the arrow (from episode i). If the action is moving 4 on top of 5 we have $p(\hat{s}^i_{t+1}|s^m_t, a) > 0 \Rightarrow w^i > 0$. Thus, the Q-function estimate $\tilde{Q}^m_d(s_t, a)$ will include $w^1 \cdot 99$ in the weighted average (line 6 in Alg. 2), making the action appealing. In contrast, without abstraction all actions get weight 0, because the full state s^i_{t+1} is not reachable from s^m_t ($p(s^i_{t+1}|s^m_t, a) = 0$). Therefore, episode i cannot be used to compute the Q-function. For this reason abstraction requires less samples to converge to a near-optimal policy.

This idea is valid in continuous domains. For example, in the *objpush* scenario, the goal is to put any object in a given region; if the goal is reached, only one object is responsible, any other object is irrelevant in that particular state.

5.3.2 Mathematical Derivation

In this section we formalize sample-based abstraction and describe the assumptions that justify the Q-function estimation on abstract states (line 6 of Alg. 2).

Abstraction applied to importance sampling

The Q-function estimation (5.9) can be reformulated for abstract states as follows. For an episode from time t, $E_t = <s_t, a_t, ..., s_T, a_T>$, let us consider an arbitrary partition $E_t = \{\hat{E}_t, E'_t\}$ such that $G(E_t) = G(\hat{E}_t)$, i.e., the total reward depends only on $\hat{E}_t$. The relevant part of the episode has the form $\hat{E}_t = <\hat{s}_t, a_t, ..., \hat{s}_T, a_T>$, while $E'_t = E_t \setminus \hat{E}_t = <s'_t, ..., s'_T>$ is the remaining non-relevant part[1]. The partial episode $\hat{E}_t$ is called *abstract* because the irrelevant variables have been marginalized, in contrast E_t is called full or complete. The Q-function estimation (5.9) is reformulated for abstract states marginalizing

[1]We assumed that the actions are relevant, otherwise they will belong to E'.

irrelevant variables:

$$
\begin{aligned}
Q_d^\pi(s_t^m, a) &= \int_{E_t} p(E_t|s_t^m, a, \pi) G(E_t) dE_t = \\
&= \int_{\hat{E}_t} \left(\int_{E'_t} p(\hat{E}_t, E'_t|s_t^m, a, \pi) dE'_t \right) G(\hat{E}_t) d\hat{E}_t = \\
&= \int_{\hat{E}_t} p(\hat{E}_t|s_t^m, a, \pi) G(\hat{E}_t) d\hat{E}_t = \\
&= R(\hat{s}_t^m, a) + \gamma \int_{\hat{E}_{t+1}} p(\hat{E}_{t+1}|s_t^m, a, \pi) G(\hat{E}_{t+1}) d\hat{E}_{t+1} = \\
&= R(\hat{s}_t^m, a) + \gamma \int_{\hat{E}_{t+1}} \underbrace{\frac{p(\hat{E}_{t+1}|s_t^m, a, \pi)}{q(\hat{E}_{t+1})}}_{w(\hat{E}_{t+1})} q(\hat{E}_{t+1}) G(\hat{E}_{t+1}) d\hat{E}_{t+1} \\
&\approx R(\hat{s}_t^m, a) + \gamma \frac{1}{\sum_{i=0}^{m-1} w(\hat{E}_{t+1}^i)} \sum_{i=0}^{m-1} w(\hat{E}_{t+1}^i) G(\hat{E}_{t+1}^i). \qquad (5.14)
\end{aligned}
$$

The above estimation is based on importance sampling just like in the non-abstract case (5.9), with similar target and proposal distributions. The main difference is the marginalization of irrelevant variables.

Importance weights for abstract episodes

Formula (5.14) is valid for any partition such that $G(E_t) = G(\hat{E}_t)$, but computing the weights $w(\hat{E}_{t+1})$ might be hard in general. To simplify the weight computation let us assume that the chosen partition guarantees the Markov property on abstract states, i.e., $p(\hat{s}_{t+1}|s_{0:t}, a_{0:t}) = p(\hat{s}_{t+1}|\hat{s}_t, a_t)$. To estimate $Q_d^\pi(s_t^m, a)$ (episode m), the weight for abstract episode $i < m$ becomes

the following:

$$w(\hat{E}_{t+1}) = \frac{p(\hat{E}_{t+1}|s_t^m, a, \pi)}{q^i(\hat{E}_{t+1})} = \frac{\int_{E'_{t+1}} p(s_{t+1:T}, a_{t+1:T}|s_t^m, a, \pi) dE'_{t+1}}{\int_{E'_{t+1}} p(s_{t+1:T}, a_{t+1:T}|s_0, \pi^i) dE'_{t+1}}$$

$$= \frac{p(\hat{s}_{t+1:T}, a_{t+1:T}|s_t^m, a, \pi)}{p(\hat{s}_{t+1:T}, a_{t+1:T}|s_0, \pi^i)}$$

$$= \frac{p(\hat{s}_{t+1}|s_t^m, a)\pi(a_{t+1}|\hat{s}_{t+1}, s_t^m, a)}{q^i(\hat{s}_{t+1})\pi^i(a_{t+1}|\hat{s}_{t+1}, s_0)} \frac{\prod_{k=t+1}^{T-1} \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_t^m, a) p(\hat{s}_{k+1}|\hat{s}_k, a_k)}{\prod_{k=t+1}^{T-1} \pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0) p(\hat{s}_{k+1}|\hat{s}_k, a_k)}$$

$$= \frac{p(\hat{s}_{t+1}|s_t^m, a)}{q^i(\hat{s}_{t+1})} \frac{\prod_{k=t}^{T-1} \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_t^m, a)}{\prod_{k=t}^{T-1} \pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0)} \tag{5.15}$$

$$\approx \frac{p(\hat{s}_{t+1}|s_t^m, a)}{q^i(\hat{s}_{t+1})} \frac{\prod_{k=t}^{T-1} \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0)}{\prod_{k=t}^{T-1} \pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0)} \tag{5.16}$$

$$\approx \frac{p(\hat{s}_{t+1}|s_t^m, a)}{q^i(\hat{s}_{t+1})} \alpha^{(m-i)}. \tag{5.17}$$

Where $q^i(\hat{s}_{t+1}) = p(\hat{s}_{t+1}|s_0, \pi^i)$ and can be approximated with (5.12) by replacing s_{t+1} with $\hat{s}_{t+1}$. The final weight formula for abstracted states is similar to the non-abstract case. The difference is abstraction of the next state $\hat{s}_{t+1}$, while the state s_t^m in which the Q-function is estimated remains a complete state.

We will now explain the weight derivation and motivate the approximations adopted. Until formula (5.15) the only assumption made is the Markov property on abstract states. No assumptions are made about the action distributions (policies) π, π^i, thus the probability of an action a_t might depend on abstracted states in previous steps. Then (5.15) is replaced by (5.16) as discussed later. Finally, the policy ratio in (5.16) is replaced in (5.17) as in HYPE without abstraction.

Let us now discuss the approximation introduced in (5.16). Using (5.16) instead of (5.15) is equivalent to using the following target distribution:

$$\frac{p(\hat{s}_{t+1}|s_t^m, a)}{q^i(\hat{s}_{t+1})} p(\hat{s}_{t+1:T}, a_{t+1:T}|s_0, \pi) = p(\hat{s}_{t+1}|s_t^m, a) p(\hat{s}_{t+2:T}, a_{t+1:T}|\hat{s}_{t+1}, s_0, \pi),$$

instead of

$$p(\hat{s}_{t+1:T}, a_{t+1:T}|s_t^m, a, \pi) = p(\hat{s}_{t+1}|s_t^m, a)p(\hat{s}_{t+2:T}, a_{t+1:T}|\hat{s}_{t+1}, s_t^m, a, \pi).$$

Since the state transition model is the same in both distributions, the only difference is the marginalized action distribution (target policy). The one used in (5.16) is

$$\pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0) \tag{5.18}$$

instead of $\pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_t^m, a)$ for $k = t, ..., T-1$. It is not straightforward to analyze this result because these actions distributions are obtained from the same policy π by applying a different marginalization. Nonetheless, it is worth mentioning that the marginalized target policy (5.18) does not depend on the specific state s_t^m, but only on abstract states and on the initial state s_0. This is arguably a desirable property for the (marginalized) target policy.

Using (5.18) as target policy, and thus (5.16) as weight, is useful when the proposal policies are equal to the target policy: $\forall i : \pi = \pi^i$. In this case the weight is exactly:

$$w(E_{t+1}^i) = \frac{p(\hat{s}_{t+1}^i|s_t^m, a)}{q^i(\hat{s}_{t+1}^i)}, \tag{5.19}$$

because the policy ratio cancels out. This formula is also applicable when $\forall i : \pi = \pi^i$ and $\pi(a|s_t) = \pi(a|\hat{s}_t)$ or at least $\pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_t^m, a) = \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0) = \pi(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k})$, that are indeed special cases of (5.18) and (5.16). Imposing or assuming $\pi(a|s_t) = \pi(a|\hat{s}_t)$ seems a reasonable choice, even though (5.18) is a weaker assumption. The optimal policy π^*, might depend only on abstract states, thus $\pi^*(a|s_t) = \pi^*(a|\hat{s}_t)$. Indeed, we expect that the optimal policy depends only on the relevant part of the state. However, we can neither assume $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$ nor $\pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k}, s_0) = \pi^i(a_{k+1}|\hat{s}_{t+1:k+1}, a_{t+1:k})$ as proposal policy. This is because the proposal policy π^i used to sample episode i has to explore with a non-zero probability all the actions: abstract states are generally not sufficient to determine the admissible actions. Thus, the dependence on the initial state s_0 is inevitable. In conclusion, the marginal target policy (5.18) is one of the weakest assumptions to guarantee a weight (5.19) for $\forall i : \pi = \pi^i$. For $\pi \neq \pi^i$ the weight becomes (5.16).

Now let us focus on (5.17) derived from (5.16). Since the policies π^i used in the episodes are assumed to improve over time, we replaced the policy ratio in (5.16) with a quantity that favors recent episodes as in the propositional case (formula (5.11)). Another way of justifying (5.17) is estimating for each stored abstract episode i, the Q-function $Q_d^{\pi^i}(s_t^m, a)$, with target policy $\pi = \pi^i$, and using only the i-th sample. With a marginalized target policy given by

(5.18), the single weight of each estimate $Q_d^{\pi^i}(s_t^m, a)$ is exactly (5.19). The used Q-function estimate can be a weighted average of $Q_d^{\pi^i}(s_t^m, a)$, where recent estimates (higher index i) receive higher weights because the policy is assumed to improve over time. Thus, the final weights are given by (5.17).

HYPE with abstraction adopts formula (5.14) and weights (5.17) for Q-function estimation. Note that during episode sampling the states are complete, nonetheless, to compute $Q_d^{\pi}(s_t^m, a)$ at episode m all previously abstracted episodes $i < m$ are considered. Finally, when the sampling of episode m is terminated, it is abstracted (line 11) and stored (line 14).

Ineffectiveness of lazy instantiation

Before explaining the proposed abstraction in detail, let us consider an alternative solution that samples abstract episodes directly, instead of sampling a complete episode and performing abstraction afterwards. If we are able to determine and sample partial states $\hat{s}_t^m$, we can sample abstract episodes directly and perform Q-function estimation. Sampling the relevant partial episode $\hat{E}_t$ can be easily performed using lazy instantiation, where given the query $G(E_t)$, only relevant random variables are sampled until the query can be answered. Lazy instantiation can exploit context-specific independencies and be extended for distributions with a countably infinite number of variables, as in BLOG [Milch et al., 2005b,a]. Similarly, Distributional Clauses search relevant random variables (or facts) using backward reasoning, while sampling is performed in a forward way. For example, to prove R_t the algorithm needs to sample the variables $\hat{s}_t$ relevant for R_t, $\hat{s}_t$ depends on $\hat{s}_{t-1}$ and the action a_{t-1} depends on the admissible actions that again depend on $\hat{s}_{t-1}$, and so on. At some point variables can be sampled because they depend on known facts (e.g., initial state s_0). This procedure guarantees that $G(E_t) = G(\hat{E}_t)$, $p(\hat{s}_{t+1}|s_{0:t}, a_t) = p(\hat{s}_{t+1}|\hat{s}_t, a_t)$ and $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$, thus (5.15) is exactly equal to (5.16) and it simplifies to $\frac{p(\hat{s}_{t+1}|s_t^m, a)}{q^i(\hat{s}_{t+1})} \frac{\prod_{k=t}^{T-1} \pi(a_{k+1}|\hat{s}_{k+1})}{\prod_{k=t}^{T-1} \pi^i(a_{k+1}|\hat{s}_{k+1})}$. Finally, the approximation (5.17) can be used. Unfortunately, this method avoids only sampling variables that are completely irrelevant, therefore in many practical domains it will sample (almost) the entire state. Indeed, evaluating the admissible actions often requires sampling the entire state. In other words, the abstract state $\hat{s}_t \subseteq s_t$ that guarantees $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$ is often equal to s_t. The solution adopted in this thesis is ignoring the requirement $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$ and approximate (5.15) with (5.16), or equivalently using (5.18) as marginalized target policy distribution.

5.3.3 Sample-based abstraction by logical regression

In this section we describe how to implement the proposed sample-based abstraction. Algorithm 2 samples complete episodes and performs abstraction afterwards. The abstraction of $\hat{E}_t$ from E_t (REGRESS function at line 11) is decomposed recursively employing backward reasoning (regression) from the last step $t = T$ till reaching s_0. We first regress the query $R(s_T, a_T)$ using s_T to obtain the abstract state $\hat{s}_T = \hat{E}_T$ (computing the most general $\hat{s}_T$ such that $R(\hat{s}_T, a_T) = R(s_T, a_T)$). For $t = T-1, ..., 0$ we regress the query $R(s_t, a_t) \wedge \hat{s}_{t+1}$ using $a_t, s_t \in E_t$ to obtain the most general $\hat{s}_t \subseteq s_t$ that guarantees $R(\hat{s}_t, a_t) = R(s_t, a_t)$ and $p(\hat{s}_{t+1}|s_t, a_t) = p(\hat{s}_{t+1}|\hat{s}_t, a_t)$. Note that $\hat{E}_t = \hat{s}_t \cup \hat{E}_{t+1}$. This method assumes that the actions are given, thus it avoids to determine the admissible actions, keeping the abstract states smaller. For this reason, REGRESS guarantees only $G(E_t) = G(\hat{E}_t)$ and $p(\hat{s}_{t+1}|\hat{s}_{0:t}, a_t) = p(\hat{s}_{t+1}|\hat{s}_t, a_t)$, in contrast $\pi^i(a|s_t) = \pi^i(a|\hat{s}_t)$ is not guaranteed. Those conditions are sufficient to apply weight formula (5.15) that is approximated by (5.16) and (5.17) as discussed in the previous section. Note that derivation (5.14) assumes a fixed partition, thus exploits only conditional independencies, but the idea can be extended to context-specific independencies.

Algorithm 3 Episode abstraction

```
1: function REGRESS(Query,Facts)                    ▷ regress Query using Facts
2:     S ← ∅
3:     for L ∈ Query, L ≠ action(_) do
4:         Find θ = mgu(L, F) with F ∈ Facts
5:         if ∃(H ∼ D ← B), ∃β s.t. Facts ⪯ Bβ and F is Hβ ∼= v then   ▷ the
   clause could have generated F
6:             Query ← Queryθβ\F ∪ Bβ
7:         else                                     ▷ Lθ = F not regressable
8:             S ← S ∪ F
9:             Query ← Queryθ\F
10:        end if
11:    end for
12:    return S
13: end function
```

The algorithm REGRESS for regressing a query (formula) using a set of facts is depicted in Algorithm 3. The algorithm tries to repeatedly find literals in the query that could have been generated using the set of facts and a distributional clause. If it finds such a literal, it will be replaced by the condition part of the clause in the query. If not, it will add the fact to the state to be returned.

Example 5.3. *To illustrate the algorithm, consider the blocksworld example in Fig. 5.2. Let us consider the abstraction of the episode on the left. To prove the*

last reward we need to prove the goal, thus $\hat{s}_2 = \texttt{on}(2,1)_2$. *Now let us consider time step* 1, *the proof for the immediate reward is* $\texttt{not}(\texttt{on}(2,1)_1)$, *while the proof for the next abstract state* $\hat{s}_2$ *is* $\texttt{on}(2,\texttt{table})_1, \texttt{clear}(1)_1, \texttt{clear}(2)_1$, *therefore the abstract state becomes* $\hat{s}_1 = \texttt{on}(2,\texttt{table})_1, \texttt{clear}(1)_1, \texttt{clear}(2)_1, \texttt{not}(\texttt{on}(2,1)_1)$. *Analogously,* $s'_0 = \texttt{on}(1,2)_0, \texttt{on}(2,\texttt{table})_0, \texttt{clear}(1)_0, \texttt{not}(\texttt{on}(2,1)_0)$. *The same procedure is applicable to continuous variables.*

5.4 Related work

In this section we will describe non-relational and relational planners related to the proposed algorithms.

5.4.1 Non-relational planners

There is an extensive literature on MDP planners, we will focus mainly on Monte-Carlo approaches. The most notable sample-based planners include Sparse Sampling (SST) [Kearns et al., 2002], UCT [Kocsis and Szepesvári, 2006] and their variations. SST creates a lookahead tree of depth D, starting from state s_0. For each action in a given state, the algorithm samples C times the next state. This produces a near-optimal solution with theoretical guarantees. In addition, this algorithm works with continuous and discrete domains with no particular assumptions. Unfortunately, the number of samples grows exponentially with the depth D, therefore the algorithm is extremely slow in practice. Some improvements have been proposed [Walsh et al., 2010], although the worst-case performance remains exponential. UCT [Kocsis and Szepesvári, 2006] uses *upper confidence bound* for multi-armed bandits to trade off between exploration and exploitation in the tree search, and inspired successful Monte-Carlo tree search methods [Browne et al.]. Instead of building the full tree, UCT chooses the action 'a' that maximizes an upper confidence bound of $Q(s,a)$, following the principle of optimism in the face of uncertainty. Several improvements and extensions for UCT have been proposed, including handling continuous actions [Mansley et al., 2011] (see [Munos, 2014] for a review), and continuous states [Couetoux, 2013] with a simple Gaussian distance metric; however the knowledge of the probabilistic model is not directly exploited. For continuous states, parametric function approximation is often used (e.g., linear regression), nonetheless the model needs to be carefully tailored for the domain to solve [Wiering and van Otterlo, 2012].

There exist algorithms that exploit instance-based methods (e.g. [Forbes and Andre, 2002; Smart and Kaelbling, 2000; Driessens and Ramon, 2003]) for

model-free reinforcement learning. They basically store Q-point estimates, and then use e.g., neighborhood regression to evaluate $Q(s, a)$ given a new point (s, a). While these approaches are effective in some domains, they require the user to design a distance metric that takes into account the domain. This is straightforward in some cases (e.g., in Euclidean spaces), but it can be harder in others. We argue that the knowledge of the model can avoid (or simplify) the design of a distance metric in several cases, where the importance sampling weights and the transition model, can be considered as a kernel.

The closest related works include [Shelton, 2001b,a; Peshkin and Shelton, 2002; Precup et al., 2000], they use importance sampling to evaluate a policy from samples generated with another policy. Nonetheless, they adopt importance sampling differently without knowledge of the MDP model. Although this property seems desirable, the availability of the actual probabilities cannot be exploited, apart from sampling, in their approaches. The same conclusion is valid for practically any sample-based planner, which only needs a sample generator of the model. The work of [Keller and Eyerich, 2012] made a similar statement regarding PROST, a state-of-the-art discrete planner based on UCT, without providing a way to use the state transition probabilities directly. Our algorithm tries to alleviate this, exploiting the probabilistic model in a sample-based planner via importance sampling.

For more general domains that contain discrete and continuous (hybrid) variables several approaches have been proposed under strict assumptions. For example, [Sanner et al., 2011] provide exact solutions, but assume that continuous aspects of the transition model are deterministic. In a related effort [Feng et al., 2004], hybrid MDPs are solved using dynamic programming, but assuming that transition model and reward is piecewise constant or linear. Another planner HAO* [Meuleau et al., 2009] uses heuristic search to find an optimal plan in hybrid domains with theoretical guarantees. However, they assume that the same state cannot be visited again (i.e., they assume plans do not have loops, as discussed in [Meuleau et al., 2009, sec. 5]), and they rely on the availability of methods to solve the integral in the Bellman equation related to the continuous part of the state. Visiting the same state in our approach is a benefit and not a limit; indeed a previously visited state s' is useful to evaluate $Q_d(s, a)$, when the weight is positive (i.e., when s' is reachable from s with action a).

For domains with an unknown number of objects, some probabilistic programming languages such as BLOG [Milch et al., 2005a], Church [Goodman et al., 2008], Anglican [Wood et al., 2014], and DC [Gutmann et al., 2011] can cope with such uncertainty. To the best of our knowledge DTBLOG [Srivastava et al., 2014] and [Vien and Toussaint, 2014] are the only proposals that are able to perform decision making in such domains using a POMDP framework. Furthermore, BLOG is one of the few languages that explicitly

handles data association and identity uncertainty. The proposed thesis does not focus on POMDP, nor on identity uncertainty; however, interesting domains with unknown number of objects can be easily described as an MDP in DDC that HYPE can solve.

Among the mentioned sample-based planners, one of the most general is SST, which does not make any assumption on the state and action space, and only relies on Monte-Carlo approximation. In addition, it is one of the few planners that can be easily applied to any DDC program, including MDPs with an unknown number of objects. For this reason SST was implemented for DDC and used as baseline for our experiments.

5.4.2 Relational planners and abstraction

There exists several modeling languages for planning, the most recent is RDDL [Sanner] that supports hybrid relational domains. A RDDL domain can be mapped in DDC and solved with HYPE. Nonetheless, RDDL does not support a state space with an unknown number of variables as in Example 5.1.

Relational MDPs can be solved using model-free approaches based on Relational Reinforcement Learning [Džeroski et al., 2001; Tadepalli et al., 2004; Driessens and Ramon, 2003], or model-based methods such as ReBel [Kersting et al., 2004], FODD [Wang et al., 2008], PRADA [Lang and Toussaint, 2010], FLUCAP [Hölldobler et al., 2006] and many others. However, those approaches only support discrete action-state (relational) spaces.

Among model-based approaches, several symbolic methods have been proposed to solve MDPs exactly in propositional (see [Mausam and Kolobov, 2012] for a review) and relational domains [Kersting et al., 2004; Wang et al., 2008; Joshi et al., 2010; Hölldobler et al., 2006]. They perform Dynamic Programming at the level of abstract states; this approach is generally called Symbolic Dynamic Programming (SDP). Similar principles have been applied in (propositional) continuous and hybrid domains [Sanner et al., 2011; Zamani et al., 2012], where compact structures (e.g., ADD and XADD) are used to represent the V-function. Despite the effectiveness of such approaches, they make restrictive assumptions (e.g., deterministic transition model for continuous variables) to keep exact inference tractable. For more general domains approximations are needed, for example sample-based methods or confidence intervals [Zamani et al., 2013]. Another issue of SDP is keeping the structures that represent the V-function compact. Nonetheless, some solutions are available in the literature, such as pruning or real-time SDP [Vianna et al., 2015].

Recently, abstraction has received a lot of attention in the Monte-Carlo planning

literature. Like in our work, the aim is to simplifying the planning task by aggregating together states that behave similarly. There are several ways to define state equivalence, see [Li et al., 2006] for a review. Some approaches adopt model equivalence: states are equivalent if they have the same reward and the probabilities to end up in other abstract states are the same. Other approaches define the equivalence in terms of the V/Q-function. In particular, we take note the following advances: (a) Givan et al. [2003] who compute equivalence classes of states based on exact model equivalence, (b) Jiang et al. [2014] who appeal to approximate local homomorphisms derived from a learned model, (c) Anand et al. [2015] who extend Jiang et al. [2014] and Givan et al. [2003] in grouping state-action pairs, and (d) Hostetler et al. [2014] who aggregate states considering the V/Q-function with tight loss bounds.

In our work, in contrast, we consider equivalence (abstraction) at the level of episodes, not states. Two episodes are equivalent if they have the same total reward. In addition, a Markov property condition on abstract states is added to make the weights in (5.14) easier to compute. Abstraction is performed independently in each episode, determining, by logical regression, the set of facts (or random variables) sufficient to guarantee the mentioned conditions. Note that the same full state s_t might have different abstractions in different episodes, even for the same action a_t. This is generally not the case in other works. The proposed abstraction directly exploits the structure of the model (independence assumptions) to perform abstraction. For this reason it relies on the (context-specific) independence assumptions explicitly encoded in the model. However, it is possible to discover independence assumptions not explicitly encoded and include them in the model (e.g., using independence tests).

5.5 Experiments

This section answers the following questions:
(Q1) Does HYPE without abstraction obtain the correct results?
(Q2) How is the performance of HYPE in different domains?
(Q3) How does HYPE compare with state-of-the-art planners?
(Q4) Is abstraction beneficial?

The algorithm was implemented in YAP Prolog and C++, and run on a Intel Core i7 Desktop. We will first describe experiments without abstraction, then compare HYPE with and without abstraction.

5.5.1 HYPE without abstraction

In this section we consider HYPE without abstraction. To answer (Q1) we tested the algorithm on a nonlinear version of the hybrid mars rover domain (called *simplerover1*) described in [Sanner et al., 2011] for which the exact V-function is available (depth $d=3$ and 2 variables: a two-dimensional continuous position and one discrete variable to indicate if the picture was taken). We choose 31 initial points and ran the algorithm for 100 episodes each. Each point took on average $1.4s$. Fig. 5.3 shows the results where the line is the exact V, and dots are estimated V points. The results show that the algorithm converges to the optimal V-function with a negligible error. This domain is deterministic, and so, to make it more realistic we converted it to a probabilistic MDP adding Gaussian noise to the state transition model. The resulting MDP (*simplerover2*) is hard to solve exactly. Then we performed experiments for different horizons, number of pictures points (1 to 4, each one is a discrete variable) and summed the rewards. For each instance the planner searches for an optimal policy and executes it, and after each executed action it samples additional episodes to refine the policy (replanning). The proposed planner is compared with SST which must replan every step. The results for both planners are always comparable, which confirms the empirical correctness of HYPE (Q1) (Table 5.1).

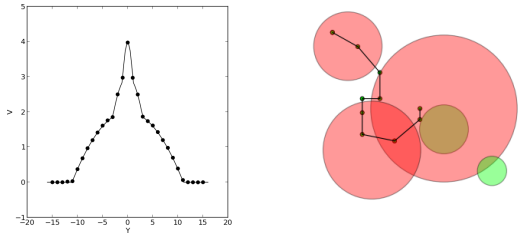


Figure 5.3: V-function for different rover positions (with fixed $X = 0.16$) in *simplerover1* domain (left). A possible episode in *marsrover* (right): each picture can be taken inside the respective circle (red if already taken, green otherwise).

To answer (Q2) and (Q3) we studied the planner in a variety of settings, from discrete, to continuous, to hybrid domains, to those with an unknown number of objects. We performed experiments in a more realistic mars rover domain that is publicly available[2], called *marsrover* (Fig. 5.3). In this domain we consider one robot and 5 picture points that need to be taken, the movement of the robot causes a negative reward proportional to the displacement and the pictures

[2] `http://users.cecs.anu.edu.au/~ssanner/IPPC_2014/index.html`

Table 5.1: Experiments without abstraction: d is the horizon used by the planner, T the total number of steps, M is the maximum number of episodes sampled for HYPE, while C is the SST parameter (number of samples for each state and action). Time refers to the plan execution of one instance, from the starting state till the goal or the maximum number of steps is reached, with a timeout of 1800s. PROST results refer to IPPC2011.

Planner	Domain	game1 $T = 40$	game2 $T = 40$	sysadmin1 $T = 40$	sysadmin2 $T = 40$
HYPE	reward	0.87 ± 0.11	0.67 ± 0.18	0.94 ± 0.07	$\mathbf{0.87 \pm 0.11}$
	time (s)	622	836	422	475
	param	$M = 1200, d = 5$	$M = 1200, d = 5$	$M = 1200, d = 5$	$M = 1200, d = 5$
SST	reward	0.34 ± 0.15	0.14 ± 0.20	0.47 ± 0.13	0.31 ± 0.12
	time (s)	986	1000	1068	1062
	param	$C = 1, d = 5$	$C = 1, d = 5$	$C = 1, d = 5$	$C = 1, d = 5$
HYPE	reward	$\mathbf{0.89 \pm 0.07}$	$\mathbf{0.76 \pm 0.19}$	$\mathbf{0.98 \pm 0.06}$	0.86 ± 0.11
	time (s)	312	582	346	392
	param	$M = 1200, d = 4$	$M = 1200, d = 4$	$M = 1200, d = 4$	$M = 1200, d = 4$
SST	reward	0.79 ± 0.08	0.27 ± 0.22	0.66 ± 0.08	0.46 ± 0.12
	time (s)	1538	1528	1527	1532
	param	$C = 2, d = 4$	$C = 2, d = 4$	$C = 2, d = 4$	$C = 2, d = 4$
PROST	reward	0.99 ± 0.02	1.00 ± 0.19	1.00 ± 0.05	0.98 ± 0.09

Planner	Domain	objpush $T = 30$	simplerover2 $d = T$	marsrover $T = 40$	objsearch $d = T$
HYPE	reward	83.7 ± 7.6	11.8 ± 0.2	249.8 ± 33.5	2.53 ± 1.03
	time (s)	472	38	985	13
	param	$M = 4500, d = 9$	$M = 200, d = T = 8$	$M = 6000, d = 6$	$M = 500, d = T = 5$
SST	reward	82.7 ± 2.7	11.4 ± 0.3	227.7 ± 27.3	1.46 ± 1.0
	time (s)	330	48	787	45
	param	$C = 1, d = 9$	$C = 1, d = T = 8$	$C = 1, d = 6$	$C = 5, d = T = 5$
HYPE	reward	86.4 ± 1.0	11.7 ± 0.2	269.0 ± 29.4	$\mathbf{3.64 \pm 1.09}$
	time (s)	1238	195	983	17
	param	$M = 4500, d = 10$	$M = 500, d = T = 9$	$M = 6000, d = 7$	$M = 600, d = T = 5$
SST	reward	82.4 ± 1.9	11.3 ± 0.3	N/A	2.48 ± 1.0
	time (s)	1574	238	timeout	138
	param	$C = 1, d = 10$	$C = 1, d = T = 9$	$C = 1, d = 7$	$C = 6, d = T = 5$
HYPE	reward	$\mathbf{87.5 \pm 0.5}$	$\mathbf{11.9 \pm 0.3}$	$\mathbf{296.3 \pm 19.5}$	3.3 ± 1.6
	time (s)	373	218	1499	20
	param	$M = 2000, d = 12$	$M = 500, d = T = 10$	$M = 4000, d = 10$	$M = 600, d = T = 6$
SST	reward	N/A	11.2 ± 0.3	N/A	0.58 ± 1.4
	time (s)	timeout	1043	timeout	899
	param	$C = 1, d \geq 11$	$C = 1, d = T = 10$	$C = 1, d \geq 8$	$C = 5, d = T = 6$

can be taken only close to the interest point. Each taken picture provides a different reward. Other experiments were performed in the continuous *objpush* MDP described in Section 5.2 (Fig. 1), and in discrete benchmark domains of the IPPC 2011 competition. In particular, we tested a pair of instances of game of life and sysadmin domains. The results are compared with PROST [Keller and Eyerich, 2012], the IPPC 2011 winner, and shown in Table 5.1 in terms of scores, i.e., the average reward normalizated with respect to IPPC 2011 results; score 1 is the highest result obtained, score 0 is the maximum between the random and the no operation policy.

As suggested by [Keller and Eyerich, 2012], limiting the horizon of the planner increases the performance in several cases. We exploited this idea for HYPE as well as SST (*simplerover2* excluded). For SST we were forced to use small horizons to keep plan time under 30 minutes. In all experiments we followed the IPPC 2011 schema, that is each instance is repeated 30 times (*objectsearch* excluded), the results are averaged and the 95% confidence interval is computed. However, for every instance we replan from scratch for a fair comparison with SST. In addition, time and number of samples refers to the plan execution of one instance. The results (Table 5.1) highlight that our planner obtains generally better results than SST, especially at higher horizons. HYPE obtains good results in discrete domains but does not reach state-of-art results (score 1) for two main reasons. The first is the lack of a heuristic, that can dramatically improve the performance, indeed, heuristics are an important component of PROST [Keller and Eyerich, 2012], the IPPC winning planner. The second reason is the time performance that allows us to sample a limited number of episodes and will not allow to finish all the IPPC 2011 domains in 24 hours. This is caused by a non-optimized Prolog implementation and by the expensive Q-function evaluation; however, we are confident that heuristics and other improvements will significantly improve performance and results.

Moreover, we performed experiments in the *objectsearch* scenario (Section 4.1), where the number of objects is unknown, even though the domain is modeled as a fully observable MDP. The results are averaged over 400 runs, and confirm better performance for HYPE with respect to SST.

5.5.2 HYPE with abstraction

To evaluate the effectiveness of abstraction (Q4) we performed experiments with the blocksworld (BW with 4 or 6 objects) and a continuous version of it (BWC with 4 or 6 objects) with an energy level of the agent and object weights. The energy decreases with a quantity proportional to the weight of the object moved plus Gaussian noise. If the energy becomes 0 the action fails, otherwise

the probability of success is 0.9. The reward is -1 before reaching the goal and $10 + Energy$ if the goal is reached. Then we performed experiments with the *objpush* scenario. This time we consider multiple objects on the table. We considered different goals: move an arbitrary object in the goal region (domain push1 with 2 objects), and move a specific object in the goal region (push2 and push3 with 3 objects). Finally, we performed experiments with the *marsrover* domain, with one robot (mars1) or two of them (mars2), and 5 picture points that need to be taken.

Table 5.2: Experiments with and without abstraction. N is the number of sampled episodes, d is the horizon used by the planner, T is the maximum number of steps, 'success' is the number of times the goal is reached.

domain	d	T	N	abstract	reward	success	time (s)
BW 4	10	10	200	NO	74.6 ± 7.4	82%	38
BW 4	10	10	200	YES	$\mathbf{80.3 \pm 7.2}$	**88%**	32
BW 6	16	16	200	NO	-16 ± 0	0%	112
BW 6	16	16	200	YES	$\mathbf{54.8 \pm 13.4}$	**68%**	42
BWC 4	10	10	200	NO	11.8 ± 2.7	84%	52
BWC 4	10	10	200	YES	$\mathbf{14.2 \pm 1.9}$	**94%**	24
BWC 6	18	18	200	NO	-18.0 ± 0	0%	186
BWC 6	18	18	200	YES	$\mathbf{8.4 \pm 2.4}$	**94%**	70
push1	20	30	1000	NO	67.6 ± 11	86%	734
push1	20	30	1000	YES	$\mathbf{84 \pm 4.7}$	**98%**	652
push2	20	30	1000	NO	-30 ± 0	0%	1963
push2	20	30	1000	YES	$\mathbf{30.5 \pm 14}$	**58%**	910
push3	20	40	500	NO	-17.4 ± 12.6	20%	638
push3	20	40	500	YES	$\mathbf{89.3 \pm 1.5}$	**100%**	122
mars1	30	40	1500	NO	$\mathbf{280.0 \pm 11.8}$	**90%**	1492
mars1	30	40	1500	YES	273.3 ± 11	86%	780
mars2	30	40	1000	NO	209.3 ± 27.7	37%	2817
mars2	30	40	1000	YES	$\mathbf{287.7 \pm 23.6}$	**87%**	902

The current implementation supports negation only for ground formulas. Regression of nonground formulas is possible when the domain is purely relational. However, it becomes challenging when there are continuous random variables and logical variables in a negated formula. If we assume that the domain is fixed (e.g., known number of objects used), logical variables can be replaced with objects in the domain, making the formulas ground. For this reason we will not consider domains with an unknown number of objects, which HYPE without abstraction can solve.

The experiments are shown in Table 5.2. The rewards are averaged over 50 runs and a 95% confidence interval is computed. The results highlight that abstraction improves the expected total reward for the same number of samples or achieves comparable results. In addition, HYPE with abstraction is always faster. The latter is probably due to a faster weight computation with abstract states and due to the generation of better plans that are generally shorter and thus faster. This suggests that the overhead caused by the abstraction procedure is negligible and worthwhile. Nonetheless, we do remark that in domains with, for example, single objects and where rewards are characterized for full states, abstraction gives no added value.

5.6 Conclusions

We proposed a sample-based planner for MDPs described in DDC, and showed how the state transition model can be exploited in off-policy Monte-Carlo. The experimental results show that the algorithm produces good results in discrete, continuous, hybrid domains as well as those with an unknown number of objects. Most significantly, it challenges and outperforms SST. Moreover, we extended HYPE with abstraction. We formally described how (context-specific) independence assumptions can be exploited to perform episode abstraction. This is valid for propositional as well as relational domains. A theoretical derivation has been provided to justify the assumptions and the approximations used. Finally, empirical results showed that abstraction provides significant improvements.

There are several possible directions for future work. For example, heuristics can be used to guide the search and indexing/hashing methods to quickly retrieve relevant stored samples during the Q-function estimation. Indeed, Q-function estimation is one of the most expensive part of HYPE.

Chapter 6

Conclusions and Future Work

We conclude the thesis with this chapter, which presents a summary of the work and possible directions for future work.

6.1 Conclusions

Probabilistic logic languages and statistical relational learning (SRL) combine logical representations, probabilistic reasoning, and machine learning. Such approaches have been successful in many application areas ranging from natural language processing to bioinformatics. However, many probabilistic logic languages do not support continuous random variables, or their support for continuous variables is limited. This makes such language less appealing for robotics and computer vision.

This thesis addresses this issue by extending probabilistic logic programming techniques to deal with hybrid relational domains for inference, learning, and planning. The general aim of the thesis is to provide a probabilistic logic framework that can tightly integrate continuous information with high-level symbolic knowledge. This integration has several advantages: it is easier to write models that define symbolic knowledge from low-level data, and describe dynamic models in terms of low-level and symbolic knowledge jointly. Once the model has been defined (or eventually learned), it can be used to perform inference and planning. Such integration allows, for example, to estimate the object positions from relations such as 'inside' and 'on', and infer the same relations from object positions. Moreover, high-level symbolic knowledge

inferred from low-level data is useful to describe such data in natural language. For example, in an object tracking scenario it is possible to query the system to list 'the blue objects inside the big box' in a language that is similar to the natural one.

We will now describe the goals and conclusions of the thesis from a technical perspective. The **first goal** of the thesis was to provide a general and efficient inference algorithm that works in complex hybrid domains. It is efficient with respect to the time to sample a (partial) world and with respect to the convergence rate. We showed how the proposed algorithm is able to provide correct results in several domains with competitive performance. In particular, it provides a meaningful answer to queries in domains (e.g., the Indian GPA problem) for which most related frameworks fail. The proposed solution involves a practical solution for conditional probabilities with zero-probability evidence. Such solution is then applied in an importance sampling algorithm without additional ad-hoc discretizations (or noise) required in other frameworks to handle zero-probability evidence. Moreover, the proposed query expansion allows to perform constraint propagation and thus it avoids sample rejections in a wide range of cases. The inference algorithm EvalSampleQuery was empirically evaluated and applied in several experiments. The results showed that EvalSampleQuery outperforms naive MC and BLOG, confirming the soundness and the competitiveness of the approach.

The **second goal** of the thesis was to extend the framework for dynamic domains. We proposed DCPF, a framework for filtering that exploits EvalSampleQuery. At the same time, DCPF avoids backinstantiation to bound the space complexity and reduce time performance variability. This makes DCPF particularly suited for online applications. DCPF was empirically evaluated and applied in several synthetic and real-world scenarios. The results showed that DCPF outperforms the classical particle filter and DBLOG for a small number of samples. In addition, the object tracking experiments showed that DCPF is promising for robotics applications. In particular, DCPF can be used to exploit relational and continuous information jointly to improve state estimation, and to convert low-level information in relational (symbolic) knowledge, valuable in several applications.

The **third goal** of the thesis was to extend the hybrid relational framework for planning tasks. We proposed the planner HYPE for hybrid relational MDPs, and showed how the state transition model can be exploited in off-policy Monte-Carlo Methods. The experiments showed that the algorithm produces good results in discrete, continuous, hybrid domains as well as in those with an unknown number of objects. Most significantly, it challenges and outperforms SST. The main drawback of HYPE is the computation complexity of the Q-function evaluation, however several optimizations can be integrated, such as

indexing for a fast retrieval of relevant samples and heuristics to guide the search.

HYPE has been extended with abstraction. In particular, we formally described how (context-specific) independence assumptions can be exploited to perform episode abstraction. This is valid for propositional as well as relational domains. A theoretical derivation has been provided to justify the assumptions and the approximations used. Finally, empirical results showed that abstraction provides significant improvements. Nonetheless, there are domains where the entire state is relevant, in such cases abstraction is not useful. Despite this, abstraction is computationally efficient, and thus it is convenient even when the abstraction regards a small set of random variables in a limited number of states.

As a final remark we hope that this thesis can help to bridge the gap between low-level continuous sensory data and the high-level symbolic representation used in artificial intelligence.

6.2 Future Work

We end this thesis with discussing some promising directions for future work. We will first discuss applications and then technical directions.

6.2.1 Applications

As discussed in the conclusions, the general aim of the thesis is bridging the gap between low-level continuous sensory data and the high-level symbolic representation. This can have impact in several domains, in particular robotics and vision. DCPF and HYPE can be used and extended for robot manipulation tasks that involve human interaction, as described in the introduction. In addition, many applications that require objects or people tracking can benefit from the proposed hybrid relational framework. For example, converting videos in natural language sentences (or just symbolic knowledge) can be performed using DCPF state estimation.

Nonetheless, robotics and vision applications are challenging for several reasons. Tags used in this thesis for an easy object detection might not be appropriate. Thus, object detection and recognition is required together with symbolic grounding that links the object symbols in the belief state to actual objects detected by the vision system. This issue, called data association or anchoring, has not been analyzed in this thesis. We believe that relational inference can bring benefits to this problem. For example, a hybrid relational ontology can be

used to discard unlikely associations or spurious observations, e.g., observing a car wheel in a kitchen is extremely unlikely, or that walls and other big objects do not move.

Many applications in robotics involve planning. HYPE can be used for this purpose. However, in complex domains, the state consisting of continuous and relational variables can easily become too large for efficient planning with HYPE. In this regard, decomposing the task hierarchically can dramatically improve the performance. This is how humans perform planning, dividing the task at several level of abstractions. Moreover, making the decomposition automatic can be an interesting direction for future work.

6.2.2 Inference and planning

We will now describe directions for future work regarding inference and planning.

In Chapter 3 we discussed how conditional probability with zero-probability evidence is defined. Unfortunately, the Borel-Kolmogorov paradox shows that there is no unique definition of a given zero-probability evidence constraint. Alternative definitions as the ones proposed by [Diaconis et al., 2013; Afshar et al., 2016] can be integrated in the future. However, from an artificial intelligence perspective, there are more fundamental questions that should be addressed in the future work: how does a different definition of zero-probability evidence influence the belief distribution of an agent and its decisions? Can a zero-probability evidence definition be better than another one for belief updating or for decision-making? These questions and the Borel-Kolmogorov paradox itself are largely ignored by the AI and robotics communities. We believe that these are important issues that need to be addressed with practical consequences, not just purely theoretical ones.

Another direction for future work is improving the sampling algorithm for DC by using more advanced sampling methods. For example, adaptive importance sampling can help to cope with complex high-dimensional domains. Indeed, it learns the optimal proposal distribution using the previous samples. This principle can be used also in complex non-parametric domains, such as those with an unknown number of objects. In addition, query expansion and the proposed solution for zero-probability evidence will remain valid. Preliminary results not discussed in this thesis are promising.

As an alternative, MCMC methods can be investigated for DC, even though it might not be easy to guarantee that the jumps generate samples consistent with the semantics of the DC program. This issue can be handled borrowing ideas from languages such as Church that apply MCMC on execution traces.

Another possible direction for future work is improving DCPF inference for dynamic domains. If the inference is performed offline, where the entire sequence of observations is known in advance, particle smoothing can be integrated in DCPF to achieve lower variance estimations. Smoothing is computationally expensive (quadratic in the number of samples), but improvements are available [Klaas et al., 2006]. Another solution for online and offline inference is state-space factorization. This helps to mitigate the curse of dimensionality. Factorization is relatively easy when the number of random variables is fixed. Nonetheless, in general hybrid relational domains described in the thesis, factorization is not trivial. Indeed, a random variable might be defined only in some worlds (samples), and the factorization has to keep the samples consistent with the DDC program semantics.

Finally, the planner HYPE can be improved in several ways. HYPE uses stored samples to evaluate the Q-function. This makes its estimation computationally expensive. For this reason, indexing/hashing methods can be implemented to quickly retrieve relevant stored samples during the Q-function estimation. Moreover, heuristics can be used to guide the search. Another issue is the curse of dimensionality for high-dimensional state-action spaces. An interesting direction regards dimension reduction techniques to learn a state representation that is more compact and easier to handle. This would make HYPE faster and more effective. Moreover, it might be useful to extend HYPE principles for policy gradient methods. In such methods the policy is parametrized; this allows the policy to generate actions in the complete continuous action space without discretization as required by HYPE.

Bibliography

[Afshar et al., 2016] H. M. Afshar, S. Sanner, and C. Webers. Closed-form gibbs sampling for graphical models with algebraic constraints. In *Proceedings of the 30th Conference on Artificial Intelligence (AAAI 2016)*, pages 3287–3293. AAAI Press, 2016. pages 50, 56, 119, 121

[Anand et al., 2015] A. Anand, A. Grover, Mausam, and P. Singla. ASAP-UCT: abstraction of state-action pairs in UCT. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)*, pages 1509–1515, 2015. pages 110, 121

[Andrieu et al., 2003] C. Andrieu, N. de Freitas, A. Doucet, and M. I. Jordan. An introduction to MCMC for machine learning. *Machine Learning*, 50(1): 5–43, 2003. pages 17, 121

[Andrieu et al., 2005] C. Andrieu, A. Doucet, and V. B. Tadic. On-line parameter estimation in general state-space models. In *Proceedings of the 44th IEEE Conference on Decision and Control, 2005 and 2005 European Control Conference. (CDC-ECC 2005)*, pages 332–337, 2005. pages 72, 121

[Apt, 1997] K. Apt. *From logic programming to Prolog.* Prentice-Hall international series in computer science. Prentice Hall, 1997. pages 1, 21, 24, 121

[Bancilhon et al., 1986] F. Bancilhon, D. Maier, Y. Sagiv, and J. D. Ullman. Magic sets and other strange ways to implement logic programs. In *Proceedings of the 5th Symposium on Principles of Database Systems (SIGACT-SIGMOD 1986)*, pages 1–15, 1986. pages 36, 121

[Beetz et al., 2012] M. Beetz, D. Jain, L. Mosenlechner, M. Tenorth, L. Kunze, N. Blodow, and D. Pangercic. Cognition-enabled autonomous robot control for the realization of home chore task intelligence. *Proceedings of the IEEE*, 100(8):2454–2471, 2012. pages 87, 121

[Boutilier et al., 2000] C. Boutilier, R. Dearden, and M. Goldszmidt. Stochastic dynamic programming with factored representations. *Artificial Intelligence*, 121(1):49 – 107, 2000. pages 29, 122

[Boutilier et al., 2001] C. Boutilier, R. Reiter, and B. Price. Symbolic dynamic programming for first-order mdps. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, IJCAI 2001, pages 690–697. Morgan Kaufmann Publishers Inc., 2001. pages 29, 122

[Browne et al.] C. Browne, E. J. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, (1):1–43. pages 107, 122

[Carvalho et al., 2010a] C. M. Carvalho, M. S. Johannes, H. F. Lopes, and N. G. Polson. Particle learning and smoothing. *Statistical Science*, 25(1):88–106, 2010a. pages 71, 72, 122

[Carvalho et al., 2010b] C. M. Carvalho, H. F. Lopes, N. G. Polson, and M. A. Taddy. Particle learning for general mixtures. *Bayesian Analysis*, 5(4): 709–740, 2010b. pages 72, 122

[Casella and Robert, 1996] G. Casella and C. P. Robert. Rao-Blackwellisation of Sampling Schemes. *Biometrika*, 83(1):81–94, 1996. pages 87, 122

[Cattelani et al., 2012] L. Cattelani, C. Manfredotti, and E. Messina. A particle filtering approach for tracking an unknown number of objects with dynamic relations. *Journal of Mathematical Modelling and Algorithms in Operations Research*, 13(1):3–21, 2012. pages 87, 122

[Choi et al., 2011] J. Choi, A. Guzman-Rivera, and E. Amir. Lifted relational kalman filtering. In *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*, pages 2092–2099, 2011. pages 86, 122

[Couetoux, 2013] A. Couetoux. *Monte Carlo Tree Search for Continuous and Stochastic Sequential Decision Making Problems*. Theses, Université Paris Sud - Paris XI, 2013. pages 107, 122

[De Raedt, 2008] L. De Raedt. *Logical and relational learning*. Cognitive technologies. Springer, 2008. pages 2, 26, 122

[De Raedt et al., 2008] L. De Raedt, P. Frasconi, K. Kersting, and S. Muggleton. *Probabilistic Inductive Logic Programming - Theory and Applications*, volume 4911 of *Lecture Notes in Artificial Intelligence*. Springer, Berlin / Heidelberg, 2008. pages 2, 26, 122

[de Salvo Braz et al., 2008] R. de Salvo Braz, N. Arora, E. Sudderth, and S. Russell. Open-universe state estimation with DBLOG. In *NIPS Workshop on Probabilistic Programming: Universal Languages, Systems and Applications*, 2008. pages 75, 86, 123

[Diaconis et al., 2013] P. Diaconis, S. Holmes, and M. Shahshahani. Sampling from a manifold. In *Advances in Modern Statistical Theory and Applications: A Festschrift in honor of Morris L. Eaton*, volume 10, pages 102–125. Institute of Mathematical Statistics, 2013. pages 50, 57, 119, 123

[Douc and Cappé, 2005] R. Douc and O. Cappé. Comparison of resampling schemes for particle filtering. In *Proceedings of the 4th International Symposium on Image and Signal Processing and Analysis, (ISPA 2005)*, pages 64–69. IEEE, 2005. pages 61, 123

[Doucet et al., 2000a] A. Doucet, N. de Freitas, K. Murphy, and S. Russell. Rao-blackwellised particle filtering for dynamic bayesian networks. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI 2000)*, pages 176–183. Morgan Kaufmann, 2000a. pages 65, 66, 123

[Doucet et al., 2000b] A. Doucet, S. Godsill, and C. Andrieu. On sequential monte carlo sampling methods for bayesian filtering. *STATISTICS AND COMPUTING*, 10(3):197–208, 2000b. pages 3, 20, 123

[Driessens and Ramon, 2003] K. Driessens and J. Ramon. Relational instance based regression for relational reinforcement learning. In *Machine Learning, Proceedings of the 20th International Conference (ICML 2003)*, pages 123–130, 2003. pages 107, 109, 123

[Džeroski et al., 2001] S. Džeroski, L. De Raedt, and K. Driessens. Relational reinforcement learning. *Machine learning*, 43(1-2):7–52, 2001. pages 109, 123

[Feng et al., 2004] Z. Feng, R. Dearden, N. Meuleau, and R. Washington. Dynamic programming for structured continuous Markov decision problems. In *Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence (UAI 2004)*, pages 154–161, 2004. pages 108, 123

[Fikes and Nilsson, 1971] R. E. Fikes and N. Nilsson. STRIPS: A new approach to the application theorem proving to problem solving. *Artificial Intelligence*, 5(2):189–208, 1971. pages 30, 123

[Forbes and Andre, 2002] J. Forbes and D. Andre. Representations for learning control policies. In *Procedings of the International Conference on Machine Learning Workshop on Development of Representations*, pages 7–14, 2002. pages 93, 107, 123

[Fung and Chang, 1989] R. M. Fung and K. Chang. Weighing and integrating evidence for stochastic simulation in bayesian networks. In *Proceedings of the 5th Conference on Uncertainty in Artificial Intelligence (UAI 1989)*, 1989. pages 17, 39, 124

[Geman and Geman, 1984] S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, 1984. pages 18, 124

[Getoor and Taskar, 2007] L. Getoor and B. Taskar. *An Introduction to Statistical Relational Learning*. MIT Press, 2007. pages 2, 26, 124

[Gilks and Berzuini, 2001] W. R. Gilks and C. Berzuini. Following a moving target-monte carlo inference for dynamic bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(1):127–146, 2001. pages 71, 124

[Givan et al., 2003] R. Givan, T. Dean, and M. Greig. Equivalence notions and model minimization in markov decision processes. *Artificial Intelligence*, 147 (1–2):163 – 223, 2003. pages 110, 124

[Gogate and Dechter, 2011] V. Gogate and R. Dechter. SampleSearch: Importance sampling in presence of determinism. *Artificial Intelligence*, 175:694–729, February 2011. ISSN 0004-3702. pages 56, 124

[Goodman et al., 2008] N. Goodman, V. K. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: A language for generative models. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI 2008)*, pages 220–229. AUAI Press, 2008. pages 19, 26, 56, 86, 108, 124

[Green, 1995] P. J. Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82:711–732, 1995. pages 19, 124

[Gutmann et al., 2011] B. Gutmann, I. Thon, A. Kimmig, M. Bruynooghe, and L. De Raedt. The magic of logical inference in probabilistic programming. *Theory and Practice of Logic Programming*, 11:663–680, 7 2011. pages 3, 26, 32, 33, 34, 36, 56, 67, 86, 108, 124

[Gyenis et al., 2016] Z. Gyenis, G. Hofer-Szabó, and M. Rédei. Conditioning using conditional expectations: the Borel–Kolmogorov paradox. *Synthese*, pages 1–36, 2016. pages 10, 43, 124

[Hajishirzi and Amir, 2008] H. Hajishirzi and E. Amir. Sampling first order logical particles. In *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI 2008)*, pages 248–255. AUAI Press, 2008. pages 86, 87, 125

[Hastings, 1970] W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. pages 18, 125

[Henrion, 1986] M. Henrion. Propagating uncertainty in bayesian networks by probabilistic logic sampling. In *Proceedings of the Second Annual Conference on Uncertainty in Artificial Intelligence UAI 1986, Philadelphia*, pages 149–164, 1986. pages 17, 125

[Higuchi, 2001] T. Higuchi. Self-organizing time series model. In A. Doucet, N. Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, pages 429–444. Springer New York, 2001. pages 71, 125

[Hölldobler et al., 2006] S. Hölldobler, E. Karabaev, and O. Skvortsova. Flucap: A heuristic search planner for first-order MDPs. *Journal of Artificial Intelligence Research*, 27:419–439, 2006. pages 29, 109, 125

[Hostetler et al., 2014] J. Hostetler, A. Fern, and T. Dietterich. State aggregation in monte carlo tree search. In *Proceedings of the 28th Conference on Artificial Intelligence (AAAI 2014)*, 2014. pages 110, 125

[Jaynes, 2003] E. T. Jaynes. *Probability Theory: The Logic of Science*. Cambridge University Press, 2003. pages 7, 125

[Jiang et al., 2014] N. Jiang, S. Singh, and R. Lewis. Improving uct planning via approximate homomorphisms. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1289–1296. International Foundation for Autonomous Agents and Multiagent Systems, 2014. pages 110, 125

[Joshi et al., 2010] S. Joshi, K. Kersting, and R. Khardon. Self-taught decision theoretic planning with first order decision diagrams. In *Proceedings of the 20th International Conference on Automated Planning and Scheduling (ICAPS 2010)*, pages 89–96, 2010. pages 29, 109, 125

[Kadane, 2011] J. Kadane. *Principles of Uncertainty*. Chapman & Hall/CRC Texts in Statistical Science. Taylor & Francis, 2011. pages 7, 10, 43, 125

[Kalman, 1960] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82, 1960. pages 20, 125

[Kantas et al., 2009] N. Kantas, A. Doucet, S. S. Singh, and J. M. Maciejowski. An overview of sequential monte carlo methods for parameter estimation in general state-space models. In *15th IFAC Symposium on System Identification*, volume 15, pages 774–785, 2009. pages 71, 126

[Kearns et al., 2002] M. Kearns, Y. Mansour, and A. Y. Ng. A sparse sampling algorithm for near-optimal planning in large markov decision processes. *Machine Learning*, 49(2-3):193–208, 2002. pages 107, 126

[Keller and Eyerich, 2012] T. Keller and P. Eyerich. PROST: probabilistic planning based on UCT. In *Proceedings of the 22nd International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 2012. pages 92, 108, 113, 126

[Kersting et al., 2004] K. Kersting, M. V. Otterlo, and L. De Raedt. Bellman goes relational. In *Proceedings of the 21st international conference on Machine learning (ICML 2004)*, 2004. pages 29, 109, 126

[Kersting et al., 2006] K. Kersting, L. De Raedt, and T. Raiko. Logical hidden markov models. *Journal of Artificial Intelligence Research*, 25:425–456, 2006. pages 86, 87, 126

[Kimmig et al., 2008] A. Kimmig, V. Santos Costa, R. Rocha, B. Demoen, and L. De Raedt. On the efficient execution of ProbLog programs. In *Logic Programming*, volume 5366 of *Lecture Notes in Computer Science*, pages 175–189. Springer Berlin / Heidelberg, 2008. pages 26, 56, 86, 126

[Klaas et al., 2006] M. Klaas, M. Briers, N. de Freitas, A. Doucet, S. Maskell, and D. Lang. Fast particle smoothing: If i had a million particles. In *Proceedings of the 23rd International Conference on Machine Learning (ICML 2006)*, pages 481–488. ACM, 2006. pages 120, 126

[Kocsis and Szepesvári, 2006] L. Kocsis and C. Szepesvári. Bandit based monte-carlo planning. In *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, pages 282–293, Berlin, Heidelberg, 2006. Springer-Verlag. pages 107, 126

[Koller and Friedman, 2009] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009. pages 11, 15, 39, 126

[Kolmogorov, 1956] A. Kolmogorov. *Foundations of the Theory of Probability*. 1956. pages 10, 43, 126

[Lang and Toussaint, 2010] T. Lang and M. Toussaint. Planning with noisy probabilistic relational rules. *Journal of Artificial Intelligence Research*, 39: 1–49, 2010. pages 109, 126

[Lemieux, 2009] C. Lemieux. *Monte Carlo and Quasi-Monte Carlo Sampling*, volume 20. Springer, 2009. pages 15, 37, 127

[Li et al., 2006] L. Li, T. J. Walsh, and M. L. Littman. Towards a unified theory of state abstraction for MDPs. In *International Symposium on Artificial Intelligence and Mathematics (ISAIM 2006)*, 2006. pages 110, 127

[Lloyd, 1987] J. Lloyd. *Foundations of logic programming*. Springer-Verlag New York, Inc., 1987. pages 1, 21, 24, 127

[Lloyd and Shepherdson, 1991] J. Lloyd and J. Shepherdson. Partial evaluation in logic programming. *The Journal of Logic Programming*, 11(3–4):217 – 242, 1991. pages 47, 127

[Lopes et al., 2010] H. F. Lopes, C. M. Carvalho, M. Johannes, and N. G. Polson. Particle learning for sequential bayesian computation. In *Bayesian Statistics*, volume 9, pages 317–360. Oxford University Press, 2010. pages 72, 127

[Manfredotti et al., 2010] C. E. Manfredotti, D. J. Fleet, H. J. Hamilton, and S. Zilles. Relational particle filtering. NIPS Workshop on Monte Carlo Methods for Modern Applications, 2010. pages 86, 127

[Mansley et al., 2011] C. R. Mansley, A. Weinstein, and M. L. Littman. Sample-Based Planning for Continuous Action Markov Decision Processes. In *Proc. ICAPS*, 2011. pages 107, 127

[Mausam and Kolobov, 2012] Mausam and A. Kolobov. *Planning with Markov Decision Processes: An AI Perspective*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2012. pages 29, 109, 127

[Mcdermott et al., 1998] D. Mcdermott, M. Ghallab, A. Howe, C. Knoblock, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL - The Planning Domain Definition Language. Technical report, CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control, 1998. pages 30, 127

[Metropolis et al., 1953] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953. pages 18, 127

[Meuleau et al., 2009] N. Meuleau, E. Benazera, R. I. Brafman, E. A. Hansen, and M. Mausam. A heuristic search approach to planning with continuous resources in stochastic domains. *Journal of Artificial Intelligence Research*, 34(1):27, 2009. pages 108, 127

[Meyer-Delius et al., 2008] D. Meyer-Delius, C. Plagemann, G. Wichert, W. Feiten, G. Lawitzky, and W. Burgard. A probabilistic relational model for characterizing situations in dynamic multi-agent systems. In *Data Analysis, Machine Learning and Applications*. Springer Berlin Heidelberg, 2008. pages 87, 128

[Milch et al., 2005a] B. Milch, B. Marthi, S. Russell, D. Sontag, D. Ong, and A. Kolobov. BLOG: Probabilistic models with unknown objects. In *Proceedings of the 19th International Joint Conference on Artificial intelligence (IJCAI 2005)*, pages 1352–1359, 2005a. pages 26, 52, 56, 75, 105, 108, 128

[Milch et al., 2005b] B. Milch, B. Marthi, D. Sontag, S. Russell, D. L. Ong, and A. Kolobov. Approximate inference for infinite contingent Bayesian networks. In R. G. Cowell and Z. Ghahramani, editors, *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics*, pages 238–245. Society for Artificial Intelligence and Statistics, 2005b. pages xiii, 54, 105, 128

[Milch, 2006] B. C. Milch. *Probabilistic models with unknown objects*. PhD thesis, University of California, Berkeley, 2006. pages 36, 67, 128

[Munos, 2014] R. Munos. *From Bandits to Monte-Carlo Tree Search: The Optimistic Principle Applied to Optimization and Planning*, volume 7 of *Foundations and Trends(r) in Machine Learning*. Now Publishers, 2014. pages 107, 128

[Murphy, 2002] K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 2002. pages 69, 128

[Murphy, 2012] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. pages 12, 13, 19, 128

[Natarajan et al., 2008] S. Natarajan, H. H. Bui, P. Tadepalli, K. Kersting, and W. keen Wong. Logical hierarchical hidden markov models for modeling user activities. In *Proceedings of the 18th International Conference in Inductive Logic Programming (ILP 2008)*, pages 192–209, 2008. pages 87, 128

[Neal, 2010] R. M. Neal. MCMC using Hamiltonian dynamics. In *Handbook of Markov Chain Monte Carlo*, volume 54, chapter 5, pages 113–162. Chapman & Hall/CRC Press, 2010. pages 18, 128

[Ng et al., 2002] B. Ng, L. Peshkin, and A. Pfeffer. Factored particles for scalable monitoring. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI 2002)*, pages 370–377. Morgan Kaufmann, 2002. pages 70, 128

[Nilsson and Małiszyński, 1995] U. Nilsson and J. Małiszyński. *Logic, Programming And Prolog.* Wiley & Sons, 2nd edition, 1995. pages 1, 21, 24, 129

[Nitti et al.] D. Nitti, V. Belle, T. De Laet, and L. De Raedt. Planning in hybrid relational MDPs. *Machine Learning.* under review. pages 90, 129

[Nitti et al., 2013] D. Nitti, T. De Laet, and L. De Raedt. A particle filter for hybrid relational domains. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2013)*, pages 2764–2771, 2013. pages 59, 129

[Nitti et al., 2014] D. Nitti, T. De Laet, and L. De Raedt. Relational object tracking and learning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2014)*, pages 935–942, 2014. pages 59, 129

[Nitti et al., 2015a] D. Nitti, V. Belle, T. De Laet, and L. De Raedt. Sample-based abstraction for hybrid relational MDPs, 2015a. European Workshop on Reinforcement Learning (EWRL 2015). pages 90, 129

[Nitti et al., 2015b] D. Nitti, V. Belle, and L. De Raedt. Planning in discrete and continuous markov decision processes by probabilistic programming. In *Machine Learning and Knowledge Discovery in Databases: European Conference, (ECML/PKDD 2015)*, volume 9285 of *Lecture Notes in Computer Science*, pages 327–342. Springer International Publishing, 2015b. pages 90, 129

[Nitti et al., 2016] D. Nitti, T. De Laet, and L. De Raedt. Probabilistic logic programming for hybrid relational domains. *Machine Learning*, 103(3):407–449, 2016. pages 32, 59, 129

[Owen, 2013] A. B. Owen. *Monte Carlo theory, methods and examples.* 2013. pages 44, 129

[Papai et al., 2012] T. Papai, H. Kautz, and D. Stefankovic. Slice normalized dynamic markov logic networks. In *Proceedings of the 26th Annual Conference on Neural Information Processing Systems (NIPS 2012)*, pages 1907–1915, 2012. pages 88, 129

[Pearl, 1988] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988. pages 11, 12, 129

[Perov et al.] Y. Perov, B. Paige, and F. Wood. The indian GPA problem. URL `http://www.robots.ox.ac.uk/~fwood/anglican/examples/viewer/?worksheet=indian-gpa`. pages 2, 51, 129

[Peshkin and Shelton, 2002] L. Peshkin and C. R. Shelton. Learning from Scarce Experience. In *Proceedings of the 19th International Conference on Machine Learning (ICML 2002)*, pages 498–505, 2002. pages 108, 130

[Pfeffer, 2001] A. Pfeffer. IBAL: A probabilistic rational programming language. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 733–740, 2001. pages 56, 130

[Pfeffer et al., 2009] A. Pfeffer, S. Das, D. Lawless, and B. Ng. Factored reasoning for monitoring dynamic team and goal formation. *Information Fusion*, 10(1): 99–106, Jan. 2009. pages 87, 130

[Pitt and Shephard, 1999] M. K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American statistical association*, 94 (446):590–599, 1999. pages 72, 130

[Precup et al., 2000] D. Precup, R. S. Sutton, and S. P. Singh. Eligibility traces for off-policy policy evaluation. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 759–766, 2000. pages 108, 130

[Przymusinski, 1988] T. C. Przymusinski. Perfect model semantics. In *Proceedings of International Conference on Logic Programming*, pages 1081–1096, 1988. pages 35, 130

[Robert and Casella, 2004] C. Robert and G. Casella. *Monte Carlo Statistical Methods*. Springer Texts in Statistics. Springer New York, 2004. pages 15, 16, 42, 130

[Russell and Norvig, 2009] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, 3rd edition, 2009. pages 12, 75, 130

[Sanner] S. Sanner. Relational Dynamic Influence Diagram Language (RDDL): Language Description. Unpublished paper. pages 31, 109, 130

[Sanner et al., 2011] S. Sanner, K. V. Delgado, and L. N. de Barros. Symbolic Dynamic Programming for Discrete and Continuous State MDPs. In *Proceedings of the 27th Conference on Uncertainty in Artificial Intelligence (UAI 2011)*, pages 643–652, 2011. pages 30, 108, 109, 111, 130

[Sato, 1995] T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming*, pages 715–729. MIT Press, 1995. pages 3, 26, 32, 34, 130

[Sato and Kameya, 1997] T. Sato and Y. Kameya. Prism: a language for symbolic-statistical modeling. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI 1997)*, pages 1330–1335, 1997. pages 26, 56, 131

[Shelton, 2001a] C. R. Shelton. Policy Improvement for POMDPs Using Normalized Importance Sampling. In *Proceedings of the 17th Conference in Uncertainty in Artificial Intelligence (UAI 2001)*, pages 496–503, 2001a. pages 108, 131

[Shelton, 2001b] C. R. Shelton. *Importance Sampling for Reinforcement Learning with Multiple Objectives*. PhD thesis, MIT, 2001b. pages 95, 108, 131

[Shirazi and Amir, 2011] A. Shirazi and E. Amir. First-order logical filtering. *Artificial Intelligence*, 175(1):193–219, 2011. pages 86, 131

[Smart and Kaelbling, 2000] W. D. Smart and L. P. Kaelbling. Practical reinforcement learning in continuous spaces. In *Proceedings of the 17th International Conference on Machine Learning (ICML 2000)*, pages 903–910, 2000. pages 93, 107, 131

[Srivastava et al., 2014] S. Srivastava, S. J. Russell, P. Ruan, and X. Cheng. First-order open-universe pomdps. pages 742–751, 2014. pages 108, 131

[Storvik, 2002] G. Storvik. Particle filters for state-space models with the presence of unknown static parameters. *Signal Processing, IEEE Transactions on*, 50(2):281–289, 2002. pages 71, 131

[Sutton and Barto, 1998] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998. pages 27, 92, 95, 96, 98, 131

[Tadepalli et al., 2004] P. Tadepalli, R. Givan, and K. Driessens. Relational reinforcement learning: An overview. In *Proceedings of the ICML-2004 Workshop on Relational Reinforcement Learning*, pages 1–9, 2004. pages 109, 131

[Tenorth and Beetz, 2009] M. Tenorth and M. Beetz. KnowRob — Knowledge Processing for Autonomous Personal Robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent RObots and Systems (IROS 2009)*, pages 4261–4266, 2009. pages 87, 131

[Thon et al., 2011] I. Thon, N. Landwehr, and L. De Raedt. Stochastic relational processes: Efficient inference and applications. *Machine Learning*, 82, 2011. pages 87, 131

[Thrun et al., 2005] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, 2005. pages 3, 131

[Vianna et al., 2015] L. G. R. Vianna, L. N. de Barros, and S. Sanner. Real-time symbolic dynamic programming. In *Proceedings of the 29th Conference on Artificial Intelligence (AAAI 2015)*, pages 3402–3408, 2015. pages 30, 109, 132

[Vien and Toussaint, 2014] N. A. Vien and M. Toussaint. Model-Based Relational RL When Object Existence is Partially Observable. In *Proceedings of the 31th International Conference on Machine Learning (ICML 2014)*, pages 559–567, 2014. pages 108, 132

[Walsh et al., 2010] T. J. Walsh, S. Goschin, and M. L. Littman. Integrating sample-based planning and model-based reinforcement learning. In M. Fox and D. Poole, editors, *Proceedings of the 24th Conference on Artificial Intelligence (AAAI 2010)*. AAAI Press, 2010. pages 107, 132

[Wang et al., 2008] C. Wang, S. Joshi, and R. Khardon. First Order Decision Diagrams for Relational MDPs. *Journal of Artificial Intelligence Research (JAIR)*, 31:431–472, 2008. pages 29, 109, 132

[Whiteley and Johansen, 2010] N. Whiteley and A. M. Johansen. Recent developments in auxiliary particle filtering. *Barber, Cemgil, and Chiappa, editors, Inference and Learning in Dynamic Models. Cambridge University Press*, 38:39–47, 2010. pages 62, 132

[Wiering and van Otterlo, 2012] M. Wiering and M. van Otterlo. *Reinforcement Learning: State-of-the-Art*. Adaptation, Learning, and Optimization. Springer, 2012. pages 27, 29, 99, 107, 132

[Wingate et al., 2011] D. Wingate, A. Stuhlmüller, and N. D. Goodman. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, pages 770–778, 2011. pages 19, 132

[Wood et al., 2014] F. Wood, J. W. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *Proceedings of the 17th International Conference on Artificial Intelligence and Statistics (AISTAT 2014)*, pages 1024–1032, 2014. pages 26, 56, 108, 132

[Zamani et al., 2012] Z. Zamani, S. Sanner, and C. Fang. Symbolic dynamic programming for continuous state and action mdps. In *Proceedings of the 26th Conference on Artificial Intelligence (AAAI 2012)*, 2012. pages 30, 109, 132

[Zamani et al., 2013] Z. Zamani, S. Sanner, K. V. Delgado, and L. N. de Barros. Robust optimization for hybrid mdps with state-dependent

noise. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2013. pages 30, 109, 132

[Zettlemoyer et al., 2007] L. S. Zettlemoyer, H. M. Pasula, and L. P. Kaelbling. Logical particle filtering. In *Proceedings of the Dagstuhl Seminar on Probabilistic, Logical, and Relational Learning*, 2007. pages 86, 87, 133

Curriculum Vitae

Davide Nitti was Born in Castellana Grotte, Bari, Italy on July 31st 1983. He went to the secondary school 'Luigi dell'Erba' with IT specialization in the same city and obtained the diploma in 2002. In 2005 he obtained a Bachelor in Computer Science Engineering at the Polytechnic of Bari, Italy. In 2009 he received his Master diploma in Computer Science Engineering in the same University, with specialization in Intelligent Systems. The thesis is titled 'Boolean Games and Description Logics for Multi-attribute Automatic Negotiation'.

In March 2011 he joined the Declarative Languages and Artificial Intelligence (DTAI) group at KU Leuven for doctoral studies under the supervision of Prof. Luc De Raedt. In January 2012 he received a 4 year scholarship from IWT (agentschap voor Innovatie door Wetenschap en Technologie) to work on his PhD on probabilistic programming for robotics. In August 2016 he will defend his doctoral thesis, titled "Hybrid Probabilistic Logic Programming".

List of publications

Journal articles

- D. Nitti, T. De Laet, L. De Raedt. *Probabilistic logic programming for hybrid relational domains*, in Machine Learning, volume 103, pages 307–449, Springer (2016).
- D. Nitti, V. Belle, T. De Laet, L. De Raedt: *Planning in Hybrid Relational MDPs.* Under review at Machine Learning Journal, Springer
- B. Moldovan, P. Moreno, D. Nitti, J. Santos-Victor, L. De Raedt. *Using Relational Affordances for Multiple-Action Two-Arm Manipulation Tasks.* Under review at Robotics and Autonomous Systems Journal

Conference papers

- D. Nitti, T. De Laet, L. De Raedt: *A particle filter for hybrid relational domains.* In: Proceedings of the International Conference on Intelligent Robots and Systems (IROS 2013), pages 2764–2771 (2013)
- D. Nitti, T. De Laet, L. De Raedt: *Relational object tracking and learning.* In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2014), pages 935–942 (2014)
- D. Nitti, T. De Laet, L. De Raedt: *Distributional clauses particle filter.* In: Machine Learning and Knowledge Discovery in Databases, volume 8726 of Lecture Notes in Computer Science, pages 504–507. Springer Berlin Heidelberg (2014)
- D. Nitti, V. Belle, L. De Raedt: *Planning in discrete and continuous Markov decision processes by probabilistic programming.* In: Proceedings of

the European Conference on Machine Learning and Knowledge Discovery in Databases (ECML/PKDD 2015), volume 9285 of *Lecture Notes in Computer Science*, pages 327–342. Springer International Publishing (2015). **Best Student Paper - Machine Learning Journal Award.**

- D. Nitti, I. Ravkic, J. Davis, L. de Raedt: *Learning the Structure of Dynamic Hybrid Relational Models.* Accepted at the 22nd European Conference on Artificial Intelligence (ECAI 2016).

Workshop Papers

- D. Nitti, T. De Laet, M. Hoffmann, I. Thon, G. Van den Broeck, L. De Raedt *A particle filter for probabilistic dynamic relational domains.* In 2nd Statistical Relational AI (StaRAI-12) workshop, (2012).
- D. Nitti, G. Chliveros, L. De Raedt, M. Pateraki, M. Hourdakis, P. Trahanias: *Application of dynamic distributional clauses for multi-hypothesis initialization in model-based object tracking.* In 9th International Conference on Computer Vision Theory and Applications (VISAPP 2014), volume 2, pages 256–261 (2014)
- D. Nitti, V. Belle, T. De Laet, L. De Raedt: *Sample-based abstraction for hybrid relational MDPs.* In: European Workshop on Reinforcement Learning (EWRL 2015)

FACULTY OF ENGINEERING SCIENCE
DEPARTMENT OF COMPUTER SCIENCE
DTAI
Celestijnenlaan 200A box 2402
B-3001 Leuven
davide.nitti@cs.kuleuven.be