Universität Bremen

Institut für Künstliche Intelligenz

Universität Bremen
Am Fallturm 1
28359 Bremen

Probabilistic Action Prospection based on Experiences

Representation, Learning and Reasoning in Autonomous Robotic Agents

Mareike Picklum

Vollständiger Abdruck der vom Fachbereich 03 (Mathematik und Informatik) der Universität Bremen zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

1. Prüfer:
Prof. Michael Beetz, PhD
Universität Bremen

2. Prüfer:
Prof. Dr. Nico Hochgeschwender
Universität Bremen

Die Dissertation wurde im April 2024 bei der Universität Bremen eingereicht und durch den Prüfungsausschuss angenommen. Die Verteidigung fand am 7. Mai 2024 statt.

Artificial Intelligence

Abstract

The concept of autonomous robotic companions assisting with tedious tasks or daily routines has long been a futuristic ambition, especially in scenarios deemed too complex for seamless operation. The multitude of variables, intricate interconnections, and numerous (side) effects on seemingly straightforward influences pose significant challenges for them to function competently without human intervention. However, if robots were equipped with knowledge about themselves, their surroundings, and the objects within it, they could address various queries about their environment and undertake tasks independently, drawing further insights from their own experiences and sensory inputs. For example, they could effortlessly respond to contextual inquiries such as "Where should I position myself in the kitchen to locate the milk carton?", "What route should I take from my current location?" and "Is the refrigerator open or closed?". Addressing uncertainty and its associated limitations is crucial for constructing a comprehensive world model that provides an autonomous agent with the necessary capabilities to operate independently – potentially leading to robots becoming valuable household aids and companions.

This thesis presents BayRoB, a probabilistic framework integrating probabilistic hybrid action models to assist autonomous agents in making informed, context-driven decisions under uncertainty. BayRoB utilizes probabilistic models to represent an autonomous robot's belief state and offers mechanisms to track changes in this state over time. The framework incorporates a novel formalism that enables the learning, representation of and reasoning over joint probability distributions representing action and object designators.

Enabling robots to adeptly handle uncertain situations significantly enhances their decision-making abilities and contributes to their capacity to anticipate action outcomes and environmental changes, thereby promoting autonomy.

The approach of integrating probabilistic hybrid models into a framework, as demonstrated by BayRoB, with learning occurring through experiential data, holds promise for fundamentally enhancing the decision-making processes of autonomous agents.

A critical aspect of this advancement lies in the incorporation of joint probability distributions, encompassing both aspects of the world and the agent itself. This integration is essential for facilitating informed decision-making rooted in experiential knowledge. By incorporating probabilistic models to efficiently learn, represent, and reason across various aspects of the agent and its environment, it becomes feasible to equip autonomous robots with cognitive abilities. This empowerment enables them to accurately predict action outcomes based on context, furnishing the agent with essential tools to make well-informed decisions when selecting optimal actions and parameters for their tasks. The presented approach is the first ever to learn and use such comprehensive joint probabilities for a robotic system.

Experiments showcase that BAYROB is capable of refining underspecified plans and allow reasoning over arbitrary matters of the agent, the available actions and their parameterizations as well as aspects of the agent's environment. A browser-based web interface allows the user to investigate the system's capabilities and reproduce the conducted experiments.

Zusammenfassung

Das Konzept autonomer Haushaltsroboter, die Menschen mühsame Tätigkeiten abnehmen oder bei täglichen Aufgaben unterstützen, ist schon lange ein visionäres Ziel, insbesondere in Szenarien, die als zu komplex für Roboter betrachtet werden. Die Vielzahl von Variablen, komplexen Zusammenhängen und zahlreichen (Neben-)Effekten von vermeintlich kleinen Veränderungen stellen bisher zu große Herausforderungen dar, um sie ohne menschliches Eingreifen zu bewältigen. Wenn Roboter jedoch mit umfangreichem Wissen über sich selbst, ihre Umgebung und allem darin ausgestattet wären, könnten sie beliebige Fragen über ihre Umgebung beantworten und Aufgaben eigenständig erledigen, indem sie weitere Erkenntnisse aus ihren eigenen Erfahrungen in Zusammenhang mit der aktuellen Situation herleiten. Zum Beispiel könnten sie mühelos auf kontextbezogene Anfragen wie „Wo sollte ich mich in der Küche positionieren, um die Milch zu finden?“, „Welche Route sollte ich von meinem aktuellen Standort aus nehmen?“ und „Ist der Kühlschrank geöffnet oder geschlossen?“ antworten. Die Bewältigung von Unsicherheit und ihren damit verbundenen Einschränkungen ist entscheidend für den Aufbau eines umfassenden Weltmodells, das einem autonomen Agenten die notwendigen Fähigkeiten bietet, unabhängig zu agieren – wodurch der Roboter potentiell zu einem wertvollen Haushaltshelfer und Begleiter werden kann.

Diese Arbeit stellt BayRoB vor, ein probabilistisches System, das hybride Aktionsmodelle integriert, um autonome Agenten bei der zuverlässigen, kontextabhängigen Entscheidungsfindung unter Unsicherheit zu unterstützen. BayRoB nutzt probabilistische Modelle, um den Überzeugungszustand eines autonomen Roboters darzustellen, und bietet Mechanismen, um Veränderungen dieses Zustands durch Veränderungen der Welt oder die Ausführung von Aufgaben zu verfolgen. Die Fähigkeit von Robotern, unsichere Situationen sicher zu handhaben, verbessert nicht nur ihre Entscheidungsfähigkeit sondern trägt auch dazu bei, Effekte und Umweltveränderungen vorherzusehen, was zur Eigenständigkeit beiträgt. Das System integriert einen neuartigen Formalismus, der das Lernen, die Repräsentation und das Schlussfolgern über Verbundwahrscheinlichkeiten für Aktionen und Objektbezeichnungen ermöglicht.

Der Ansatz, probabilistische, hybride Modelle in ein System zu integrieren, dessen Lernen durch erfahrungsbasierte Daten erfolgt, wie von BayRoB demonstriert, verspricht eine grundlegende Verbesserung der Entscheidungsprozesse autonomer Agenten. Ein entscheidender Aspekt dieses Fortschritts liegt in der Integration von Verbundwahrscheinlichkeiten, die sowohl Aspekte der Welt, möglicher Aktionen als auch des Agenten selbst umfassen. Diese Integration ist entscheidend für die Förderung informierter Entscheidungsfindung, die auf erfahrungsbasiertem Wissen beruht. Durch die Integration probabilistischer Modelle, die das effiziente Lernen und Repräsentieren verschiedener Aspekte des Agenten und seiner Umgebung erlauben sowie das Herleiten weiterer Informationen erleichtern, wird es möglich, autonome Roboter mit kognitiven Fähigkeiten auszustatten. Das ermöglicht es ihnen, Handlungsergebnisse kontextabhängig präzise vorherzusagen und dem Agenten wesentliche Werkzeuge zur Verfügung zu stellen, um fundierte Entscheidungen bei der Auswahl optimaler Aktionen (und deren Parameter) für ihre Aufgaben zu treffen. Der vorgestellte Ansatz ist der erste, der derart umfassende Verbundwahrscheinlichkeiten für ein Robotersystem lernt und verwendet.

Experimente zeigen, dass BayRoB in der Lage ist, ungenaue Pläne zu verfeinern und über beliebige Aspekte des Agenten, der verfügbaren Aktionen und ihrer Parameterisierungen sowie Bereiche der Roboterwelt zu schließen. Eine browserbasierte Schnittstelle ermöglicht es, die Fähigkeiten des Systems zu nutzen und die durchgeführten Experimente zu reproduzieren.

Acknowledgments

As I reach the summit of my doctoral journey, I find it essential to express my heartfelt gratitude to those whose unwavering support has played an crucial role in the realization of this academic endeavor. The completion of this thesis marks not only a personal milestone but also a collective achievement made possible by the guidance, encouragement, and sacrifices of a remarkable group of individuals. In extending my acknowledgments, I would like to recognize the significant contributions of my supervisor, family, and dear ones who have been constant pillars of support throughout this challenging yet rewarding academic pursuit. This acknowledgment is a sincere tribute to those whose impact has shaped not only my academic journey but also my personal growth. Thank you for being a crucial part of this transformative experience.

I would like to express my deepest gratitude to my supervisor, Michael Beetz, whose guidance, expertise, and unwavering support have been invaluable throughout the entire journey of my PhD research. Your mentorship has played a pivotal role in shaping the direction of this thesis. I would also like to express my gratitude to Nico Hochgeschwender for graciously agreeing to serve on my thesis committee.

I am sincerely grateful for the invaluable assistance provided by my family as well as my boyfriend's in caregiving, which has allowed me the precious time to focus on advancing my thesis work. I extend heartfelt appreciation in particular to my mother Erika and my sister Verena, for their unconditional love, encouragement, and sacrifices that allowed me to pursue my academic aspirations. Their unwavering belief in my abilities has been a constant source of motivation. I am truly grateful for your support.

Special thanks go to my boyfriend, Daniel, for being a pillar of strength and understanding the demands of this academic endeavor and for being a source of both emotional and intellectual support. Beyond offering encouragement, you generously shared your insights, offered constructive feedback, and lent your expertise, making a significant contribution to the depth and quality of this work. Your patience, encouragement, and belief in me have made this journey more meaningful and enjoyable. Special thanks to our precious son, who patiently endured our distracted moments while we focused on

thesis discussions: your resilience were our guiding lights through it all. We are endlessly grateful for your unwavering love.

I would also like to thank my dear friend Anna and my boyfriend Daniel for proofreading my thesis. Your insightful suggestions have greatly enhanced the quality of my work.

I am grateful to my colleagues for the stimulating discussions, collaborative efforts, and the sense of camaraderie that made the research environment enriching and enjoyable. Your insights and feedback have been invaluable.

Lastly, I want to express my appreciation to all those who have contributed in various ways, directly or indirectly, to the completion of this thesis. Your support has been vital, and I am truly grateful for the collective efforts that have shaped this academic achievement. I am truly fortunate to have had such a supportive network of individuals, and I am sincerely thankful for each one's contribution to my academic journey.

April 2024
Mareike Picklum

This work has received funding from the Collaborative Research Center (CRC) SFB1232 by the German Research Foundation (Deutsche Forschungsgemeinschaft (DFG)) (project number 276397488), from the European Union Seventh Framework Programme (FP7) projects RoboHow (grant number 288533) and SHERPA (grant number 600958) and from Research Grants DFG Programme PIPE (project number 322037152).

Contents

FIGURES

TABLES

Algorithms and Code

Appendix

chapter One

Introduction

In the not-so-distant future, *Artificial Intelligence (AI)* will be part of our everyday lives, far beyond the form that we already see today, as tireless aides that help us navigate unknown regions[1], as smart companions answering questions we type into our phones or formulate in spoken language[2,3,4,5,6] or as one of the countless little helpers developed to make our lives easier and more enjoyable[7,8,9]. What these (to a greater or lesser extent) *intelligent* agents have in common is that they do not take on *physical* but rather organizational or virtual tasks. But what about tedious and time-consuming household chores? Soon, robotic agents will play a crucial role in our future household landscape, where intelligent autonomy permeates the very essence of our daily routines.

Autonomous robots, equipped with advanced sensors, AI, and precise mechanics, have the potential to profoundly impact the way humans conduct their daily lives. The allure of incorporating them into our homes lies not only in their ability to alleviate the burdens of tiring and laborious tasks, but also in their capacity to enrich our lives by enabling us to focus on more meaningful and fulfilling endeavors. The idea of robots taking on hard or tedious tasks is not merely a futuristic fantasy, but an imminent reality that could redefine the boundaries of domestic routines. Imagine a future where household chores like vacuuming, cleaning, dishwashing, and even laundry are seamlessly handled by autonomous robots, allowing individuals to focus on much more important and enjoyable things in life. This liberation from everyday duties can empower individuals to engage in creative pursuits, spend quality time with loved ones, or in-

[1] Google Maps, navigation software
[2] ChatGPT
[3] Apple Siri
[4] Amazon Alexa
[5] Google Assistant
[6] Microsoft Bing
[7] ELSA speak and Duolingo, AI to help users learn/improve foreign language skills
[8] Wysa and Youper, AI mental health apps
[9] Socratic, powered by Google AI to help students with their homework

vest in self-improvement activities that might otherwise be sidelined by the demands of daily upkeep.

social benefit ▷

The introduction of autonomous robots into the household environment can also carry the potential to significantly improve the lives of people who require assistance in their daily routines. Elderly and differently-abled people may substantially benefit from robotic assistants as they get back a piece of independence from others. These robots can provide companionship, shoulder hard or tedious tasks, and even monitor health parameters, thereby promoting a sense of independence and vastly enhancing the life quality for those who need it most. The combination of technology and caregiving not only lightens the load on caregivers but also provides independence and dignity for people with limitations. From a broader perspective, the deployment of autonomous robots in households can also contribute to sustainable and enjoyable living practices, thereby rendering them valuable not only to people who *require* but also those who *desire* assistance.

Envision a scenario where you have a robotic companion adept at assisting with daily tasks, such as cooking, setting tables, and tidying up the kitchen afterward. This robot would possess the ability to autonomously navigate the kitchen, identify obstacles, and adjust its course accordingly. Moreover, it would be equipped with knowledge about the kitchen layout and the locations of essential items like tableware, food, and utensils. The robot could discern whether certain objects were within reach, and it would be aware of the status of drawers and cabinet doors. With each completed task, the robot would accumulate experience, using it to learn and enhance its comprehension, or *model*, of the environment. This model would enable the robot to address various questions about its surroundings and perform tasks. For instance, it could effortlessly respond to contextual queries like “Where should I stand in the kitchen to see the milk carton?”, which would pose a typical query when tasked with the very common task of finding objects, in this case the milk carton. Following up, questions like “How do I get there from my current position?” and “Is the fridge open or closed?” show the necessity for possessing a comprehensive world model enabling an agent to answer all those questions. In terms of performing a physical task, a question could also be “how do I flip a pancake such that it lands on my spatula again, without breaking it?” which requires even more sophisticated reasoning about joint poses, the use of force and so on. So, what are the possibilities and advantages that would emerge? In particular, it would be possible to *implement behavior changes based on reasoning about anticipated effects*, or, in probabilistic terms, to query for the most probable parameter settings for a certain (desired) outcome, e.g.:

$$\underset{pose_{\text{arm}},\ force,\ldots}{\arg\max} \quad P(pancake\_on\_spatula = \top, pancake\_intact = \top \mid pose_{\text{arm}}, force, ...) \tag{1}$$

with all the information needed for such a probabilistic query being learnt from data. Such information would contribute substantially to enable a robot to efficiently execute tasks in real-world scenarios. It also provide a key capability for the agent to adapt to new conditions, autonomously make decisions and solve problems, since (under uncertainty) Bayesian inference provides the optimal solution.

So, why hasn't this become a reality yet? Why can't we just put a robot in our house and tell it to do things that are annoying yet seem to be very easy? One challenge lies in generating a comprehensive world model. The primary obstacle is the high-dimensionality of the real world due to its dynamic and continuously changing nature. In acting in real-world scenarios, the robot faces what is called the *open world challenge*, requiring it to adapt to an array of infinite scenarios and interact effectively with its environment and other agents. This implies a vast state space that the robot must handle. Consider a simple example calculation: If we restrict the robot's possible positions to a 100×100 grid on the kitchen floor and discretize its facing direction into 360 angles, there are already 3.6 million potential states. Introducing factors such as open or closed doors and drawers, visible or hidden objects, leads to a combinatorial explosion, since the state space grows exponentially with the number of variables. For instance, with just 5 doors or drawers and 10 visible objects, the total number of states balloons to $3{,}600{,}000 \cdot 2^5 \cdot 2^{10} = 117{,}964{,}800{,}000$ (in words: one hundred seventeen billion nine hundred sixty-four million eight hundred thousand!). The challenges and difficulties that arise when working with such high-dimensional data is often referred to as the *curse of dimensionality*.

These vast numbers illustrate why robotic agents often operate in highly discretized environments under controlled, thus very limited, conditions. This complexity is a key reason why we are still a long way from realizing futuristic visions of androids seamlessly handling everyday tasks in a human-like manner. Despite handheld agents becoming commonplace, recent surveys, comprising a Delphi study and an accompanying population survey (Engel 2020) regarding perspectives of AI suggest that only a minority perceives robots and AI as reliable, safe, and trustworthy technologies, particularly for close-contact services in domestic or caregiving settings. The prevailing view is that robots have yet to demonstrate competence in performing such tasks compared to humans. Generally speaking, activities, that are easy for a human being can be very hard for a robotic agent:

> *"But as the number of demonstrations has mounted, it has become clear that it is comparatively easy to make computers exhibit adult level performance on intelligence tests or playing checkers, and difficult or impossible to give them the skills of a one-year-old when it comes to perception and mobility"*
>
> *–Moravec (1988)*

A human knows from *experience*, and without explicitly making it present that a task usually consists of a number of subtasks, which again consist of even simpler subtasks that have to be executed appropriately. A robot, by default, does not know that it has to do that, let alone *how* to do it, and expects its programmer to feed it all the information it needs to execute the task at this time, in this environment, with its capabilities and tools provided. What contributes to the robot being *intelligent* is the capability to break away from the necessity of human intervention and autonomously act in a human environment by solving tasks all by itself. This requires *understanding* the task and

deep learning ▷

identifying *what* needs to be done and *how*. Deep Learning is frequently utilized for all sorts of tasks, with the underlying premise being that the larger the volume of data accessible, the greater the likelihood of having the requisite information for the task execution. This methodology has demonstrated remarkable efficacy across various domains, including image recognition and classification, parameter learning (Voigt, Johannsmeier, and Haddadin 2020), *Natural Language Processing (NLP)* or, just recently, robotic manipulation (Mitash et al. 2023; Hidalgo-Carvajal et al. 2023). Whenever the key to a solution lies in recognizing recurring patterns, deep learning persuades with remarkable results. These black-box approaches are very well-suited for restricted applications within well-defined bounds, however, they do not pose real common-sense reasoning.

Household robotics, as opposed to the technological developments of industrial and scientific applications in the past, is still a field that introduces an entirely new field of challenges, as agents need to interact with the open world, adapt to the smallest of changes in their physical environment and detect and recover from failures by themselves. Robotic household companions have not found their way into our daily lives yet because they lack the ability to act competently in this environment. In an open environment with open task descriptions, the primary challenge lies in the need for flexible and robust decision making during the execution of a task as it is too complex to model every single imaginable scenario beforehand, at development time. This entails expanding knowledge during runtime through experiential learning and adapting to changing conditions. While robots are superior to humans when it comes to repetitive tasks that require precision and strength at high speed, the competence with which a human can seemingly effortlessly undertake tasks in complex unstructured environments remains unparalleled by robots.

Undoubtedly, the foundation of intelligent human behavior and decision-making rests upon a bedrock of common-sense knowledge. This knowledge is acquired through a triad of essential principles: learning from observations and experience, engaging in meaningful discourse with domain experts, and absorbing information from scholarly sources. While we have seen significant advances machine learning has made especially in commercial applications, it falls short in achieving the stability and resilience necessary for robust and versatile AI systems. Leading researchers and technologists argue that our current AI capabilities are insufficient to meet the ambitious expectations we hold for the future of AI technology (Marcus and Davis 2019). To bridge this gap and usher in the era of more reliable and adaptable AI systems, it is imperative to incorporate common-sense reasoning into artificial systems. This foundational ability, shared by humans, allows us to perceive, understand, and evaluate everyday matters effortlessly.

Furthermore, understanding the intricacies of cognition, which encompasses the inference of new information from sensory input and prior knowledge, is pivotal (Chater, Tenenbaum, and Yuille 2006). Cognition plays a crucial role in the *Strategic Research Agenda (SRA)* and the *Multi Annual Roadmap (MAR)* presented by SPARC as it empowers agents to make sense of an uncertain world, both in the present and the future, influencing their actions (SPARC 2014; 2016). The *20-Year Community Roadmap for Artificial Intelligence Research in the US* (Gil and Selman 2019) by the *Computing*

Community Consortium (CCC) of the *Computing Research Association (CRA)* also lists *Integrated Intelligence* as a research priority to realize societal benefits of AI which includes the development of models of human cognition.

> *"Cognition is the process by which an autonomous system perceives its environment, learns from experience, anticipates the outcome of events, acts to pursue goals, and adapts to changing circumstances."*
>
> *–Vernon (2014)*

One could argue that the common-sense knowledge that humans acquire enables them to make rational decisions based on statistical evidence, given sufficient data acquired through experience. But while being amazingly appropriate for a given situation in the majority of cases, decision making in the human mind sometimes produces puzzlingly unreasonable or irrational results. Humans sometimes make decisions that are biased by the *availability heuristic*, which supports a certain hypothesis over another when there is more data available about it, as it distorts the perception of its relevance. Kahneman (2017) argues that our minds are susceptible to systematic errors because we are not able to see beyond or even acknowledge our biased view. He describes the two-systems approach to judgment and choice by differentiating System 1 and System 2, which he thinks of as to agents, producing fast (intuitive) and slow (deliberate) thinking. System 1 functions automatically, requiring minimal to no conscious effort while lacking a sense of deliberate control. System 2 directs attention toward mentally demanding tasks, such as complex calculations, where effort is required. The functioning of System 2 is frequently linked to the subjective sense of autonomy, decision-making, and focused concentration. In the two-systems approach, it is assumed that humans merely use resemblance as a convenient heuristic to reach a decision, which is precisely not the reliance on statistical evidence. This, however, is what makes it difficult to mimic human behavior in computational systems.

Researchers have increasingly appreciated the importance of probabilistic reasoning and knowledge representation in the field of Bayesian cognition sciences, using a probability theoretical approach to model human cognition. There is a lot of information hidden in large amounts of data and building joint probability distributions is one way to extract knowledge from it. The term *Bayesian Brain* describes just that: the theory that there are hidden structures in the human brain that perform Bayesian inferences (Friston 2012) about its perceptions and experiences which then dictate its owner's behavior. The aspiration of researchers in this field is to create robots that exhibit common-sense intelligence and human-like reasoning to emulate their behavior. As demonstrated by Griffiths, Kemp, and Tenenbaum (2023), probability theory can be viewed at as an elaborate mathematical apparatus for elucidating and enacting theories of cognition. They argue that Bayesian models offer a means to tackle profound inquiries pertaining to uniquely human cognitive processes. Tenenbaum et al. (2011) point out that the capacity of probabilistic generative models to demonstrate how individuals can truly acquire abstract structured knowledge is their most distinguishing

◁ *Bayesian cognition theory*

aspect and argue that the Bayesian approach helps advance our understanding of cognition, part of it being the functioning and growth of the human mind. However, generalizations in probabilistic knowledge bases is challenging due to the representational and computational complexity.

> *"A cognitive architecture determines the organisation of the system's cognitive functions. It provides the infrastructure for embedding knowledge, acquiring new knowledge, and using that knowledge to understand the world, to act purposefully, and to anticipate the need for action. It can also provide a framework to allow new skills to be developed through experience."*
>
> *–SPARC (2014)*

The desiderata for cognitive systems are multifaceted, requiring a harmonious blend of logical, statistical, and causal reasoning that incorporates symbolic and subsymbolic elements. In essence, as we strive for the development of AI systems that can match human intelligence and decision-making capabilities, the integration of common-sense reasoning and a deeper understanding of cognition is not just a necessity but the key to unlocking the full potential of AI. It is through these innovative approaches that we can pave the way for AI systems that are not only intelligent but also resilient, versatile, and capable of making informed decisions in an ever-evolving world. Vernon, Beetz, and Sandini (2015) recommend the use of joint episodic memory to facilitate prospection and goal-directed action in cognitive robotics and argue that the combination of episodic and procedural memory enables the internal simulation to be influenced by the present context, semantic memory, and the agent's set of values.

1.1 Probabilistic Cognitive Action Models

In this thesis, probabilistic hybrid models over actions and their parameterizations are developed as a step towards a cognitive architecture to enable robotic agents to adeptly perform everyday human activities. Probabilistic models are the apt choice for this endeavor, as they align with the idea that human cognitive reasoning operates in a probabilistic manner. These models allow to estimate the likelihood of different actions and their outcomes, providing a foundation for decision-making that mirrors the complex probabilistic computations that underlie human cognition.

EASE ▷

There has been tremendous progress in emulating human knowledge- and information processing in robotic agents. The *Collaborative Research Center (CRC) Everyday Activity Science & Engineering (EASE)* has already substantially contributed to the field of artificial cognitive systems by developing cognitive robots that are capable of executing everyday activities with extraordinary manipulation skills. The conceptual framework of EASE can be seen as a cognitive architecture incorporating concepts of human cognition. *Narrative-enabled Episodic Memories (NEEMs)* refer to specialized data structures that empower robotic agents to extract information from extensive collections of

observations, experiences, or detailed accounts of activities. NEEMs serve the purpose of identifying representations that can leverage the inherent structure within these activities, making it possible to transition tasks into problem domains that are computationally more manageable than the original ones. Essentially, NEEMs help robots organize and make sense of their experiences for improved problem-solving. NEEMs aim to model an artificial memory system designed for artificial agents drawing inspiration from the human episodic memory system. *Pragmatic Everyday Activity Manifolds (PEAMs)* are representations used to capture and describe various aspects of everyday activities in a low-dimensional, localized manner, similar to mathematical manifolds. These PEAMs are employed to help agents accomplish their intended tasks efficiently while maintaining computational feasibility. In essence, they enable agents to navigate and understand the intricacies of everyday activities in a more manageable and practical manner.

Nyga (2017) introduces a natural-language instruction interpreter capable of addressing the challenges posed by vagueness and ambiguity in human language which is able to deduce any missing information necessary for a robot to carry out the given instruction effectively by generating robot-interpretable plans. The approach poses an implementation of a symbolic relational probabilistic knowledge base over actions along with its parameterizations. To achieve the plan generation, the task of instruction comprehension is framed as a reasoning problem within first-order probabilistic knowledge bases. Specifically, the system employs *Markov Logic Networks (MLNs)* as a formalism to represent uncertain knowledge. Symbolic concepts incorporating ontological knowledge from taxonomies are implemented which heavily make use of semantically similar relational structures to infer unknown concepts. The presented approach is therefore limited to symbolic descriptions.

Dealing with uncertainty requires finding solutions for some of the hardest tasks in computer science, such as dealing with the complexity of decision spaces, as uncertainty tends to add magnitudes of intricacy to a given problem. Applications in the real world also suffer from partial observability due to the sheer vastness - and therefore impossibility to represent its entirety - of the open world. The absence of probabilistic hybrid models capable of seamlessly handling both symbolic and subsymbolic features represents a significant problem in the field of AI and *Machine Learning (ML)*. Symbolic reasoning is crucial for tasks requiring explicit logical inference, knowledge representation, and high-level abstraction, while subsymbolic techniques like deep learning excel in pattern recognition and complex data processing. Yet, most real-world problems demand a combination of these two paradigms to effectively merge structured information and raw sensory data. Without such versatile models, AI systems struggle to flexibly navigate the spectrum of cognitive tasks, hindering their ability to provide meaningful, context-aware solutions. Addressing this gap in hybrid modeling is essential for advancing the frontiers of AI and enabling systems to tackle the multifaceted challenges of the modern world more effectively.

The necessity for hybrid models and formalisms capable of efficiently representing joint probability distributions serves as a fundamental driving force for the research within this thesis. Using probabiliy theory to represent knowledge allows to extract the optimum from the underlying data, as one can query for the highest expected utility for

the given task. This work aims at learning what happens, when the underlying data are experience data from robots, whether one can learn and reason probabilistically from it and what is the best intended sequence of actions to solve a task given the experience data. The challenge lies in the very complex structure of actions and sequences. In particular, actions and sequences suffer from high dimensionality, which typically renders probabilistic inference infeasible. To represent a joint probability for a given task one needs to 1) determine how to get sufficient data that is required to appropriately represent it, 2) handle the fact that some important information might not be observable and 3) figure out how to perform reasoning effectively and efficiently. This work presents Bayesian Robot Brain (BAYROB), a framework using a probability theoretic approach to generate and improve action models from experience which is aligned with the principle of NEEMs mentioned before. The approach implies that decisions for certain actions can be made in an informed way. In particular, the generated action models can be used for prospection to enhance the planning skills of an autonomous agent. This work integrates in the research focus of EASE and will make use of knowledge collected from an agent's own experiences and those of others.

In the approach presented here, action cores are used, which are (potentially higher-level) action descriptions. BAYROB adds a subsymbolic component to the parameterization of action cores, thereby creating hybrid action models. Let's assume we have two (in this case rather low-level) action cores, `move_base` and `turn`. The `move_base` action has a single numeric control parameter `dist` for the distance to travel, the `turn` action gets a numeric parameter `angle` to determine how many degrees to turn. Each call of one of these two actions is considered to be *one* robot movement at a certain point in time, even though a robot might be able to execute them simultaneously. Now what one is particularly interested in, is the outcome of the execution of such an action. A human has sophisticated anticipation capabilities (Williams 2018; Szpunar, Spreng, and Schacter 2014; Jeannerod 2001), such that the outcome of their actions is typically equal or very close to their expectations. However, telling a robot to do something is a far cry from getting the result one would expect. Every robot manipulation comes with a degree of uncertainty, conditioned by inaccuracies in its sensors, its actuators or extrinsic influences. To capture and track such uncertainties, it is necessary to create a realistic model of the behavior of itself and the physical effects actions have on the world around it.

Beetz and Grosskreutz (2005) present a framework for modeling and predicting concurrent behavior in autonomous mobile robots using a formalism that uses continuous feedback control processes to model more realistic robot behavior and physical effects. The aim is to improve the adaptability and reliability of autonomous robots especially in complex and dynamic environments. The robot behavior is predicted with high probability, aiding in plan revision and decision-making during execution. The system can identify potential flaws or undesired outcomes and revise the plan in real-time. Essential in the authors' approach is the representation of continuous feedback control processes, non-deterministic effects and exogenous events, which is key to appropriately act in the open world.

One way to encapsulate specific conditions that a robot perceives as true about itself and its surroundings, along with the anticipated physical outcomes stemming from its

own actions or external events, is to maintain a belief state. The advantages of probabilistic models make it tempting to represent this belief state as a joint probability distribution covering both the world and the robot. This distribution encompasses variables that describe the current state of the agent and its entire environment. However, especially in real-world scenarios, constructing such models often poses computational and representational challenges, or it necessitates making substantial assumptions about the underlying distributions, which may not accurately mirror the actual real-world conditions.

A robot opening a fridge: While this scenario is nothing out of the ordinary in movies, in reality we are still far from a world, in which robotic agents perform everyday human activities. | *Figure 1*

BAYROB incorporates a formalism that not only enables the concise representation of complex probability distributions but also makes them computable in hybrid domains, marking a substantial advancement in the area of probabilistic modeling. By addressing the inherent challenges of representing joint probabilities in domains that blend discrete and continuous variables, this work opens up new avenues for efficiently modeling and analyzing real-world phenomena with a high degree of precision. The devel-

opment of this formalism promises to greatly enhance the capabilities of probabilistic models in various fields, including decision-making, error detection and -recovery, and more, ultimately contributing to the advancement of science and technology.

A belief system employing this formalism represents action cores and their consequences as probability distributions over the variables associated with the action core. To move closer to a more human-like anticipation model, the ability to predict the outcome of specific actions becomes pivotal. For instance, in the previously mentioned example, the model for the action core `move_base` encapsulates the agent's expectation of how its position changes when it moves a certain distance in the forward direction. Chapter 4 will elucidate how, by leveraging probabilistic knowledge without resorting to simulation, the transition from one belief state to the next can be achieved through the addition of these distributions. But does that also work for multiple (time) steps? If the answer is yes, one could generate action *sequences*. The implications on the decision-making skills of such a formalism employed in a (robotic) agent are manifold.

Anticipation · The agent would be capable to project the effects of its actions into the future and therefore enhance its anticipation skills. By enabling the agent to predict the consequences of various actions before execution, it can make more informed decisions. Given the current position of the agent and a (parameterized) action to execute, the agent can anticipate (under the uncertainty of the underlying model), what its updated position after a successful task completion will be. It can also anticipate effects in the outside world, for instance, in a dynamic environment, a robot can predict the position of a moving object and plan its actions accordingly, leading to better interaction and coordination. This capability also aids in risk assessment, allowing the robot to proactively avoid potential collisions or hazards. In the context of complex tasks, such as navigating through obstacles or manipulating objects, the ability to foresee the implications of each action fosters efficiency and adaptability, as the robot can adjust its strategy based on projected outcomes.

Example CRAM plan: The task to move to a specific location will be divided into a sequence of parameterized subtasks | *Code 1*

```
1  defplan path(goal_location, robot_location)
2      perform (an action
3                  (type move-forward)
4                  (distance ?dist))
5      perform (an action
6                  (type turn)
7                  (angle ?deg))
8      perform ...
```

Complex Tasks · Beginning at a very low level, one single motor action, e.g. one `move_base` step alone does not get a robot very far, given a complex household task. The ability to project a certain outcome of an action, however, allows the robot to look into the future and forms the basis for another, subsequent action. Knowing its own current position and its new position after executing the `move_base` action, allows the robot to plan ahead and generate the step after the next one and project its new position.

Chained together these sequences of actions allow the agent to anticipate trajectories of its own position or those of moving objects around it. Code 1 shows an exemplary plan in pseudoized *CRAM Plan Language (CPL)* for finding a path to a specific location. The single steps can be generated by starting at the current location of the robot and consecutively selecting the action that, projecting its outcome, most likely results in a state that brings the agent closer to the goal. The new state then serves as a starting point for the next step, or probabilistically speaking, as the *evidence* for the next projection.

Applied to higher-level, more abstract tasks, the consecutive execution allows to anticipate the overall result of an entire action sequence, ultimately enabling the agent to perform much more sophisticated operations. With the underlying probabilistic model, the action sequences can be generated guided by the user's preferred criterion, be it the confidence level of the success of the overall task, speed (tolerating a risky behavior of the robot including possibly breaking dishes), or any other definition of high utility. The increase in uncertainty and entropy that arises from prospecting future states probabilistically from an already uncertain state poses a significant challenge, which has to be addressed when generating action sequences using a distribution-based approach. Section 4.6 will demonstrate, showcased by various examples in Section 6.4 that indeed, employing the same approach repeatedly enables the generation of an entire sequence of actions.

Effect Control · The projection of consequences also allows the agent to select actions with the highest probability of generating a desired effect, thus optimizing its decision-making process. When tasked with cleaning a room, the robot can anticipate the consequences of different cleaning methods, such as using a vacuum cleaner versus a broom. It will consider factors like cleanliness level, energy consumption, and noise, ultimately choosing the method that aligns best with the homeowner's preferences. In a kitchen setting a robot can evaluate various routes and actions to choose the the one not only with the highest likelihood of success given the goal description (e.g. a set-up breakfast table), but also with the lowest number of repetitive steps, the shortest paths, the least broken cups etc. It can also anticipate the effects of various cooking techniques, such as frying versus baking, taking into account taste, nutrition, and cooking time to make informed choices. This ability to foresee outcomes lets the robot adapt to changing circumstances and make decisions that optimize its performance. By selecting actions and, in particular, action *parameters*, with the highest utility based on projected outcomes, the robot can control the effects of its actions and therefore operate more effectively, efficiently, and safely in a wide range of applications, ultimately enhancing its overall performance.

Error Treatment · Anticipation skills are vital when it comes to identifying and avoiding failures. Which path is more preferable or which cooking technique is favored over another highly depends on the task, its circumstances and the utility defined by the instructor and can change quickly in an open world. If the preferred action can not be executed for some reason, for example, because certain kitchen utensils are missing or other preconditions for a task completion are not met, the agent can foresee that the task will fail and select another action that is more likely to succeed. The identification of failures is one example and a challenge of its own as it requires the developer of robotic agents to be somewhat omniscient of the open world. There is an intimidating

spectrum of possible failures which can lead to a decreased number of actions that the robot can confidently execute. Robots heavily rely on sensors to gather information about the world. However, these sensors can be prone to inaccuracies, noise, and limitations in certain situations. When the robot's perception is compromised by noisy or incomplete data, it may struggle to accurately assess its environment, leading to suboptimal decisions and a reduced set of viable actions. For example, if a robot's sensory data is unreliable, it might avoid risky actions that require precise perception, thereby limiting its range of feasible behaviors. One failure can also lead to a chain reaction of other failures. For example, a sensor malfunction could cause the robot to misinterpret its environment, leading to misguided actions that exacerbate the problem. When facing unexpected circumstances, the robot might also become cautious and stick to a smaller set of actions it deems safe, rather than exploring more diverse options. Depending on its error handling, severe failures might force the robot to rely more heavily on human intervention or supervision. This loss of autonomy limits the robot's ability to operate independently and perform tasks efficiently. So how can we avoid the need for being all-knowing and yet build an agent that is self-confident enough to solve our tasks? Part of the solution lies in the insight that failures introduce *uncertainty* into the decision-making process leaving us with the understanding that in order to be a confident actor in the open world an agent has to be allowed to be *uncertain* about things and learn from previous mistakes and experience - its own and those of others. In an open-world scenario, we often lack precise knowledge about the specific errors that might manifest. Nevertheless, by leveraging a probabilistic model developed through accumulated experience, we can estimate the likelihood of potential errors occurring. This understanding of when and why failures occur, based on an analysis of the underlying circumstances, enables us to probabilistically identify clusters of failures. These clusters can later be categorized into distinct error classes. The ultimate objective is to establish a self-training system that automatically associates newly emerging errors with the appropriate error class, thereby facilitating a custom-tailored approach to error handling. This informed error detection and recovery process has the potential to wield significant influence, particularly in the domain of robotic systems operating within open environments.

1.2 The Intelligence in Intelligent Agents

In recent years the field of robotics has witnessed remarkable advances that have transformed various industries such as production lines[10] or state-of-the-art science labs[11]. These advancements have led to substantial improvements in efficiency, precision, and overall performance, driving unprecedented levels of innovation and productivity. Modern industrial robots are equipped with advanced sensors and AI-driven algorithms that enable them to adapt to changing production conditions, thereby minimizing downtime and optimizing production schedules. This level of adaptability not only ensures consistent product quality but also enhances the overall efficiency of manufacturing operations.

[10] KUKA

[11] Boston Dynamics

In the domain of science laboratories, robotics has also played a pivotal role in advancing research and experimentation. High-precision robotic systems are now employed in various scientific disciplines, including pharmaceuticals, biotechnology, and materials science. These systems are capable of handling delicate procedures with unparalleled precision, significantly reducing the margin of error and ensuring reproducibility in experiments. As a result, researchers can obtain more accurate data and draw reliable conclusions, ultimately accelerating the pace of scientific discovery. Laboratory automation, powered by robotics, has also enabled researchers to conduct experiments on a larger scale and with higher throughput. This is particularly evident in fields such as genomics (Keeney 2011; Tegally et al. 2020; Aldridge et al. 2013; Laghaee et al. 2005) and drug development (Michael et al. 2008; Schneider 2018), where automated robotic platforms can process large volumes of samples and compounds in a fraction of the time it would take using traditional manual methods. This scalability not only expedites the research process but also facilitates the exploration of a broader range of possibilities.

Furthermore, the rise of collaborative robots, or *co*bots, has transformed the manufacturing landscape. In different cases of application in industrial production lines (Michalos et al. 2022), agents have to operate alongside humans, augmenting their capabilities and taking on repetitive or physically demanding tasks[12]. The integration of cobots has not only improved production precision and efficiency but also enhanced worker safety by reducing the risk of injuries associated with strenuous tasks.

Still, we should not forget that these robots typically possess a relatively limited level of independence and are highly sensitive to changes in their surroundings due to their specialization in a specific task. Marcus and Davis (2019) contend that contemporary AI is inherently limited in scope. It excels at the specific tasks it has been designed for but falters when confronted with unfamiliar situations or challenges. One underlying reason for this limitation is the insurmountable challenge of gathering sufficient data to train machines for every conceivable situation. Consequently, no single *narrow* AI system can amass enough data to comprehensively address the full spectrum of circumstances. The very essence of comprehending a narrative does not align with the narrow, purely data-driven AI paradigm because the world itself is inherently diverse. However, humans often ascribe intelligence to computers when they exhibit superficially human-like behavior, and they easily attribute similar underlying mechanisms that apply to humans. An excellent example for this phenomenon poses OpenAI's[13] infamous *Generative Pre-trained Transformer (ChatGPT)*[14,15], a chatbot that was made available to the broad public in late November 2022. It was quickly hyped as the best AI chatbot ever released to the general public[16] and went viral due to its seemingly unlimited potential uses. It impressed with remarkable capabilities that vary from generating text and translating different languages to perform mathematical calculations and even programming. The conversational competence exceeds that of previously seen chatbots

[12] cmp. the human-robot cooperation of KUKA & BMW and BMW Press

[13] OpenAI

[14] ChatGPT

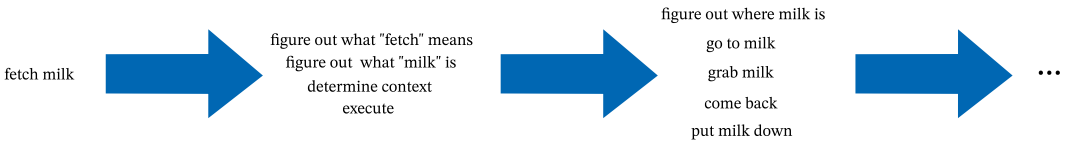
[15] The Guardian on ChatGPT: What is AI chatbot phenomenon ChatGPT and could it replace humans?

[16] New York Times on ChatGPT: The Brilliance and Weirdness of ChatGPT

by far. ChatGPT even published its own paper (King and ChatGPT 2023). However, critical voices highlight the weaknesses of the system (Plebani 2023; Kasneci et al. 2023; Koubaa et al. 2023) that has no access to the internet[17] and is therefore limited to its training data, containing only information up to 2022. It also spreads misinformation, as its answers cannot be taken as facts, therefore, critical thinking is strongly advised. OpenAI itself states that ChatGPT "may produce inaccurate information about people, places, or facts". While users on social media make it a sport to find funny glitches, gibberish responses and errors, one has to admit that ChatGPT almost always delivers them in plausible-sounding sentences.

It is important to keep in mind that the intelligence displayed by supposedly intelligent agents is not genuine, and the mere appearance of intelligence does not necessarily imply true cognitive capability. One tends to overestimate the capabilities of AI fundamentally, being tricked by seemingly complicated tasks the agent can handle. Marcus and Davis (2019) argue that genuine intelligence necessitates capabilities such as reasoning, language comprehension, and analogical thinking, in addition to pure speech and object recognition. These abilities are crucial for achieving a deeper understanding of tasks and for reasoning about the complex interrelationships among objects and individuals that are causally linked. This becomes essential in open-world environments where machines must deduce human intentions, even when those intentions are not explicitly articulated or are ambiguously formulated. In the complicated interplay of human-robot communication, tasks defined in human language often embody rather abstract goal specifications that imply the decomposition into a couple of subtasks for effective execution. This reflects the complexity of human cognition which necessitates robots to interpret, *understand* and break down these tasks into manageable sublevel actions. It is much more natural to ask someone to "Prepare a delicious dinner for our guests!" than to ask for the menu planning, preparation and cooking, the table setting, a possible ambience setup, the serving and timing as well as the cleanup, where each of these subtasks again consist of lower level tasks such as analyzing dietary preferences and restrictions of guests, selecting a variety of food, breaking down each dish into individual cooking steps and so forth.

An alledgedly simple task has to be broken down to numerous subtasks. This has to be carried on until a level is reached that contains robot-interpretable and robot-executable instructions. | *Figure 2*



The robot has to be able to deal with that non-specificity in order to competently act in a human environment. In particular, it has to find a sequence of actions that, when executed, eventually produces an outcome that meets the goal description and satisfies potential additional requirements such as possible time constraints or other quality criteria by itself. Figure 2 shows an example for a supposedly simple task that has to

[17] While the free version GPT-3 still does not have an internet connection (as of Feb. 2024), Plus- and Enterprise users can use plugins such as Bing in GPT-4 to gain access to the internet.

be broken down into smaller subtasks to be robot-executable. The complexity and parameterization level of the subordinate tasks highly depend on the robot's capabilities and the tools it is provided with. For example, a robot that is equipped with two grippers will solve a task differently than one with only one gripper as it might, for example, carry multiple items at once when serving the dishes. The execution order of the subtasks plays a crucial role. Imagine a robot that is charged with the task to prepare the dinner and already knows that there are the subtasks "Menu planning", "preparation and cooking" and "table setting" involved. Executing the "preparation and cooking" action before planning the menu would result in a situation, in which the robot is not able to execute the current or any of the follow-up tasks. If the robot is able to detect its error, it might try to backtrack and fix the mistake, otherwise the overall plan will fail. Keeping track of its own as well as the world's state and how it changes when the robot manipulates it, the agent would have been able to plan ahead and prevent the situation described above.

For a robot to execute the exemplary task above, it would require a complex integration of various capabilities. It has to understand the initial abstract task given in human language, requiring NLP. The robot needs a *Knowledge Base (KB)* to access dietary information, recipes and cooking instructions which needs to be represented in a robot-interpretable way, thereby necessitating an appropriate *knowledge representation*. *Sensory perception* enables the agent to identify and locate ingredients and other objects so the robot can find its way around in different environments. The decompositon into sublevel actions and logical sequencing requires *planning* while the handling of tools, ingredients and other objects calls for a high level of manipulation skills. The robot needs to be aware of its environment and keep track of the changes to observe consequences of its own actions by maintaing a *world state* (apart from its own state). In case something goes wrong, either due to some failure or other unforeseen events, the robot has to be able to detect and handle errors to minimize the damage on the overall task execution. Each of these competences involved forms an entire scientific discipline keeping researchers all over the world busy.

1.3 Contributions & Delimitations

This thesis investigates scalable probabilistic hybrid models, focusing specifically on the concise and computable representation of joint probability distributions and their applications in learning and reasoning for cognitive robots. The main focus lays on the BAYROB system which allows to identify the following main contributions:

CONTRIBUTIONS

This work presents BAYROB, an operational system introducing the process of combining joint distributions to generate action models from the experiences of robots. This methodology facilitates the continuous tracking of an agent's belief state over time. The system uses a formalism that enables the concise representational and computational handling of joint probability distributions within hybrid domains. This aids in making informed decisions, ultimately enabling robots to effectively adapt their own skillsets and plans to new environments, becoming more proficient and more efficient during their operational runtime. This approach forms the foundation for enhancing robot control programs by introducing probability theory over joint probabilities within the framework of probabilistic, static, hybrid automata.

The system is validated experimentally in this thesis by employing separate reasoning systems in two different domains, i.e. 1) in autonomous robotics, where the system aids robotic agents to make more informed decisions, rendering them reliable actors in open-world scenarios and 2) in the materials science field, where the main goal is to develop novel materials based on a requirement profile, specifying the desired properties of a material. These examples serve as a testament to the innovative nature of the research and its potential impact on various domains.

While joint probability distributions are identified as powerful means to deduce meaningful yet hidden information from data, their complexity renders inference on them infeasible for the majority of applications. The formalism employed in BAYROB not only allows to relax those restrictions, but also does that in a white-box fashion, such that generated models are human-readable and interpretable. The presented approach is transferable to various application domains and therefore poses a valuable contribution to the scientific community, in particular in the field of developing autonomous agents.

An open source implementation of *Joint Probability Trees (JPTs)* to learn, represent and reason over hybrid joint probability distributions, which is developed in collaboration with Daniel Nyga and Tom Schierenbeck, is provided. The implementation of JPTs is available as a python package[18] on pypi or as source code on GitHub[19].

However, it is important to note the boundaries and limitations set forth in this work. This thesis does not intend to provide an exhaustive analysis of robot *planning*, nor does it examine the broader implications of designing a fully integrated (robot) software system including natural-language understanding, sensory perception, robot manipulation and many more, as this is well beyond the scope of a doctoral thesis.

Additionally, while this work is part of a larger research motivated by developing cognitive agents emulating human behavior and decision making, this study will not extensively cover the broader discourse of cognitive science. The intricacies of how the

[18] pyjpt
[19] JPTs on GitHub

human brain processes information, while essential, are beyond the immediate scope of this work. Furthermore, the computability and scalability if probability distributions are touched upon, but a comprehensive analysis of the computational complexity theory of the presented approach is not within the purview of this thesis.

In essence, this study concentrates on the presented system BAYROB and its employed probability theoretical formalism as an approach to direct an intelligent robot's reasoning and decision-making towards a more autonomous, human-like behavior. The outlined delimitations help maintain the focus and clarity required to address the specific objectives of this research while acknowledging the broader context in which the presented formalism can be applied. Having outlined the delimitations of this doctoral thesis,

1.4 Reader's Guide

Each individual chapter of this work is as self-contained as possible, allowing for independent reading. The structure of the remainder of this thesis is outlined as follows.

Probabilistic Knowledge Processing · Chapter 2 offers a brief overview of the mathematical framework that serves as the foundation for this thesis. It examines techniques in probabilistic knowledge processing and outlines the core principles necessary for understanding the topic.

Scalable Probabilistic Hybrid Models · Chapter 3 contains a concise examination of the fundamental formalisms in knowledge representation, learning, and reasoning, particularly in the context of probabilistic approaches. It then revisits essential concepts in probability theory and *Probabilistic Graphical Models (PGMs)*, with a specific focus on *Probabilistic Circuits (PCs)*, laying the foundations for the theoretical concepts described in the remainder of the document. The main focus of this chapter lies on the introduction of JPTs as a special case of PCs as they play a central role in the reasoning system introduced in this work.

BAYROB - Bayesian Robotic Brain · Chapter 4 introduces the BAYROB system for combining joint distributions to generate action models from the experiences of robots. It commences with formulating the generation of action sequences as a search poblem. This chapter outlines the principal architectural components of the BAYROB system, illustrates its probability-theoretic approach and proposes a method for generating action models and using them to anticipate the outcome of robot actions. It presents experiments in which the approach has been validated successfully and discusses its various prospected applications.

Probabilistic Knowledge Bases for Material Discovery · Chapter 5 examines an illustrative application of the *Materials CALO (MATCALO)* principle, the BAYROB system employed in the domain of materials science. It has a substantial focus on the insights and outcomes outlined by Picklum and Beetz (2019), the deductions of which form the cornerstone of this chapter, highlighting the method's contributions across a wider array of applications.

Evaluation · Chapter 6 will evaluate the BAYROB system by investigating how well the model translates experiential knowledge into probabilistic hybrid action models, presenting how the system can be used to refine underspecified robot plans and introducing a comprehensive web app that facilitates the access to the system's services.

Related Work · Chapter 7 presents relevant literature, focusing specifically on foundational research, emphasizing works related to learning and knowledge representation, along with the adopted formalism of JPTs and the broader research context with respect to the BayRob system in its entirety.

Conclusions · Chapter 8 will summarize the discoveries made in this research and place them in the context of the BAYROB system's future prospects.

Colored Boxes · Some contents are highlighted to direct the reader to a certain definition, quote or additional piece of information that is considered notably significant. These contents have been strategically designed to seamlessly blend into the surrounding text, preserving the reading flow, while emphasizing particular aspects.

Listings · The document includes a number of supplementary material. The Glossary on p. I ff and the Acronyms listing on p. V ff, provide explanations for context-specific terms that require further clarification within the scope of this work and feature frequently used abbreviations, respectively. Mathematical symbols used in the formal definition of the conceptual framework descriptions are listed on p. XI ff. To maintain a visually pleasing format, the sources of hyperlinks in the document are not presented at the respective location but are instead compiled in an alphabetically-ordered list located at p. IX et seqq. This arrangement ensures that even readers of a hardcopy edition can readily access the linked resources.

Links in Footnotes · Footnotes in general provide supplementary information to a piece of text that is deemed valuable but, if incorporated into the main text, could disrupt the fluidity of reading. They often incorporate hyperlinks that are directly accessible in the digital version. The link sources can also be found in the Links listing on p. IX ff.

Appendix · The appendix on p. A ff contains additional images, images that are already embedded in the document but at a larger scale, installation instructions for the web services presented in this work, and information about media sources. If not stated otherwise in the part, all images, plots and other media are created and owned by the author.

chapter two

Probabilistic Knowledge Processing

This chapter provides a concise introduction to the mathematical framework which this thesis relies upon. It reviews state-of-the-art methods in probabilistic knowledge processing and summarizes the fundamentals essential for comprehending the subject.

2.1 Knowledge Representation Hypothesis

Seeking for a formalism to equip a robotic agent with human-like cognitive capabilities and the ability to learn from experience implies the existence of a suitable knowledge representation framework, or KB, to efficiently store and retrieve information. Shapiro (2009) considers *Knowledge Representation and Reasoning (KR&R)* as a field focused on understanding, designing, and implementing methods for depicting information in computers. This enables programs (agents) to utilize this information for various purposes, including deriving implicit information from it, engaging in conversations with people using natural language, determining the next course of action, planning future activities, and addressing problems in domains that typically demand human expertise. According to Brachman and Levesque (1985), any process that can engage in intelligent reasoning about the world must include, at least partially, a set of structures resembling language, which effectively represent the knowledge and beliefs attributed to the process. The formalization of the *Knowledge Representation Hypothesis*

> *"Any mechanically embodied intelligent process will be comprised of structural ingredients that a) we as external observers naturally take to represent a propositional account of the knowledge that the overall process exhibits and b) independent of such external semantical attribution, play a formal but causal and essential role in engendering the behavior that manifests that knowledge."*
>
> *–Brachman and Levesque (1985)*

states that any intelligent processes will have to use symbolic structures, e.g. databases that represent knowledge, or, as John McCarthy phrases it:

> *"A program has common sense if it automatically deduces for itself a sufficiently wide class of immediate consequences of anything it is told and what it already knows"*
>
> *–McCarthy (1959)*

The field of KR&R explores the challenge of representing essential knowledge and engages in reasoning with it. According to McCarthy (1959), an agent needs to possess three essential capabilities in order to effectively utilize knowledge about the world, that are

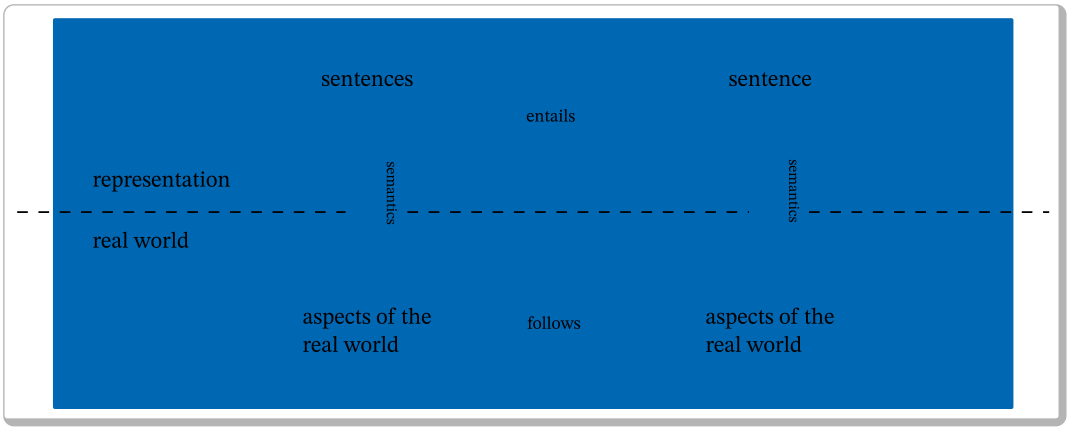
1. a collection of statements or propositions regarding the world, known (or believed) by the agent to be true, which encompasses atomic propositions, rules, and equivalences,
2. an interface designed for the inclusion or removal of new facts from the existing set in this collection and
3. an interface to deduce new beliefs based on those already present in the collection.

The latter involves utilizing representations of statements to derive new propositions. Essentially, the capability to *reason* enables the explicit articulation of propositions that are implied by those already known. The fundamental concept underlying the application of knowledge representation and (logical) reasoning in problem-solving is depicted in Figure 3, which is conceptually taken from Russell and Norvig (2010, Chapter 7). *Representation* refers to a system of symbolic encodings of propositions believed to be true in the real world. Therefore, if A represents B, A serves as a placeholder for B and is typically more accessible. The connection from a representational symbol to its real-world counterpart is termed its *semantics*.

Propositional rules outline the relationships between symbols (where symbols stand for general propositional statements) and serve as a basis for reasoning. The term *reasoning* refers to the process of manipulating symbols within a representation to infer a new representation. These rules are also referred to as *sentences*. For instance, a rule such as "If the agent is at position (x_{in}, y_{in}), facing the kitchen cabinet, then it can de-

tect a cup" can be employed for reasoning, leading to the conclusion that $detected(\text{cup})$ holds whenever $(X_{in}, Y_{in}) = (x_{in}, y_{in})$ is considered true.

Correspondence between the real world and its representation: The syntactic representation (top) of the real world (bottom) allows to deduce *new* knowledge which has to correspond to aspects of the real world that follow from other aspects whose representation has been reasoned about. | *Figure 3*



Unfortunately, the propositions deduced from a representation do not necessarily align with the aspects of the real world as purely logical representations of aspects of the world are typically not applicable to non-deterministic environments. As a result, their practical utility is constrained when dealing with applications characterized by significant ambiguity and uncertainty, where statements tend to be true in some, or even most cases, but never in all of them.

2.2 Uncertainty in Knowledge Representation

In non-deterministic environments, such as the real world, uncertainty is inherent. There are various forms of uncertainty, each presenting itself in distinct ways. *Factual uncertainty* involves doubts about factual knowledge, where propositions may be either true or false. For example, uncertainty regarding the robot's position, expressed in (x, y) coordinates, falls into this category. On the other hand, *regulatory uncertainty* pertains to rules governing the world that are either noisy or provide insufficient explanations. An illustration of this is when moving forward results in slight deviations from the intended driving direction.

Reasons contributing to uncertainty are manifold. *Noisy actuators* result in the imprecision of executed robot actions (such as moving forward) which is less accurate than the robot believes them to be. *Noisy sensors* are highly susceptible to all sorts of influencing factors. Cameras are highly susceptibile to errors in varying lighting conditions and occlusions, lasers can be disturbed by reflective surfaces or interferences in the sensor's line of sight which distort or block the laser signals. Sensors can be sensitive to changing weather conditions, temperature variations, power fluctuations and environmental disturbances such as shocks and vibrations, leading to inaccurate readings. These uncertainties in the sensor readings significantly impact the reliability and pre-

cision of gathered information, making it challenging to obtain accurate and consistent data. Another type of uncertainty is attributed to the *limited understanding* of the world and how it works. The vast amount of knowledge, though substantial, is often unstoreable or unrepresentable, and much of it remains imperceptible to the agent, resulting in numerous *unobserved variables*. The *non-deterministic* nature of both agent behavior and the environment renders it impossible to formulate strict logical rules to reason about, as incorporated in many classic knowledge representation and reasoning systems.

Handling uncertainty is essential for any intelligent operating in the real world. This entails the capability to explicitly address uncertainty and employ mathematical techniques that facilitate automated decision-making when confronted with uncertain conditions. Intelligent agents can improve their capacity for reasoning, learning from data, and making decisions in real-world scenarios where uncertainty is inherent by integrating probabilistic methods. In the following, the foundations of probabilistic knowledge processing that form the basis for the work described in this thesis will be briefly revisited.

2.3 Basics of Probability Theory

Probability theory stands out offering a potent mathematical tool for managing uncertainty. It plays a pivotal role in addressing uncertainty in a way that it establishes a formal framework for quantifying uncertainty where probabilities serve as representations of the likelihood of various outcomes, enabling intelligent agents to articulate and comprehend the uncertainty associated with diverse situations or events. Employed in practical applications, machine learning techniques allow to derive probabilistic models from data and enable intelligent agents to express uncertainty in predictions and enhance decision-making as more data becomes available.

2.3.1 Key Concepts

The central idea in probability theory revolves around the notions of *random events* and *random variables*. An elementary – or *atomic* – event A, denotes an occurrence that comprises a singular outcome within the experiment or *sample space* Ω such that the collection of all these outcomes constitute the entire sample space.

random variables ▷

A *probability distribution* P serves to quantify the concept of probability by assigning a value in $[0, 1]$ to events, representing the probability with which the event occurs. In particular,

a) each single event is assigned a probability greater or equal to 0: $P(A) \geq 0, \forall A \subseteq \Omega$,
b) the probability of the entire sample space is 1: $P(\Omega) = 1$ and
c) the probability of a subset of N mutually exclusive events in the

sample space is the sum of the individual probabilities: $P\left(\bigcup_{n=1}^{N} A_n\right) = \sum_{n=1}^{N} P(A_n)$.

Formally, a random variable X is a mathematical function that assigns some value to every potential outcome of a random experiment, where the value is an element in the *domain* of the random variable, $dom(X)$, i.e. the set of possible values X can take. The domain may be the set of real numbers $\mathbb{R}$, the set of Boolean values $\mathbb{B} = \{\text{True}, \text{False}\}$ (or short: $\{\top, \bot\}$) or any other set of numerical or categorical values such as $\{\text{red}, \text{green}, \text{blue}\}$ or $\{\text{John}, \text{Mary}, \text{Chris}, \text{Anna}\}$.

Using random variables to denote a certain outcome, say, the Boolean random variable X denotes whether it is raining and assigning it a value from its domain allows to represent propositions about events such as "It is raining" ($X = \top$) or "It is *not* raining" ($X = \bot$). When assigning these propositions a quantitive value by writing it as a probability $P(X = \top) = p$ it is possible to represent the degree, to which the proposition is *believed* to hold in the real world. As $p \in [0, 1]$, the probability of $P(X = \bot)$ must be equal to $1 - p$.

Logical operators in *Propositional Logic (PL)* allow combining Boolean random variables to create more elaborate propositions. The conjunction $X \wedge Y$ denotes that X and Y both hold, $P(X \wedge Y)$ the probability of X and Y being true at once. In terms of probabilities, the conjunction is often equivalently represented as $P(X, Y)$ and is called the *joint probability distribution* of X and Y. Probabilities like the above, that do not impose any kind of constraints or conditions are called *prior*, *marginal*, or *unconditional* probabilities. In contrast to unconditional probabilities, a *conditional* or *posterior probability* refers to a probability of one or more random variables, given the value of another is already known. For example, if X and Y are two jointly distributed random variables, the conditional distribution of X given Y, denoted as $P(X|Y)$, is the probability distribution of X when the value of Y is known (say, $\top$). Given the set of all events, this describes the instances where $Y = \top$ and then refine the selection to those instances where $X = \top$ as well, comparing the sizes of these event sets relative to each other:

◁ *prior & posterior probabilities*

$$P(X \mid Y) = \frac{P(X \wedge Y)}{P(Y)} \tag{2}$$

When information about the variable Y does *not* affect an agent's belief in another variable X, then the conditional probability $P(X \mid Y)$ is equal to the unconditional probability $P(X)$. X and Y are termed *independent* in this case, and $P(X \wedge Y) = P(X) \cdot P(Y)$ holds. Conversely, if knowledge about Y *does* influence the agent's belief in X, then X and Y are considered *dependent*.

The fundamental principle to calculate the joint probability of multiple events or the joint distribution of multiple random variables is called *chain rule* and is defined as

$$P(X \wedge Y) = P(X \mid Y) \cdot P(Y) \tag{3}$$

which directly follows from the definition of conditional probabilities in Equation 2. It enables the representation of a joint probability distribution by factoring it into conditionals and *marginals*. A *marginal* distribution is the probability distribution of selected random variables within a larger set of jointly distributed random variables. This distribution is derived by examining the probabilities associated with specific variables

while excluding others, essentially "marginalizing" over the unwanted variables. The term "marginal" indicates that we are examining the distribution along the margins, concentrating on particular variables of interest while omitting the others.

2.3.2 Inference

If one were to find the probability of some event X, but can only observe some events Y_i that form a partition of the sample space, the *law of total probability* is a useful tool which sums, or *marginalizes* over one or more random variables from the joint distribution:

$$\begin{aligned} P(X) &= \sum_{y \in dom(Y)} P(X \wedge Y = y) \\ &\overset{\text{Eq. 3}}{=} \sum_{y \in dom(Y)} P(X \mid Y = y) \cdot P(Y = y) \end{aligned} \tag{4}$$

Bayes' theorem ▷

Bayes' theorem, one of the cornerstone theorems in probability theory, describes the probability of an event based on prior knowledge of conditions that might be related to the event. Mathematically, it is expressed as:

$$\underbrace{P(Y \mid X)}_{\text{posterior}} = \frac{\overbrace{P(X \mid Y)}^{\text{likelihood}} \cdot \overbrace{P(Y)}^{\text{prior}}}{\underbrace{P(X)}_{\text{evidence}}} \tag{5}$$

It proves valuable in adjusting beliefs about the likelihood of different explanations or causes as fresh evidence emerges by incorporating prior knowledge and observed data and it facilitates the transition between abductive and deductive reasoning tasks. In an abductive reasoning task, we typically infer the best explanation or cause for observed evidence. In contrast, deductive reasoning involves deriving specific conclusions from general principles.

When building a probabilistic KB one can make use of a set of logical rules that represent propositions believed to be true about certain aspects of the world and form a joint probability distribution $P(\Sigma)$ over a set of atomic propositions Σ. To achieve this, one has to introduce a Boolean random variable for each of these atomic propositions and store the co-occurrences for each combination of such atomic events in what is called a contingency table. In theory, the introduced tools such as Bayes' theorem (Equation 5), the chain rule (Equation 3) and the law of total probability (Equation 4) would allow calculating any posterior $P(Q|E)$ from such a full joint distribution, where $Q \subseteq \Sigma$ is an arbitrary query and $E \subseteq \Sigma$ an arbitrary evidence. In practice, this works only for the smallest problems as the generation and handling of such a full joint probability poses significant challenges, primarily related to scalability. The foremost obstacle arises from the exponential growth in table size as the number of variables or events increases. This exponential expansion renders it computationally infeasible for practical applications involving a multitude of interconnected variables. Another difficulty is linked to the data prerequisites for precisely estimating probabilities across all conceivable combi-

nations. In various real-world situations, specific event combinations may be rare or entirely missing in the accessible data, compromising the reliability of predictions. The computational complexity associated with computing, storing, and manipulating large joint probability tables is therefore a big challenge. As the table size increases, algorithms for updating, querying, or learning from the table become progressively hard, demanding considerable computational resources.

To tackle the limitations of representing full joint distributions, alternative probabilistic models like *Bayesian Networks (BNs)*, *Markov Networks (MNs)*, or other ML techniques are commonly employed, providing more scalable and computationally efficient methods for representing probabilistic relationships and achieving a balance between precision and practical feasibility in real-world applications.

In probabilistic knowledge bases, a common form of inference involves computing the posterior distribution for a set of query variables $Q \subseteq X$ of all variables X in the distribution given observed evidence variables $E \subseteq X$. In many cases the primary concern is to identify the most probable variable assignment given the provided evidence, whereas obtaining the complete posterior distribution for the queries is not required. Here, the concepts of *Maximum A Posteriori (MAP)* and *Most Probable Explanation (MPE)* inferences are are employed. MAP and MPE both aim at determining the most likely assignment $\hat{p}$ for non-evidence variables Q (also referred to as *MAP variables*), considering the provided evidence, i.e.

◁ *inference*

◁ *MAP and MPE*

$$\hat{p} = \arg\max_{q \in Q} P(q \mid E) \tag{6}$$

In literature the use of the terms MAP and MPE slightly differs in that they are either used synonymously or deviate in their definition of Q. According to Park and Darwiche (2004), MPE refers to the *joint* MAP problem, where $Q = X \setminus E$ includes *all* non-evidence variables, while MAP focuses on finding the most likely assignment of values to only a subset $Q \subseteq X \setminus E$ (so-called *MAP variables*) among the non-evidence variables, given the evidence. Therefore, MPE poses a special case of MAP. However, other literature (Koller and Friedman 2009) use the terms synonymously and refer to the case where $Q \subseteq X \setminus E$ as *marginal MAP*. Following the former definition of Park and Darwiche (2004), both MAP and MPE pose computational challenges. MAP is very hard as it involves computing the marginal probability distribution over the MAP variables, which is a type of sum product problem. The subsequent step is to maximize over the marginal distribution, turning it into a maximization problem, essentially a max sum product problem. On the other hand, MPE is treated as an instance of the max product problem, where there is no need to compute the sum.

2.4 Probability Distributions

Unfortunately, the strategy of creating a complete joint distribution by generating Boolean variables for each atomic event has an additional constraint – it is ineffective in continuous settings. While one can generate a Boolean random variable for each value assignment where the set of values is finite, this is not possible for (continuous)

numerical values (e.g. values $\in \mathbb{R}$). It is therefore required to incorporate other types of distributions, some of which will be introduced in the following. Recall, a random variable is the quantified outcome of an experiment which can be either discrete or continuous. A probability function is a mathematical function that assigns probabilities to the potential outcomes of a random variable. It is typically denoted as $f(x)$. Depending on the type of the random variable, the probability function is either a *Probability Mass Function (PMF)* (for discrete variables) or a *Probability Density Function (PDF)* (for continuous variables). The *Cumulative Distribution Function (CDF)* $F(x)$ is employed to determine the probability that a random variable is less than or equal to a specified value

$$F(X) = P\Big(X \overset{20}{\leq} x\Big) \tag{7}$$

This section introduces some of the most frequently used distributions and their characteristics. It is important to note that the list of distributions presented here is far from being exhaustive. The aim is to give an overview over existing distributions, with a focus on those, that are of relevance in the context of this work.

2.4.1 Numeric Distributions

Numeric probability distributions are crucial in quantifying uncertainty and representing the likelihood of various outcomes. The distributions can be over distinct, separate (*discrete*) values or for a continuum of values (*continuous*). The essential characteristics linked to some numeric probability distributions will be explored in the following.

2.4.1.1 Discrete Distributions

A discrete random variable X is described by its PMF

D

$$f(x) := P(X = x), \tag{8}$$

which will also be called its distribution. The PMF returns the probability that the random variable X is equal to a specific value x. As the name suggests, the set of values X can take, its *domain*, denoted as $dom(X)$, contains (finite or countably infinite) *discrete* values. The PMF returns a positive numerical value for each value from the random variable's domain, i.e.

$$\forall x \in dom(X) : f(x) > 0 \tag{9}$$

Furthermore, the sum of all these values sums to 1:

[20] The probability of achieving a specific value for a continuous variable is zero ($P(X = x) = 0$), consequently, $F(X) = P(X \leq x) = P(X < x)$ for a continuous variable X.

$$\sum_{x \in dom(X)} f(x) = 1 \tag{10}$$

When examining random variables, it can be interesting to measure the *central tendency*, a representative value around which the random variable tends to cluster on average. This can be interpreted as a predicted or average outcome, making it a valuable tool for decision-making and risk assessment. The *expected value* $\mathbb{E}(X)$ of a discrete random variable is a measure representing the average or mean value (μ) one would anticipate from a large number of repetitions of an experiment. It can be considered the *weighted average* of all possible outcomes, where each outcome is weighted by the likelihood of its value, that being the probability of occurence:

◁ *expected value*

D

$$\begin{aligned} \mathbb{E}(X) &:= \sum_{x \in dom(X)} x \cdot f(x) \\ &= \sum_{x \in dom(X)} x \cdot P(X = x) \end{aligned} \tag{11}$$

The expected value therefore provides a representative value that gives an idea of the center of the distribution of the random variable. The expected value tempts to compare different random variables to each other, however, two random variables sharing the same expected value can still be very different from one another. A random variable's variability, the so-called *variance*, is therefore another important characteristic to consider, as it describes how the values are distributed around the expected value. While some random variables have the majority of possible values scattered closely around the expected value, others never take their mean as an actual value, for example a variable X with $P(X = -100) = \frac{1}{2} = P(X = 100)$. Obviously the expected value lies at $\mathbb{E}(X) = -100 \cdot \frac{1}{2} + 100 \cdot \frac{1}{2} = 0$ but $0 \notin dom(X)$. The variance of a random variable, denoted as $Var(X)$, intuitively describes the deviation from the mean μ (with $\mu = \mathbb{E}(X)$), where the inconvenience of handling the absolute value is accounted for by regarding the mean *square* deviation:

◁ *variance*

D

$$\begin{aligned} Var(X) &:= \mathbb{E}\left((X - \mu)^2\right) \\ &= \sum_{x \in dom(X)} (x - \mu)^2 \cdot P(X = x) \end{aligned} \tag{12}$$

The *standard deviation* σ of a random variable X is the square root of the variance:

D

$$\sigma := \sqrt{Var(X)} \tag{13}$$

It can be shown that calculating the variance using

$$Var(X) = \mathbb{E}(X^2) - \mathbb{E}(X)^2 \tag{14}$$

is equivalent to applying the definition in Equation 12 (Schickinger and Steger 2002, Chapter 1), which can be easier to compute.

The proof for the equality of $\mathbb{E}\Big((X-\mu)^2\Big) = \mathbb{E}(X^2) - \mathbb{E}(X)^2$ can be found in Schickinger and Steger (2002, Chapter 1).

Bernoulli Distribution · The arguably most basic discrete random variable is a binary categorical variable which is characterized by having only two possible outcomes, such as $\{\top, \bot\}$ or $\{1, 0\}$. A binary variable can be derived from a non-binary random variable by establishing criteria for "success" and "failure". Take, for instance, a random variable X representing the act of rolling a fair six-sided die with $dom(X) = \{1, 2, 3, 4, 5, 6\}$ and $\forall x \in dom(X) : P(X = x) = \frac{1}{6}$. If the focus lies on the event A, defined as "rolling a two", then the "success" corresponds to rolling a two, while the "failure" encompasses any value other than two. The binary variable X' then has the two possible outcomes $A : X = 2$ and $B : X \in \{1, 3, 4, 5, 6\}$.

Generally, the PMF of a Bernoulli variable with $dom(X) = \{0, 1\}$ is represented as

$$f(x) = \begin{cases} p & \text{for } x = 1 \\ 1 - p & \text{for } x = 0 \end{cases} \tag{15}$$

where p is the probability of success. Defining $q := 1 - p$, the expectation and variance are determined as

$$\mathbb{E}(X) = 1 \cdot p + 0 \cdot (1 - p) = p \tag{16}$$

and

$$\begin{aligned} Var(X) &= \mathbb{E}(X^2) && - \mathbb{E}(X)^2 \\ &= 1^2 \cdot p + 0^2 \cdot q - p^2 \\ &= p && - p^2 \\ &= p(1 - p) \\ &= pq. \end{aligned} \tag{17}$$

Binomial Distribution · While the Bernoulli distribution models a single random experiment with two possible outcomes, the Binomial distribution extends the Bernoulli distribution to multiple trials under the i.i.d assumption (where i.i.d stands for independent, identically distributed) and therefore represents the number of successes in a fixed number of independent Bernoulli trials. A random variable defined as a sum of n independent Bernoulli-distributed random variables $X := \sum_{i=1}^{n} X_i$ with an identical success probability p is said to have a binomial distribution with parameters n and p, denoted as $X \sim \text{Bin}(n, p)$. An example for such a distribution is the toss of a fair coin 100 times. In this case, $n = 100$ and $p = \frac{1}{2}$. The PMF of a binomial distribution is defined as

$$\begin{aligned} f(x) &= \binom{n}{x} p^x \cdot q^{n-x} \\ &= \frac{n!}{x!(n-x)!} p^x \cdot q^{n-x} \end{aligned} \tag{18}$$

where again, $q \coloneqq 1 - p$. A more detailed deduction of Equation 18 can be found in Section 4.5. The expected value of a binomially distributed variable is calculated by

$$\begin{aligned} \mathbb{E}(X) &= \mathbb{E}(X_1) + \mathbb{E}(X_2) + ... + \mathbb{E}(X_n) \\ &\overset{21}{=} n \cdot p \end{aligned} \tag{19}$$

while the variance is

$$\begin{aligned} Var(X) &= Var(X_1) + Var(X_2) + ... + Var(X_n) \\ &\overset{21}{=} n \cdot p \cdot q. \end{aligned} \tag{20}$$

Discrete Uniform Distribution · The discrete uniform distribution is a probability distribution that describes the probability of each outcome in a finite set of equally likely outcomes, i.e. a random variable X is said to have a discrete distribution if the range of X consists of distinct values, and each value has the same probability of occurrence, making it a symmetric and uniform distribution. It is therefore characterized by a constant PMF for each possible value in the sample space, i.e. if

$$dom(X) = \{x \mid x \in \mathbb{Z}\}, |dom(X)| = n \tag{21}$$

then $X \sim \mathcal{U}(n)$ implies

$$f(x) = \begin{cases} \frac{1}{n} & \text{for } x \in dom(X) \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

Here, assume $x = 1, 2, ..., n$ and $n \in \mathbb{N}$ being the number of values of X, i.e. all $x \in \mathbb{N}$ from 1 to n. Generally, the expectation of the discrete uniform distribution is computed as

$$\begin{aligned} \mathbb{E}(X) &= \frac{1}{n} \sum_{x \in\ dom(X)} x \\ &\overset{22}{=} \frac{n+1}{2} \end{aligned} \tag{23}$$

The general case of the variance looks quite complex

$$Var(X) = \frac{1}{n} \left(\sum_{x \in\ dom(X)} x^2 - \frac{1}{n} \left(\sum_{x \in\ dom(X)} x \right)^2 \right) \tag{24}$$

but with

$$\begin{aligned} \mathbb{E}(X^2) &= \sum_{x \in\ dom(X)} x^2 \cdot \frac{1}{n} \\ &= \frac{(n+1) \cdot (2n+1)}{6} \end{aligned} \tag{25}$$

[21] due to the linearity of the expected value $\mathbb{E}(X)$

[22] according to Carl Friedrich Gauss's formula for summing an arithmetic series: $\frac{n(n+1)}{2}$

it is then reduced to a simplified form due to the assumptions about the domain of X above:

$$\begin{aligned} Var(X) &= \mathbb{E}(X^2) - \mathbb{E}(X)^2 \\ &= \frac{(n+1)\cdot(2n+1)}{6} - \left(\frac{n+1}{2}\right)^2 \\ &= \frac{(n+1)\cdot(n-1)}{12} \\ &= \frac{n^2-1}{12} \end{aligned} \tag{26}$$

The discrete uniform distribution is typically employed to model the outcomes of fair games, such as rolling a fair six-sided die, flipping a fair coin, or drawing a card from a well-shuffled deck. In these cases, each outcome has an equal probability of occurring.

2.4.1.2 Continuous Distributions

A continuous random variable – in contrast to a discrete random variable, which involve countable and distinct outcomes – can take an infinite number of values within a specified interval. Examples for continuous random variables are representations of heights or weights of randomly selected persons, a measured temperature or the continuous position of an agent in its environment. Distributions of continuous variables are modeled using the PDF. Unlike in discrete distributions, probabilities of continuous distributions are associated with intervals rather than individual points. The PDF provides a relative measure of how likely the variable is to be near a particular value, i.e. $f(x) \neq P(X = x)$, therefore, the probability of a continuous random variable taking any exact point value is technically zero. Since the probability that X falls within a certain interval $[a, b]$ is given by the integral of the PDF, requiring X to be exactly equal to a specific value c results in the limits of this interval being identical i.e. $a = b = c$, rendering the interval infinitesimally small and thus, the integral evaluates to zero.

However, analogous to discrete distributions, where the sum of all probabilities of all the possible outcomes equals 1 (cmp. Equation 10), the area under the curve of the PDF of a continuous variable is equal to 1:

$$\int_{\infty}^{\infty} f(x)\,\mathrm{d}x = 1 \tag{27}$$

Uniform Distribution · The equivalent to the uniform discrete distribution from Section 2.4.1.1 is the continuous uniform distribution, which spreads mass uniformly over an interval $[a, b]$ and its PDF is given by:

$$f(x) = \begin{cases} \frac{1}{b-a} & \text{if } x \in [a, b] \\ 0 & \text{otherwise} \end{cases} \tag{28}$$

The expectation function and variance are given by

$$\begin{aligned}\mathbb{E}(X) &= \int_a^b \frac{x}{b-a}\,\mathrm{d}x \\ &= \frac{b+a}{2}\end{aligned} \tag{29}$$

and

$$\begin{aligned}Var(X) &= \int_a^b \frac{1}{b-a}\left(x - \frac{b+a}{2}\right)^2 \mathrm{d}x \\ &= \frac{(b-a)^2}{12}\end{aligned} \tag{30}$$

Normal Distribution · The Normal or *Gaussian* distribution is one of the most prominent continuous distributions. Many natural phenomena tend to display an approximately normal distribution. The distribution of heights, weights, and other physical characteristics in a population often follows a normal distribution with its characteristic bell shape due to the influence of multiple independent factors. The normal distribution possesses straightforward and precisely defined mathematical characteristics. Its PDF and CDF are expressed in closed-form, which simplifies mathematical computations and statistical analyses. The Central Limit Theorem states that, regardless of the initial shape of the population distribution, the distribution of sample means will tend to approximate a normal distribution as the sample size increases. This only applies if certain conditions are met, such as that samples must be randomly selected from the population, the sample size should be sufficiently large (where there is no clear definition of "large") and that the samples must be drawn independently from each other.

A continuous variable is normally distributed ($X \sim \mathcal{N}(\mu, \sigma^2)$) with parameters $\mu \in \mathbb{R}$ (the center of the curve) and $\sigma \in \mathbb{R}^+$ (the spread about the center), if it has the PDF

$$f(x) = \frac{1}{\sqrt{2\pi\sigma}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \tag{31}$$

It can be shown that the expectation is given by

$$\mathbb{E}(X) = \mu \tag{32}$$

and the variance is

$$Var(X) = \sigma^2. \tag{33}$$

The proof can be found in Schickinger and Steger (2002, Chapter 2).

2.4.2 Symbolic Distributions

Multinomial Distribution The multinomial distribution extends the concept of the binomial distribution to include scenarios with k instead of only two possible outcomes, with probabilities $\pi_1, ..., \pi_k$. Note that it is necessary to have $\sum_{i=1}^{k} \pi_i = 1$ hold. Just like the binomial distribution, a fixed number of n independent trials, is required.

However, rather than focusing solely on counting the occurrences of a single "success" outcome, we use X_j to represent the count of occurrences for the jth outcome. This leads to the formation of the multivariate random vector $X_1, ..., X_k$.

The PMF is given by

$$f(x_1, ..., x_k) = \frac{n!}{x_1! \cdot x_2! \cdot ... \cdot x_k!} \cdot \pi_1^{x_1} \cdot \pi_2^{x_2} \cdot ... \cdot \pi_k^{x_k} \tag{34}$$

where $x = (x_1, ..., x_k)$ (Bishop 2006). The expected value and variance of X_j as well as the covariance between two outcome counts are only defined if the distribution's values have true numeric semantics. The expected value is then determined calculating

$$\mathbb{E}(X_j) = n \cdot \pi_j \tag{35}$$

and the variance by calculating

$$Var(X_j) = n \cdot \pi_j \cdot (1 - \pi_j) \tag{36}$$

Additionally, the covariance between the outcome counts X_i and X_j is given by

$$Cov(X_i, X_j) = -n \cdot \pi_i \cdot \pi_j \tag{37}$$

where the inverse correlation is logical from an intuitive standpoint, considering the constant sum $X_1 + X_2 + ... + X_k = n$. Put simply, as one outcome becomes more frequent, the occurrences of other outcomes must decrease for the preservation of this total. It is worth highlighting that if all other outcomes are grouped as "failure," each individual count X_j follows a binomial distribution with n trials and success probability π_j.

mode ▷

For multinomial distributions with symbolic values the expected value is not defined. In this case, one typically resorts to returning the *mode* (Finucan 1964), which is the most frequently occurring value in the distribution. This measure provides some insight into the central tendency of the data. If there are multiple values with an equal (maximum) probability, they are denoted as the *joint node*.

2.4.3 Comparing Distributions

There are many measures to quantify the similarity or dissimilarity (i.e. distance) of distributions, a selection of which will be introduced in the following. In practical applications, the choice of a distance or similarity measure depends on the specific characteristics of the data and the goals of the analysis. A distance function $\varrho : P \mapsto P$ on a given set P is called a *metric*, when the following axioms are satisfied (Čech and Katětov 1969):

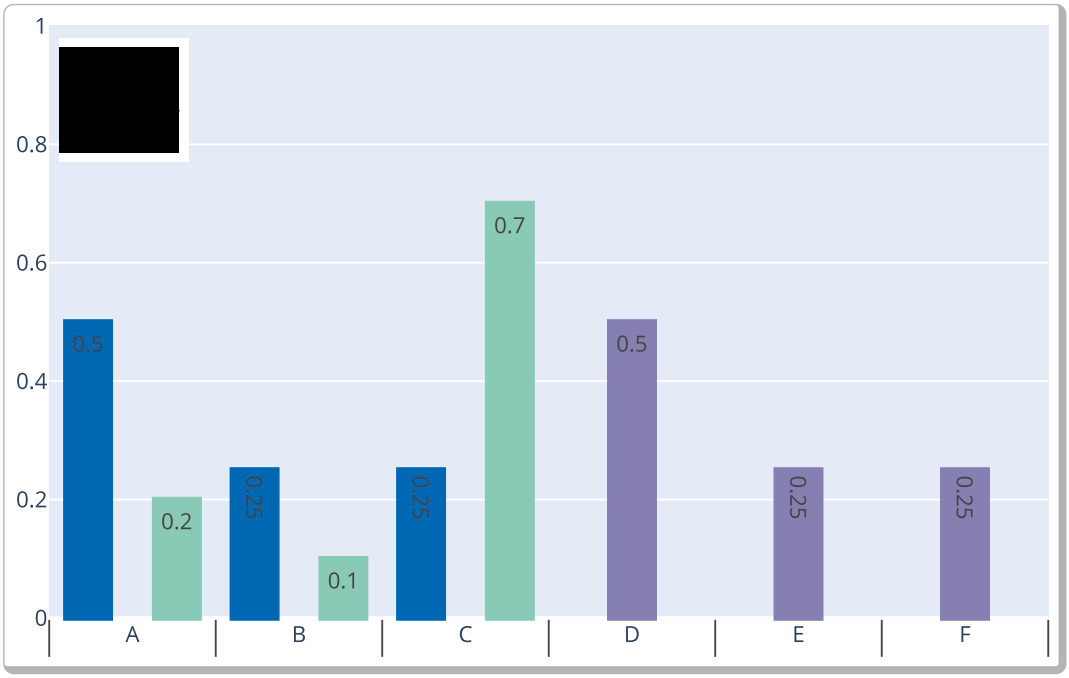
$$\begin{aligned} &[1] \quad \varrho(x, x) = 0, x \neq y \Rightarrow \varrho(x, y) > 0; \\ &[2] \quad \varrho(x, y) = \varrho(y, x); \\ &[3] \quad \varrho(x, y) + \varrho(y, z) \geq \varrho(x, z). \end{aligned} \tag{38}$$

The Kullback-Leibler divergence (Kullback 1959) for example, represents the expectation of the log difference between the probability of data in one distribution with a second (assumed to be approximating) distribution. Unfortunately, the measure is not symmetric (Equation 38 [2]) and therefore not a true metric.

Similarly to Kullback-Leibler, the Bhattacharyya distance (Bhattacharyya 1946) and the related Hellinger distance (Hellinger 1909) can be effective in certain contexts, but their limitations in terms of symmetry make them less suitable for some tasks. They are therefore not employed in BAYROB.

Similarity of discrete distributions: The domain sizes of all three distributions $dist_1$ o $dist_3$ are equal but only $dist_1$ (blue) and $dist_3$ (green) share identical domain values. $dist_2$ (violet) does not share domain values with either $dist_1$ or $dist_3$ and would be considered maximally dissimilar to the other distributions.

| *Figure 4*



Similarity measures in other applications, such as in set theory, might give an intuition on how a quantitative comparison of two distributions can be achieved. The Jaccard coefficient measures the similarity of n sets S_i by applying the *intersection over union* principle, i.e. by calculating the ratio of the number of common (intersecting) elements and the size of the union of the n sets:

$$sim_{jac}(S_1, ..., S_n) = \frac{|\bigcap_i^n S_i|}{|\bigcup_i^n S_i|} \tag{39}$$

Since the returned value ranges from 0 to 1, it can be used to measure a distance as well, by just calculating $dist_{jac}(S_1, ..., S_n) = 1 - sim_{jac}(S_1, ..., S_n)$. But does that work in distributions as well? Figure 4 exemplary shows the intuition behind it in discrete distributions. Since $dist_1$ (blue) and $dist_3$ (green) have identical domains, but differ-

ent probabilities for the respective values, one would consider them to be more similar than the distributions $dist_1$ and $dist_2$ (violet). How can we quantify this in terms of the above equation?

When thinking of a discrete distribution of a set of labels with assigned probabilities, the intersection over union principle can work as well: interpreting the intersection as the least common multiples (i.e. minimum) of the probabilities of the respective labels and the union as the greatest common multiples (i.e. maximum), the division of the respective sums of these values represents the divergence of values in the common domain:

$$sim_{discr}(D_1, ..., D_n) = \frac{\sum\limits_{x \in\ dom(D)} \min(p_i(x))}{\sum\limits_{x \in\ dom(D)} \max(p_i(x))} \tag{40}$$

The similarity measure is only defined on distributions with an identical domain for a simple reason. Assuming, the measure would consider the probability values of the labels in the shared domain, disregarding the others, the measure would not reflect the dissimilarity posed by the difference in the domains itself, leading to unintuitive results. Given another distribution $dist_4$ with $dom(dist_4) = \{C, D, E\}$ and $P(C) = .25, P(D) = .1, P(E) = .65$, therefore partly sharing a domain with $dist_1$ and $dist_3$, the similarity measure would output $sim_{discr}(dist_1, dist_4) = \frac{0.25}{0.25} = 1$, interpreting the two obviously very different distributions as identical. Consequently, by default, if the intersection of the domain of two or more distributions such as in the case of $dist_1$ and $dist_3$ is empty, the distributions are considered maximally dissimilar.

The Jaccard coefficient is defined on discrete sets, therefore the adaptation in Equation 40 works for discrete distributions only, but BAYROB employs hybrid domains, such that we need to find an adaptation for continuous distributions. Figure 5 shows the PDFs of two Gaussian-distributed random variables X_1 and X_2 as well as the PDFs of their mixture and sum. It is desirable to find a similarity (or distance) measure that quantifies the differences in shape and momentums adequately.

Following the argumentation of similarities of sets and discrete distributions above, the similarity measure for continuous domains could be defined incorporating the distribution functions' integrals as follows:

$$sim_{cont}(d_1, d_2) = \frac{\int\limits_{\mathbb{R}} \min(d_1, d_2)}{\int\limits_{\mathbb{R}} \max(d_1, d_2)} \tag{41}$$

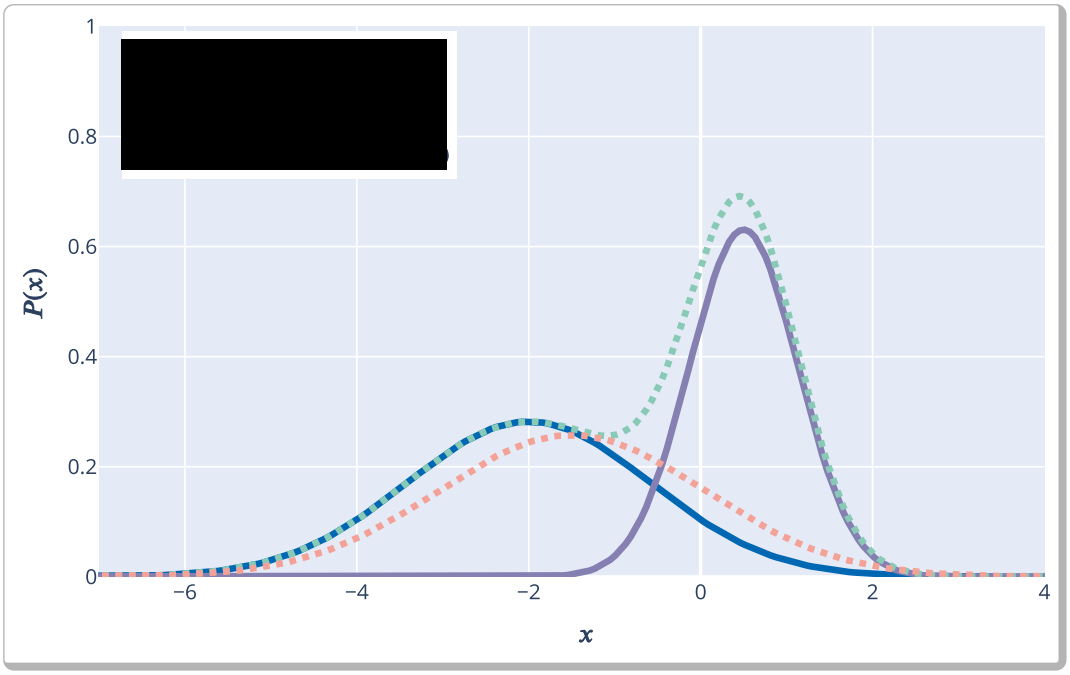
which again is very similar to the Ruzicka similarity (Deza and Deza 2006, Chapter 17), a similarity on $\mathbb{R}^n$, defined by

$$sim_{ruz}(x, y) = \frac{\sum\limits_i \min(x_i, y_i)}{\sum\limits_i \max(x_i, y_i)} \tag{42}$$

sometimes referred to as the quantitative data Jaccard similarity.

Just like in the discrete case above, if the shapes of two distributions are very similar but the moments differ, they can, in the worst case, be considered maximally dissimilar by the Jaccard similarity measure. Take as an example Gaussian distributions with an identically shaped bell curve that only differ in their respective means, like the ones shown in Figure 6 a).

Similarity of continuous distributions | *Figure 5*



While for multinomial distributions it is reasonable to assume maximal dissimilarity for distributions that do not share an identical domain, this does not intuitively apply to continuous distributions. Imagine, as an example, a Gaussian-distributed variable X, representing the x-coordinate of a robot's position. The spread of the distribution, i.e. in this case the width of the bell, then determines the uncertainty of the robot's position in the horizontal direction, while the mean denotes the absolute position around which the robot believes it is located. Comparing two such distributions with different means might then represent for example, how far a potential goal location is away from the current position. A value of $sim_{jac}(S_1, ..., S_n) = 0$, denoting minimal similarity (or $1 - sim_{jac}(S_1, ..., S_n) = 1$ denoting maximal dissimilarity) would not correspond to the intuitive impression that the current distribution may not be the exact goal location, but might be closer to or further from it than the one *before* some action was executed. In particular, it would not be possible to compare the (anticipated) resulting distributions of different actions and make an informed decision about what action is the best, i.e. takes the robot closer to the goal. In fact, $sim_{jac}(D_1, D_2) = sim_{jac}(D_1, D_3) = sim_{jac}(D_2, D_3) = 0$ for the distributions in Figure 6 a).

The *Wasserstein* distance (Villani 2009; Deza and Deza 2006), sometimes called *Earth Mover's Distance* or *optimal transport* distance, can be seen as a quantification of the effort it takes to transform one distribution into another. Distributions are considered piles of earth that need to be reshaped, where the mass that has to be moved to align one

pile with another represents the *cost*, i.e. distance of the piles. In mathematical terms, the Wasserstein distance is expressed as the product of the distance travelled (comparing momentums) and the amount of mass being transported (comparing shapes). It is defined on a metric space (M, d), with M being a set and d a metric on M, as:

D

$$W_p(d_1, d_2) := \left(\inf_{\gamma \in \mathrm{T}(d_1, d_2)} \int_{M \times M} d(x, y)^p \, \mathrm{d}\gamma(x, y) \right)^{\frac{1}{p}} \tag{43}$$

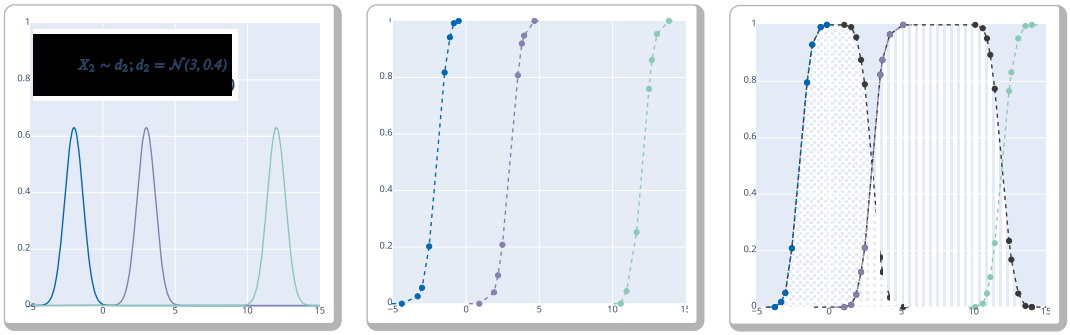
where inf denotes the infimum (greatest lower bound), and $\mathrm{T}(d_1, d_2)$ the set of all couplings of the input probability measures with marginals d_1 and d_2. More figuratively, the distance describes the area between the CDFs of the compared distributions (see the patterned areas in Figure 6 b). Compared to the distance of 1 for each of the pairs returned by the Jaccard distance, the Wasserstein distance corresponds to the intuitive understanding of the differences of the three distributions: The two leftmost distributions d_1 and d_2 have the smallest distance to each other ($W_p(d_1, d_2) \approx 5.003$), the outermost ones d_1 and d_3 have the largest with $W_p(d_1, d_3) \approx 13.964$. The distance between d_2 and d_3 is calculated as $W_p(d_2, d_3) \approx 8.960$.

Identically-shaped Gaussian distributions (a) and their CDFs (b): The patterned areas represent the areas of the (absolute) differences between distributions. | *Figure 6*

Identically-shaped Gaussian distributions, shifted | *a)*

The CDFs of the respective distributions on the left | *b)*

$| d_1 - d_3 |$ (checkerboard) and $| d_2 - d_3 |$ (stripes) | *c)*



The Wasserstein metric works on the CDF of numeric distributions. The equivalent for multinomial distributions would then be the summing up the pairwise (cumulated) probabilities. That, however, would require the multinomial distribution to have some order defined, as is the case in numeric distributions. Depending on that order, the results of the distance calculation may vary significantly, such that the approach is not directly applicable to multinomial distributions. Alternatively, one can simply take the sum of the pairwise absolute differences of each of the domain values' relative frequencies, i.e.

$$dist_{earth\_move}(D_1, D_2) = \sum_{x \in \, dom(D)} | \, p_{D_1}(x) - p_{D_2}(x) \, | \tag{44}$$

This approach is similar to the X^2 (Chi-squared) test (first introduced by Pearson (1900)) for categorical data but without squaring the differences. Instead, it seems to be more natural to take the absolute values as a true "distance". In the special case of a Boolean dstribution, this approach could be interpreted as the effort it takes to take a certain amount of probability mass from the $\top$ pile plus the same effort it takes to move it to the $\bot$ pile (or vice versa).

In BAYROB, the Wasserstein distance from Equation 43 and its adapted variant for multinomial distributions from Equation 44 as well as the Jaccard similarity for discrete distributions Equation 40 and its numeric counterpart Equation 41 are employed.

2.5 Probabilistic Graphical Models

PGMs offer a concise means of representing probability distributions by capitalizing on their structure, enabling encoding across a high-dimensional space with numerous variables. The representation of a joint probability distribution over a set of even the smallest number of random variables (even if they are only binary-valued) is typically computationally unmanageable as it is too large to store and impossible to grasp for the human mind (Koller and Friedman 2009). PGMs offer a sophisticated framework that seamlessly integrates uncertainty and logical structure to concisely represent complex real-world situations and leverage the independence properties inherent in them. Their graph-based representation consists of nodes, representing domain variables, and (directed) edges denoting their respective probabilistic connections. Depending on the underlying representation mechanism, outlined below, these graphs can be either directed or undirected, categorizing PGMs into two families: BNs and MNs. What unifies them is their capacity to represent and facilitate inference over probability distributions, typically in a human-interpretable and understandable manner.

The graph structures facilitate effective inference strategies, allowing to ask all sorts of questions about the distributions they represent. Notably, algorithms for computing posterior probabilities of subsets of domain variables, given some evidence, can be employed, directly leveraging the graph structure. Since calculating posteriors often suffices for many inference tasks, there is no need to compute the entire joint distribution. This aspect makes inference in PGMs a compelling tool, especially in fields such as medical diagnosis (McLachlan et al. 2020), risk assessment in finance (Chan, Chu, and So 2023), or safe autonomous navigation (Iberraken and Adouane 2022).

To develop such models from data and expert knowledge, effective algorithms are necessary – and fortunately, they exist. Therefore, PGMs not only provide representation of and reasoning over probabilities but also learning thereof, proving essential for constructing intelligent systems.

2.5.1 Bayesian Networks

BNs, also called *belief networks* or *causal networks*, are used to efficiently represent a joint distribution over a set of random variables $\boldsymbol{X} = \{X_1, ..., X_n\}$ in terms of a di-

(conditional) independence ▷

rected acyclic graph structure. In order to understand the graphical representation of the distribution, one has to grasp the notion of (conditional) *independence* of variables. The variables X_i with $i \in 1, ..., n$ are independent, if

$$P(X_1, ..., X_n) = P(X_1) \cdot P(X_2) \cdot ... \cdot P(X_n) \tag{45}$$

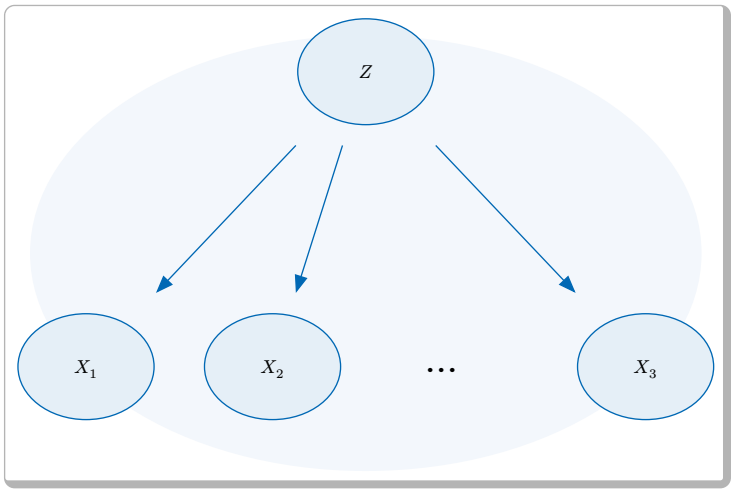
holds. As a notation for independence, $(\boldsymbol{X} \perp \boldsymbol{Y})$ for any disjoint subsets $\boldsymbol{X}$ and $\boldsymbol{Y}$ of the variables can be found frequently.

Analogously, a variable X_i is *conditionally* independent on a set of variables $\boldsymbol{X_i}$ *given* yet another variable Z, i.e. $(X_i \perp \boldsymbol{X_i} \mid Z)$ with $\boldsymbol{X_i} = \{X_1, ..., X_n\}/\{X_i\}$,[23] if

$$P(X_1, ..., X_n, Z) = P(Z) \prod_{i=1}^{n} P(X_i \mid Z) \tag{46}$$

for all variable assignments of the X_is and Z in their respective domains. The factorization in Equation 46 represents the the *naïve Bayes assumption*, i.e. the assumption that, in a classification scenario, the features (here: X_i) are conditionally independent given the instance's class (Z). This is shown by the graph structure for the naïve Bayes model in Figure 7. The same intuition is built upon by BNs. The graph structure is still as compact and leverages the distribution's conditional independencies but without being restricted to the naïve Bayes' strong independence assumptions.

The graph representation of a naïve Bayes model | *Figure 7*



While in other model representations, the edges of the graph structures are not necessarily directed, in BNs they are, rendering the structure a *Directed Acyclic Graph (DAG)*. Each node X_i that has incoming edges depends directly on all of its parent nodes, i.e. the ones the incoming edges origin from, denoted by the set $Par(X_i)$. Each node is associated with a distribution table, the *Conditional Probability Distribution (CPD)*, that specifies the probability of the respective variable conditioned on all its immediate parents. The graph structure therefore represents the joint distribution's factorization as

[23] where / denotes "not in", i.e. $\boldsymbol{X_i}$ is the set of variables $X_1, ..., X_n$ without X_i.

well as a set of its conditional independence assumptions simultaneously, and its joint distribution is given by

$$P(X_1, ..., X_n) = \prod_{i=1}^{n} P(X_i \mid Par(X_i)). \tag{47}$$

Figure 8 introduces an easy-to-understand example of a robot's error detection using a warning light that illustrates the intuition behind BNs. The robot is powered by a $Battery$, whose charging status (high or low) can not be observed directly. However, there are other observable indicators that allow drawing conclusions about the battery status such as the $Motor$ of the robot, which is powered by the battery as well and may be running or stopped. When the battery is very low, the motor sometimes continues buzzing for a little bit, without being fully functional. On the other hand, when the battery is fully charged, the motor might still stop in 5% of the cases. Additionally, an LED $Light$ flashes red when the battery reaches a critical level or green when everything is ok. Unfortunately, sometimes the light does not flash at all, for example when the LED's $Diode$ is faulty (which happens in 20% of the cases) or when the battery status is even too low to power the light. An intern observes the state of the light and sends a $Report$ to the robot's operator which is a simple "ok" when the light is green and "warning" otherwise. Due to the lighting conditions (and his laziness), however, the intern may be wrong about the state of the light, resulting in incorrect reports. The figure shows not only the BN representing this scenario, but also the corresponding CPDs of the random variables $Diode$, $Battery$, $Light$, $Motor$ and $Report$, which represent how the variables in the graph depend on their respective parents (if there are any). In the case of the variables $Diode$ and $Battery$, no parents are given, so the distributions are marginal distributions, while the others are conditional probability distributions of the variable given the different combinations of joint value assignments of the variable's parents. The given BN for this example allows to compute the probability of any possible value assignment of the five random variables, i.e. their joint probability. As introduced in Section 2.3, the joint probability of multiple events or the joint distribution of multiple random variables can be calculated employing the chain rule, which can be applied to BNs, therefore the joint distribution for this example is given by

$$\begin{aligned} P(Diode, Battery, Light, Motor, Report) = {} & P(Diode) \\ & \cdot P(Battery) \\ & \cdot P(Light \mid Diode, Battery) \\ & \cdot P(Motor \mid Battery) \\ & \cdot P(Report \mid Light) \end{aligned} \tag{48}$$

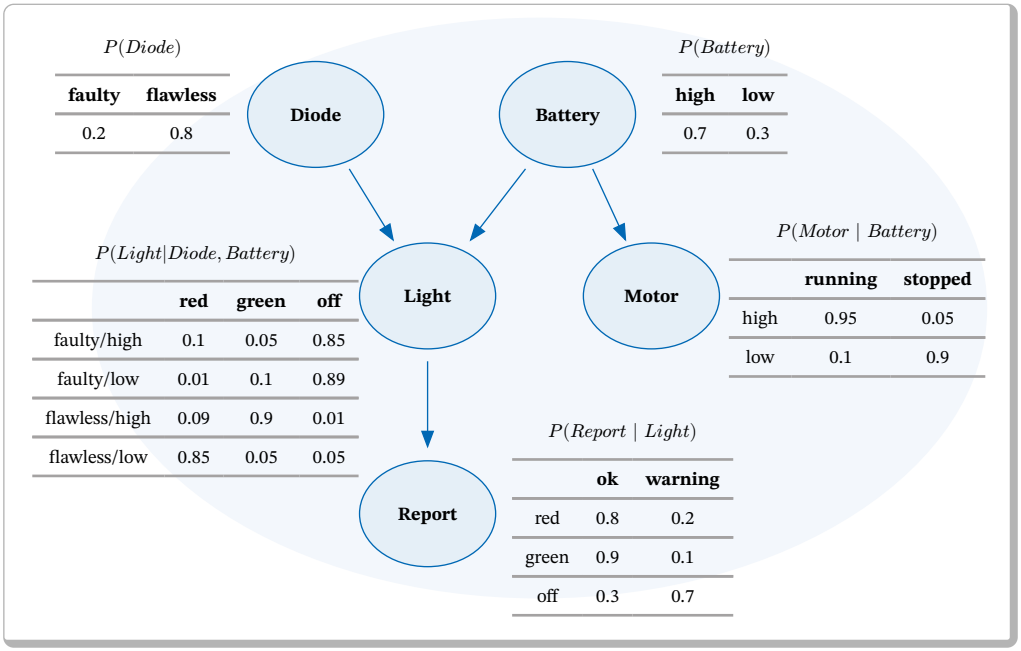
As BNs represent joint probability distributions over the represented random variables (nodes), it is possible to pose any kind of query over (subsets of) variables. Due to the semantics of the directed edges, one can identify different reasoning patterns, depending on the direction of the asked query, e.g. "downwards" or "upwards" from evidence to query, respectively. The *causal reasoning* can be interpreted as a prediction of effects given the observed cause as evidence, such as querying for the probability of a positive report ("ok"), given the LED diode was observed to be faulty. Compared to the marginal

◁ *causal and diagnostic reasoning*

probability of $P(Report = \text{ok}) \approx 0.73$, the observed diode lowers the probability for a positive report significantly $P(Report = \text{ok} \mid Diode = \text{faulty}) \approx 0.38$.

For the other direction, the "upwards" direction, one tries to find the probability of the cause given some observed effect, e.g. trying to find the probability of a low battery given the light was observed to be off or flashing red. The probability after the observations increases from $P(Battery = \text{low}) \approx 0.3$ to ≈ 0.38 for $P(Battery = \text{low} \mid Light = \text{off})$ and even ≈ 0.76 for $P(Battery = \text{low} \mid Light = \text{red})$. This pattern is called *evidential* or *diagnostic* reasoning, where the term *diagnostic* is deduced from medical diagnosis, where one tries to identify a disease that best explains the observed symptoms.

The BNs for the robot error detection framework with the variables' CPDs | *Figure 8*



Another pattern, called *intercausal reasoning* describes the phenomenon that if one effect can be induced by multiple causes, these causes can interact in the sense that observing one of them allows drawing conclusions about the others. In the robot error network example, this can be observed when trying to determine the probability of a fully charged battery. While the prior is fairly high $P(Battery = \text{high}) = 0.7$, the probability drops significantly when observing a red flashing light: $P(Battery = \text{high} \mid Light = \text{red}) \approx 0.24$. It is reasonable to believe that the error indicator light should be green on a high battery status, while a red light might indicate a critical battery level. However, adding the observation of a running motor (and therefore obviously strong enough battery), the probability increases again and is even higher than the prior $P(Battery = \text{high} \mid Light = \text{red}, Motor = \text{running}) \approx 0.75$. Obviously, either the observations must be wrong or there must be another explanation for the red flashing light. Looking at the BN, another potential cause for a red light is the LED diode and in fact, observing a faulty diode would increase the probability of a high battery status even more to $P(Battery = \text{high} \mid Light = \text{red}, Diode = \text{faulty}) \approx 0.96$. In other words, the faulty diode *explains away* the flashing light, such that a low battery

explaining away ▷

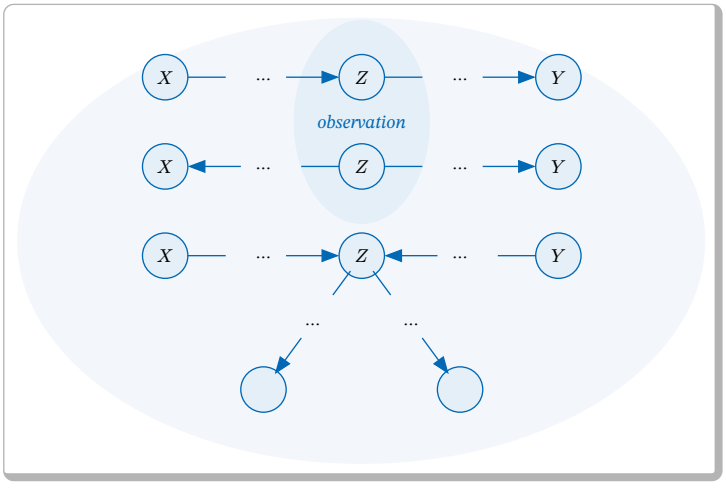
is not the most probable cause anymore. Therefore, even though $Diode$ and $Battery$ are not directly dependent, as there is no edge between the nodes, observing $Light$ as evidence, makes them dependent.

The graph structure of the BN allows to directly determine whether two variables X and Y are conditionally independent or not given some evidence. This is checked by examining whether paths between variables paths are *blocked* or *unblocked* based on the observed variables, which is equivalent to conditional independence. A path is considered blocked, when one of three preconditions is met (Koller and Friedman 2009): There is a node z on the path that

1. has an incoming and an outgoing edge (i.e. z is an ancestor node of one of the two variables X and Y, and a descendant of the other) and z represents a variable that is observed, which means z is part of the evidence, or
2. has two *outgoing* edges (i.e. z is a common ancestor of both X and Y) and z is observed, or
3. has two *incoming* edges (i.e. z is a common descendant of both X and Y) but z is *not* observed and neither are its descendants.

This concept is called *d-separation* and is shown in Figure 9. As an example, observing the report in addition to observing the light when trying to assess the battery status does not provide any additional information (case 1). This is intuitive since the BN structure represents direct dependencies. Because there is no edge from the battery node to the report node, there is no direct relationship between the two represented variables. In other words, the report is conditionally independent from the battery (and the diode and the motor), given that the light was observed: $(Report \perp Diode, Battery, Motor \mid Light)$, therefore the probabilities $P(Battery = \text{low} \mid Light = \text{off}) = P(Battery = \text{low} | Light = \text{off}, Report = \text{warning}) \approx 0.38$.

d-separation criterion in BNs | *Figure 9*



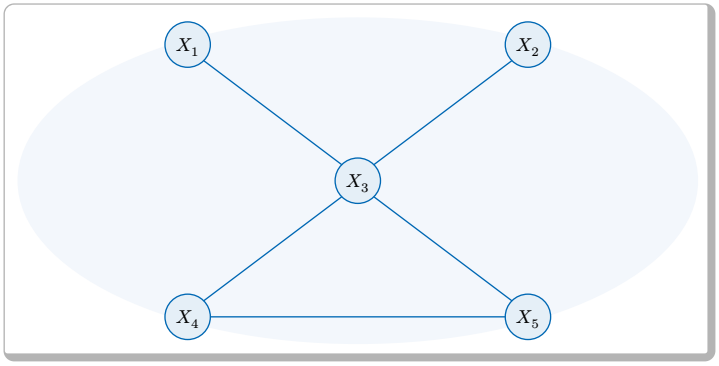
When constructing BN, one has to recognize that the direction of edges does not signify a unilateral dependency. Unlike traditional causal diagrams where arrows denote a clear cause-and-effect relationship, BNs accommodate symmetric dependencies and conditional probabilities which reflects the probabilistic nature of the modeled rela-

tionships. Edges in a BN represent statistical dependencies, indicating that the knowledge of one variable provides probabilistic information about another and therefore capture the flow of information. This does not imply a strict causal hierarchy. In the real world, events are often correlated within complex patterns, which is reflected by these statistical relationships. Constructing a BN therefore requires a deep understanding of the probabilistic dependencies between variables, that go beyond simple causes and effects.

2.5.2 Markov Networks

In contrast to directed models such as BNs, undirected models allow to represent probability distributions where the probabilistic influence between variables do not have have a clear directionality. MNs, also called *Markov Random Fields (MRFs)*, as BNs, are composed of nodes representing random variables and edges representing direct probabilistic interaction between them as shown in Figure 10.

Exemplary Markov Network | *Figure 10*



However, the symmetric influence two connected variables have on each other requires a different parameterization to represent the kind of relationship than CPDs do in BNs. Therefore, the multilateral influences are described in terms of functions ϕ (called *factors*) mapping each possible value assignment X of directly connected nodes to a non-negative real value v:

D

$$\phi : \boldsymbol{X} \mapsto \mathbb{R}^{+} \tag{49}$$

A group of fully connected variables that pose a fully connected subgraph in the network is called a *clique*. The factor assigned to a clique is sometimes called the *clique potential*, describing the local relationships of that group. The global relationship model over the entire graph is then posed by the product over all factors of the *maximal* cliques (cmp. Figure 11), i.e. the ones that contain as many fully interconnected variables as possible. Note that factors are not to be confused with probabilities, as they are not normalized. Their probabilistic semantics, however, become meaningful only

when considering the contributions of all potentials in the network. The global relationship model is then normalized, thus forming a legal (Gibbs) distribution (Koller and Friedman 2009; Bishop 2006) over the set of all variables $\boldsymbol{X}$ in the network

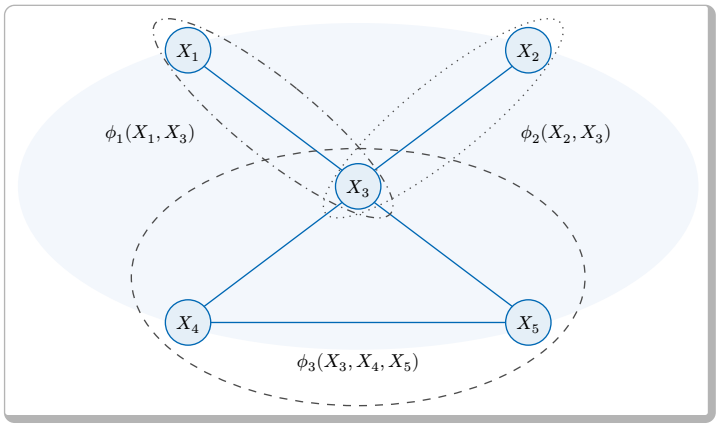
$$P(\boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z} \prod_{c \in G} \phi_c(\boldsymbol{x}_c) \tag{50}$$

where c refers to a (maximal) clique in the network graph G, ϕ_c is thus the factor of the clique c and $\boldsymbol{x}_c$ the assignment from the world $\boldsymbol{x}$ for the variables that occur in c. The normalization factor Z, called the *partition function* is then

$$Z = \sum_{\boldsymbol{x} \in \mathcal{X}} \prod_{c \in G} \phi_c(\boldsymbol{x}_c) \tag{51}$$

with $\mathcal{X}$ being the set of all possible assignments (worlds).

The three maximal cliques $\{X_1, X_3\}, \{X_2, X_3\}$ and $\{X_3, X_4, X_5\}$ with their respective factors (clique potentials) ϕ_1, ϕ_2 and ϕ_3 | *Figure 11*



Unfortunately, as the partition function involves summing the probability masses of all potential scenarios, exact inference in MNs becomes computationally intractable, especially for larger problem instances. Consequently, approximate algorithms are commonly employed in both parameter learning and inference.

While in BNs one has to check the cases of d-separation to determine whether two (sets of) variables are independent, MNs allow direct graph separation, i.e. two disjoint sets of variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ are conditionally independent given a set of variables $\boldsymbol{Z}$ (with $\boldsymbol{Z}$ also being disjoint from $\boldsymbol{X}$ and $\boldsymbol{Y}$, respectively), if there is no path between any node $X \in \boldsymbol{X}$ and $Y \in \boldsymbol{Y}$ with only nodes on that path that are not in $\boldsymbol{Z}$. In other words, if the set $\boldsymbol{Z}$ was entirely removed from the network, there would be no connection between $\boldsymbol{X}$ and $\boldsymbol{Y}$ anymore. In the example from Figure 11, X_1 and X_2 are conditionally independent given X_3. Note that articulated by the values of the factors, directly connected variables (e.g. X_1 and X_2) can still be a priori independent. In general terms, a node X is independent from any other node in the network, given its immediate neighbors (called the *Markov Blanket (MB)*), which is intuitive, as the nodes in the MB separate X from all other nodes.

A MN can be represented as a *Factor Graph (FG)*, which adds factor nodes to the network that are associated with factors and whose direct neighbors are the variables are the members of the clique the factor is assigned to.

The way a FG encodes the relationships of variables can sometimes make certain patterns less obvious. Factors are like tables that describe how variables are related. Similar to how patterns can be found in data, these relationships can have specific structures that depend on particular values of the variables involved. However, these structures may not be immediately clear when looking at the standard representation of a FG. To make these patterns more visible, an alternative way of describing these relationships can be used, which involves converting the relationships into log-space helping to reveal hidden patterns or structures in the data. More explicitly, a factor can be transformed to

$$\phi(\boldsymbol{X}) = \exp(-\varepsilon(\boldsymbol{X})) \tag{52}$$

with $\varepsilon(\boldsymbol{X}) = -\ln \phi(\boldsymbol{X})$ being the *energy function* (Koller and Friedman 2009). Then

$$P(\boldsymbol{X} = \boldsymbol{x}) \propto \exp\left(-\sum_{c \in G} \varepsilon_c(\boldsymbol{x}_c)\right). \tag{53}$$

By introducing an *indicator feature* f_c for each maximal clique c, taking value 1 if a certain value assignment of the clique meets a certain requirement and 0 otherwise, its energy function can be represented as a product of a constant weight w_c and the feature.

$$P(\boldsymbol{X} = \boldsymbol{x}) = \frac{1}{Z} \exp\left(-\sum_{c \in G} w_c \cdot f_c(\boldsymbol{x}_c)\right) \tag{54}$$

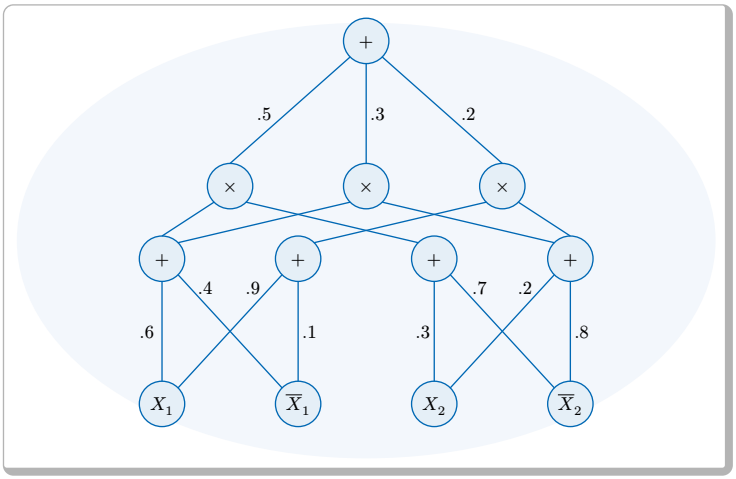
with w_c being a weight which turns the product over the clique potentials in Equation 50 into a sum of weighted features and is a more compact representation due to the reduction of large domains to a (binary) feature function and a weight.

The choice of inference methods on MNs is typically a trade-off between accuracy and computational efficiency. Depending on whether the model structure is well-suited for efficient algorithms (loops in their graphical representation lead to cycles in the dependency structure and make interactions between variables highly interdependent) and the overall size and complexity of the model, it might be advisable to resort to approximate methods such as belief propagation (Bishop 2006), or *Markov Chain Monte Carlo (MCMC)* sampling techniques like Gibbs (Russell and Norvig 2010). In particular, calculating the partition function, which involves the computationally expensive summing over all possible value assignments of the variables, is a key component of many inference algorithms and can be intractable especially for large datasets typically found in real-world applications.

2.5.3 Sum-Product Networks

Poon and Domingos (2011) represent the partition function of PGMs by introducing multiple layers of hidden variables by a deep architecture called *Sum-Product Network (SPN)*. SPNs, like BNs and MNs, represent probability distributions over random variables. However, they comprise three different types of nodes rather than just one. The *sum nodes* represent weighted sums of their child values, the weights being positive, normalized values. The *product nodes*, analogously, represent the product of their child values, capturing relationships between variables. The *leaf nodes* then represent the probability distributions over the individual variables, as known from other network types. SPNs are therefore rooted directed acyclic graphs, whose root values represent an unnormalized probability distribution and can be learnt recursively by either splitting it into a product of SPNs over independent sets of variables or into a sum of SPNs learned from subsets of the instance, if they cannot be found. The idea behind SPNs is that one can take advantage of the fact that graphical models with multiple hidden layers enable effective inference across a broader range of distributions.

An exemplary sum product network (adapted from Poon and Domingos (2011)) | *Figure 12*



An SPN $S(x_1, ..., x_n, \overline{x}_1, ..., \overline{x}_n)$ (short: $S(x)$) is defined as a function over a set of indicator variables x_i with their respective negations $\overline{x}_i$ for each variable X_i. Figure 12 shows an exemplary SPN (adapted from Poon and Domingos (2011)) with only two boolean variables X_1 and X_2, i.e. $S(x_1, x_2, \overline{x}_1, \overline{x}_2)$. The weights at the edges make the network model a weighted mixture of random variables. The joint distribution represented by the network is computed as

$$\begin{aligned} S(x_1, x_2, \overline{x}_1, \overline{x}_2) = & \; 0.5 \cdot (0.6x_1 + 0.4\overline{x}_1) \cdot (0.3x_2 + 0.7\overline{x}_2) \\ & + 0.2 \cdot (0.6x_1 + 0.4\overline{x}_1) \cdot (0.2x_2 + 0.8\overline{x}_2) \\ & + 0.3 \cdot (0.9x_1 + 0.1\overline{x}_1) \cdot (0.2x_2 + 0.8\overline{x}_2) \end{aligned} \tag{55}$$

To compute probabilities in the network, the indicator variables are set to 1 if they are compatible with the evidence and 0 otherwise. For example, calculating $P(X_1 = \top, X_2 = \bot)$, i.e.

$$\begin{aligned} S(x_1, x_2, \overline{x}_1, \overline{x}_2) &= S(1, 0, 0, 1) \\ &= 0.5 \cdot (0.6 + 0) \cdot (0 + 0.7) \\ &\quad +0.2 \cdot (0.6 + 0) \cdot (0 + 0.8) \\ &\quad +0.3 \cdot (0.9 + 0) \cdot (0 + 0.8) \\ &= 0.522 \end{aligned} \tag{56}$$

In general, the above calculation will return an unnormalized, yet proportional value, which has to be divided by the value of the partition function, which is computed the same way but with *all* indicator variables set to 1, i.e. $S(1, 1, 1, 1)$ (short: $S(*)$). However, in the example from Figure 12, the weights are already chosen in a way that the partition function sums up to 1 such that 0.522 is the final result.

Marginals are computed by setting the indicator variables to 1 if they are compatible with the evidence and 0 for the indicator variable of the respective negated variable. All other remaining indicator variables, whose variables do not occur in the evidence, are set to 1. For example, finding $P(X_2 = \bot)$, i.e. marginalizing over X_1 is done by calculating

$$\begin{aligned} S(x_1, x_2, \overline{x}_1, \overline{x}_2) &= S(1, 0, 1, 1) \\ &= 0.5 \cdot (0.6 + 0.4) \cdot (0 + 0.7) \\ &\quad +0.2 \cdot (0.6 + 0.4) \cdot (0 + 0.8) \\ &\quad +0.3 \cdot (0.9 + 0.1) \cdot (0 + 0.8) \\ &= 0.75 \end{aligned} \tag{57}$$

in the example above. Yet again, in the general case, the result has to be normalized using the partition function $S(*)$. Since conditional probabilities can be expressed as the ratio of a joint and a marginal probability, their calculation can be done by combining two of the calculations exemplified above.

According to Poon and Domingos (2011), all manageable PGMs can be formulated as SPNs, which are inherently more versatile. They decompose complex probability distributions into a product of simpler distributions, which makes it computationally efficient. SPNs can be learned from data using generative and discriminative (Gens and Domingos 2012) methods, and there are algorithms for structure learning (Gens and Domingos 2013; Rooshenas and Lowd 2014) and parameter learning (Poon and Domingos 2011) where parameter learning means estimating the weights of the edges of the network and is typically achieved using *Expectation Maximization (EM)* or *gradient descent*. SPNs perform approximate inference in the case of cyclic dependencies. Otherwise cycles have to be replaced by multivariate distributions whose representation is generally untractable (Gens and Domingos 2013). They have been applied in areas such as natural language processing and image and speech recognition.

2.5.4 Probabilistic Circuits

Similarly to the probabilistic models introduced before, *Probabilistic Circuits (PCs)* capture complex probability distributions. However, in contrast to PGMs, they do not ex-

plicitly represent (in-)dependencies between random variables in terms of direct connections. The graph rather represents complex dependencies through *circuits*, which are composed of nodes representing basic operations such as products and sums. The combinations of these operations and the structure of the circuit then determine complex probability distributions. Choi, Vergari, and Broeck (2020) present PCs as a universal and integrated computational framework suitable for handling tractable probabilistic modeling. PCs provide a unified framework that encompasses other graphical models such as SPNs or probabilistic decision graphs, simultaneously facilitating tractable reasoning over parts of their respective model classes. However, in contrast to classical PGMs like BNs and MRFs, which possess a clear representational semantics, PCs are distinctly operational, as units within computational graphs explicitly indicate how to evaluate the probability distributions they represent, essentially providing a guide for answering probabilistic queries.

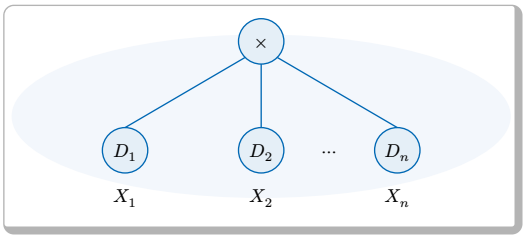
The components of PCs are sums, products and distributions. The most basic graph representation would therefore be posed by a single node, denoting an arbitrary distribution function over one or more random variables. Given some evidence e, the node will return the corresponding PDF of its distribution. Through factorization, more complex probability distributions can be represented, which is denoted by adding a product node to the graph as a connecting node of the factors, i.e., the nodes representing the single distributions that factorize the overall distribution. Assuming these to be independent, the joint probability distribution P_m over k random variables is then defined as

$$P_m(\boldsymbol{X}) = \prod_{i=1}^{k} P_{m_i}(\boldsymbol{X}_i) \tag{58}$$

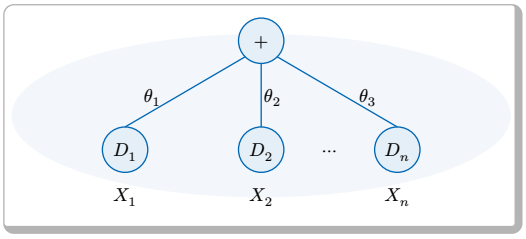
with P_{m_i} being the distribution over a subset $\boldsymbol{X_i}$ of $\boldsymbol{X}$. The graph for a fully factorized distribution (cmp. Figure 13 a)) then contains one node for each single distribution and one product node connecting the distribution nodes.

Examples of PCs | *Figure 13*

A fully factorized PC over n random variables X_i and their respective distributions | *a)*



A mixture model PC with a single sum node and the mixture weights θ_i | *b)*



Inference over the model is now much easier to compute since its broken down into smaller inference tasks. However, the full factorization poses a special case and if one wants to gain expressiveness using a more general factorization, the third type of nodes is required: Adding sum nodes (cmp. Figure 13 b)) allows to represent (weighted) mixtures of models in PCs. The weights θ_i in combination with the graph $\mathcal{G}$ comprising distribution, sum and product nodes form the two components (parameters and struc-

ture) defining a PC. By combining multiple layers of sum nodes and product nodes, arbitrarily complex distributions over the represented random variables can be built. $\mathcal{G}$ is a rooted DAG, though the edges are often not represented as directed paths, due to the convention of plotting them in children-before-parents fashion either from left-to-right or bottom-to-top reading direction.

When considering different types of probabilistic models for certain inference tasks, one has to find a trade-off between tractability and expressiveness, since higher expressiveness typically comes with lower tractability and vice versa. Through the reading direction, the graph encodes the process of evaluating the distribution, making it well-suited for answering probabilistic queries with reduced computational cost compared to some other traditional models. By combining mixtures and factorizations, PCs are both expressive and tractable if several constraints are met:

Decomposability · A product node is considered *decomposable* if its children rely on non-overlapping sets of variables. Consequently, a PC is decomposable, if all of its product nodes are composable.

structural properties of PCs ▷

Smoothness · A sum node is called *smooth* when all its children have identical scopes[24]. Again, the overall PC is considered smooth, when all its sum nodes are smooth.

The process of breaking down large integration problems into simpler computations is facilitated by these two properties, enabling the tractable computation of marginal queries.

If the PC satisfies further structural properties, even the tractable computation of MAP queries is possible:

Determinism · A sum node is *deterministic* if only one of its children produces a nonzero output for any given input. Consequently, the PC is deterministic, if this property holds for all its sum nodes.

and

Consistency · A product node is *consistent*, if any variable shared among its children is present in only one leaf node. Are all product nodes in a PC consistent, the PC is consistent.

Since the distribution nodes can represent arbitrary distributions, PCs provide a unified computational framework allowing to represent different models in a common framework. They provide unparalleled expressiveness by representing complex probabilistic relationships through a hierarchical structure of nodes. Their efficiency in computing probabilities and conducting inference tasks, even within high-dimensional spaces, underscores their tractability. This combination of expressiveness and tractability positions PCs as the optimal choice for modeling and reasoning under uncertainty across diverse domains. In Chapter 3, a type of PCs is introduced, which is particularly computationally efficient, making it the preferred formalism for the implementation in BAYROB.

[24]The scope of a node n is the set of variables it depends on, i.e. the union of all its child nodes' scopes.

2.6 Knowledge Acquisition

So far, different kinds of distributions along with their characteristics have been introduced. However, the key question is, how do we acquire these distributions? In the following, different kinds of (parameter) learning techniques will be outlined briefly.

2.6.1 Generative and Discriminative Learning

When constructing a probabilistic model, the knowledge engineer must consider the tasks the model is intended to perform. Typically, only a subset of variables is queried given a set of observed variables, making it unnecessary to model the complete joint distribution across all variables. In many situations, representing the posterior of the query variables given the evidence is sufficient for the inference task and is favored over modeling the entire joint probability. Models that express knowledge as a joint probability, $P(X, Y)$, involving input variables X and the label Y, are known as *generative models*. These models can generate all variables independently of their semantics (query or evidence variables). They apply Bayes' theorem (Equation 5) to calculate $P(Y \mid X)$ and then select Y based on the highest likelihood (Ng and Jordan 2001). In contrast, *discriminative models* directly represent the posterior probability $P(Y \mid X)$. Both generative and discriminative models can be used for predictive tasks, but discriminative models do not provide a distribution over evidence variables, limiting their ability to draw conclusions about them. However, in many cases, it is possible to identify variables that are solely evidential, rendering the encoding of their distribution unnecessary. Discriminative models are therefore often preferred for classification tasks, where the focus is on *discriminating* data instances. Tasks such as text classification (Zhang, Jiang, and Li 2019) or object detection (Vidal-Naquet and Ullman 2003) are typically examples for such tasks, as the focus lies on learning the boundaries between classes and predicting the presence of objects, respectively, while the distributions over the images or text body is neither of particular interest nor is it efficient to represent it in a joint distribution. Generative models are better suited for tasks involving the *generation* of new samples or capturing the underlying data distribution, such as in image generation. As the name suggests, models in this task are employed to create new samples resembling the represented distribution, such as human faces (Richardson et al. 2021).

2.6.2 Maximum Likelihood Principle

As mentioned in Section 2.3, a distribution quantitatively defines the probabilities of some event, i.e. a random variable taking on a specific value. To generate such a distribution, these quantitative values can either be set manually or learned from data. The former – of course – relies on the subjective judgment of the developer and does not necessarily reflect aspects of the real world. In contrast, learning from data, or *experience*, represents regularities that are based on actual observations. Here, the quality of the underlying dataset plays a pivotal role for the quality of the learnt model. A learning task typically involves determining some model parameters θ that best explain

the observations. The objective is to fine-tune these parameters to improve the model's capacity to capture and understand the inherent patterns or relationships within the dataset and therefore approximate the distribution inherent to the problem. Each datapoint d_i in the dataset $\mathcal{D}$ is considered a complete assignment of values to the variables represented by the model and are assumed to be drawn indepently from each other. The plausibility of the observed data $\mathcal{D}$ under the assumed model parameters θ can be quantified using the the *likelihood function* $\mathcal{L}$, which measures the probability of observing the given data, given a specific set of parameter values:

$$\mathcal{L}(\theta \mid \mathcal{D}) = \prod_{i=1}^{N} P(d_i; \theta) \tag{59}$$

The best set of parameters is then determined by maximizing this function (Eq. 59) w.r.t. θ

$$\begin{aligned} \hat{\theta} &= \arg\max_{\theta \in \Theta} \mathcal{L}(\theta) \\ &= \arg\max_{\theta \in \Theta} \prod_{i=1}^{N} P(d_i; \theta) \end{aligned} \tag{60}$$

likelihood and MLE ▷

to get the *Maximum Likelihood Estimate (MLE)*, which provides a point estimate for the parameters, and under certain conditions (such as large sample sizes), possesses desirable statistical properties, including consistency and asymptotic normality. Choosing the values for the parameters such that they maximize the likelihood of observing the given data under the assumed statistical model is commonly referred to as the *maximum likelihood principle*. The method is widely used in various fields for parameter estimation. While the likelihood function is a product of individual probabilities for each data point, the *log* likelihood transforms the product of probabilities into a sum of logarithms, providing computational simplicity, numerical stability (when dealing with small probabilities), and efficiency in optimization, as many algorithms are designed for additive structures. At the same time, likelihood and log likelihood functions have the same maxima due to the logarithm function being strictly monotonically increasing. Therefore, the log likelihood is often preferred for practical and computational reasons.

2.6.3 Entropy-based Methods

entropy ▷

Another way of constructing informative models for classification or regression tasks is to select features that contribute the most to the organization and separation of the data. In doing so, the concept of *entropy* is leveraged, which measures the uncertainty (or *disorder*) for an attribute, represented by random variable X, in a dataset D. The entropy, first instroduced by Shannon (1948) and sometimes also called *information*, is defined as

D

$$\mathrm{H}(X) := -\sum_{i=1}^{n} P(x_i) \cdot \log P(x_i) \tag{61}$$

where n is the number of values X can take, i.e. $n = |dom(X)|$.

To understand the intuition behind entropy, it is best to explain the idea with a simple coin example, where the correct prediction of the outcome of a coin toss results in a win and the incorrect prediction leads to a loss of money, in this example, assume a prize (or loss) of 1€ per toss. The theory behind entropy is then determining the value of prior information about the outcome of the toss. Given a *biased* coin, which shows *Head* in 99% of the tosses and *Tail* in only 1% (i.e. the random variable X can take two different values *Head* and *Tail*, with the respective probabilities 0.9 and 0.01), the expected information per toss according to Equation 61 would be determined by calculating

$$1.0€ \cdot 0.99 - 1.0€ \cdot 0.01 = 0.98€. \tag{62}$$

In other words, since the coin toss almost always results in showing *Head*, the prediction is very easy and will almost always be correct.

Given a *fair* coin that shows *Head* and *Tail* both in half of the cases, the calculation amounts to

$$1.0€ \cdot 0.5 - 1.0€ \cdot 0.5 = 0.0€, \tag{63}$$

which means no realiable prediction about the outcome can be made, making it not a very useful source of information. Thus, the less information is provided regarding the outcome, the greater the value of prior information about it becomes. This observation is then used to partition datasets in the learning process of prediction models. By quantifying the impurity of a dataset allows to compare different partitionings of the same dataset. The goal is to select a partition of the dataset in a way that minimizes its expected entropy. Assuming that the selection of the attribute X splits the original dataset into n subsets D_i, one for each of the values of X, then each of these subsets has its own entropy $\mathrm{H}(X_i)$. The expected entropy of the data *after* the selection of the attribute X is then considered the weighted average (cmp. Quinlan (1986))

D

$$E(\mathrm{H}(X)) := \sum_{i=1}^{n} P(x_i) \cdot \mathrm{H}(X_i) \tag{64}$$

and can be used to determine the *information gain* of a split at a particular attribute, which is the reduction in entropy of a split compared to the variable's entropy in the dataset before the split (cmp. Quinlan (1986)):

D

$$G(X) := \mathrm{H}(X) - E(X). \tag{65}$$

Another way of determining the best (discrete) split variable is calculating the *Gini impurity* (Jost 2006) for each attribute, which is 1 minus the sum of squares of success probabilities of each of the attribute's values

D

$$Gini(X) := 1 - \sum_{i=1}^{n} P(x_i)^2 \tag{66}$$

and then choose the one with the lowest (i.e. closest to 0) score. Logically, the goal is to reduce uncertainty in the data. Partitioning the datasets is typically applied in tree-based methods, where the objective lies in constructing a tree structure representing a hierarchical partitioning of the input space. They are composed of nodes (inner nodes as well as leaves) and branches, which earned them the name. Tree-based models are mainly separated into two groups:

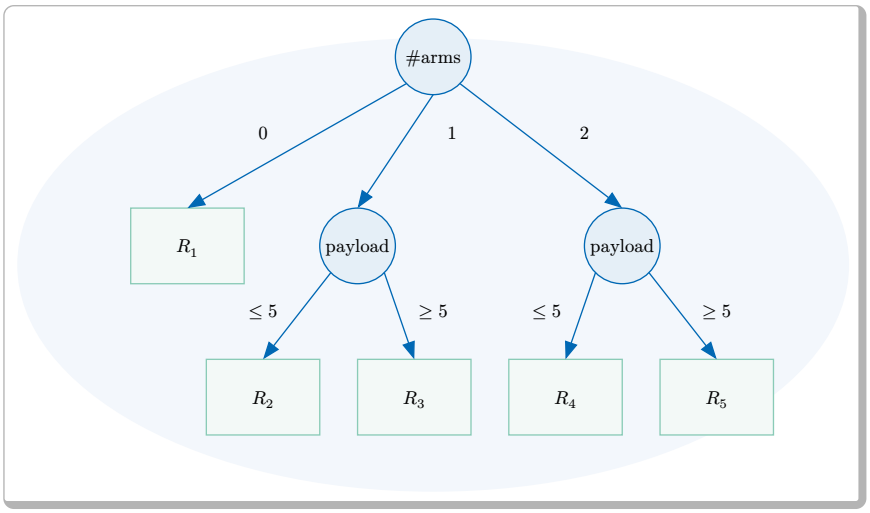
Decision Trees (DTs) are used for classification tasks aiming at assigning instances to predefined categories (or *classes*). Inner nodes in a DT represent decision criteria for a certain feature, based on which the data is split. The leaf nodes represent the class labels. The decision criterion is determined based on impurity measures, such as the abovementioned information gain or the Gini impurity. These methods are used to find the best attribute for splitting the data at the respective node, branching into n subtrees, where n is the number of different values that particular attribute can take. The algorithm therefore selects the attribute that maximizes information gain, leading to a more organized and informative partitioning of the data. Given a feature assignment, one can follow a path along the DT, starting at the root node, following the splits according to the feature assignment and will then end up at a particular leaf node determining the class for the input assignment. A well-known algorithm for constructing DTs is *Iterative Dichotomiser 3 (ID3)* (Quinlan 1986), which iterates over all attributes of the dataset and (greedily) selects the the one with the highest information gain. The data is then split according to the values of the chosen attribute and the algorithm is executed recursively on the created partitions. Its successor *C4.5* (Quinlan 2014) adds the support for continuous feature variables (though the targets are still limited to categorical rather than continuous variables), and allows missing attribute values.

Regression Trees (RTs) object to regression tasks, predicting a continuous numerical value. Similar to DTs, the nodes represent conditions based on features, but the splitting is typically binary, where one side matches the condition (e.g. $X \leq x$) and the other does not. Learning algorithms search for the optimal threshold that maximizes information gain. The leaf nodes represent predicted numerical values. The measures to determine the best attribute and split point is chosen aiming at minimizing the prediction error, such as the *Mean Squared error (MSE)*.

Figure 14 shows an example of a tree for classifying robot instances based on only two properties, the number of arms $\#arms$ the robot possesses and the $payload$ (in kilograms) it can carry. While robot R_1, in this example a small robotic dog, can be classified directly based on its characteristic of not having arms (in the sense of robotic grippers), the others require additional information for coming to a conclusion. R_2 and R_3 both have a single arm, however, while one of them is very powerful such as an industrial

robot, the other cannot carry heavy objects, hence it might be designed for precision tasks rather than lifting payload. The same goes for the remaining 2-armed robots R_4 and R_5, where one might be a houshold robot designed to carry lightweight loads such as food items or kitchen utensils, the other might be a humanoid designed for presentation rather than executing physical tasks.

Example of a decision tree with two decision criteria $\#arms$ and $payload$ (in kilograms) to classify robot instances | *Figure 14*



When constructing DT or RT models, certain stopping criteria are applied to govern the properties of the resulting tree. Selecting unfavorable stopping criteria, the tree might over-generalize or overfit, or simply be too large to be computed. Therefore one can for example limit the depth of the constructed tree, the number of samples in a leaf or achieve a certain level of purity in the splits. Still, the algorithms may create overly complex trees that are prone to overfitting which can be addressed by pruning techniques.

The *Classification and Regression Trees (CART)* algorithm (Breiman et al. 1984) recursively splits the dataset in two subsets to purify the data (cmp. Algorithm 1). The function SAME-CLASS(D) returns true when all of the samples in the dataset D are in the same class. Is that the case, a leaf node is created (CREATE-LEAF(c)) representing the class label of the dataset. The function DETERMINE-CLASS(D) returns the class that best describes the dataset, e.g. a majority vote (in case the class is not the same for all instances in the dataset). FIND-BEST-SPLIT(D) returns a feature and a value of that feature that maximize the Gini impurity or the information gain, depending on whether the feature variable is continuous or discrete. If no such best split feature exists, the algorithm again halts and creates a leaf node with the respective representative class label, otherwise a decision node is created for that feature using CREATE-NODE(s, v). The recursion is then performed using the newly created data splits (SPLIT(s, v, D)). The returned subtrees are then set as child nodes (SET-CHILDREN($l, r, node$)) for the created decision node. The algorithm stops when some stopping criterion is met, such as the minimum number of samples required per leaf or the maximum depth of the generated tree.

◁ *CART*

CART: The pseudoized algorithm for learning tree structures | *Algorithm 1*

CART(D):

Input: D, a dataset

Output: a decision tree

```
1  if SAME-CLASS(D) then
2      return CREATE-LEAF(DETERMINE-CLASS(D))
3  end if
4
5  best_split, split_value ← FIND-BEST-SPLIT(D)
6  if not best_split then
7      return CREATE-LEAF(DETERMINE-CLASS(D))
8  end if
9
10 node ← CREATE-NODE(best_split, split_value)
11 left_split, right_split ← SPLIT(best_split, split_value, D)
12 left ← CART(left_split)
13 right ← CART(right_split)
14 node ← SET-CHILDREN(left, right, node)
15
16 return node
```

chapter three

Scalable Probabilistic Hybrid Models

This chapter introduces JPTs, an innovative approach designed to render the learning and analysis of joint probability distributions feasible within real-world contexts. JPTs offer comprehensive support for both symbolic and subsymbolic variables, seamlessly integrated within a unified hybrid model. Unlike conventional methods, JPTs do not necessitate a priori knowledge of variable interdependencies or specific distribution families. Instead, JPT representations harness tree structures that effectively partition the problem domain into pertinent subregions, a process derived from empirical training data rather than predetermined dependency assumptions.

The architecture of JPTs yields substantial benefits, as both the learning and reasoning processes exhibit linear scalability. The trees empower transparent insight into any posterior probability $P(Q|E)$, enabling lucid explanations for all inference outcomes. Demonstrative experiments underscore the pragmatic utility of JPTs in scenarios involving high-dimensional, heterogeneous probability spaces, even when dealing with millions of training instances. In light of this, JPTs emerge as a promising and viable alternative to traditional probabilistic graphical models, bridging the gap between theory and application.

In this chapter, the task of summarizing the fundamental insights presented by Picklum et al. (2023) is approached. The aim here is to encapsulate the core of this work, offering a concise yet illuminating overview of its main discoveries, research methods, and ramifications. By revisiting the content of this significant work, the goal is to provide readers with a clear comprehension of the pivotal contributions that can have a great impact on the landscape of this subject area.

The development of JPTs took place in collaboration with Daniel Nyga and Tom Schierenbeck.

3.1 Introduction to JPTs

Joint probability distributions offer a wide range of high-potential applications in engineering, science, and technology (Chater, Tenenbaum, and Yuille 2006; Knill and Pouget 2004). Besides families of continuous distributions, PGMs, such as BNs and MRFs (Koller and Friedman 2009), are the de-facto standard in probabilistic knowledge representation. They provide graph-based languages to model dependencies and independencies of variables, and local joint or conditional distributions that quantify the statistical dependencies. However, the practical applicability of PGMs suffers from the representational and computational complexity of learning and reasoning. Exponential runtime for learning and reasoning often can only be avoided by introducing strong independence assumptions that must be known prior to learning and may turn out to be too great simplifications of a model to be of practical use (Besag 1975; Jain 2012). As a simple example, consider a probability space $\langle X, Y, C \rangle$ of two numeric variables, X and Y, and one symbolic variable C, $dom(C) = \{Red, Blue\}$ as illustrated in Figures 15a) and b). Let the symbolic values *Red* and *Blue* demaracate two clusters that are approximately normally distributed. Classic methods for density estimation postulate a mathematical model and apply the maximum likelihood and expectation/ maximization principles in order to find the model parameters that fit the data best. However, this learning process is expensive since the unconstrained parameter space is huge and most learning methods do not exploit the structure of the training data and their underlying distribution.

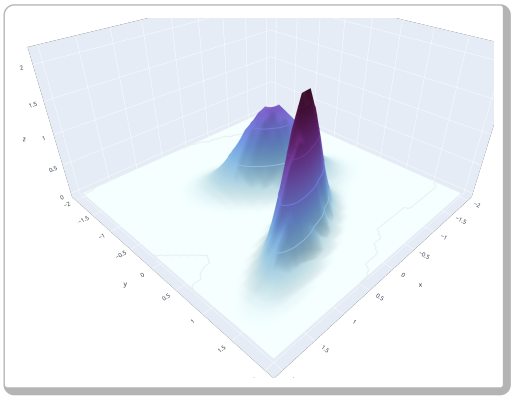
JPTs are model-free shallow deterministic probabilistic circuits and they exploit the structure of the data to be learnt from in a greedy fashion in order to construct a tree structure that partitions the problem space recursively into subspaces. In its leaves, marginal distributions, represented by CDFs over all variables in the respective subspace are maintained, which can be superimposed in order to obtain a sound and globally consistent posterior belief. As opposed to most PGMs, dependencies among variables in JPTs do not need to be known at design time and only very mild assumptions about models are required. Figure 15 d) shows such a tree structure that has been learnt from the data in Figure 15 a), and the marginal distribution in Figure 15 c) shows indeed close resemblance to the ground truth distribution in Figure 15 a). JPTs allow the computation of any posterior distribution in a transparent and explainable way, such that the rationale for any inference result can be provided in a human-interpretable form.

JPTs and PCs ▷

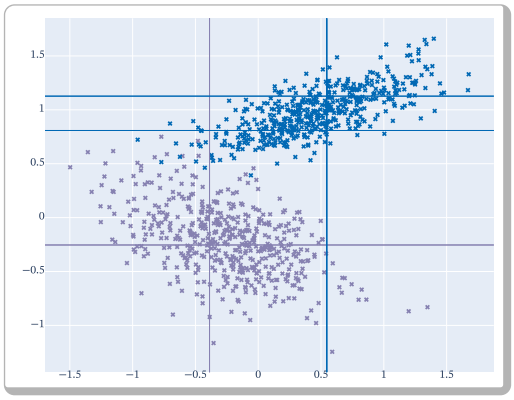
In the following, the concept of Joint Probability Trees as a novel framework for knowledge representation and reasoning in probabilistic hybrid domains will be formally introduced and algorithms for the learning of and reasoning about JPTs will be presented. The concept of quantile-parameterized distributions is then adapted for the efficient learning of univariate continuous distributions without prior assumptions about their functional forms, and third, the performance of JPTs is investigated and showcased empirically (by courtesy of Tom Schierenbeck) on publicly available datasets.

Example of a joint probability distribution of two numeric variables (X, Y) and one symbolic variable (color). Enlarged versions of the figures can be found in the supplementary material. | *Figure 15*

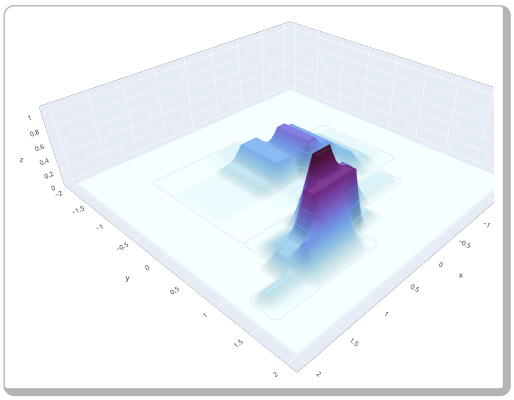
The ground truth distribution | *a)*



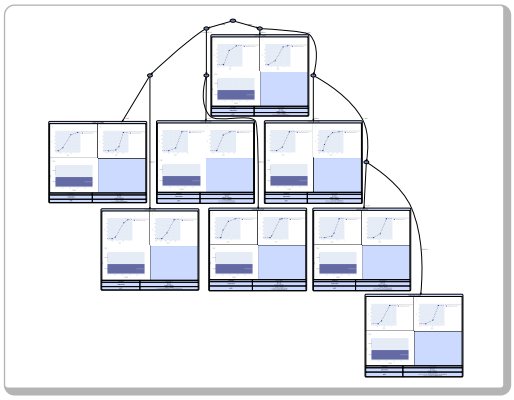
Scatterplot of the sample data | *b)*



The marginal $P(X, Y)$ represented by the JPT in d) | *c)*



The tree structure of the corresponding JPT | *d)*



3.2 Conceptual Framework

In this section, the concept of JPTs is introduced more formally. Let us denote the D -dimensional problem space under consideration as $X = \langle X_1, ..., X_D \rangle$, where X is a vector of random variables X_i, whose domains are denoted by $dom(X_i)$. The set of possible worlds, i.e. all possible complete variable assignments, is denoted by $\mathcal{X}$, and a specific assignment to all variables in X, by $x = \langle x_1, ..., x_D \rangle$, where $x \in \mathcal{X}$. As a representational formalism, JPTs make use of tree-like structures like classification and regression trees (Breiman, Friedman, Olshen, and Stone 1984). Trees have a couple of desirable properties that put themselves forward to be used as a knowledge representation formalism. Most notably, they (1) are simple to understand and interpret, (2) can be thought of as white box models that foster explainable decision making, (3) are compact and sparse representations that allow efficient learning and reasoning. These

features are leveraged through the model structures implementing a recursive partition of the problem space under consideration. Let $T : \mathcal{X} \mapsto \Lambda$ be a tree-like structure that associates an input sample $x \in \mathcal{X}$ with one of its leaves $\Lambda = \{\lambda_1, ..., \lambda_N\}$. The recursive partitioning through a tree structure guarantees that T is exhaustive and mutually exclusive.

As a consequence, the result of the application of T, $T(x)$, can be treated as a random variable that indicates which leaf λ an arbitrary sample x will be associated with. An auxiliary variable L, $dom(L) = \Lambda$ is thus introduced, that extends the problem space X by L and hence forms a new probability space $X' = \langle X, L \rangle$. Applying the law of total probability, one can marginalize the prior probabilities of X over X' by

$$P(X = x) = \sum_{\lambda \in \Lambda} P(X|L = \lambda) \cdot P(L = \lambda). \tag{67}$$

The leaf priors $P(L = \lambda)$ can be easily obtained from the portions of training data that are covered by the respective leaf during training and can be permanently stored in the leaf data structure. The leaf-conditional distributions $P(X_1, ..., X_D|L = \lambda)$, however, are more difficult to represent as they still comprise the joint distributions over X. Here, the naïve Bayes assumption that postulates conditional independence among all X_i is introduced, once L is known. This assumption is reasonable as the leaves in T represent contiguous subregions in $\mathcal{X}$ that are formed during learning by minimizing the mutual information of variables (or maximizing information gain, respectively), which can be proven equivalent to statistical dependence (Murphy 2022). Assuming independence of variables in the leaf nodes in turn allows to represent the priors over X_i conditioned by the respective subspace in every leaf node of T in an extremely compact way:

$$P(X = x) = \sum_{\lambda \in \Lambda} P(L = \lambda) \prod_i P(X_i = x_i|L = \lambda). \tag{68}$$

Equation 68 is called the JPT distribution, which can be used in order to compute the marginal probability of a possible world $x \in \mathcal{X}$. It can be regarded as a mixture model, whose mixing coefficients are represented by the leaf priors, and the local distributions are given by the priors in the respective leaves.

3.2.1 Reasoning in Joint Probability Trees

Computing the marginalization over the tree leaves in Equation 68 may seem like unnecessary effort as, once x is known, λ is fully determined by T. However, mixing the leaf distributions becomes inevitable if the values of only a fraction of variables $E \subset X$ are known in advance and the posterior probability of a subset $Q \subseteq X$, $P(Q|E)$, needs to be computed. Extending Equation 68 to canonical posterior inference can be achieved in a straightforward way by introducing background evidence to the JPT distribution:

$$P(q|e) = \sum_{\lambda \in \Lambda} P(\lambda|e) \prod_i P(q_i|\lambda, e_i), \tag{69}$$

where $q = \langle q_1, ..., q_D \rangle$ is a vector of values of the query variables and $e = \langle e_1, ..., e_D \rangle$ is a vector of constraints e_i for the variables X_i. Note that, because of the naïve Bayes assumption, q_i only depends on the constraints on the respective same variable X_i, i.e. e_i, instead of the entire vector e. The most interesting part of the posterior in Equation 69 is the factor $P(\lambda|e)$, the probability that a sample satisfying the constraints e will be associated with leaf λ. While the pure leaf priors $P(\lambda)$ can be obtained from their associated data portions, the conditional $P(\lambda|e)$ is slightly more complex to compute. However, the tree structure of T can be exploited as an efficient approximation as follows. According to Bayes' theorem, $P(\lambda|e) \propto P(e|\lambda) \cdot P(\lambda)$ holds. The distribution $P(L|e)$ can thus be computed by evaluating $P(e|\lambda) \cdot P(\lambda)$ for every λ and normalizing the results to form a proper distribution. $P(e|\lambda)$ can in turn be factorized according to $\prod_i P(e_i|\lambda)$ due to the conditional independence assumption, which corresponds to the prior distributions stored in every leaf of T. A significant performance gain can be achieved by testing both q and e against the path conditions of every leaf λ. If either of the two violates a path condition, the entire subtree can be pruned for a specific reasoning problem.

◁ *JPT tree structure*

3.2.2 Learning of Joint Probability Trees

In order to learn a JPT from data, a variant of the popular and well-known tree learning methods is proposed, which is introduced in this section. C4.5 (Quinlan 1993) and CART (Breiman, Friedman, Olshen, and Stone 1984) are very successful variants of induction algorithms for classification and regression trees. Essentially, a tree is being built by splitting the input data into subsets constituting the input data for the successor children. The splitting consists of a variable or a variable/value pair that marks a pivot position in the observable feature space optimizing some criterion of impurity with respect to the split data sets. The split constitutes the root node of the tree and the process is repeated on each of the child subsets recursively and terminates when the impurity with respect to the target variables within a subset at a node is minimal or when splitting no longer reduces the impurity. The result of the procedure is a tree structure T, whose inner nodes represent *decision nodes* at which a single input vector x is evaluated according to the pivot variable attached to the respective node and the subsequent child node to proceed with is determined by the value of that variable. If the child node does not have any further children, a *terminal node* is reached in which the predictive values of the target variables are stored, which constitutes the return value $T(x)$ of the tree. For a more detailed discussion of tree learning it is referred to Loh (2014).

Generative Learning · Ordinary classification and regression tree learning is defined over dedicated feature (input) and target (output) variables of the problem space and therefore it is also called a discriminative learning setting. In each node, the best possible split of the remaining data is looked for in the set of feature variables and their domains, and every possible split is evaluated with respect to its potential to reduce the impurity of the target variables. Although discriminative learning of JPTs is also possible, the focus here lies on the generative learning case. In contrast to discriminative CART learning, in JPTs, the entire set of variables X functions both as feature and as

target variables. This means that in every decision node during the learning process, the node impurity is evaluated with respect to all available variables X_i, and all variables X_i are considered as potential split candidates.

As JPTs support both symbolic and numeric variables, a measure of impurity is needed to account for this hybrid character of the model. Here, the difficulty arises that typical error measures for numerical data, e.g. the MSE and impurity measures for symbolic data, e.g. entropy, reside in different and incompatible value ranges. In order to harmonize the two worlds of symbolic and subsymbolic impurity, a combined measure of normalized, relative impurity improvement is proposed as follows. Following the definition in Equation 61, for a distribution over a symbolic random variable X, its entropy is defined by

$$H(P(X)) = -\sum_{x \in dom(X)} P(x) \cdot \log P(x). \tag{70}$$

impurity improvement ▷

In CART learning, a possible split is considered better the more it reduces the expected entropy over the children induced by the split. As a multinomial variable has its highest entropy in the uniform distribution, the entropy can be normalized with respect to the maximal entropy a distribution of the same domain size can have, i.e., $H(\mathcal{U}(X))$, where $\mathcal{U}(X)$ denotes the uniform distribution over X. The impurities of both numeric and symbolic variables can be normalized through the percentage by which they would be reduced by a split, which we denote as $H_{\text{rel}}(P(X))$ and $MSE_{\text{rel}}(P(X))$. The two measures of impurity improvement of symbolic and subsymbolic variables are combined by a weighted average to form the total impurity improvement I over a data set $\mathcal{D}$ when the split of the data is performed on the variable X_i,

$$\begin{aligned} I(\mathcal{D}, X_i) = &\frac{|X_{\text{sym}}|}{|X|} \sum_{X_j \in X_{\text{sym}}} H_{\text{rel}}\left(P_{\mathcal{D},X_i}(X_j)\right) \\ &+ \frac{|X_{\text{num}}|}{|X|} \sum_{X_j \in X_{\text{num}}} MSE_{\text{rel}}\left(P_{\mathcal{D},X_i}(X_j)\right), \end{aligned} \tag{71}$$

where X_{num} and X_{sym} denote the sets of numeric and symbolic variables in X, respectively, and $P_{\mathcal{D},X_i}(X_j)$ denotes the distribution over X_j induced by the data set $\mathcal{D}$ when split at variable X_i.

When the splitting criterion does not lead to an improvement of the impurity within a node or some learning threshold is reached, a terminal node is generated by the learning algorithm. Terminal nodes in JPTs hold the marginal univariate distributions over all variables in X that can be induced by the data in the current node. In the next section, it is discussed in greater detail how these distributions can be represented and learnt efficiently.

Discriminative Learning · JPTs can also be learnt in a discriminative fashion. Discriminative learning can be advantageous, when it is possible to commit to dedicated sets of input and output variables. In such cases, the learning process can be more efficient and the learnt model can be more compact and more accurate than its generative

counterpart. The JPT learning algorithm reduces to ordinary CART learning, when the set of variables is split into dedicated feature and target variables.

Structure Learning · In ordinary PGMs like Bayesian or Markov networks, learning the structure of the graphs, i.e. the dependency model of the variables under consideration, represents a difficult learning problem on its own. As most of the learning methods in PGMs make strong assumptions about a fixed graphical model, learning the network structure is typically a problem even harder than learning the model parameters alone (Koller and Friedman 2009). It is important to note that, in JPTs, the model structure T is elicited from the data distribution during the learning process and thus T encodes the dependencies among the variables. This entails two essential benefits: First, no additional computational effort needs to be carried out in order to obtain the graphical model, and second, no prior knowledge about the domain of discourse must be incorporated prior to learning. In particular, in data mining and knowledge discovery applications, this is a highly desirable property of a learning algorithm.

3.2.3 Example

Let us illustrate the concept of JPTs by means of the example already presented in the Introduction, see Figure 15. The JPT that has been acquired from these data is shown in Figure 15 d), an enlarged version of which can be found in the appendix (Figure 87). Every leaf in the tree structure corresponds to one rectangular subregion in the partition of the problem space in Figure 15 b). Although the JPT learning does not make any assumptions about the functional form of the distribution, the two clusters in the distribution are represented reasonably well. Every leaf has also attached the prior distributions over all three variables in the respective subregion. Visualizations of the distributions are shown as CDF plots for the distributions over the two numeric variables and as histograms for the distributions over the symbolic variable. For better readability, an enlarged version of each of the images is put into the supplementary material accompanying this document.

3.3 Learning & Reasoning in Continuous Domains

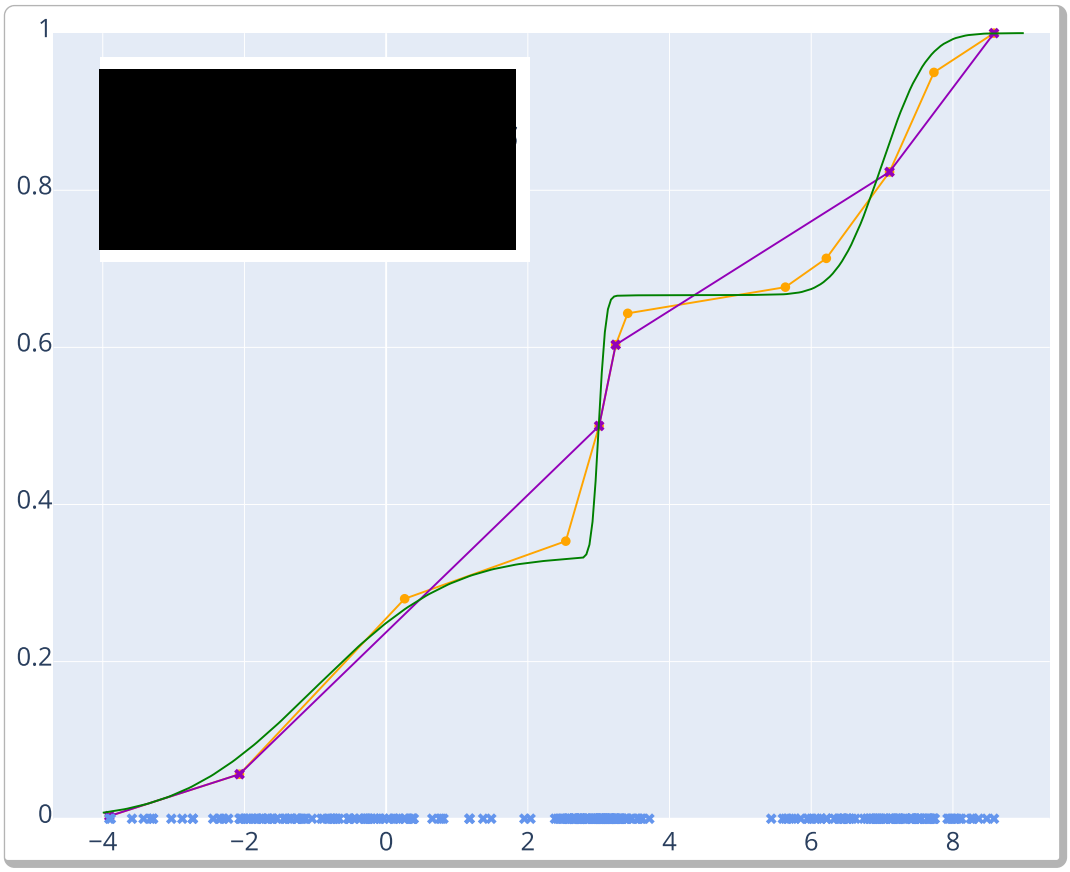
Committing oneself to a specific functional form of the PDF beforehand and hence to its respective model-tailored learning procedures (e.g. a normal distribution with its mean and covariance) as required by many state-of-the-art methods, may be subject to misrepresent the underlying data. More complex models to represent more sophisticated distributions, as provided by Gaussian Mixture Models or kernel-based methods tackle this problem but typically require iterative methods like EM as they cannot be optimized in a closed form.

3.3.1 Quantile-parameterized Distributions

Keelin and Powley (2011) introduce the concept of *quantile-Parameterized Distributions (QPDs)*, which is adapted in this paper for the purpose of representing and acquiring continuous univariate probability distributions. In QPDs, the CDF is represented and learned instead of the PDF. The CDF, in turn, is the integral over the PDF and represents the γ-quantile probabilities, $P(X \leq x_\gamma)$, of the distribution. The principal advantage of learning the CDF over learning the PDF is that *any* CDF can be easily learnt from data as follows. Let $\mathcal{D} = \{d_1, ..., d_N \mid i < j \Rightarrow d_i \leq d_j\}$ denote the sorted, indexed set of 1-dimensional data samples from the continuous domain under consideration. A dataset $\tilde{\mathcal{D}} = \{\langle d_i, \gamma_i \rangle\}$ can be constructed where $\gamma_i = \frac{1}{|\mathcal{D}|}|\{d \mid d \in \mathcal{D}, d \leq d_i\}| = \frac{i}{|\mathcal{D}|}$ is the γ_i-quantile of the data set $\mathcal{D}$. $\tilde{\mathcal{D}}$ serves as training data for a supervised regression task, the result of which corresponds to the CDF $F(x)$ of the desired probability distribution. In principle, any regression model can be used to fit and represent the CDF. In order to approximate the CDF of a distribution, the use of *Piecewise Linear Function (PLF)* is proposed. A PLF $f(x)$ is a function defined on a finite number of intervals in $\mathbb{R}$, each of which has a linear function f_i attached. The set of intervals partition the domain of the function. The function value of f at a particular point $\hat{x}$ is given by the value of the function f_i whose attached interval encompasses $\hat{x}$.

piecewise linear functions ▷

Samples (blue) drawn from a mixture of three Gaussian CDFs (green, log-likelihood of -640.18), the CDF approximated by a PLF with $\varepsilon = 0.05$ (orange, log-likelihood of -685.91) and the CDFs approximated by a PLF with $\varepsilon = 0.01$. (purple, log-likelihood of -624.49) The hinges of the PLF are marked by orange dots and purple crosses. | *Figure 16*



For several reasons the use of a PLF as a regressor of the CDF of a probability distribution is appealing.

1. the concatenation of linear functions is very general, so that a PLF is capable of approximating functions of arbitrary shape and precision, and thus the resulting CDF can be considered free of model assumptions
2. computations, manipulations and interpretations are intuitive and simple
3. there are efficient algorithms to acquire PLFs, such as *Multivariate Adaptive Regression Spline (MARS)* (Friedman 1991)
4. individual pieces can be learnt independently on partitions of the data and composed afterwards to form the final distribution

Figure 16 shows an example of a mixture of three Gaussian distributions and the approximation of its CDF in the form of a PLF that has been acquired from samples drawn. The figure shows that the PLF is not only similar to the CDF of the three Gaussian distributions, but also competitive in terms of the likelihood.

CDF-Learn | *Algorithm 2*

CDF-LEARN($\tilde{\mathcal{D}}$):

Input: $\tilde{\mathcal{D}} = \{\langle d_i, \gamma_i \rangle\}$: a sorted set of γ_i-quantiles

Output: a set of hinge points of a PLF

1 **if** $\text{MSE}\left(\tilde{\mathcal{D}}\right) < \varepsilon$ **then**
2 **return** $\emptyset$
3 **end if**
4 Determine $\tilde{\mathcal{D}} = \left\langle \tilde{\mathcal{D}}_1^{i^*}, \tilde{\mathcal{D}}_2^{i^*} \right\rangle$, with $\tilde{\mathcal{D}}_1^{i^*} = \{d_1, ..., d_{i^*}\}, \tilde{\mathcal{D}}_2^{i^*} = \{d_{i^*}, ..., d_N\}$ such that $i^* = \arg\min_i \mathbb{E}\left(\text{MSE}\left(\tilde{\mathcal{D}}^{i^*}\right)\right)$
5 **return** $\{i^*\} \cup$ CDF-LEARN$\left(\tilde{\mathcal{D}}_1^{i^*}\right) \cup$ CDF-LEARN$\left(\tilde{\mathcal{D}}_2^{i^*}\right)$

3.3.2 Efficient Learning of Cumulative Distributions

In this section, an efficient algorithm for fitting CDFs in the form of PLFs is introduced. It is based on recursive partitioning of the data set $\tilde{\mathcal{D}}$ and inspired by regression tree learning (Breiman, Friedman, Olshen, and Stone 1984). Starting with $\tilde{\mathcal{D}}$, a point $< d_{i^*}, \gamma_{i^*} > \in \tilde{\mathcal{D}}$ is determined, which minimizes the MSE of a PLF of the form

$$f(d) = \begin{cases} f_1(d) \text{ if } d_1 \leq d \leq d_{i^*} \\ f_2(d) \text{ if } d_{i^*} \leq d \leq d_N \end{cases}, \tag{72}$$

for all $d \in \tilde{\mathcal{D}}$. This process is recursively repeated on the subsets $\tilde{\mathcal{D}}_1 = \{d_1, ..., d_{i^*}\}$ and $\tilde{\mathcal{D}}_2 = \{d_{i^*}, ..., d_N\}$ until an error bound ε is being undercut. Note that, if $\varepsilon = 0$, the

process will terminate when all points in $\tilde{\mathcal{D}}$ have been selected as an optimal split point once. In this case, the learnt PLF reaches the highest possible likelihood, where every point is perfectly matched. The set of all points represents the interval boundaries of the PLF. The CDF-LEARN algorithm is listed in Algorithm 2. $\text{MSE}(\mathcal{D})$ determines a function that returns the MSE of all points in $\mathcal{D}$ applied to a linear function through the minimal and maximal points in $\mathcal{D}$. The CDF-LEARN pass yields the hinge points in $\mathcal{D}$ that can be connected to form a linear spline of the CDF. The CDF is constantly 0 for all $d < \min \mathcal{D}$ and constantly 1 for all $d \geq \max \mathcal{D}$.

3.3.3 Reasoning about Cumulative Distributions

A-priori Reasoning · In the previous section QPDs were introduced as a model-free representation of probability distributions and the advantages of learning the respective CDF was outlined. Representing the CDF F directly is advantageous over using the PDF, as marginal probabilities $P(X \leq x_0)$ and $P(X > x_0)$ can be obtained directly by evaluating F without the computationally expensive step of integrating the PDF. Prior probabilities over intervals $[x_l, x_u]$ can be computed by $P(x_l \leq X \leq x_u) = F(x_u) - F(x_l)$.

A-posteriori Reasoning · The calculation of a posterior $P(X|x_l \leq X \leq x_u)$ can be implemented in a similarly straightforward fashion. By decomposing the piecewise linear CDFs according to the posteriors' condition, the distribution now represents only the required interval $[x_l, x_u]$. Simply cropping and extracting a part of the function at the interval boundaries would leave us with an invalid QPD, since in most cases it will not represent probabilities ranging from 0.0 to 1.0. To normalize the distribution, the cropped part of the function is shifted to the base axis and stretched such that the properties of a valid distribution function are restored, i.e. $F(x_l)$ and $F(x_u)$ will evaluate to 0.0 and 1.0, respectively.

Confidence-rated Output · In many practical applications, obtaining a mere predictive value of a target variable is insufficient. Especially in safety-critical applications, it is crucial that predictions can have a confidence value attached expressing their dependability. For example, it is useful to report a confidence interval that encompasses the expectation of a variable X, $\mathbb{E}(X)$, with a certain probability – the confidence level. In order to compute such a confidence interval $[x_l, x_u]$ given a confidence level ϑ, the inverse of the CDF, also called *Percent Point Function (PPF)* can be used,

$$F^{-1}\left(F(\mathbb{E}(X)) - \frac{\vartheta}{2}\right) \leq \mathbb{E}(X) \leq F^{-1}\left(F(\mathbb{E}(X)) + \frac{\vartheta}{2}\right). \tag{73}$$

In the case of PLFs, inverting the CDF is cheap and involves only the inversion of every linear component of the CDF. In Section 3.4, an example of confidence-rated outputs in JPTs is presented.

3.3.4 Learning and Reasoning in Symbolic Domains

Probability distributions over symbolic variables are represented by histograms over the domains of the respective variables. If a path from the root of the tree to a leaf node contains a decision node constraining a symbolic variable, it is superfluous to store prior distributions over the respective variables in the leaves of the respective subtree, the impurity (entropy) of the variable is minimal already. In order to compute the leaf-conditionals $P(q_i|\Lambda, e_i)$ in Equation 69 for all symbolic variables X_i, it is sufficient to eliminate the inadmissible values of X_i from the respective domain and re-normalize the histogram distribution.

3.4 Experiments

In recent years, various approaches for representing and learning probability distributions have been proposed, each of which is based on a specific set of constraints and assumptions (cf. Chapter 7). It is therefore difficult if not impossible to provide reasonable and robust evaluation criteria for comparing the predictive performance of a joint distribution learnt with different models.

Likelihood, on the one hand, is a well-known measure for a model's ability to fit a data set. However, computing the exact likelihood in most previous works strongly depends on the model assumptions made and the preprocessing of the data sets. As – to the best of our knowledge – JPTs are the first approach that allows for hybrid symbolic/ continuous model-free, joint probability distributions without prior assumptions, a direct comparison with previous works (Gens and Domingos 2013; Dang, Vergari, and Broeck 2020; Vergari et al. 2021) is difficult if not impossible.

Additionally, JPTs are evaluated by comparing their predictive performance with respect to each variable in an experiment to a discriminative approach that has been trained on the respective variable exclusively with the same representational complexity (e.g. the number of samples in a leaf node). This is a relatively hard setup, because the *single* JPT model is to compete with specialized discriminative models, which are allowed to tailor their representational resources to one particular variable.

Iris Data Set · In a first experiment JPT are compared against CART using the Iris dataset. In this machine learning benchmark problem, the goal is to learn to distinguish three different kinds of flowers. The data comes with four numerical features and one symbolic variable. As a model parameter, the minimal number of samples in a leaf node is set for both JPT and CART to 20% of the available data. The predictive results obtained in a 10-fold cross validation are shown in Table 1. Although the single JPT model needs to cover all five variables and one CART model was trained for each of the variables, JPT significantly outperforms CART in every variable.

MNIST Data Set · Another popular dataset for ML model evaluation is known as *MNIST* (LeCun and Cortes 2010). This dataset contains images of the digits 0 to 9, as written by various people. The images' dimensions are 8×8 and have grayscale values

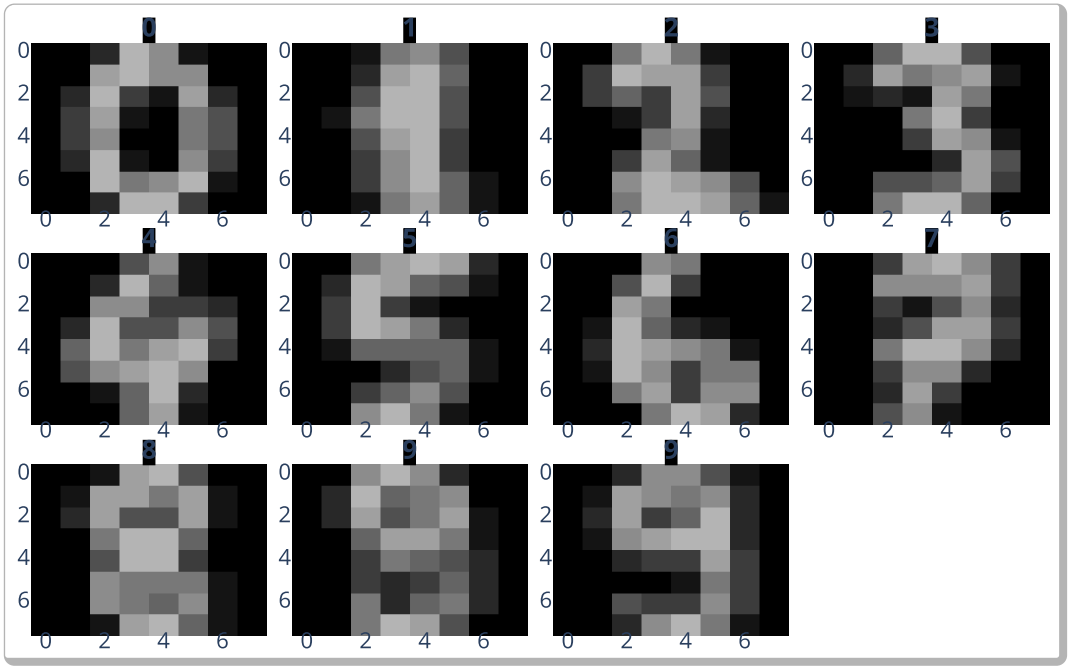
in the range $0 - 255$. Traditionally, the task in this dataset is to correctly assign one of the labels $0 - 9$ to every image in the collection, which is a discriminative problem statement. In this experiment, it is demonstrated that JPTs can perform both discriminative classification tasks and generative sample generation. The learnt JPT comprises 11 leaves, whose expectation over the 65 variables are shown in Figure 17. It can be seen that the expected value over the pixel values reasonably matches the expectation over the class labels. A visualization of the tree structure itself can be found in the appendix.

Iris data set: *top: Mean Absolute Error (MAE)* for the numeric variables in the Iris data set obtained from JPT (left) and CART (right). *Bottom:* F-score of the symbolic variable in the Iris data set | *Table 1*

Num. Variable	**JPT**	(+/-)	**CART**	(+/-)
sepal length (cm)	0.281207	0.0571313	0.342762	0.0414057
sepal width (cm)	0.22864	0.0500534	0.277941	0.0726099
petal length (cm)	0.237465	0.0764522	0.311952	0.0919128
petal width (cm)	0.147048	0.0501356	0.155645	0.0468443

Sym. Variable	**JPT**	(+/-)	**CART**	(+/-)
species	0.967213	0.0530674	0.959016	0.0676142

Expectations $\mathbb{E}(X_1), ..., \mathbb{E}(X_{64})$ over the probability distributions in the 11 leaf nodes of a learnt JPT (minimum 100 samples in each leaf). The clusters represented by the images in the leaf nodes reasonably match the associated class labels, which are displayed above the respective images. | *Figure 17*



Regression · In this experiment, it is demonstrated that JPTs can be used to accurately perform non-linear regression analyses, for which traditionally popular methods like *Ordinary Least Squares (OLS)*, RT, or neural network models are chosen. As ground

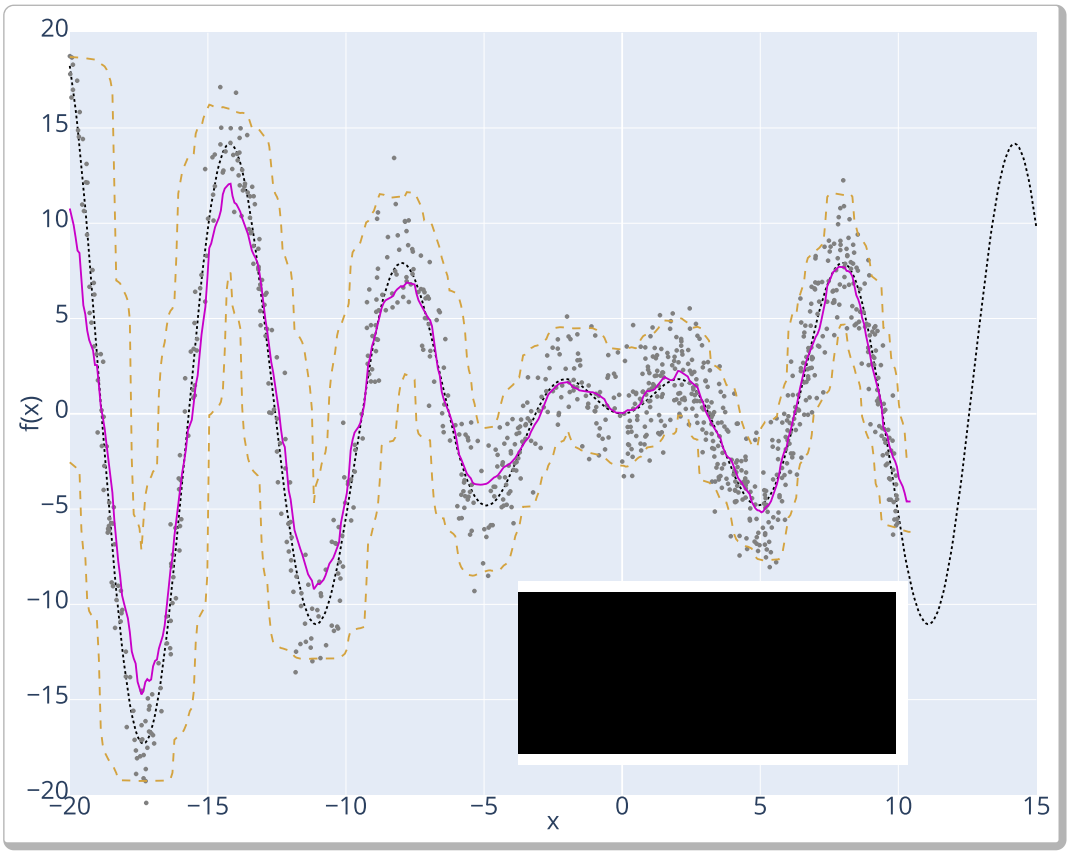
truth regressor, the function $f(x) = x \sin x$ is chosen, from which 1000 data points were sampled uniformly distributed with additive Gaussian noise. Regression analysis with joint distributions in JPTs can be achieved by evaluating the expected value of y given the value of x, $\mathbb{E}(y|x, \vartheta)$, as an estimate of $f(x)$, where ϑ is a confidence level that is used to calculate the confidence bounds of the answer as described in Section Section 3.3.3. Figure 18 shows the ground thruth, the sampled training data, the JPT prediciton as well as its upper and lower confidence bands. A quantitative comparison to CART is shown in Table 2.

MAE for predictions in the regression experiment using JPT and CART with different learning parameters. The column *#samples* contains the fraction of available data samples that need to be covered by every leaf node. | *Table 2*

#samples	JPT	CART
20.00%	3.0041032	4.821578
10.00%	2.1888970	4.0494846
5.00%	1.6213786	2.3802814
2.00%	1.2783260	1.3080680
1.00%	0.7854674	0.9564107

Regression fitting the function $f(x) = x \sin x$: The ground truth function $f(x)$ is shown in dotted gray, the training samples with Gaussian noise as gray dots, the prediction $\mathbb{E}(y \mid x, \vartheta = .95)$ given by the learnt JPT in purple, and the yellow dashed lines represent the upper and lower percentiles of the prediction. |

Figure 18



Airline Departure Delay · JPTs can efficiently be learnt even with very large datasets. As an example, the publicly available *Airlines Departure Delay Prediction* dataset provided by OpenML (Vanschoren et al. 2013) is used. This dataset is particularly of interest, as it comprises 10 million instances each of which consists of 7 mixed-type feature values (3 nominal, 4 numeric). There are no missing values in that dataset. The results in Table 3 show that even though the JPT model represents all variables as opposed to the specialized CART models each of which only represents one variable, the JPT performs comparably well on all of the 7 variables and even outperforms the CART models in three of them. This is remarkable, because it shows, that JPTs can compete with specialized discriminative models.

Empirical Evaluation · An extensive evaluation of JPTs was conducted on eight popular datasets from the UCI machine learning repository (Dua and Graff 2017), in which JPTs were learnt with different hyperparameters and determined the size of the resulting models as well as the likelihoods they achieve. As a hyperparameter for learning, the minimum number of samples was varied in each leaf node to different portions of the available training data to induce different levels of model complexity. The evaluation has been conducted on test sets created by randomly sampling 10% of the data. The experiments show that, with increasing complexity (i.e. decreasing minimum number of samples per leaf), the achieved likelihood of the learnt models reliably increases significantly both in the training set and in the test set. The detailed experimental results are listed in Table 15 in the appendix. The rightmost column contains the number of test samples with 0 likelihood. This typically happens in model-free distributions where test samples may lie outside the convex hull of the training data and the CDF-LEARN algorithm assigns 0 probability mass to these regions.

Experimental results on the Airline dataset: MAE and deviations of the numeric variables of JPT and CART (top) and F-score and deviation of the symbolic variables (bottom). | *Table 3*

Num. Variable	**JPT**	(+/-)	**CART**	(+/-)
DayOfWeek	1.89726	0.0118047	1.70767	0.00376937
CRSDepTime	144.518	0.615262	163.463	0.550986
Distance	293.35	2.82525	397.128	2.83425
CRSArrTime	139.678	0.825618	172.927	0.6609

Sym. Variable	**JPT**	(+/-)	**CART**	(+/-)
UniqueCarrier	0.178936	0.000313542	0.200474	0.000179531
Origin	0.0639007	3.28761e-05	0.0967244	1.47472e-05
Dest	0.0629808	1.9464e-05	0.0966721	1.41371e-05

3.5 Discussion

Joint probability distributions have great potential to serve as powerful problem-solving tools for machine learning applications. As opposed to most methods in the field of ML, joint distributions do not rely on dedicated input and output variables but can

be queried for any aspect contained in the model given any evidence. Most existing methods, however, require strong assumptions that all variables are either symbolic or numeric or that their distributions have the same functional form. This makes the accompanying methods tailored to the specific densities. However, in many real-world problems, the semantics of variables calls for support of heterogeneous modalities of both numeric and categorical variables in a single model. JPTs meet this requirement of hybrid symbolic and subsymbolic reasoning, which has also been identified as one of the key demands of future AI applications (Marcus and Davis 2019). A further significant advantage of JPTs over state-of-the-art PGMs is that JPTs do not rely on assumptions about dependencies or families of distributions at all. The variable dependencies are represented by the model structure that is generated as a byproduct of the learning procedure and the use of piecewise linear functions to approximate the CDFs of numeric variables allows highest flexibility with minimal model assumptions. The tree structure itself consists of conjunctions of variable constraints, each of which describes a specific subregion in the problem space. This makes the model structure interpretable and understandable for humans, and, as a consequence, its allows transparent reasoning and more robust, more traceable, and more explainable decision making (Goebel et al. 2018). The practical applicability of classic PGM is impeded by their representational and computational complexity that the models typically imply. The inferential complexity is $\#P$-complete in the general case (Koller and Friedman 2009) and learning is intractable since it involves inference. To circumvent these computational challenges, strong assumptions about variable independence and approximate inference mechanisms have to be adduced. In JPTs, inference is exact and can be performed in linear time with respect to the model size, i.e. the number of leaves in the tree. In addition, it is easy to limit the complexity of the model by, for instance, the computational resources that are available. This makes JPTs extremely flexible, scalable and parallelizable. JPTs are therefore shallow *deterministic* probablistic circuit, such that the circuit remains tractable for MPE inference.

◁ *JPTs can represent arbitrary shapes of distributions*

In summary, JPTs address a selection of properties of machine learning methods that have been identified as pivotal challenges of AI and ML in the literature of recent years. In this section, a selection thereof was reviewed and discussed in a qualitative manner, in which way JPTs are capable of addressing these desiderata. For a more detailed discussion of activities and methods in a typical lifecycle of ML models refer to the survey by Ashmore, Calinescu, and Paterson (2021).

chapter four

BayRoB - Bayesian Robotic Brain

This chapter addresses the probability-theoretic approach behind the BayRoB system and how belief states in autonomous agents can be represented as joint probability distributions. The general concept, incorporating JPTs as an efficient and powerful framework and the opportunities this approach provides, are outlined in the following. The presented concepts – along with their evaluation – constitute the key contributions of this work.

Chapter 1 has already mentioned various capabilities a robotic agent that is supposed to act in a realworld scenario has to have. In particular, a robot that is supposed to act autonomously is required to make decisions on its own, as it is not possible to know all possible scenarios the agent might face beforehand. It has to infer what objects are required to accomplish its task, where to find them and how to handle them which again requires additional knowledge about how the world and all the things in it *work*. This includes knowledge about physics (things fall to the ground when you let them go, liquids flow downwards, stacked objects might tumble), about everyday regularities, often referred to as *common knowledge*, (hot objects better be grasped by its handles if present, perishables are typically stored in cool places such as fridges) and about specifics of its immediate environment and the objects in it (the door of this fridge opens to the left, the cupboard contains 3 plates and 2 cups). Now with – hypothetically – all this knowledge present, an agent still has to make a reasonable decision in any situation possible. But what is reasonable? Russell and Norvig (2010) describe good agent behavior as *rational* behavior, that enables it to do the *right* thing. Rational behavior is of course not something that is easily determined, as it depends on how a successful outcome of a specific situation is defined and what knowledge, capabilities and percepts the agent has at the time of the task execution.

> *"For each possible percept sequence, a rational agent should select an action that is expected to maximize its performance measure, given the evidence provided by the percept sequence and whatever built-in knowledge the agent has."*
>
> *–Russell and Norvig (2010)*

This chapter is oriented towards this definition and the formal definition of rational (robot) agent models provided by Russell and Norvig (2010) and Beetz and Nyga (2023), to be equipped with the necessary terminology for the following conceptual framework. The focus will lie on states and state transitions as they are the fundamental concepts in terms of using joint probability distributions for making informed decisions.

4.1 A Rational Robotic Agent

In the context of rational robots, we perceive a robot agent designed to accomplish its goals by engaging with its surroundings. This agent acquires information about its environment via sensors and uses physical actions to bring about changes in the environment's state. The agent's behavior is guided by a function that translates sensory input and prior knowledge into actionable instructions for the robot. The decision-making processes of robots and the consequences of their actions on the environment are considered an ongoing interaction between the robot and its operational setting. An interaction cycle of the agent with its environment looks like the one in Figure 19:

An interaction cycle can be formalized as described by (Beetz and Nyga 2023):

A set of symbols is used to represent various elements: $\mathcal{O}$ (observations) for the robot's percepts, $\mathcal{A}$ for the available robot actions, and $\mathcal{X}$ for the states of the environment. The interaction between the robot agent and the environment are structured around three critical functions: the *perceptual filter*, the *state transition* and the *agent function*.

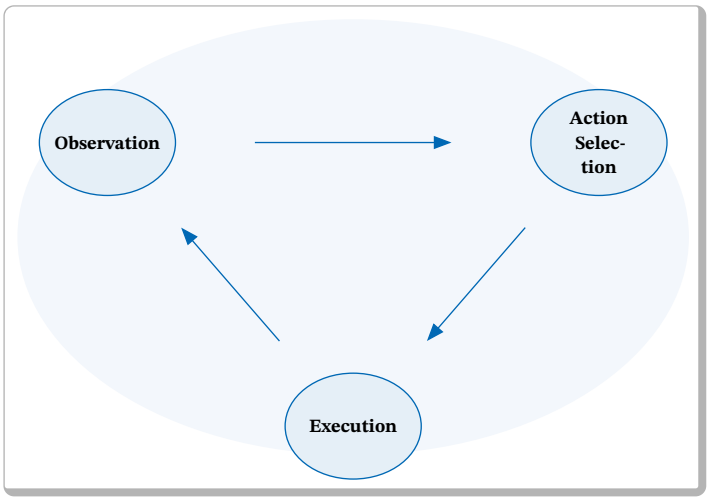
The *perceptual filter*, denoted as $f_p : \mathcal{X} \mapsto \mathcal{O}$, embodies the robot agent's perception capabilities. It takes the current state of the environment as input and maps it into the space of potential observations. In practical scenarios, the environment can typically only be observed partially, such that $O \neq X$. This occurs due to constraints on the robot's sensor range and limitations through presence of occlusions, and the fact that sensors only measure specific physical attributes of the world.

The *state transition* function, $f_e : \mathcal{X} \times \mathcal{A} \mapsto \mathcal{X}$, models how the robot agent's actions impact the environment. This function takes the environment state (s) and an action (a) executed in that state, and produces the resulting environment state (s').

Now, how does the world evolve as the robot carries out sequences of actions? Let us assume a clock generates a series of discrete timesteps, $T = \{1, 2, ...\}$, serving as time indices. Running the robot's perception-action loop generates a sequence of timesteps, each corresponding to an environment state. This sequence represents the progression

of the environment's states. This sequence is denoted here as the function $X^T : T \mapsto \mathcal{X}$, mapping each timestep to a specific environment state. Similarly, the robot agent's perceptual filter generates an observation trajectory, denoted as $O^T : T \mapsto \mathcal{O}$. Eventually, the agent function maps the information collected through observations up to a given timestep t into the action chosen for execution in the current cycle, $f : \mathcal{O}^t \mapsto \mathcal{A}$. Consequently, the agent function generates a sequence of actions that the robot agent will execute over time, $A^T : T \mapsto \mathcal{A}$.

Observation-Action Selection-Execution interaction cycle: in each time step or *iteration*, an agent 1) observes the current environment state, 2) determines the subsequent action to undertake and 3) executes the selected action with the intention of modifying the state of the environment. | *Figure 19*



The robot's interaction with its environment can therefore be denoted as a pair $\langle A, E \rangle$, where the environment E is represented as a tuple $\langle X, X_0, f_e, f_p \rangle$, and the robot A as a tuple $\langle \mathcal{O}, \mathcal{A}, f \rangle$, with the following conditions:

1. $X^T(0) = X_0$
2. $X^T(t+1) = f_e(X^T(t), A^T(t)), \forall t \in T$,
3. $O^T(t) = f_p(X^T(t)), \forall t \in T$,
4. $A^T(t) = f(O^t), \forall t \in T$.

In this representation, $\mathcal{X}$ denotes the set of potential environment states with an initial state X_0, $\mathcal{O}$ represents the set of observations and $\mathcal{A}$ represents the set of feasible actions. Given a specific agent A and an environment E, the state sequence generated by the agent's function f in E is referred to as the $effects(f, E)$.

The definitions above establish the formalization groundwork for the interaction between a robotic agent and its environment. However, something very important is missing so far: the *quality* of the produced action sequence. An agent is typically tasked with reaching certain goals, specifically certain changes in the environment, that are desired by the instructor. It is hence required to introduce a performance measure, indicating the quality or *utility* of a constructed action sequence: $U : \mathcal{X}^T \mapsto \mathbb{R}^+$, where $\mathcal{X}^T = \{X^T \mid X^T : T \mapsto \mathcal{X}\}$ stands for the set of all state sequences. It assigns a real-

valued score to every possible state sequence. This allows to evaluate the performance of the agent function in an environment E by defining $V_U(f, E) = U(effects(f, E))$.

In most cases, f_p and f_e are non-deterministic, as the real-world scenarios are typically prone to uncertainty, which makes it impossible to fully observe the actual state of the environment, rendering the prediction of certain action effects infeasible. A possible solution for this limitation is to use a distribution over the possible environments instead. The performance would then be the expected utility of the environment produced by f:

$$V_U(f, \mathcal{E}) = \sum_{E \in \mathcal{E}} P(E) \cdot U(effects(f, E)) \tag{74}$$

with $P(E)$ denoting the probability of the world being in state E.

Given the utility function U, an objective metric for evaluating an agent's decision quality regarding desired task outcomes is provided. This allows the comparison of the effectiveness of different agent functions and also helps to establish the concept of rationality. With $V_U(f, E)$ representing the expected outcome achieved by an agent function f within a set of environments $\mathcal{E}$ as per the utility function U, an agent is said to be ideal and rational if, for any given sequence of sensory inputs, it selects the action that it expects to maximize its performance measure with, specifically, its agent function f^* is defined as

$$f^* = \arg\max_f V_U(f, E). \tag{75}$$

A robotic agent in the context of this work should not simply react to its sensory input like a *reflex agent* (Russell and Norvig 2010) would, but take into account certain knowledge it has about itself, the world in general and the objects in it. Only taking current percepts into account when making decisions would not suffice, for various reasons. Objects that are not visible through occlusion once the agent moves, would not be included in its visual observations anymore and are therefore practically nonexistent to it despite being present and available. The agent should *remember* where it has seen objects and what changes it has caused in its environment so it can use that knowledge later, if required. Beetz and Nyga (2023) call a robot agent *knowledge-enabled* if it is equipped with symbolic representations of the internal robot agent model and can infer action sequences that are predicted to achieve a given goal. This necessitates maintaining an internal *world model*, capturing all spatial relations between objects as well as their states, such as electronic devices being turned on or off, doors and drawers being open or closed and so on. The representation of this model is key for an agent making the *right* decisions.

knowledge-enabled robots ▷

4.2 Running Example

A simple household scenario will serve as a running example to illustrate the concepts introduced so far. Consider a simple rectangular floorplan with the limits ("walls") of $[0, 100]$ in both x and y direction, serving as the kitchen world a robot agent can move

in, where the (x, y)-coordinates on the kitchen floor represents the robots' belief of its own base position (x_{in}, y_{in}). The agent can move to any position within the kitchen's boundaries that is not obstructed by an obstacle. The kitchen contains a number of furniture items such as a `kitchen_unit`, a `fridge`, a `stove`, a `kitchen_island` and two `chairs`. The kitchen unit contains two kitchen cabinets with doors to store different kinds of items. Kitchenware such as cups and bowls are stored in the left cabinet, non-perishable food items like cereal in the right one. A drawer below the left cabinet contains cutlery. The fridge can be used to store perishable items as well as items that are best served cold such as milk and beer. The stove on the right of the kitchen floor plan can be used to cook and has another small cabinet for pots and pans. The three-piece suite in the center of the kitchen is composed of the kitchen island, serving as a table, and two chairs. Each of these furniture items are considered immovable and therefore pose obstacles for the robot, that it needs to navigate around. The kitchen floor plan is sketched in Figure 20.

The agent in our kitchen scenario has beliefs about itself and its environment. It can track its own position, detect furniture items and objects, if it is close enough and its view is not obstructed and it recognizes a collision with an obstacle or wall. It can execute different actions, some of which have already been introduced before: The agent can a) move a given `distance` into its current facing direction using the action `move_base`, b) it can turn around a given `angle` using `turn` and it has c) `perception` capabilities that will be described in the following.

Overall, the state of the agent and its world is represented by the variables in Table 4.

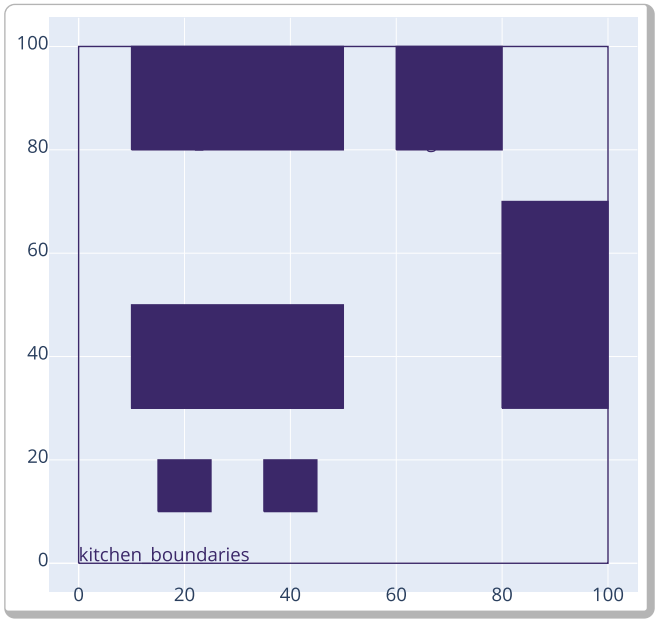
The variables of the running example | *Table 4*

Variable	**Type**	**Domain**
X_{in}	numeric	$[0, 100] \subset \mathbb{R}$
Y_{in}	numeric	$[0, 100] \subset \mathbb{R}$
Δ_{pos_x}	numeric	$[-1, 1] \subset \mathbb{R}$
Δ_{pos_y}	numeric	$[-1, 1] \subset \mathbb{R}$
$Collided$	boolean	$\{\top, \bot\}$
$XDir_{in}$	numeric	$[0, 100] \subset \mathbb{R}$
$YDir_{in}$	numeric	$[0, 100] \subset \mathbb{R}$
$Angle$	numeric	$[-45, 45] \subset \mathbb{R}$
Δ_{dir_x}	numeric	$[-1, 1] \subset \mathbb{R}$
Δ_{dir_y}	numeric	$[-1, 1] \subset \mathbb{R}$
$Daytime$	multinomial	{morning, post-breakfast, pre-lunchtime, lunchtime, post-lunchtime, pre-dinnertime, dinnertime, post-dinnertime, night}
$Open(\langle\text{container}\rangle)$, container $\in$ {fridge_door, cupboard_door_l, cupboard_door_r, kitchen_unit_drawer, stove_door}	boolean	$\{\top, \bot\}$

$Detected(\langle \text{object} \rangle)$, object $\in$ {cup, cutlery, bowl, sink, milk, beer, cereal, stovetop, pot}	boolean	$\{\top, \bot\}$
$Nearest\_Furniture$	multinomial	{kitchen_unit, fridge, kitchen_island, chair1, chair2, stove}

The continuous variables X_{in} and Y_{in} as well as $XDir_{in}$ and $YDir_{in}$ represent the agents' position in the kitchen and its facing direction, respectively. The variables indexed with *out* instead of *in* are the outcomes of the respective action executions and will be discussed in more detail in the descriptions of the separate datasets below. $Daytime$ is a multinomial variable and can take 9 different representing the discrete times of day a kitchen is typically more or less heavily-frequented. Multiple variables are used to represent the boolean state of the storage places mentioned above, i.e. whether the cutlery drawer, doors of the cabinets and fridge are open or closed and whether a certain object is perceived. There exist separate boolean variables for each possible value of container (for the variable $Open(\langle \text{container} \rangle)$) and object (for $Detected(\langle \text{object} \rangle)$), which can be taken from Table 4.

Overview of the kitchen world | *Figure 20*



Note that the state of the detected variable for certain objects may be linked to the state of the open variable for others, since items may not be visible by the agent if they are in a closed cabinet or drawer. In the remainder of the document the notation $Detected(\langle \text{object} \rangle)$ and $Open(\langle \text{container} \rangle)$ will be treated as a placeholder for listing all assignments of container and obj, for ease of readability. The discrete variable $Nearest\_Furniture$ is an additional piece of semantic information that gives some indication of the agent's (symbolic) belief about its location. Its values contain the labels of the 6 furniture items in the kitchen world. Lastly, the boolean variable $Collided$

represents whether the agent has collided with either an obstacle or the kitchen walls when executing the most recent `move_forward` action. In the case of a collision, its position remains the same, i.e. the action execution is considered unsuccessful.

Design Decisions

Spurious Correlations · Models trained with robot experience data sometimes represent aspects of the world that a human would interpret as false. This is not necessarily a problem of the model but of the data it has been trained with. Datasets automatically generated from robotic environments' sensors typically include robot poses, perception data, and other sensor outputs for only a limited number of experiments. Using the example of a kitchen, the robot may not have explored most parts of the kitchen, resulting in data being concentrated along certain paths connecting frequently visited furniture, like the fridge and the kitchen island, or the kitchen unit and the kitchen island.

This type of data often includes numerous samples representing positive dependencies between variables. For instance, it may indicate that the fridge door must be open for the robot to view or retrieve items from it. However, there is a risk of learning false dependencies, such as "all other doors and drawers must be closed to see the milk" or "the cup can only be detected if the cutlery is *not* detected," simply because there are no examples in the dataset proving otherwise. While there may be correlations between the occurrences of events, there is not necessarily a causal relationship between them. The observed effect is also related to the frame problem, which arises from the fact that merely specifying which conditions are altered by actions does not guarantee that all other conditions remain unchanged.

◁ *causality vs. correlation*

◁ *frame problem*

To avoid learning these false dependencies, specific design decisions must be made to ensure that the resulting model accurately reflects the intended representation of the world. One could for example create separate models for each furniture item such that the state of the other objects and items is not known to the model in the first place. One could also try to generate samples for all possible variable combinations that could theoretically occur in real situations, which might be computationally infeasible, recalling the computation of the count of potential worlds as discussed in the introduction. If the learning algorithm allows it, one could manually define (in-)dependencies between variables, introducing external knowledge not present in the data. Which strategy to choose is highly dependent on the current problem to solve, the available data and is typically subject to experimentation. It is hard to determine what are relevant dependencies between variables and finding a solution to determine them automatically is not solved within the context of this work. In our kitchen scenario, as a pragmatic solution multiple strategies are deployed which are based on the background knowledge about the nature of the experiment and therefore, the data.

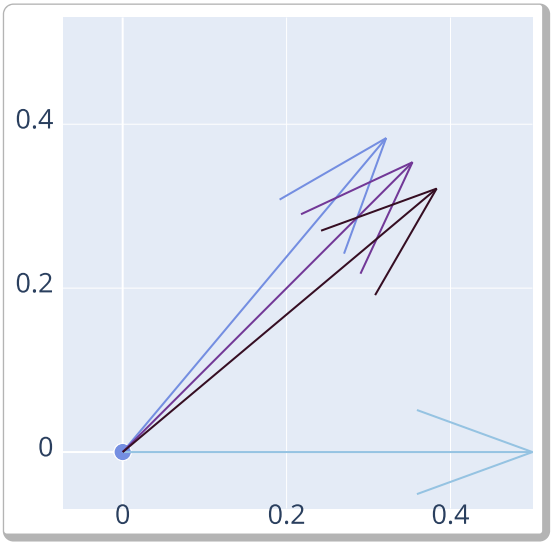
General vs. Specialized Models · It is desirable to generate a model that allows an agent to predict the outcome of its actions across various environments. Unfortunately, more generalized models typically come with higher uncertainty which can be retained to a certain extent by incorporating more datapoints covering diverse situations an agent might encounter – of course at the expense of a heightened model complexity. Generating the optimal model for a task therefore involves striking a balance among

model parameters, quantity and quality of data points and desired generalization capabilities.

In this context, knowing the absolute positions of certain furniture items and objects is crucial for the agent. Consequently, incorporating absolute positions into the model becomes necessary, even though it complicates the model's transferability to different environments. One step towards a more generalized model, however, involves storing starting positions as absolute coordinates and representing directions as normalized vectors, while the outcomes of the `move_base` action (x_{out}, y_{out}), as well as the results of a `turn` $(xdir_{out}, ydir_{out})$ are represented as *delta* values – the *difference* between the agent's updated position and its previous location, or the new facing direction after a turn and the previous one, respectively. This allows to generalize over the outcomes of unobstructed movements but still represent collision-prone locations.

Direction update for generating `turn` data points given an initial direction vector of $(1, 0)$ and the `turn` parameter angle $= -45°$. For simulating uncertainties in robot movements due to inaccuracies in the actuators, some noise is added to the `angle` factor, i.e. the direction will not be exactly, but $\approx (0.7, 0.7)$ |

Figure 21



`turn` Dataset

The data for the `turn` model contains data points representing the agent's rotation around different angles. The turn action of the agent will only update its facing direction, its position is left untouched. In particular, a rotation without a move forward will never trigger a collision, even if the robot's position is right at the edge of an obstacle or wall. Since the facing direction is position-independent, the dataset contains only vectors with values of the variables

$$XDir_{in} \times YDir_{in} \times Angle \times \Delta_{dir_x} \times \Delta_{dir_y} \tag{76}$$

as data points, i.e. the (absolute) facing direction *before* executing the `turn` action, the action parameter $angle$ and the delta to the facing direction *after* the update.

The agent should be able to know, from an arbitrary facing direction, what its new facing direction would be after rotating around a given angle. This would require to generate samples for any combination of facing direction and angle. To approximate this, the data for the `turn` model is generated by uniformly sampling initial directions $(XDir'_{in}, YDir'_{in})$ along with (also uniformly sampled) angles between $[-45, 45]$ degrees, each of which then represents one data point. The range of $[-45, 45]$ is here set to be the range the agent might be able to turn in a single action. A wider angle can be reached by carrying out multiple turn actions consecutively. Some Gaussian noise ($\sigma = 0.01$) is added to the input angle to account for inaccuracies in the robots motor activity. The new facing direction is then calculated by multiplying $(XDir'_{in}, YDir'_{in})$ with the rotation matrix:

$$\begin{aligned} \begin{bmatrix} \Delta_{dir'_x} \\ \Delta_{dir'_y} \end{bmatrix} &= \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} XDir'_{in} \\ YDir'_{in} \end{bmatrix} \\ &= \begin{bmatrix} XDir'_{in} \cdot \cos(\theta) - YDir'_{in} \cdot \sin(\theta) \\ XDir'_{in} \cdot \sin(\theta) + YDir'_{in} \cdot \cos(\theta) \end{bmatrix} \end{aligned} \tag{77}$$

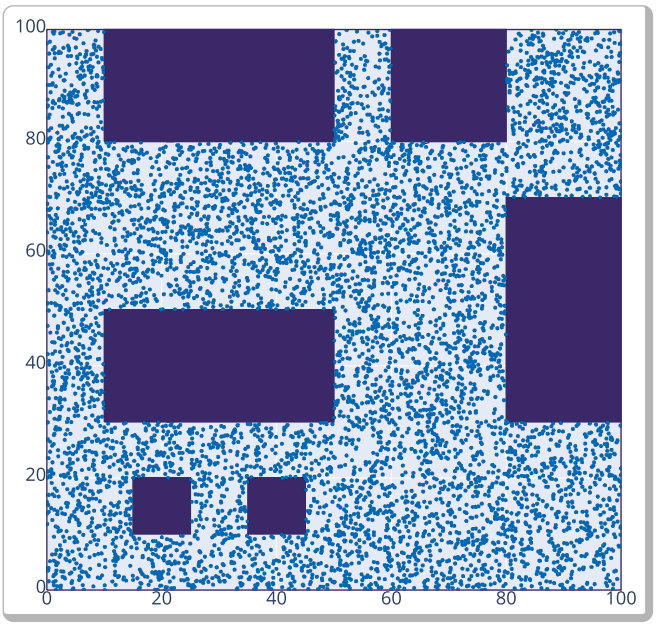
with θ being the (radian) angle to rotate.

As mentioned before, a data point consists of the initial facing direction, the angle as the action parameter and the Δ_{dir} as the difference between the new facing direction and the initial facing direction, i.e.

$$\Delta_{\text{dir}} = \begin{bmatrix} \Delta_{dir_x} \\ \Delta_{dir_y} \end{bmatrix} = \begin{bmatrix} \Delta_{dir'_x} \\ \Delta_{dir'_y} \end{bmatrix} - \begin{bmatrix} XDir_{in} \\ YDir_{in} \end{bmatrix} \tag{78}$$

The dataset contains 35,000 data points.

The ground truth data for the kitchen scenario | *Figure 22*



`move_base` Dataset

The process of generating the data points employs a systematic approach to capture the diversity of spatial scenarios. The data for the `move_base` model is shown in Figure 22 and is used to train a model that can predict, for any obstacle-free position on the floor plan, and preferably for as many facing directions as possible, where the the agent will be located after making one step forward. It is therefore required to sample fine-meshed positions over the entire kitchen space. For each of those positions, the robot might face in different directions and for each of those, the robot is supposed to have some idea what lies ahead of it. First, 10.000 positions (X_{in}, Y_{in}) uniformly distributed across the entirety of the kitchen area (i.e. the 100×100 floor plan) are sampled. Next, an additional layer of variation is introduced by randomly sampling 360 facing directions for each of these positions. These vectors allow us to explore the full range of orientations from each position. The new positions are computed by propelling forward from the original sampled position into the respective direction. To emulate the inherent uncertainties in real-world scenarios, a parameterized distance (or *step size*) is incorporated. Furthermore, to account for potential motor inaccuracies or uncertainties, a touch of randomness is introduced by adding Gaussian noise to the distance. This meticulous process ensures that our generated data points not only cover the spatial area comprehensively but also encapsulate the unpredictabilities encountered in practical settings.

Each datapoint consists of a vector of 7 values

$$X_{in} \times Y_{in} \times XDir_{in} \times YDir_{in} \times \Delta_{pos_x} \times \Delta_{pos_y} \times Collided \tag{79}$$

i.e. the position of the agent *before* executing the move action, its current facing direction, its delta to the new position *after* the action execution, and its detection whether a collision has occurred. From the position and direction, the rotations are calculated using Equation 77. From each of the position-direction combinations a data point is generated by determining the new position through the calculation

$$\begin{bmatrix} \Delta_{pos'_x} \\ \Delta_{pos'_y} \end{bmatrix} = \begin{bmatrix} X_{in} + XDir_{in} \cdot \text{distance} \\ Y_{in} + YDir_{in} \cdot \text{distance} \end{bmatrix} \tag{80}$$

with $\text{distance} \approx 1$, see Figure 23.

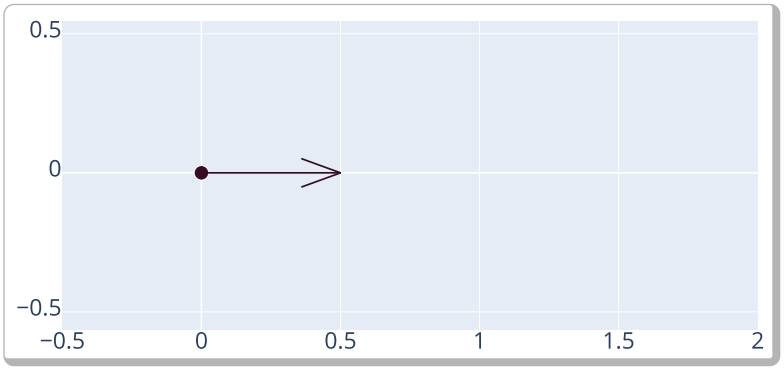
Analogously to the Δ_{dir} in the `turn` dataset, the position Δ_{pos} is determined by subtracting the starting position from the position after the `move_base` action:

$$\Delta_{\text{pos}} = \begin{bmatrix} x_{out} \\ y_{out} \end{bmatrix} = \begin{bmatrix} x'_{out} \\ y'_{out} \end{bmatrix} - \begin{bmatrix} x_{in} \\ y_{in} \end{bmatrix} \tag{81}$$

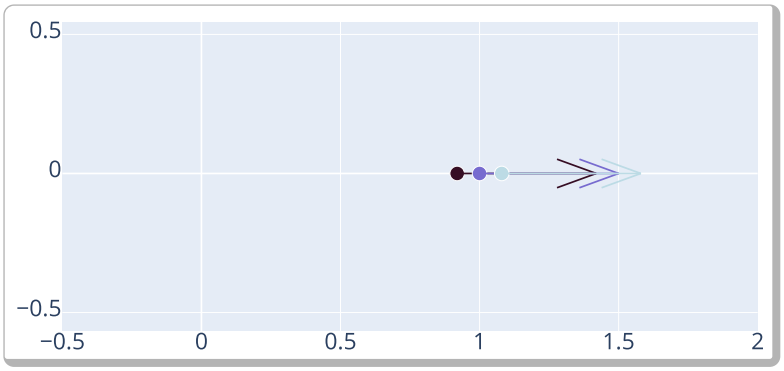
Now all vector values for the `move_base` data point representation (79) have been defined an the data point is generated. Following this strategy, the dataset contains $100 * 100 * 360 = 3{,}600{,}000$ data points. Note that for all initial positions that either lie close to a wall with a facing direction towards the wall or within an obstacle, a collision occurs. In that case, the deltas are represented by 0 mean Gaussian noise to account for minimal position updates caused by vibrations as an effect of the collision. Figure 22 shows a plot of the ground truth data used for training the `move_base` model.

Position update for generating `move_base` data points given an initial position $(0, 0)$ and distance $= 1$. Similarly to the `angle` parameter in `turn`, noise is added to the `distance` factor, i.e. the position after the update will be $\approx (1, 0)$ | *Figure 23*

Position: $(0, 0)$, Direction: $(1, 0)$ | *a)*



Position: $\approx (1, 0)$, Direction: $(1, 0)$ | *b)*



`perception` Dataset

The dataset is generated by sampling positions around the obstacles in the kitchen and collecting information about the perceptions of the agent, in particular, the state of the environment, e.g. what objects are visible from the current position, which containers are open or closed and what time of day it is. The `perception` model joins all these information in a joint distribution over the variables:

$$\begin{aligned} & X_{in} \\ \times\ & Y_{in} \\ \times\ & XDir_{in} \\ \times\ & YDir_{in} \\ \times\ & Daytime \\ \times\ & Open(\langle \text{container} \rangle) \\ \times\ & Detected(\langle \text{object} \rangle) \\ \times\ & Nearest\_Furniture \end{aligned} \tag{82}$$

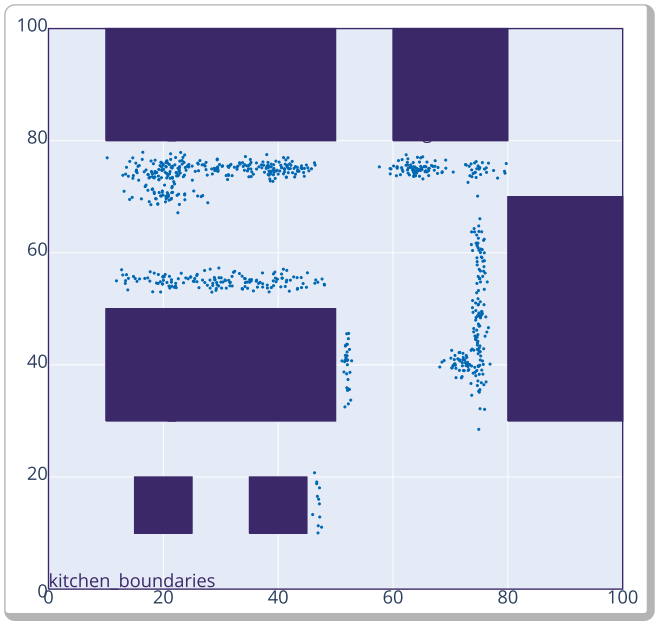
In the sampling procedure for this action model, two distinct sets of samples are gathered, each contributing unique insights to the data collection. In both sets, the variable

Daytime is sampled uniformly and the variable *Nearest_Furniture* is set to the respective furniture around which the position was sampled.

In the first set, positions near furniture items are strategically sampled, specifically situating the robot in scenarios where it is facing the furniture item and can overlook its surface. To simulate real-world conditions, doors and drawers remain predominantly closed, limiting the robot's vision to the exterior. However, in a select few instances, doors or drawers may be opened only when the agent is in close proximity. This set of samples primarily captures the robot's perspective of obstacles and potential objects in direct line of sight.

Conversely, the second set employs a different strategy and focuses on the visibility of objects stored in cabinets and drawers, with samples drawn from Gaussian distributions around specific locations. These locations are intentionally varied from those in the first set and are constrained to those from where the interior of cabinets or drawers can be visually accessed by the robot. Depending on the time of day and the robot's position, items within the furniture can be detected in these samples. The key distinction lies therefore in the proportion of the state of the $Open(\langle \text{container} \rangle)$ variable, which is more frequently set to $\top$ in the second set. The respective outliers in the two sets, however, soften the bias in terms of false dependencies mentioned earlier of the learnt model.

The ground data for the `perception` model contains data points with positions sampled around the obstacles, plus additional data sampled from areas in front of the furniture containing containers. Number 1 marks the positions from which an agent can open the drawer of the kitchen unit and can see its contents. Numbers 2 and 3 mark the positions for the left and right cabinets of the kitchen unit, respectively. Number 4 and 5 mark the positions for opening the doors of the fridge and the stove. | *Figure 24*



Depending on the value of the time of day, some objects may be located at different positions, e.g. on the kitchen island (when the table is set for breakfast). They are there-

fore only visible to the robot when certain criteria are met. Below are some of the rules representing the value of the variable $Detected(\langle \text{object} \rangle)$:

$$
\begin{aligned}
Detected(\langle \text{object} \rangle) \Leftrightarrow\ & \text{object} \in \{\text{cup}, \text{cutlery}, \text{bowl}, \text{milk}, \text{cereal}\} \\
& \quad \wedge\ Nearest\_Furniture = \text{kitchen\_island} \\
& \quad \wedge\ Daytime = \text{morning} \\
& \vee\ \text{object} = \text{milk} \\
& \quad \wedge\ Nearest\_Furniture = \text{fridge} \\
& \quad \wedge\ Open(\text{fridge\_door}) = \top \\
& \quad \wedge\ Daytime \neq \text{morning} \\
& \vee\ \text{object} \in \{\text{cup}, \text{bowl}\} \\
& \quad \wedge\ Nearest\_Furniture = \text{kitchen\_unit} \\
& \quad \wedge\ Open(\text{cupboard\_door\_left}) = \top \\
& \quad \wedge\ Daytime \neq \text{morning} \\
& \qquad \vdots \\
& \vee\ \ldots \\
& \qquad \vdots \\
& \vee\ \text{object} = \text{sink} \\
& \quad \wedge\ Nearest\_Furniture = \text{kitchen\_unit} \\
& \vee\ \text{object} = \text{beer} \\
& \quad \wedge\ (Nearest\_Furniture = \text{kitchen\_island} \\
& \quad \wedge\ Daytime = \text{night} \\
& \quad \vee\ Nearest\_Furniture = \text{stove}) \\
& \vee\ \text{object} = \text{stove\_top} \\
& \quad \wedge\ Nearest\_Furniture = \text{stove} \\
& \vee\ \text{object} = \text{pot} \\
& \quad \wedge\ Nearest\_Furniture = \text{stove} \\
& \quad \wedge\ Daytime \in \{\text{pre-lunchtime}, \text{pre-dinnertime}\} \\
& \vee\ \text{object} = \text{pot} \\
& \quad \wedge\ Nearest\_Furniture = \text{kitchen\_island} \\
& \quad \wedge\ Daytime \in \{\text{lunchtime}, \text{dinnertime}\} \\
& \vee\ \text{object} = \text{pot} \\
& \quad \wedge\ Nearest\_Furniture = \text{kitchen\_unit} \\
& \quad \wedge\ Daytime \in \{\text{post-lunchtime}, \text{post-dinnertime}\}
\end{aligned}
\tag{83}
$$

The two sets of samples are aggregated eventually to form the full training data for the `perception` model. The generated dataset then contains about 772 examples and can be seen in Figure 24, where the positions from which furniture doors and drawers can be reached are numbered.

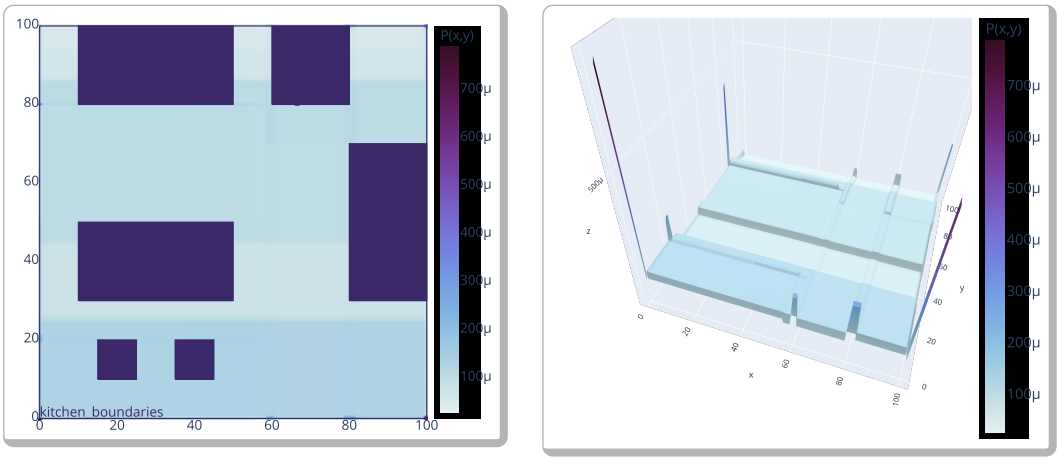
Having introduced the simplified world an exemplary agent can act in, one can now consider probabilistic hybrid action models as means to make probabilistic statements about how an action execution impacts the agent's belief state.

4.3 Robotic Belief States as Joint Distributions

As terms like *world model*, *state* or *belief* may have different meanings (at least regarding the practical implementation) dependent on the context they occur in, this section will briefly explain how they should be interpreted in the context of this work. A *world model* in the context of robotic agents typically refers to a representation that the agent constructs to understand and make predictions about its environment. It compasses various aspects of the robot's perception and knowledge about the world, including the state of the environment, objects, entities, and their interactions.

The initial distribution $P(X_{in}, Y_{in})$ as heatmap and 3D surface views. The heatmap shows some light patches around areas with furnitures, indicating low-probability positions. The 3D-view highlights the difference between low and high probability positions | *Figure 25*

The heatmap | *a)* The 3D-view | *b)*



An integral element within a world model is what is often referred to as the *belief state*. Belief states are often represented as sets of statements a robotic agent believes to be true about the surrounding world and serve as a probabilistic or uncertain portrayal of the robot's existing comprehension of its environment. It integrates data obtained from a range of sensors, prior knowledge, and observations to gauge the current condition of the world. This belief state generally encompasses details about object locations and attributes, the robot's own state, as well as any pertinent information essential for decision-making purposes. The belief state remains in a constant state of transition, capable of being modified as the robot accumulates new sensory information and carries out actions. It plays a significant role in the decision-making process, as the robot leverages its belief state to reason about the prospected outcomes of various actions and ultimately make informed decisions grounded in its understanding of the world. Belief states can be modeled as probability distributions to account for the agent's uncertainty about the current state of the environment.

When examining certain aspects of the world, it is natural to find them in either symbolic or subsymbolic forms. A useful world model should closely reflect the natural representation of these aspects. For instance, when deciding whether to employ the left or right gripper of a robot for a specific task, it is logical to depict the grippers as categorical values of a symbolic variable, just as with object categories. The success or failure of an action can, arguably, be represented as a boolean variable, unless it serves as a more elaborate performance metric rather than a simple yes/no decision, in which case a subsymbolic representation might suffice. Parameters like robot and object positions, container fill levels, and temperatures often span continuous domains and are best represented using subsymbolic variables. In non-(or semi-)hybrid systems, continuous variables are at times discretized for ease of handling. Additionally, simplifying assumptions are frequently adopted, often assuming that features follow Gaussian distributions when more complex and differently shaped distributions cannot be managed without appropriate tools. In rare cases categorical values are assigned to ranges of numeric values to allow interpreting them in subsymbolic domains. Interpreting symbolic values as numbers, however, is semantically meaningful only if there exists some defined order among the values of the symbolic variable, while discretizing a continuous variable may overly simplify the domain. Converting variables from one domain into another or introducing simplifying assumptions is typically a pragmatic choice driven by the absence of systems that are capable of handling truly hybrid domains. Nevertheless, such choices often lead to a decrease in accuracy, as these changes may not reflect the real-world conditions. There is a genuine need for hybrid models that can represent both symbolic and subsymbolic variables to mirror these conditions appropriately.

4.4 Action Models as Joint Distributions

Chapter 3 has already outlined the advantages and capabilities of using JPTs such that it is now straightforward to employ them to effectively represent an agent's belief state, as we have established a method for compactly and computably representing joint probability distributions over the environment, actions and the agent itself. The belief state in BAYROB is represented as a joint distribution over all possible non-delta and non-parameter variables:

$$\begin{aligned} &P(X_{in}, Y_{in}, XDir_{in}, YDir_{in}, Collided, Daytime, \\ &Open(\langle\text{container}\rangle), Detected(\langle\text{object}\rangle), Nearest\_Furniture), \end{aligned} \tag{84}$$

i.e. the agent's state is defined by its position and orientation in the kitchen environment as well as the state of the furniture elements and objects in the kitchen. The `move_base`, `turn` and `perception` actions can alter subsets of these variables and therefore influence the agent's state. The `move_base` action model is represented as a joint distribution

$$P\Big(\Delta_{pos_x}, \Delta_{pos_y}, Collided \mid X_{in}, Y_{in}, XDir_{in}, YDir_{in}\Big), \tag{85}$$

the `turn` action model as

$$P\Big(\Delta_{dir_x}, \Delta_{dir_y} \mid XDir_{in}, YDir_{in}, Angle\Big) \tag{86}$$

and the `perception` model as

$$\begin{aligned} P(X_{in}, Y_{in}, XDir_{in}, YDir_{in}, Daytime, Open(\langle \text{container} \rangle), \\ Detected(\langle \text{object} \rangle), Nearest\_Furniture). \end{aligned} \tag{87}$$

The learnt distribution over the (absolute) positions for the `move_base` action in Figure 25 a) and b) gives an idea of the kitchen floorplan with some of the non-walkable areas (e.g. at the top edge) being lighter colored due to missing information about (x_{in}, y_{in}) positions, since most part of that area is covered by obstacles (kitchen unit and fridge).

The `perception` model allows to ask complex things like "If I detected a milk container and it is currently early in the morning, where will I most likely be located?". Figures 28 a) to b) show that given the information that the agent has detected milk, its belief about its current position is updated dependent on the time of day. While it is most likely to come across a milk container on the kitchen island in the morning, where the breakfast table is typically set, it is not likely to find it there during night time. At night, it is much more likely to find it stored in the fridge to keep it cool. This model links the inherent robot state such as its position and facing direction with the world, i.e. it stores information about the agent's spatial relation to (visible) objects and furniture as well as environment conditions (time of day).

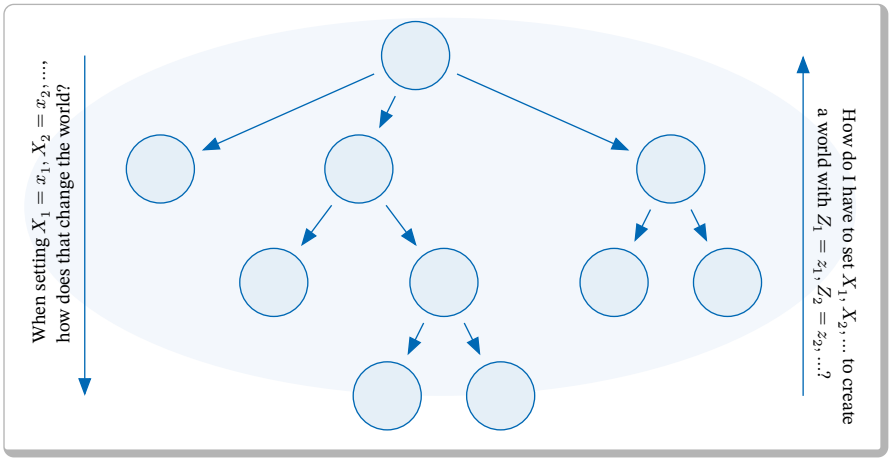
4.4.1 Action Intelligence in BAYROB

When learning models to represent some aspects of the world in the context of machine learning, one typically thinks in terms of *query* and *evidence*, i.e. of variables that are parameters to be set and variables that determine an outcome depending on these parameters. When performing inference tasks on a trained model, one is typically interested in asking two different questions as pictured in Figure 26. The *forward* question to the model is to anticipate an outcome of a certain parameter setting. In our case of action models this would correlate to asking questions such as: "If I determined my current position to be at (x, y) facing right and I moved forward 3 steps, at which position will I most likely end up?"

The parameter variables are *applied* to the model to determine the value of the query variables. However, in many cases the *backward* question is much more interesting, i.e. asking for a state of the world, that led to the current situation, or in our setting: "If I am currently at (x, y) facing right and I *want* to end up at a position, where I can detect (and therefore possibly grab) milk, what do I have to do?" This inquiry revolves around generating a specific outcome in the world, and the objective is to identify an appropriate parameter configuration to achieve that desired result. Essentially, this task can be viewed as the reverse application of a model. When constructing such models, certain considerations come into play. Firstly, not all inference algorithms may facilitate reverse inference. Given a function $f(x)$, it is not always mathematically feasible to have a corresponding function $F := f^{-1}$. Secondly, it is essential to predefine which variables are designated as evidence or parameters and which ones are considered

query variables, while the primary interest lies in the capability to query the model with any desired question. Whether it involves observing an entity like milk or engaging in a specific activity, such as being at or reaching a particular location, every variable in the model should function as both a query and a evidence variable simultaneously, depending on the context.

The forward-backward queries in JPTs: Querying in the forward direction of a JPT (`apply`), returns the predicted outcome of a certain parameter setting, while the backward (`reverse`) direction returns a set of parameter-value mappings that most likely produce a certain outcome, when applied. | *Figure 26*



Action models and belief states in BAYROB are joint distributions incorporating JPTs, which do exactly that: allowing to formulate any kind of question formulated as a query in terms of an arbitrary variable combination given another arbitrary variable-value assignment as evidence. A *conditional* JPT is generated by the pseudoized algorithm depicted in Algorithm 3.

The CONDITIONAL-JPT algorithm takes a variable assignment e and a JPT t as inputs. Probabilistically, it computes the conditional probability $P(x|e)$, where x represents the variables in t that are not part of e. If e cannot be satisfied by t, the algorithm returns an error. The algorithm iterates over a list of nodes, which initially only contains the root node of (the copy of) t. If the current node is a decision node, it is removed if it does not have any children (e.g. due to pruning in an earlier iteration) or its two children are inserted into the node list to be processed in later iterations. If the node contains only one child node, the algorithm continues to avoid processing them multiple times. If the current node is a leaf node, the probability of the node given the evidence is calculated. If the probability is 0, the node is inconsistent with the evidence and therefore removed from the tree, otherwise the priors are recalculated and the algorithm moves on to the next iteration. After pruning all inconsistent nodes and updating the prior distributions, the probability mass of the entire tree is computed. If the calculation fails, the evidence is considered unsatisfiable by the tree, resulting in an error. Otherwise, the overall distributions of the tree are determined and the tree is returned, projecting only the nodes consistent with the evidence.

◁ *conditional JPT*

The algorithm incorporates the execution of a number of functions that are not outlined as pseudo-code, but will be briefly explained in the following.

CONDITIONAL-JPT: The algorithm to apply evidence on a JPT and get a new JPT that represents $P(x|evidence)$. | *Algorithm 3*

CONDITIONAL-JPT(e, t):

Input: e, a variable assignment, mapping variables to their observed values
t, a JPT

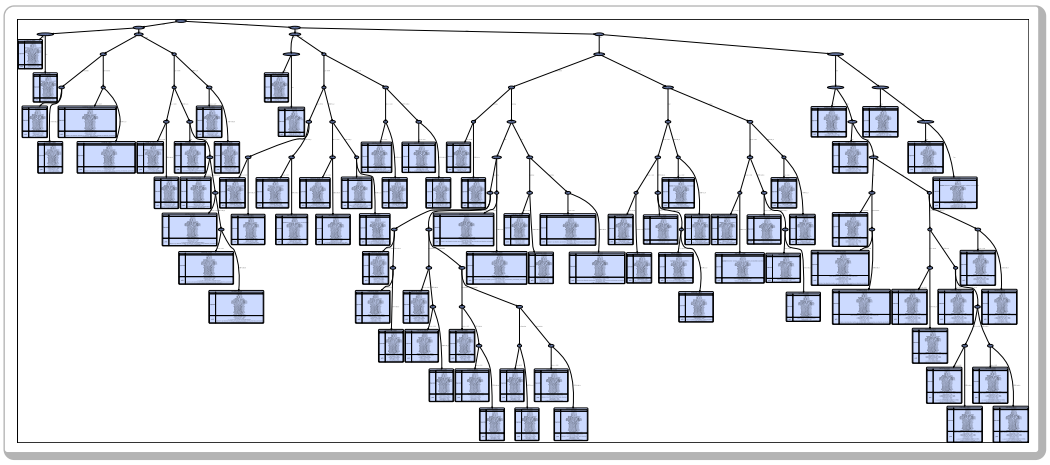
Output: a JPT representing $P(x|e)$ with $x = \text{VARIABLES}(t) \setminus e$ or an ERROR, if the evidence is unsatisfiable

```
1  t_ ← COPY(t)
2  fringe ← INSERT(t_.root, fringe)
3
4  while not EMPTY(fringe) do
5      node ← REMOVE-FIRST(fringe)
6      if IS-DECISION-NODE(node) then
7          if not HAS-CHILDREN(node) then
8              t_ ← REMOVE-FROM-TREE(node, t_)
9          else if |CHILDREN(node)| = 2
10             fringe ← INSERT-ALL(EXPAND (node), fringe)
11             continue
12         end if
13     else
14         prob ← CALCULATE-PROBABILITY(node, e)
15         if prob > 0
16             prior ← CALCULATE-PRIOR(node, prob)
17             t_ ← SET-PRIOR(prior, node, t_)
18         else
19             t_ ← REMOVE-FROM-TREE(node, t_)
20         end if
21     end if
22 end while
23
24 if not prob-mass ← CALCULATE-PROBABILITY-MASS(t_) SUCCESSFUL
   then
25     return ERROR
26 end if
27 t_ ← REDISTRIBUTE-PROBABILITY-MASS(t_)
28 t_ ← SET-PRIORS(CALCULATE-PRIORS(t_), t_)
29 return t_
```

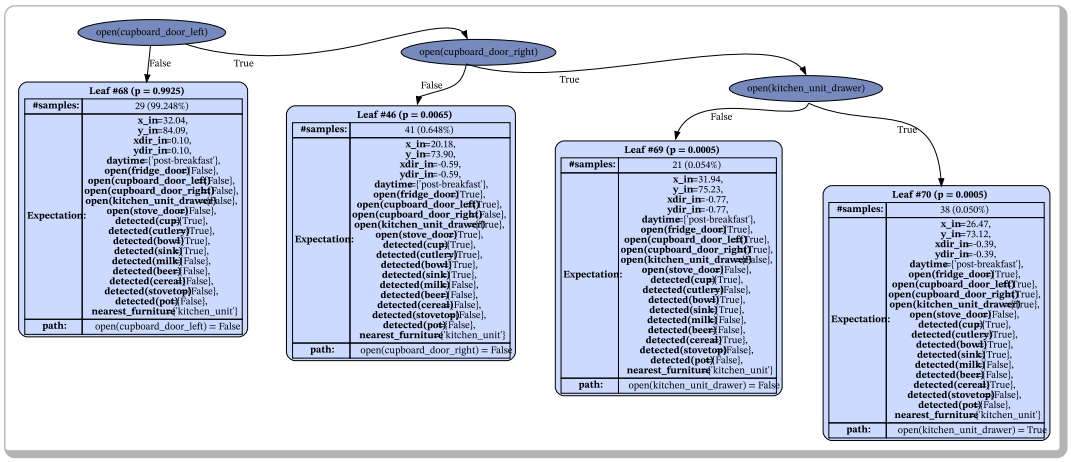
- ▷ EMPTY($iterable$) is a boolean function that returns $\top$ if the passed parameter (a list or queue) contains no elements and $\bot$ otherwise.
- ▷ INSERT($element, iterable$) and INSERT-ALL($elements, iterable$) append one or multiple elements to a list of queue, while
- ▷ INSERT($element, position, assigment$) and INSERT-ALL ($element, position, assignment$) inserts/appends them to an assignment at a specific position
- ▷ REMOVE-FIRST($iterable$) removes the first (left-most) element from a given list or queue.
- ▷ EXPAND($node$) returns all child-elements of a node
- ▷ IS-DECISION-NODE($node$) checks whether the current node is a decision or leaf node
- ▷ HAS-CHILDREN($node$) checks whether the given node has child-nodes
- ▷ REMOVE-FROM-TREE($node, tree$) removes a given node from the tree structure

The entire `perception` tree and the same tree conditioned on $Detected(\text{bowl}) = \top \wedge Daytime = \text{post-breakfast}$. | *Figure 27*

Full JPT | *a)*



Conditional JPT | *b)*



- ▷ CALCULATE-PROBABILITY-MASS($tree$) uses the results of the probability calculations of each node to determine the probability mass of the entire tree
- ▷ CALCULATE-PROBABILITY($node, evidence$) determines the probability of a node given the evidence, by multiplying the probability for each variable's distribution being consistent with the evidence, exploiting the independence assumption. The result is multiplied with the node's prior. The probability is 0 if any of the node's variable's distributions is inconsistent with the evidence.
- ▷ the SUCCESSFUL check throws an error if the previously performed calculation (CALCULATE-PROBABILITY-MASS) is unsuccessful, as the evidence is then considered unsatisfiable by the tree
- ▷ REDISTRIBUTE-PROBABILITY-MASS($tree$) is used to normalize the probabilities of the nodes and adjust their distributions according to the evidence
- ▷ CALCULATE-PRIOR($node, probability$) determines the prior for a single leaf
- ▷ SET-PRIOR($prior, node, tree$) sets the prior for a single leaf
- ▷ CALCULATE-PRIORS($tree$) determines the prior for the overall tree
- ▷ SET-PRIORS($priors, tree$) sets the prior for the overall tree

Figure 27 shows the entire JPT representing the `perception` data distribution compared to the resulting tree conditioned on $Detected(\text{bowl}) = \top \wedge Daytime = \text{post-breakfast}$, indicating a significant reduction of the tree size in the presence of additional information.

JPT posterior ▷

If one is interested in only a subset of variables, the function JPT-POSTERIOR($vars, e, t$) (Algorithm 4) can be used. As the name suggests, it returns the posteriors of the requested $vars$ of the tree t, given the evidence e (or an error, if the computation fails). The algorithm iterates over all leaves that are consistent with the evidence, computes their respective probability for the evidence and the leaf's priors. For each variable specified in $vars$ the distribution is cropped to the value range specified in the evidence. After processing all consistent leaves, the calculated likelihoods are multiplied with the respective leaf priors to determine the distributions' weights. If the weights can not be normalized, the algorithm returns an error, since the evidence is considered unsatisfiable. As a final step, the leaves' distributions for the requested variables are consolidated into a single distribution by performing a weighted merge. The resulting variable-distribution mapping is then returned.

Some of the functions explained above will be re-used here as well, while some more will be introduced in the following.

- ▷ APPLY($evidence, tree$) returns all the tree's leaf nodes that have each variable either *not* occur in the path from the root node to this node or match (e.g. lie in the interval of) the value in the path
- ▷ PAIRS($sequence, sequence$) zips together two sequences of equal length to allow iterating over the value pairs
- ▷ NORMALIZE($sequence$) normalizes a sequence of values such that the values of the resulting sequence sum up to 1.
- ▷ CROP-DISTRIBUTION($node, variable, evidence$), similarly to REDISTRIBUTE-PROBABILITY-MASS($tree$), adjust a node's distributions according to the evidence, but returns the adjusted distributions instead of updating them in-place

JPT-POSTERIOR: Compute the JPT's posterior distribution of every variable in $vars$. The result contains independent distributions. | *Algorithm 4*

JPT-POSTERIOR($vars$, e, t):

Input: $vars$, a list of query variables of the posterior to be computed
e, a variable assignment, the evidence given for the posterior to be computed
t, a JPT

Output: a variable assignment containing independent distributions $P(vars|e)$ or ERROR

Static: $distributions$, a mapping of variables to lists of distributions, initially empty
$likelihoods$, a sequence of likelihoods, initially empty
$priors$, a sequence of priors, initially empty
$weights$, a sequence of weights, initially empty
$result$, a mapping of variables to posterior distributions, initially empty

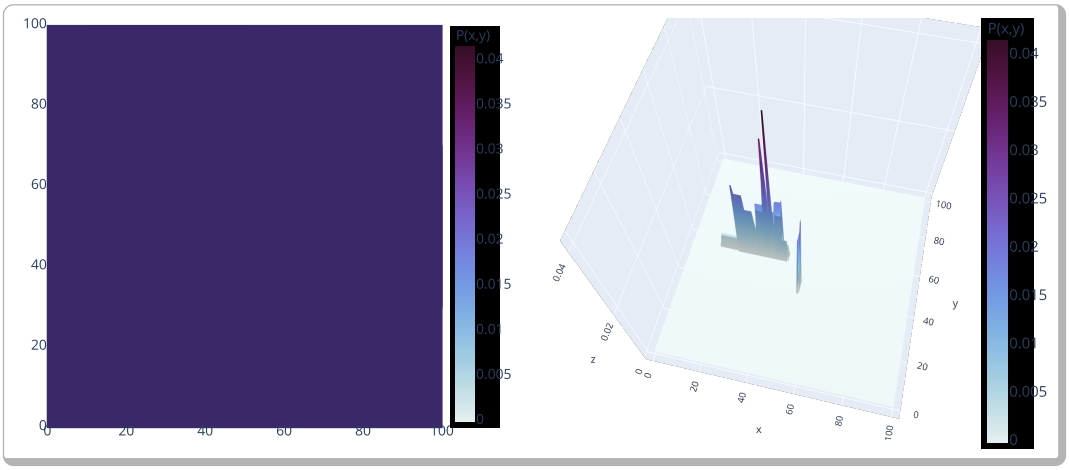
```
1  for each leaf in APPLY(e, t) do
2      likelihoods ← INSERT(CALCULATE-PROBABILITY
       (leaf, e), likelihoods)
3      priors ← INSERT( CALCULATE-PRIOR(leaf), priors)
4
5      for each var in vars do
6          dist ← CROP-DISTRIBUTION(leaf, var, e)
7          distributions ← INSERT(dist, var, distributions)
8      end for
9  end for
10
11 for each (l, p) in PAIRS(likelihoods, priors) do
12     weights ← INSERT(l · p, weights)
13 end for
14
15 if not weights ← NORMALIZE(weights) SUCCESSFUL then
16     return ERROR
17 end if
18
19 for each (var, dists) in distributions do
20     result ← INSERT(WEIGHTED-DISTS(var, dists, weights), var, result)
21 end for
22
23 return result
```

▷ WEIGHTED-DISTS$(variable, dists, weights)$ consolidates the leaves' distributions for a specific variable into a single distribution by performing a weighted merge.

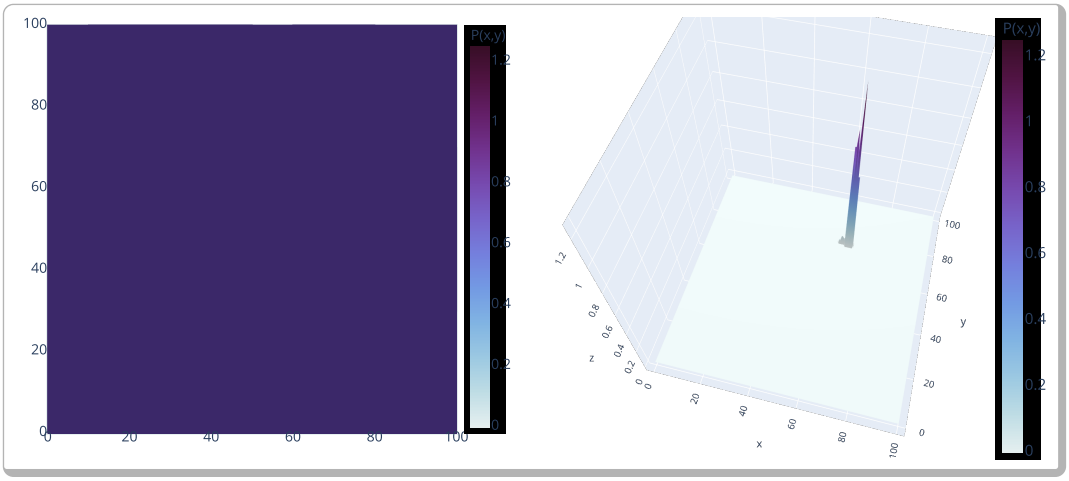
Figure 28 shows the posteriors over the position variables X_{in}, Y_{in} given different values of $daytime$ and having detected either milk or a bowl. With having detected milk in the morning (Figure 28 a)) one can see that the the agent is believed to most likely be positioned around the kitchen table, as the milk is probably part of the breakfast setting during this time. Changing the daytime to night (Figure 28 b)) adjusts the belief about the position to now be somewhere near the fridge, where milk is typically stored to keep it cool at times when it is not used. Another example (Figure 29)) shows that the agent is most likely positioned in the upper left area of the kitchen. This also makes sense, as after breakfast, the bowl is probably being washed up after use and can therefore be detected near or in the sink.

The position distribution conditioned on different perceptions: The distribution $P(X_{in}, Y_{in})$ conditioned on the variables $Daytime$ and $Detected(\langle \text{object} \rangle)$ as heatmap and 3D surface views. | *Figure 28*

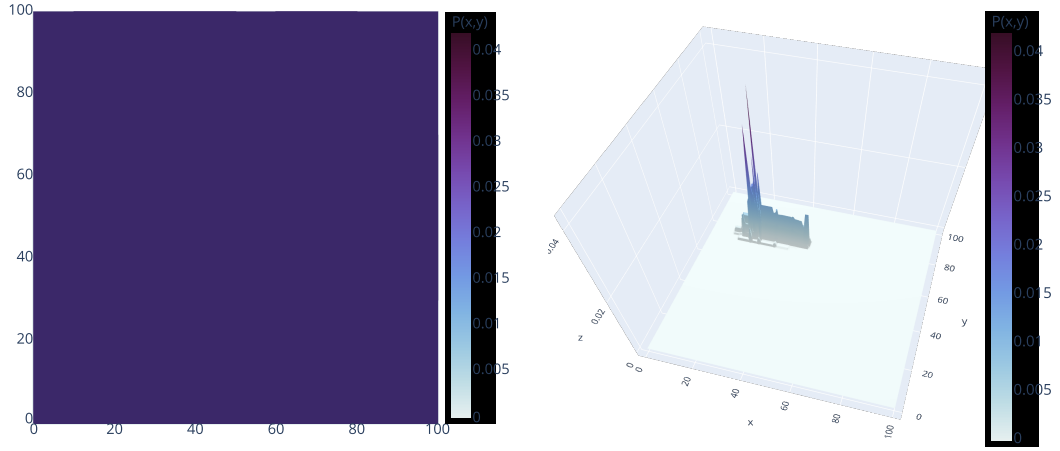
Where am I most likely positioned when I see milk in the morning? $P(X_{in}, Y_{in} \mid Daytime = \text{morning}, Detected(\text{milk}) = \top)$ | *a)*



Where am I most likely positioned when I see milk in the night? $P(X_{in}, Y_{in} \mid Daytime = \text{night}, Detected(\text{milk}) = \top)$ | *b)*



Where am I most likely positioned when I see a bowl after having breakfast? $P(X_{in}, Y_{in} \mid Daytime = \text{post-breakfast}, Detected(\text{bowl}) = \top)$ | *Figure 29*



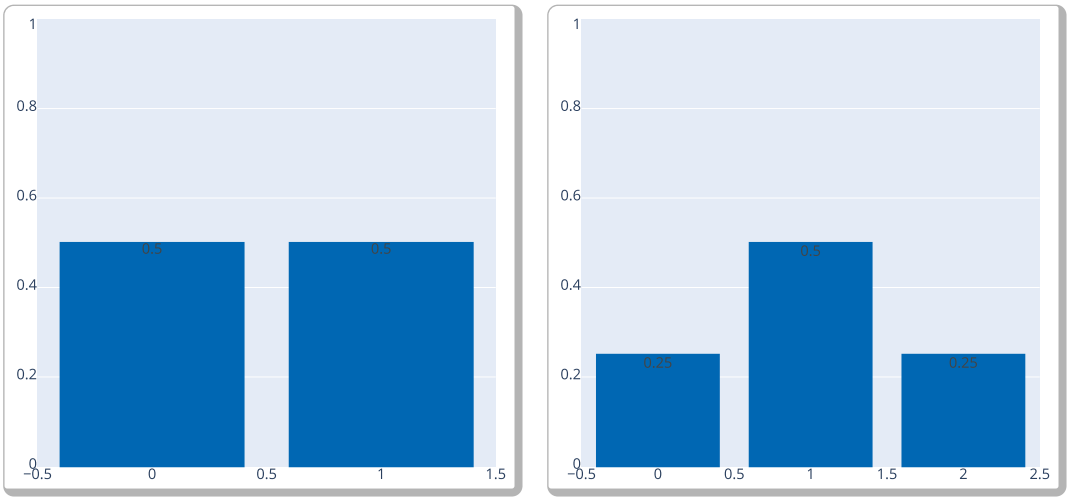
The conditional probability distributions and posterior calculations play a crucial role in updating an agent's belief state and therefore in making informed decisions based on anticipated outcomes of actions in certain contexts. The belief state updates will be dealt with in greater detail in Section 4.5.

4.5 Updating Belief State Distributions

The environment is of course subject to change induced by actions taken by the agent or external events. Consider the example from Section 4.2, in particular the two actions `move_base` and `turn`. Both of them are subsymbolic representations of a robotic movement (move `dist` into direction `d`, turn around `angle` degrees).

Sum of Bernoulli Distributions: The distribution in b) is the result of the sum of two Bernoulli distributions depicted in a) | *Figure 30*

$Ber(0.5)$ | *a)* $Bin(2, 0.5)$ | *b)*



This term reflects that actions can be taken at any point within a continuous range rather than being restricted to specific, discrete choices. Executing these actions directly influences the position of the agent, as well as the direction it faces, therefore, they need to be represented in the agent's belief state about itself. For simplicity, it is assumed that the agent's belief state consists only of these two features: *position* and *direction*. When executing a `move_base` action, parameterized with $\mathrm{dist} = 3$ given the current position of the agent as an (x, y) vector, one can easily calculate the new position by moving it 3 units into its facing direction, thereby generating a new state by simply adding a *delta* to the current state.

An action model in BAYROB, represented as a (numeric) probability distribution, can act the same way. It performs an addition of distributions to update the features of the belief state according to the parameters of the action. Note, that this makes sense for the position and direction distributions, but may not necessarily reflect a natural update of other random variables. Adding two distributions can be achieved by convolving them, combining them into a new probability measure.

Convolution allows to assign a meaningful probability to the sum of values that are subject to randomness and, in particular, corresponds to the sum of two independent random variables. Therefore, if the considered probability measures have a probability function or a probability density function, the convolution of the probability measures can be related to the convolution (of functions) of the probability functions or probability density functions. In the general case, the sum Z of two independent *discrete* random variables X and Y can be denoted as

$$P(Z = z) = \sum_{k=-\infty}^{\infty} P(X = k) \cdot P(Y = z - k). \tag{88}$$

A special case is posed by Bernoulli distributed random variables and was already introduced in the paragraph about Binomial distributions in Section 2.4.1.1. If n variables are *all* identically Bernoulli distributed, their n-fold convolution represents a binomial variable, i.e. $\sum_{i=1}^{n} Ber(p) \sim Bin(n, p)$ and in particular $\sum_{i=1}^{2} Ber(p) \sim Bin(2, p)$.

This can easily be verified and is portrayed by a small example: Consider 2 fair coins C_1 and C_2 with $dom(C_i) = \{0, 1\}$ (head, tail) and $p = P(C_i = 0) = P(C_i = 1) = 0.5$, $i \in \{1, 2\}$, i.e. C_1 and C_2 are Bernoulli distributed $C_i \sim Ber(p)$. Let X be a binomial random variable: $X \sim Bin(2, p)$. The sum of the coin distributions $Z = \sum_{i=1}^{2} X$ results in:

$$
\begin{aligned}
P(Z=z) &= P\left[\sum_{i=1}^{2} C_i = z\right] \\
&\overset{25}{=} \sum_{c\in\, dom(C_i)} P(C_1=c)\cdot P(C_2=z-c) \\
&= \sum_{c\in\, dom(C_i)} \left[\binom{1}{c} p^c (1-p)^{1-c}\right] \cdot \left[\binom{1}{z-c} p^{z-c}(1-p)^{1-z+c}\right] \\
&= p^z(1-p)^{2-z} \sum_{c\in\, dom(C_i)} \binom{1}{c}\cdot\binom{1}{z-c} \qquad (89) \\
&= p^z(1-p)^{2-z}\left[\underbrace{\binom{1}{0}}_{=1}\cdot\binom{1}{z} + \underbrace{\binom{1}{1}}_{=1}\cdot\binom{1}{z-1}\right] \\
&\overset{26}{=} \binom{2}{z}\cdot p^z\cdot(1-p)^{2-z} \\
&= P(X=z)
\end{aligned}
$$

In the specific example above with $p=(1-p)=0.5$ the domain of the sum of the variables C_i would extend to $dom(X)=\{0,1,2\}$ (cmp. Figure 30) with the respective probabilities $\{0.25, 0.5, 0.25\}$:

$$
\begin{aligned}
Bin(2,0.5) &= \sum_{k=0}^{n}\binom{n}{k}\cdot p^k\cdot(1-p)^{n-k} \\
&= \sum_{k=0}^{n}\binom{2}{k}\cdot 0.5^k\cdot 0.5^{2-k} \qquad (90) \\
&= \binom{2}{0}\cdot 0.5^0\cdot 0.5^{2-0} + \binom{2}{1}\cdot 0.5^1\cdot 0.5^{2-1} + \binom{2}{2}\cdot 0.5^2\cdot 0.5^{2-2} \\
&= 0.25 + 0.5 + 0.25
\end{aligned}
$$

It follows that if two independent random variables $X\sim Bin(n_x,p)$ and $Y\sim Bin\big(n_y,p\big)$ then for $Z:=X+Y$: $Z\sim Bin\big(n_x+n_y,p\big)$.

In the case of the distributions being continuous rather than discrete, the convolution of two distributions makes use of the distributions' PDFs to calculate the integral over:

$$
f\circ g(z) = \int_{-\infty}^{+\infty} f(x)\cdot g(z-x)\,\mathrm{d}x \qquad (91)
$$

In BAYROB, the sum of the variables representing the distributions for the x-coordinate X_{in} of the agent's position and its delta Δ_{pos_x} are then

[25] In the general case, as shown in Equation 88 one would have to sum over the entire integer space $\mathbb{Z}$, i.e. $P(Z=z)=\sum_{c=-\infty}^{\infty} P(C_1=c)\cdot P(C_2=z-c)$ but in this case, $P(X=x)=0$ for any $x\notin dom(C_i)$, therefore the sum can be reduced to only span the variables from the domain of the Bernoulli variable(s)

[26] According to Pascal's rule: $\binom{n}{k}=\binom{n-1}{k-1}+\binom{n-1}{k}$, for $0<k<n$.

$$f_{X_{in}} \circ f_{\Delta_{\mathrm{pos}_x}}(z) = \int_{-\infty}^{+\infty} f_{X_{in}}(x) \cdot f_{\Delta_{pos_x}}(z-x)\,\mathrm{d}x \tag{92}$$

where $f_{X_{in}}$ denotes the PDF of X_{in} and $f_{\Delta_{pos_x}}$ the PDF of Δ_{pos_x}.

4.5.1 Single Action Updates

anticipation ▷

Equation 91 allows to make statements about how a probabilistic hybrid action model learnt from experience data affects the robots' belief state. In particular, one can anticipate how the execution of a certain action will alter the robots' belief about the state of itself and its environment.

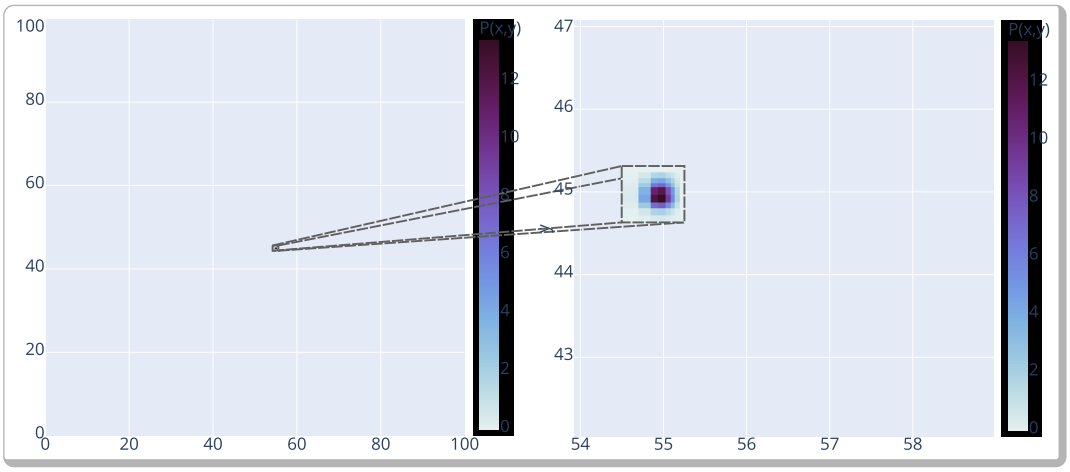
Figure 31 shows the process of that formula being applied to the distributions of a single variable, to project the outcome of the execution of one single action into the future. In this case, the variables about which the belief state is being updated are the ones representing the agent's position, (X_{in}, Y_{in}). As mentioned before, adding the delta to this variable represents moving one step forward, so the distributions $P(X_{in}, Y_{in})$ and $P(\Delta_{pos})$ are convolved to achieve a new position distribution $P({X_{in}}', {Y_{in}}')$. The two images in Figure 31 a) show the initial distribution plotted into the entire 100×100 grid and its magnified view to enable the inspection of the distribution's shape. Figure 31 b shows the Δ_{pos} distribution given the current position and facing direction. The elongated shape shows the uncertainty of the `move_base` model in terms of the distance travelled clearly. The convolution of the two distributions is shown in Figure 31 c) with a magnified view on the right. The images are scaled identically to its counterpars in the top row, so the development of the Gaussian-shaped initial distribution to the elongated shape in the convolved distribution can be followed.

Similarly, in Figure 32 one can observe the distribution update of a single `turn` action. Here, not only the shape of the distribution, but in particular its position in the plot is of interest. The variables $(XDir_{in}, YDir_{in})$ are initialized with $XDir_{in} \sim \mathcal{N}(1, \sigma^2)$ and $YDir_{in} \sim \mathcal{N}(0, \sigma^2)$; $\sigma^2 = 0.01$, denoting a facing direction to the right, as is indicated by the position of the distribution in the plot on the border between the right upper and lower quadrants.

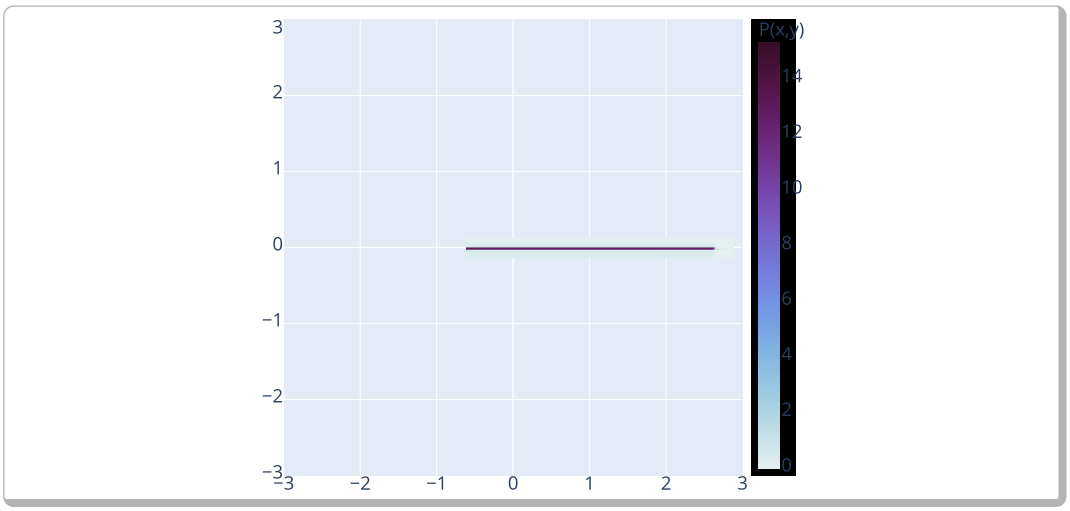
After the update, a turn of $-20°$, one can observe the distribution not only to widen vertically, but also to shift further towards the upper right quadrant, which corresponds to a turn to the left. In BAYROB, updating *continuous* and discrete *numeric* variables, that posess distributions representing an initial (*in*) and a delta (*out*) form, is performed using this strategy. However, this kind of linear update does not semantically make sense for all numeric distributions and, in particular, does not work for multinomial (and boolean) distributions. Here, the strategy is to *replace* the distribution by another distribution representing the outcome of the action execution.

A single update step for a `move_base` action: The images represent the initial, Δ_{pos} and updated (convolved) distributions for a single `move_base` step. The variables (X_{in}, Y_{in}) are initialized as $X_{in}{\sim}\mathcal{N}(55, \sigma^2)$ and $Y_{in}{\sim}\mathcal{N}(45, \sigma^2)$, the facing direction with $XDir_{in}{\sim}\mathcal{N}(1, \sigma^2)$ and $YDir_{in}{\sim}\mathcal{N}(0, \sigma^2)$; $\sigma^2 = 0.01$. The left column shows the original size of the plots, while the right magnifies the distributions for improved visibility, where a) and c) are scaled identically to easily interpret the development of the distribution. | ***Figure 31***

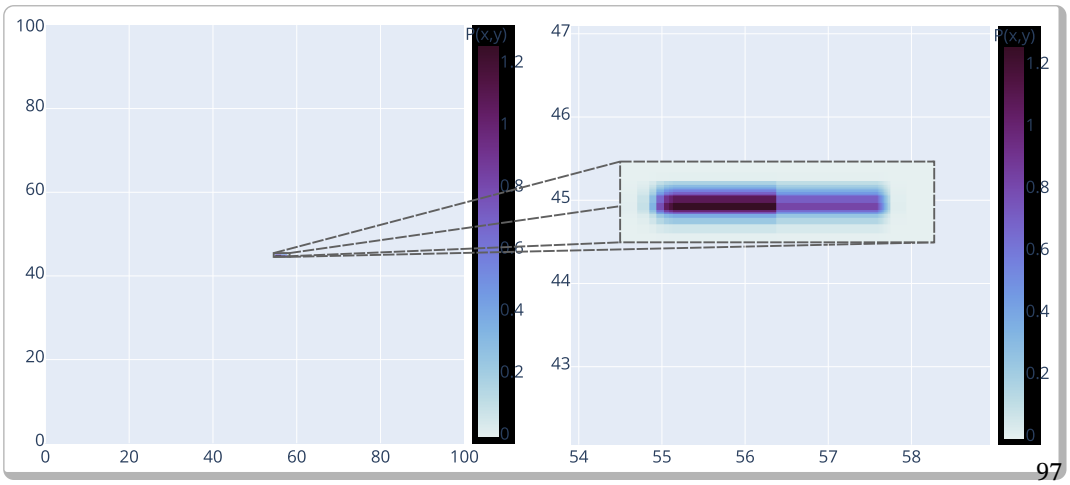
The initial position distribution $P(X_{in}, Y_{in})$ | ***a)***



The delta distribution $P(\Delta_{pos} \mid X_{in}, Y_{in}, XDir_{in}, YDir_{in})$ | ***b)***

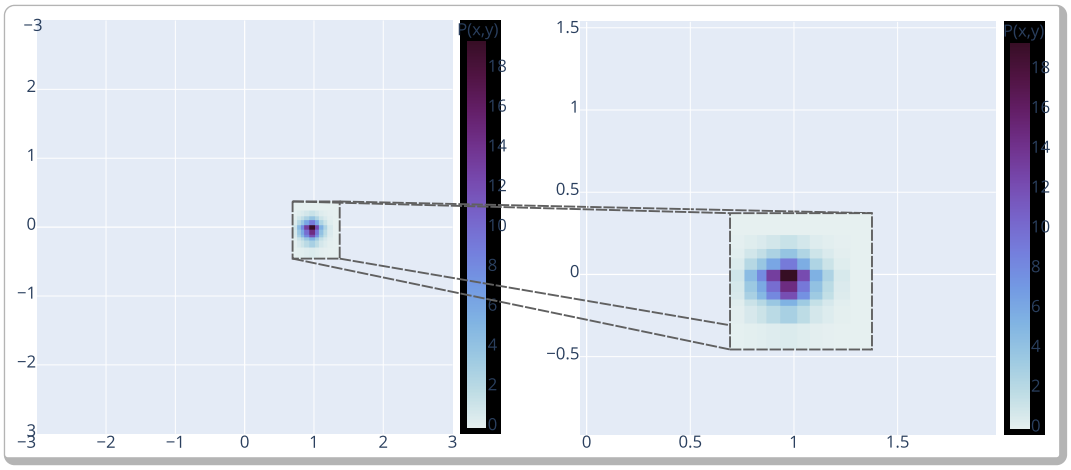


The convolutions of the distributions in Figures 31 a) and b), i.e.
$P(X_{in}{}', Y_{in}{}') = P(X_{in}, Y_{in}) + P(\Delta_{pos})$ | ***c)***

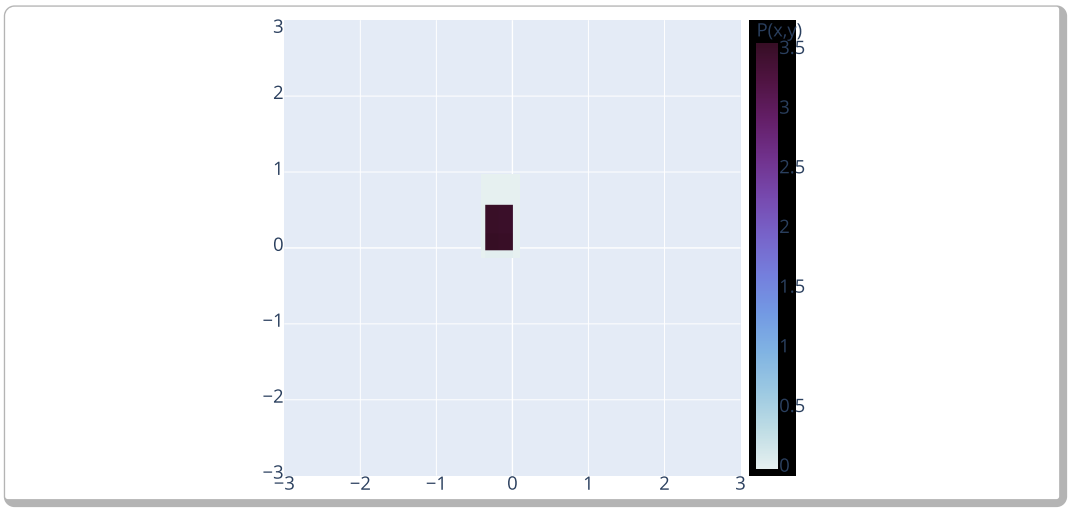


A single update step for a `turn` action: The images represent the initial, Δ_{dir} and updated (convolved) distributions for a single `turn` action parameterized with $\text{angle} = -20$. The variables $(XDir_{in}, YDir_{in})$ are initialized with $XDir_{in} \sim \mathcal{N}(1, \sigma^2)$ and $YDir_{in} \sim \mathcal{N}(0, \sigma^2)$; $\sigma^2 = 0.01$. Again, the left column shows the original size of the plots, while the right magnifies the distributions for improved visibility, where a) and c) are scaled identically to easily interpret the development of the distribution. | *Figure 32*

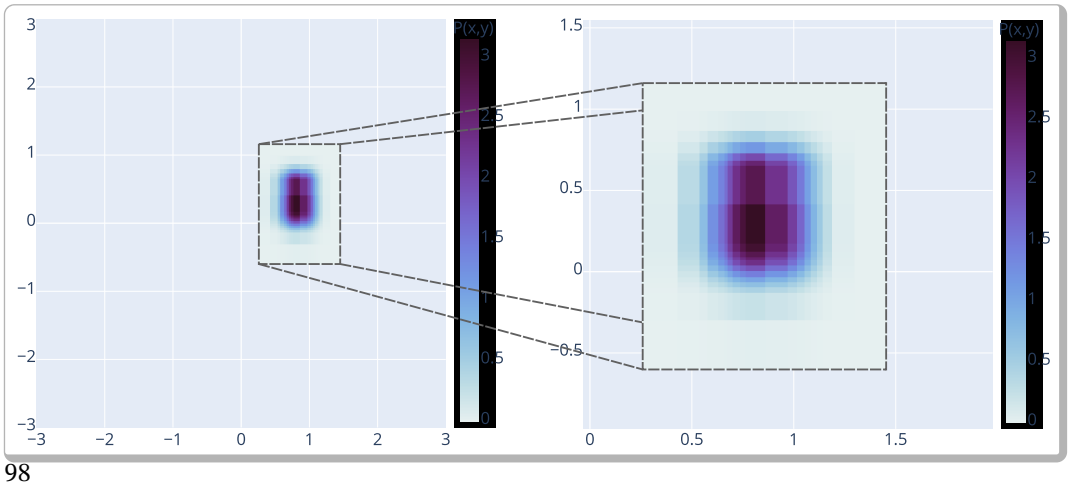
The initial distribution $P(XDir_{in}, YDir_{in})$ | *a)*



The delta distribution $P(\Delta_{dir})$ | *b)*



The convolutions of the distributions in Figures 32 a) and b), i.e.
$P(XDir_{in}', YDir_{in}') = P(XDir_{in}, YDir_{in}) + P(\Delta_{dir})$ | *c)*

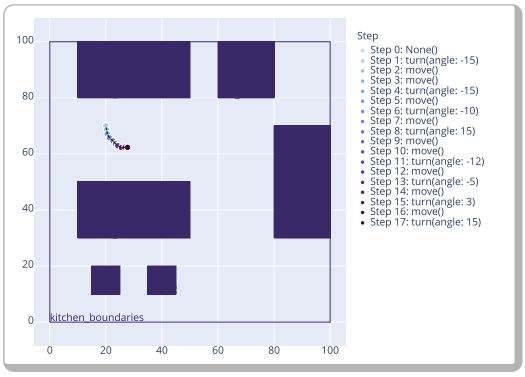


4.5.2 Multiple Subsequent Action Updates

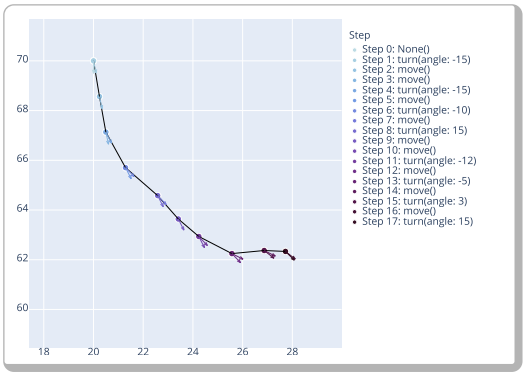
After showing that convolution can indeed be utilized to perform a semantically meaningful belief state update, it seems natural to assume that chaining updates of single actions allows to look even further into the future and prospect the outcomes of multiple actions performed sequentially. Imagine a robot being confronted with multiple robot instructions as exemplified by the (pseudoized) *Cognitive Robot Abstract Machine (CRAM)* plan in Code 2. The plan contains multiple 1-step `move-base`, as well as differently-parameterized `turn` actions. Figure 33 shows an exemplary (predicted) trajectory of a robot movement as triggered by the plan.

The trajectory of an agent as predicted by BAYROB by projecting the effects of numerous `move_base` (circles) and `turn` (arrows) actions. The positions are the expected values of the position variables for the respective step in the path. The distribution updates for the 9 `move_base` steps are shown in Figure 34 (heatmaps) and Figure 35 (3D-surface renders) | *Figure 33*

The trajectory of the movement | *a)*



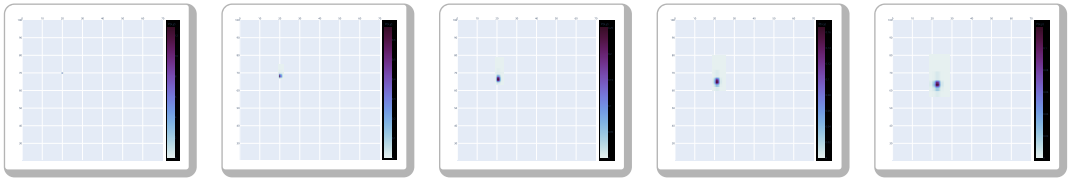
The magnified trajectory | *b)*



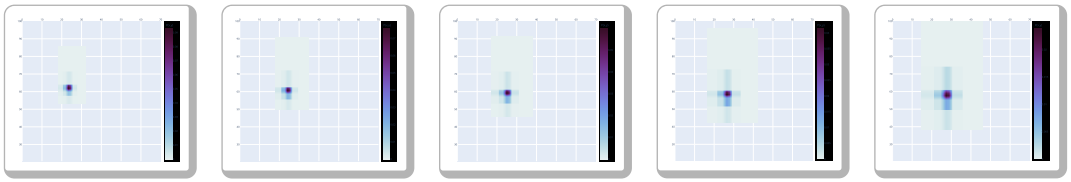
The transitions of the position distribution from the initial state over the 9 `move_base` steps can be observed in Figure 34 and Figure 35, which show the results of the convolution of the two discrete random variables X_{in} and Y_{in} of the agent and their respective Δ-distributions as determined from the `dist` parameter of a `move_base` action to update the agent's belief state. The distribution plots for the `turn` steps are intentionally left out in the plot series, as they do not trigger a position update and therefore no change in the distribution of X_{in} and Y_{in}. While the 3D-surface renders highlight the change in shape of the distributions, the changes in the heatmaps correspond to the predicted movements of the robot in Figure 33, and clearly indicate the increasing uncertainty.

The distribution update over the 9 (`move_base`) steps of the path in Figure 33. Note that the steps 1, 4, 6, 8, 11, 13, 15 and 17 are intentionally left out as the `turn`-actions do not trigger a position update and the images are therefore identical to the respective previous step. The update of the belief state implies an increase of uncertainty in each step, which is indicated by the wider range of projected possible x_{in} and y_{in} values and the decrease in color intensity. | *Figure 34*

Step 0 (init) | *a)* Step 2 | *b)* Step 3 | *c)* Step 5 | *d)* Step 7 | *e)*

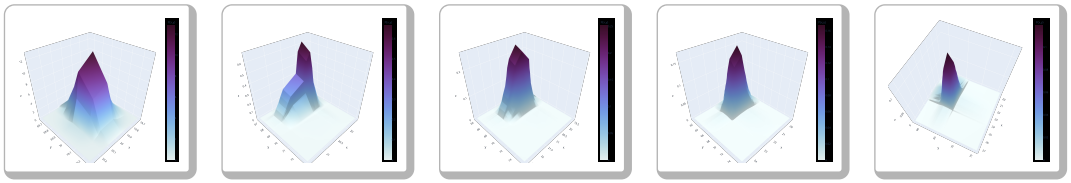


Step 9 | *f)* Step 10 | *g)* Step 12 | *h)* Step 14 | *i)* Step 16 | *j)*

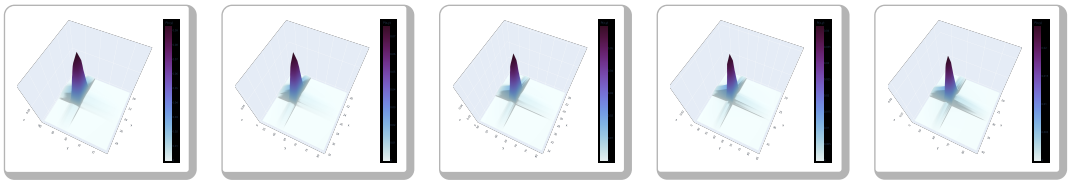


The 3D-surface renders for the distributions in Figure 34 | *Figure 35*

Step 0 (init) | *a)* Step 2 | *b)* Step 3 | *c)* Step 5 | *d)* Step 7 | *e)*



Step 9 | *f)* Step 10 | *g)* Step 12 | *h)* Step 14 | *i)* Step 16 | *j)*



CRAM plan for exemplary action sequence: Figure 33, Figure 34 and Figure 35 show the resulting path and distribution updates for this action sequence. | *Code 2*

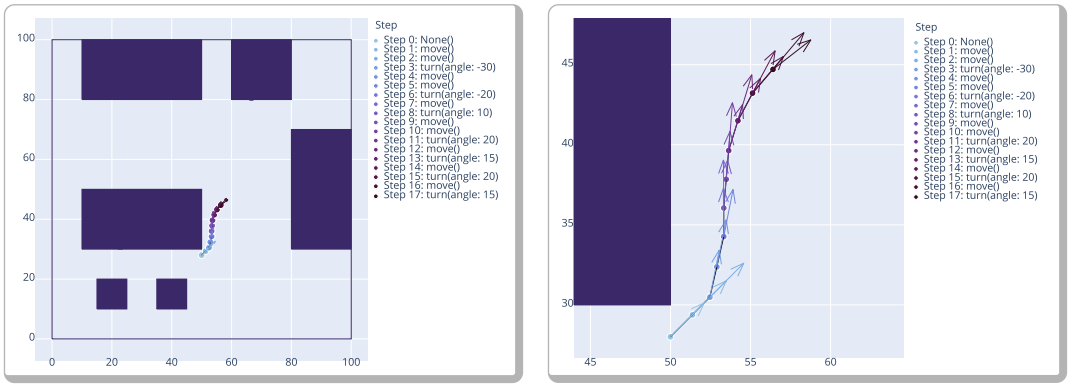
```
(defplan path(goal_location, robot_location)
    perform (an action
                (type turn)
                (angle -15))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type turn)
                (angle -15))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type turn)
                (angle -10))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type turn)
                (angle 15))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type turn)
                (angle -12))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type turn)
                (angle -5))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type turn)
                (angle 3))
    perform (an action
                (type move_base)
                (distance 1))
    perform (an action
                (type turn)
                (angle 15))
```

4.5.3 Addition vs Shift

Alternatively to convolving distributions, a possible belief state update could be realized by *shifting* distributions by the delta represented by Δ_{pos} (or Δ_{dir}, respectively). Mathematically, updating the belief state is achieved by translating the distribution function by a value Δ, i.e. $f(x + \Delta)$, where positive values of Δ will cause the function to "move to the left", negative value will "move" it to the right. As the function takes a single value for Δ instead of an entire distribution, one could choose the expectation of Δ_{pos} (or Δ_{dir}, respectively).

The prediction of a trajectory of an agent as a result of shifting the distributions to achieve a belief state update. | *Figure 36*

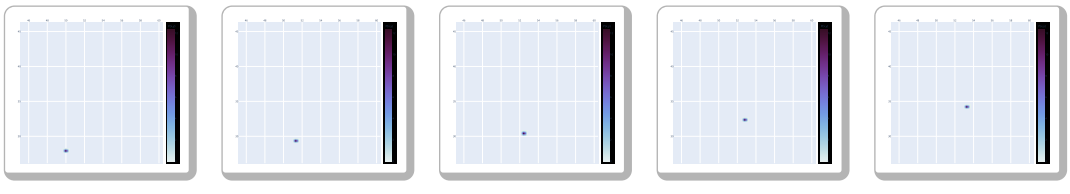
Entire kitchen scenario | *a)* Magnified path | *b)*



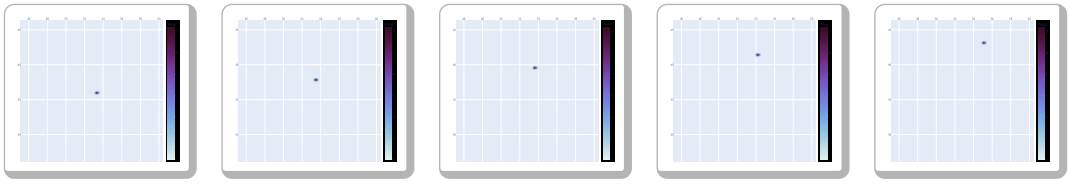
Obviously, this strategy has the advantage, that it can be calculated much easier than the convolution and the time/cost spent on this operation is thus drastically reduced.

The distribution shift over the 9 (`move_base`) steps of the path in Figure 33. The update of the belief state implies the change in position of the plot but does not involve any increase of uncertainty or any other change in shape of the distribution. | *Figure 37*

Step 0 (init) | *a)* Step 2 | *b)* Step 3 | *c)* Step 5 | *d)* Step 7 | *e)*



Step 9 | *f)* Step 10 | *g)* Step 12 | *h)* Step 14 | *i)* Step 16 | *j)*



However, while the generated path (triggered by the same plan in Code 2), looks very similar, the distribution itself never really changes in terms of shape, i.e. the uncer-

tainty always stays the same (Figure 37). This behavior thus does not correspond to the natural understanding of the development of an agent's belief state after performing actions.

An agent being somewhat uncertain about its position in the first place will be even more uncertain about it after executing an action that, in itself, adds uncertainty to the equation through e.g. inaccuracies in actuators or sensory output. In BAYROB, the emphasis is placed on the accuracy of predictions rather than computational efficiency. This design choice led to a preference for convolving distributions over shifting them. However, to maintain a computable size of the belief state distributions, an reduction is performed after the convolution, in which the PLF of the distribution is replaced by one with fewer function segments through iteratively replacing subsequent function segments by an approximation thereof.

4.6 Plan Refinement with JPTs

In the previous section, it has been shown that the representation of a belief state in terms of joint distributions allows to anticipate the outcome of an action execution and it has also already been mentioned that applying the model into the other direction is of particular interest. Given the current belief state, what does the agent have to do in order to achieve a specific goal specification? Which actions have to be performed and how do the parameters have to be set? The capability of allowing such backward inferences is a major advantage a generative model has over a discriminative one. The backward inference aims at finding all possible states from which the current state could be reached by executing an arbitrary *single* action. When realizing this kind of inference, one can make use of the structure of the underlying probabilistic model, in particular, the tree-structure of the learnt JPT. Based on the assumption that the predecessors of a tree node have been naturally created by the learning algorithm in forward direction and that the algorithm was driven by real experience data, one can assume that a semantic correlation exists between a node and the adjacent nodes in close proximity. Leveraging the tree structure, which is applicable in both forward and reverse directions is thus evident and will be outlined in the following.

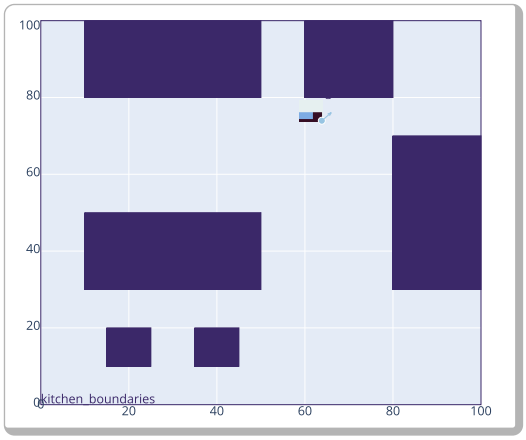
4.6.1 Single Backward Action Updates

In contrast to finding *successor* states that (most probably) represent a desired outcome, finding *predecessor* steps can be seen as the attempt to describe a certain situation that could have led to another situation, which breaks down to the simple question of "How did I get here?". If, for example, the current situation is that an agent has perceived milk, a possible answer to how that could have happened would be something like "Well, if it is *morning* and you are standing near the *kitchen island* somewhere around position $(40, 50)$ and ... and you used some *detection* action, *then* you could have seen milk". Other possible answers are of course all the different combinations of positions, times of day and states of drawers and doors *before* some action either changed the detection state of the milk from $\bot$ to $\top$ or left it unchanged if it was already $\top$. Figure 38 shows

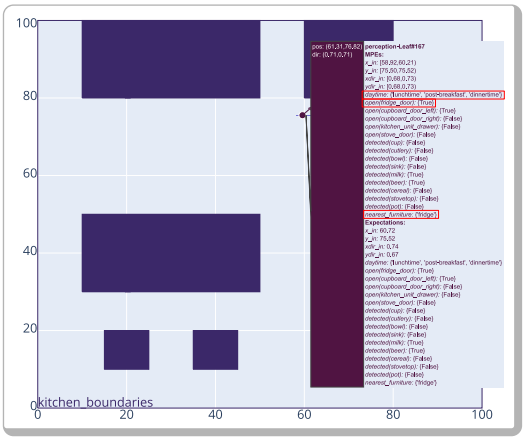
some of the immediate predecessor steps of this particular example. The predecessors are determined by checking whether the probability that the result of the forward execution of each leaf of the action models satisfies the query is greater than 0. If this is the case, the robot's belief state is updated according to the *input* distributions of the leaf i.e. the ones representing the state of the variables *before* the action was executed. If the probability is 0, the leaf is skipped and not further considered as a potential predecessor candidate. In conformance with the training data, the milk can only be detected *in* the fridge, see Figures 38 a) and b) or *on* the kitchen island, see Figures 38 c) and d).

A subset of predecessor candidates for the goal $Detected(milk) = \top$ | ***Figure 38***

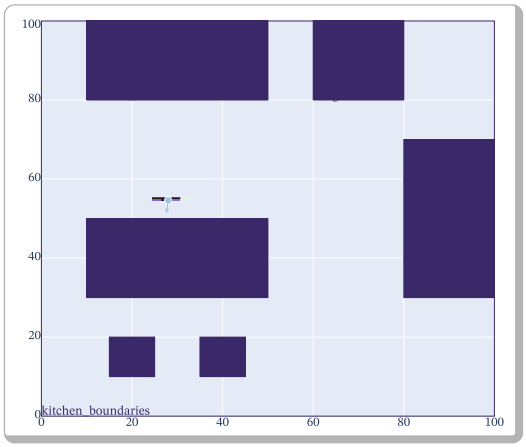
Most candidates are located where one would typically stand in front of the (open) fridge, facing top-right | *a)*



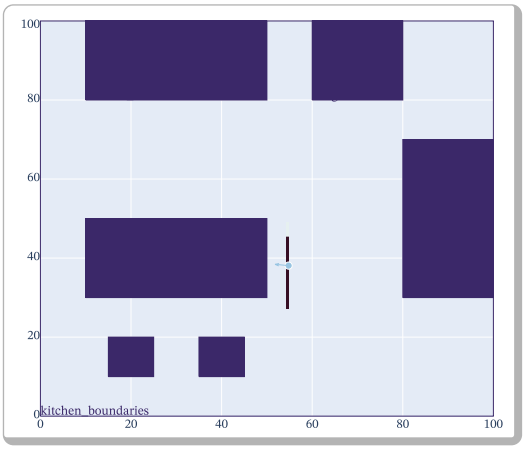
These candidates also have a strong tendency for $Open(fridge\_door) = \top$ and $Daytime \neq$ morning | *b)*



In the morning, milk can typically be detected from the upper edge of the kitchen island... | *c)*



... or from the right edge of it, facing towards the table, respectively | *d)*



The most probable positions are therefore scattered around the table and the area, where one would stand when the fridge door is opened, facing towards the target, respectively. If the fridge is closed, however, the milk can not be detected albeit the position and facing direction being ideal. And still, even if the fridge door is open, milk is not always stored in the fridge at all times, as it might currently be in use. Therefore, the time of day also plays a role for this specific request, as can be seen in the highlighted

distributions in the infobox in Figure 38 b). For better readability, a magnified version of the annotations is shown in Figure 39.

The magnified annotations of the predecessor candidate in Figure 38 b) | *Figure 39*

error treatment ▷

The backward action update is particularly useful when reasoning over erroneous behavior, i.e. when trying to determine, which preconditions led to an action to fail. Proficiency in error detection, prediction and, consequently, avoidance is essential for an autonomous agent to choose a configuration of actions and parameters with the highest probability of success. The backward action update is therefore vital for identifying types of errors that frequently emerge from certain situations and minimizing their probability of occurrence.

4.6.2 Multiple Backward Action Updates

Analogously to the multiple forward action updates in Section 4.5.2, it is reasonable to believe that once a selection of potential predecessor states for a goal specification was found, the pre-predecessors can be determined as well, guided towards a certain intial (current) state of the robot, such that ultimately a sequence of belief states can be determined. It is then just a small step towards thinking of these action updates as a *search* for a *path* between two belief states or a belief state and a (fuzzy) goal specification. Wrapping up the observations from the previous sections, one finds all the tools to describe the forward- or backward inference as a well-defined search problem (Russell and Norvig 2010, Chapter 3).

Problem Formulation · A search problem typically revolves around discovering a solution within a designated *search space*. This space encompasses the entirety of possible states that the system or problem can assume. Each state in the kitchen example denotes a specific configuration or arrangement and - as the name suggests - contains all possible variable assignments (belief states). The *initial state*, the starting point from where the search begins, is simply the robot's current belief state, e.g. its current position and facing direction in the kitchen. Thus, the *goal state* is a belief state that satisfies the goal specification, e.g. a certain position the robot intends to reach or a situation setting such as being in a position from where milk can be seen (in order to grab it). A *successor function* or *action model* defines the possible actions that can be executed from a given state. It generates the set of successor (or predecessor) states that can be reached from the current state. This function is defined by the respective (multiple) forward and backward action updates introduced in the course of this chapter. Each action execution comes with a *cost* which quantifies the effort required to transition from one state to another. The sum of the costs of the execution of an entire action sequence is called the *path costs* of a possible solution. Naturally, one tries to minimize the cost, such that sequences of actions (if more than one can be found) with lower costs are preferred over more expensive ones.

If an algorithm always finds the solution with the lowest costs, it is characterized as *optimal*. Note that the "best" (or cheapest) solution highly depends on the problem and in fact, there might not even exist optimality in certain settings. Other criteria to evaluate the quality of a search algorithm include – among others – its *completeness* and its *time* and *space complexity*. Completeness describes the capability of an algorithm to reliably find a solution *if it exists,* which is essential in many applications. Time and space complexity measure the efficiency of an algorithm in terms of an estimation of the amount of time an algorithm takes to complete the search or the amount of memory required

to hold (partial) candidate solutions available. Lower time and space complexity are generally preferred.

The A* search algorithm stands as a fundamental tool in pathfinding and optimization, valued for its efficiency and versatility. Its appeal emerges from its ability to intelligently guide its path exploration by combining two crucial factors: the *step cost* incurred to reach a particular state and a *heuristic* estimate of the remaining distance to the goal. This dual-informed approach enables A* to intelligently prioritize paths, ensuring to find the most cost-effective route and drastically reducing the number of expanded nodes in comparison to uninformed algorithms along the way. A* is complete and optimal (given an admissible, that is, not overestimating heuristic), making it a preferred choice in various domains. The critical parts of the A* algorithm, outlined in Algorithm 5, are the functions for the goal check (IS-GOAL$(node)$), the successor generation (GENERATE-SUCCESSORS$(node)$) and the calculation of the heuristic and step costs (aggregated in CALCULATE-SCORE$(node)$). The goal check is an identity check of the current node and the goal node (or a check whether the current node meets some fuzzy goal criteria, however) and constitutes one of two termination criteria of the algorithm, the other being that there are no more nodes to expand in the open list without having reached the goal yet. The successor generation determines all possible follow-up nodes that can be reached from the current node. The calculation of the costs is an aggregation of the step costs (the costs that have been accumulated so far for getting to the current node) and the expected remaining costs from the current node to the goal.

In BAYROB, the search for a solution is performed in an A*-like fashion, but without guaranteeing optimality. The cost measure in this setting is not only depending on the problem itself, but also on the type of question being asked in every single search execution. So what does cost mean exactly? Depending on the context, this may vary. When trying to find a sequence of actions resulting in a fully set breakfast table, one might go for the *quickest* solution, involving transporting multiple objects (cups, plates, cutlery) at once instead of moving between the cupboard and the kitchen table for every single item. Tasked with the assignment to fetch some drink from the fridge, it might be advisable to choose the *safest* plan with a high probability of success and low probability of dropping breakable containers and spilling liquids. Costs in BAYROB are therefore determined using two different measures: the *confidence*, which is the probability that the current state can be reached from a candidate state (or vice versa), and the distance to the goal (i.e. a belief state satisfying the goal specification). The distance aggregates the single distances of the different components of the belief states and requires finding a measure that takes into account the hybrid domains of these variables.

◁ *complex tasks & effect control*

While the distance between distributions of the continuous variables can be measured using the Wasserstein distance (cmp. Equation 43 in Section 2.4.3), this does not work for multinomial distributions. Taking $1 - sim_{cont}(d_1, d_2)$ (cmp. Equation 40) might be a solution here, but then a solution has to be found on how to combine *normalized* distance of multinomial distributions with the *unnormalized* distance of the continuous ones. In BAYROB, there are different approaches combined, depending on what the distance measure is used for.

The general algorithm of A* can be executed in a forward and reverse search direction, given all its operations are invertible, which can be very difficult and is sometimes not feasible at all. If the operations are not invertible, it is not possible to generate parent (i.e. predecessor) nodes for a current state. Action models in BAYROB support inference in both directions as shown in Section 4.5.1 and Section 4.6.1, and the A*-like search applying these types of inference will be outlined in the following.

A*: The algorithm for the A* search | *Algorithm 5*

A*(s, g):

Input: s, a start node representing the current state
g, a goal node to find a path to

Output: a path from s to g or ERROR

Static: $open$, a priority queue, initially empty
$closed$, a set of visited nodes, initially empty

```
1  open ← s
2  closed ← ∅
3
4  while not EMPTY(open) do
5      node ← REMOVE-FIRST(open)
6      closed ← INSERT(node, closed)
7      if IS-GOAL(node) then
8          return PATH(node)
9
10     for each node_ in GENERATE-SUCCESSORS(node) do
11         if CONTAINS(closed, node_) then
12             continue
13         end if
14
15         if CONTAINS(open, node_) then
16             continue
17         end if
18
19         node_ ← CALCULATE-SCORE(node_)
20         open ← INSERT(node_, open)
21     end for
22 return ERROR
```

Forward Direction · The forward direction of an A*-like search in BAYROB boils down to finding every possible action that can be executed in the current state. This translates to the *application* of the tree model to a given query, i.e. computing the conditional trees which only contain the nodes that satisfy the conditions determined by the (uncertain) variable values of the agent's current belief state, as described by Algorithm 3 in Section 4.4.1. The leaves of the resulting trees serve as potential candidates for computing successor nodes. A successor state is generated using Algorithm 6: The function GENERATE-CANDIDATES($node$) conditions each action model (tree) on the variable values of the state represented by the given node and returns a list of the pooled leaves of the resulting conditional trees. For each of the leaves, a new node (which in BAYROB represents a belief state) is created, being an exact copy of the given node at first. The distributions are then updated by either adding the leaf's delta distributions of that variable (which means the current variable is updated by the action represented by the leaf) or replacing it by the leaf distribution if there is no delta distribution for that particular variable. The latter is typically the case for all multinomial distributions. The list of the newly generated states is then returned as a list of potential successors for the given start node.

The overall search algorithm is based on Algorithm 5, with a few particularities due to the nature of the variables being distributions. The A* algorithm only works (and is optimal and complete), if the goal test and the cost functions are defined appropriately. When working with distributions, the goal test can of course not check for exact identity with the goal specification as the goal specification is required to be a mapping of variables to ranges (or sets) of values while the states are mappings of variables to distributions. Even if the current state and the goal both contained only distributions, one would hardly ever be able to generate a state through the execution of multiple actions that matches *exactly* a given goal distribution.

A similar challenge applies when trying to determine, whether a node has already been processed before (i.e. the $closed$ list contains the respective node) or was already added to the $open$ queue (lines 11 to 17 in Algorithm 5). This check is necessary to prevent processing the same nodes over and over again, resulting in an infinite loop. It is therefore advisable to substitute *equality* in terms of nodes/states in BAYROB for *sufficiently close similarity* between them to prevent computational (and memory) overload. As an example, one could aggregate the individual similarities of the distributions of the (common) variables between two states, whereas the aggregation operation could be the mean (or minimum or any other kind of function) of the individual values. In BAYROB, the similarity of two states is determined by the mean of the individual (Jaccard) similarities of the shared distributions. This measure is used whenever it is required to determine, whether some state sufficiently resembles another state or the goal. In the forward search, the decision whether or not a node should be added to the $open$ priority queue or the $closed$ list of already visited nodes is reduced to a simpler check, which compares which leaves of which trees have been consulted to create the states. Note that this may lead to some potential interesting nodes being dropped, since the state distributions do not exactly represent the leaf distributions in the trees, but may vary depending on the input distributions (through the convolution of distributions). However, in practice, this has shown to be a useful decision, since many very

similar states were created, that did not qualitatively differ significantly, which caused computational and memory overload without considerably improving the results.

BAYROB-GENERATE-SUCCESSORS: The algorithm to generate successor nodes for an A*-like (forward) search | *Algorithm 6*

BAYROB-GENERATE-SUCCESSORS(s):

Input: s, a start node representing the current state

Output: a set of potential successor nodes

Static: $successors$, a set of potential successor nodes

```
1  successors ← ∅
2
3  for each leaf in GENERATE-CANDIDATES(node) do
4      s_ ← COPY-STATE(s)
5      for each (dname, d) in DISTRIBUTIONS(leaf) do
6          if IS-DELTA(dname, leaf) then
7              d_ ← DISTRIBUTION(s_, dname) + d
8          else
9              d_ ← DISTRIBUTION(s_, dname) = d
10         end if
11
12         s_ ← UPDATE-OR-INSERT(s_, dname, d)
13         successors ← INSERT(s_, successors)
14     end for
15 return successors
```

The goal test of the forward direction first checks, whether the current node contains distributions for all variables that are specified in the goal. Is that not the case, the goal test fails because there is no way the goal specification can be satisfied by the current node. Otherwise the goal test assesses whether the existing belief state aligns with a fuzzy goal specification. Given that the belief state is represented as a distribution, a natural approach seems to be evaluating the probability of the specified value range within the distribution and then comparing it against a predetermined threshold. However, this method encounters challenges as determining an appropriate threshold proves to be difficult. The complexity arises from the fact that, over successive update steps, uncertainty within the belief state tends to increase drastically. Consequently, even a well-regarded belief state may necessitate an extremely low threshold, making it impractical for effective goal evaluation. The evolving uncertainty within the belief state underscores the need for more nuanced and adaptive approaches in the goal

test to account for the dynamic nature of the underlying distributions. To address this challenge, a refined approach in goal testing involves computing the MPE of the belief state's distribution. This MPE is then intersected with the specified value range of the goal specification, revealing the overlap between the belief state and the goal. In a final step, the result of this intersection, which signifies how much of the belief state aligns with the goal, is compared with the MPE of the distribution computed in the first place. This comparison not only gauges the overlap with the goal but also accounts for the broader distribution, indicating how much of it falls outside the goal.

Given a sufficient amount of examples available, one would usually derive the heuristic for a search algorithm guided by data. However, for the sake of simplicity in the current example, a manual calculation of a distance measure is opted for to be employed as a heuristic function. The calculation of a heuristic in general can be understood as a means to quantify the effort required to get from one point to another, or, to turn the current state into a state that matches the goal criteria. In other words, it needs to determine, how much of each of the distributions has to be changed to transform them into "acceptable" distributions. That sounds a lot like the idea behind the earth moving explanation of the Wasserstein distance introduced in Section 2.4.3 which is why it is taken to determine the heuristic for the numeric distribution variables. For the other distributions in BAYROB, however, the adaptation for multinomial distributions is used (cmp. Equation 44). The heuristic is then the mean of the distances between the state and the goal. Analogously, the step costs are defined as the distance between two subsequent states.

BAYROB-GENERATE-PREDECESSORS: The algorithm to generate predecessor nodes for an A*-like (reverse) search | *Algorithm 7*

BAYROB-GENERATE-PREDECESSORS(s):

Input: s, a start node representing the current state

Output: a set of potential predecessor nodes

Static: $predecessors$, a set of potential predecessor nodes

```
1  predecessors ← ∅
2
3  for each s_ in BAYROB-REVERSE(node) do
4      predecessor ← INSERT(s_, predecessor)
5  end for
6  return predecessor
```

Reverse Direction · In contrast to the forward direction, the action models are not constrained by conditioning the entire tree on a given query. As the name suggests, the goal is to find all possible predecessor steps for the current (initially the goal) state and to build an action sequence by stepwise prepending those predecessor steps until the

initial state is reached. In the context of BayRoB, the approach in the reverse direction is to find all actions, that, when applied, result in a state that is sufficiently similar to the current state. Since the outcome of an action is only implicitly available by the forward application of the leaf, the search for predecessors may involve multiple computationally heavy operations which have to be applied to each leaf of each action model in every search iteration. This depends on the type of the respective distributions and variables. In the case of numeric variables that are represented by designated initial and delta distributions as is the case for the positional and directional distributions, the computation of the leaf's outcome involves the addition, cropping and subtraction of distributions (cmp. Algorithm 7). This procedure aims at generating potential predecessor states that have a high probability of resulting in the current state given the action and parameters represented by the respective leaf. The outcome of these operations are then compared to the current state to determine their degree of accordance. If the result matches the current state, the newly generated state is accepted as a potential predecessor state and serves as a new goal in a subsequent search step. The stepwise prepending of predecessor steps until the init state is reached then results in a sequence of actions as path from the init state to the goal. A final forward execution of the found path then serves as a feasibility check and allows to compare the actual predicted result with the desired goal state.

The heuristic and step costs are calculated the same way as in the forward direction, with the difference that the heuristic determines the distance from the current state to the init state instead of the goal state.

Bi-Directional Search · The forward and reverse search can be combined into a common search to profit from the respective advantages of both directions. After initializing the open lists of both the forward and reverse direction algorithms, the bi-directional search checks in every iteration, whether the respective current nodes are identical, which would indicate that the two algorithms found a common node and therefore a path from the initstate to the goal state. If the nodes are different, the current node of the forward direction is set as the (temporary) goal for the reverse algorithm and vice versa, such that it is ensured that both algorithm work towards a common node. The respective current nodes are then expanded and the successor (or predecessor) nodes are computed and added to the open queues. A solution in this strategy is found if

- the current belief states of the forward and reverse search are identical[27] (the algorithms "meet in the middle"), or
- the current belief state of the forward algorithm matches the goal specification or
- the current belief state of the reverse algorithm matches goal specification.

The results found by the algorithms are then aggregated to form a single path from the initial state to the (originally specified) goal state.

[27] As mentioned before, it is almost impossible to find identical distributions (and thus states) in the present approach, therefore "identical" is interpreted as "sufficiently close" here, i.e. the similarity of two belief states is lower than a predefined threshold.

BAYROB-REVERSE: The algorithm to generate candidates by reversing tree inference in A*-like (reverse) search | *Algorithm 8*

BAYROB-REVERSE(s):

Input: s, a start node representing the current state

Output: a set of candidates

Static: $candidates$, a set of potential predecessor nodes

```
1  candidates ← ∅
2
3  for each action_model do
4      for each leaf in LEAVES(action_model)
5          s_ ← STATE()
6          for each (dname, d) in DISTRIBUTIONS(leaf) do
7              if IS-DELTA(dname, leaf) then
8                  continue
9              else if HAS-DELTA(dname, leaf) then
10                 d_ ← d + DELTA(dname, leaf)
11                 p ← PROBABILITY(d_, query)
12                 if not p > 0 then
13                     continue to next leaf
14                 end if
15                 d_ ← CROP(d_, query)
16                 d_ ← d_ − DELTA(dname, leaf)
17             else
18                 p ← PROBABILITY(d, query)
19                 if not p > 0 then
20                     continue to next leaf
21                 end if
22                 d_ ← d
23             end if
24
25             s_ ← UPDATE-OR-INSERT(s_, dname, d_)
26         end for
27     end for
28     candidates ← INSERT(s_, candidates)
29 end for
30 return candidates
```

chapter five

Probabilistic Knowledge Bases for Material Discovery

This chapter provides comprehensive exploration of another example for the usage of the BayRoB principle, drawing primarily upon the insights and findings presented in Picklum and Beetz (2019). The following pages are dedicated to analyzing the key concepts, research methodologies, and significant discoveries that have emerged from the work documented in Picklum and Beetz (2019). The conclusions drawn form the foundation of this chapter and emphasize the contributions of the methodolgy on a broader range of applications.

5.1 MatCALO

Innovations in industry and technology constantly call for the development of novel, increasingly powerful materials. The field of materials science is concerned with finding materials that meet predefined property specifications which vary with the intended use of the material. In general, a material has a certain set of properties that are subject to a number of influencing factors. As an example, the chemical composition, i.e. the individual elements the material consists of, plays an important role for the distinct characteristics the resulting material has. But also the sequence of different treatments a material undergoes has substantial impact on the final material's properties. Each processing step may change the structure of a material such that it develops different properties depending on the material itself and the parameterization of the treatment. Multiple processes conducted subsequently in a *process chain* then produce the final set of properties the material will have. However, some of the processing steps may cancel out the results of previous steps, therefore finding the right process chain is crucial in producing the desired material.

The emerging field of *materials informatics* combines domain knowledge from the material sciences with computational and representational means of computer science.

However, despite remarkable progress in both analytical and data-driven approaches towards material design, the discovery and engineering of new materials is still a time, cost, and labor-intensive process.

One obvious reason for this is the intractably huge exploration space, which originates from the number of qualitative and quantitative process parameters of different material treatments. For example, considering the number of different possible processes (e.g. thermal, mechanical or chemical treatments) with their respective parameterizations (e.g. temperatures, processing times) as well as the proportions of the composition yield an unimaginably large number of possible combinations. From a computational point of view, this process of exploring new materials suffers from what is commonly referred to as the curse of dimensionality'', the exponential growth of a search space in the number of its dimensions. In addition, the empirical evaluation of a new material compound is time and labor intensive as it involves the manufacturing and processing of sufficiently large samples of a material, as well as the investigation and measurement of its properties.

Ellendt and Mädler (Ellendt and Mädler 2018) have proposed a novel approach towards material design, called *Farbige Zustände* (*colored states*), which aims at the investigation of materials and processes on microscopic-size samples that can be produced with innovative high-throughput methods at low cost. However, these micro samples are often too small in size to directly measure properties like the fatigue strength against. Instead, a set of descriptive features of micro samples is used to predict the material properties from microscopic to macroscopic samples. Finding mappings between these features in the microscopic level and corresponding properties in the macroscopic level plays a central role in this development process because it allows to experimentally explore material groups while overcoming resource limitations at the same time.

The experimental setup within the approach is designed for the systematic analysis of a selection of well-known metal alloys such that the results can be compared to findings from literature. The relations can later be transferred to less thoroughly investigated materials.

In this chapter, the conception and prototypical implementation of MatCALO is presented, an intelligent, assistive system that is capable of supporting materials scientists in their work by generating hypotheses on how to process certain materials to obtain desired properties. The overarching vision behind this system is the idea that a material scientist approaches the system with a set of desired properties a material should meet. The researcher queries the system with these requirements and gets back a confidence-rated set of hypotheses about which material compositions and processing steps are likely to yield the respective results. A high-level overview of this framework is shown in Figure 40. To this end, the system maintains a large database of experimental data which will be used to generate a set of candidate answers. In a later state the system will additionally use semantic knowledge to complement the trained model in refining its results, presenting alternative answers and generating final hypotheses. This implies the requirement of finding a suitable and machine-understandable representation of semantic knowledge.

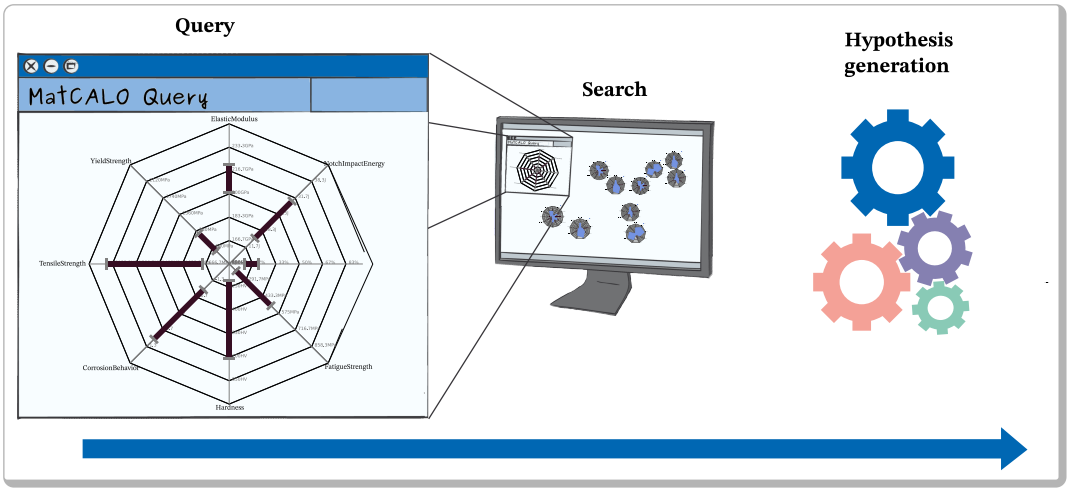
The scientific contributions of this work are therefore

- the investigation of materials science control structures and their processing in terms of an intelligent system that allows to query different aspects in order to find undiscovered relations
- the introduction of an approach to generate process plans for the targeted construction of novel materials
- an open-source prototypical implementation of the software MATCALO which combines this knowledge and provides access to it

An overview of research in the field of materials informatics is presented in Section 5.2. Section 5.3 covers the conceptual description of the research question that are tackled. The terms introduced in the scope of this work will be delineated the domains this work is restricted to are defined. Section 5.3.1 details the research question by formalizing the conceptual description before Section 5.3.2 presents the hypothesis generation by means of a representative example. The semantic representation of knowledge is addressed in Section 5.3.3. The current state of the prototypical implementation of MATCALO as well as information the interpretation of various visualizations is demonstrated in Section 5.3.4. A proof-of-concept evaluation along with the discussion of the results and an outlook to future developments can be found in Sections 5.4 and 5.5

High-level overview of the framework: the query entered by the user first triggers a search for candidate hypotheses. The hypotheses generation then uses additional semantic knowledge to refine the results.

Figure 40



5.2 State of the Art

While material discovery has been conducted purely empirically for thousands of years, only in the past few centuries the analytical investigation of materials emerged as noted by Agrawal and Choudhary (2016), aiming at formulating general laws of materials behavior.

Computer science contributed to this development by providing computational models and simulations in recent decades. Still, the field of materials informatics is rather new and gained relatively little attention in comparison to other scientific fields, in which computer science already features prominently since years and has become indispensable as stated by Rodgers and Cebon (2006). Popular examples are cheminformatics and bioinformatics which even form new scientific disciplines allowing to solve tasks that have been previously deemed infeasible. A famous example is the sequencing of the human genome, rendered possible by combining classical research in biology with computer science, machine learning, statistics, mathematics, and engineering technologies. This milestone in modern biology laid the foundation for the study of genetic diseases and the functional principles of biological processes. It is now possible to identify candidate genes responsible for certain tasks and understand their interactions in a biosystem. Ontologies representing certain aspects of biology, genetics and chemistry are heavily used for well-structuring and querying complex data.

Nørskov and Bligaard (2013) create the link to the materials science by describing the search for the properties of a material that define its functional properties as the quest for the 'materials genome'. Knowing the genome would rapidly speed up finding new catalysts for different purposes that reduce energy consumption and waste products and, eventually, build a sustainable chemical sector. There is not enough data to find all links between catalyst structures to rates of reaction conditions and that it is infeasible to build such a knowledge base. The authors rather propose the use of machine learning, which allows to find patterns in data and use them to make predictions. They also define the genome to be not only specified by underlying data but also a collection of machine learning methodologies such as analysis tools, search methods and learning algorithms to create synthetic data based on such predictions. Takahashi et al. (2019) introduce the concept of catalyst informatics and present it as the new generation of catalyst design methods. They propose three key concepts to establish catalyst informatics, 1) the catalyst data, which must be consistent and diverse while collected and organized following global rules, 2) the catalyst data to catalyst design, which deals with machine learning to predict (i.e. design) catalysts using patterns found in the data and 3) the platform for catalyst informatics which is expected to be publicly available and convenient to use in order to bring catalyst informatics forward.

Walsh (2015) raises the question on how to identify arrangements of the elements of the periodic table, whose number of possible combinations easily grow into millions, to efficiently produce interesting properties and names various examples for the successful use of materials prediction techniques.

Rajan (2015) argues that Big Data concepts are important to identify essential structure-property relationships but also points out that current efforts in materials informatics often focus only on increasing the volume of data, while being negligent of the remaining metrics velocity, variety and veracity. However, these metrics, combined with different tools of machine learning have shown to still help designing new materials dealing with uncertainty and sparse data. Rajan (2012) also names ensemble soft computing methods, being defined as fuzzy logic, neural network theory and probabilistic reasoning, as potentially superior to traditional 'hard' models. He argues that

they reveal novel information not explicitly present in the data instead of just finding the embodied relationships.

Takahashi and Tanaka (2016) discuss different aspects that need to be taken into consideration in materials informatics. This includes the collection of data, where they emphasize the necessity of collecting not only positive results but also negative and unwanted ones to allow machine-learning processes to reveal trends and behaviors which would be left undiscovered using only positive examples. They identify the inaccessibility of some databases as the largest issue when it comes to the development of a global knowledge base and express the requirement for global-standard data collection and database creation. The authors also identify the discovery of key descriptors representing materials properties as a crucial step in materials informatics and promote the development of web interfaces to make tools for material modeling, calculation and machine learning available to other researchers to create useful and accessible platforms to share knowledge and data and eventually develop materials informatics further.

Scientists propose using the Semantic Web to agree on a common structure of how data is represented allowing the broad community to easily access the findings of other scientists and reuse open software tools as described by Wang et al. (2016) and Taylor et al. (2006) and Casher and Rzepa (2006). A first step towards formalizing the chemistry domain is the *Gainesville Core Ontology*[28]

which was built to describe a typical Computational Chemistry experiment. The idea behind such approaches is to ease the advances in sharing information and therefore increase reproducibility and reusability of experimental findings.

Agrawal and Choudhary (2016) introduce the *fourth paradigm of science*, i.e. the data-driven discovery of novel materials that follows from the urge to handle complex and diverse data of materials science. The authors present popular predictive modeling algorithms and discuss the workflow of materials informatics by means of three examples where invertible PSPP relationships were learned.

There are already multiple approaches on using computational methods on material science approaches. Data-driven approaches such as the one presented by Gauthaml et al. (2011) attempting to predict materials properties based on combining materials informatics and physics-based modeling gain more and more attention while Seshadri and Sparks (2016) promote the design of databases relating to functional materials to accelerate the discovery and deployment of novel materials and propose the employment of a standardized, machine-readable (JSON-based) materials information file format containing information about material properties. The authors discuss the manual aggregation of data from literature in order to learn from the contained experiments and argue that the designed databases need to be searchable and interactive to be used effectively. They highlight the importance of appropriate visualization which gives a better insight in the collected data and present their own web services for their *thermoelectrics database*[29] (already introduced and explained in greater detail earlier by Gaultois et al. (2013)) and their *battery database*[30] allowing users to plot data with vari-

[28]Gainesville Core Ontology
[29]UCSB Thermoelectrics Database
[30]Utah Battery Database

able parameterizations. The authors propose to archive raw data from plots when future papers are published, to avoid the problem of having to re-digitize originally digital content from the publications. The possibility of aggregating data from crowd-sourcing is discussed which rises the question of data curation. Still, when employing the mentioned strategies for easing data aggregation, the authors leave open how to integrate already existing data contained in previously published literature.

Gaultois et al. (2016) present a *web interface*[31] that recommends novel thermoelectric compounds using machine learning techniques on a database of about 25000 prescreened materials. The tool is hosted on the website of *Citrine Informatics*[32], who claim to host the world's largest materials database that they also provide an online search engine for.

The *Materials Genome Initiative (MGI)*[33] was launched in 2011 to accelerate the discovery, development and deployment of novel materials to support industry and research in their work. Numerous agencies and institutes are involved in this initiative and support it with resources and infrastructure. Jain et al. (2013) introduce a project emerging from this initiative which deals with the development of various applications that support scientists in designing better materials. The applications use data mining technologies to compute materials properties such that scientists can target their research at promising compounds proposed by the system. The authors aim at a system that allows *rapid-prototyping* of materials simulated by computers to overcome the costly and time-consuming experimental development in the lab. Users can register for an account and use the provided tools in the *web interface*[34] or create an *Application Programming Interface (API)* key to access the data and integrate with their own infrastructure.

Janowicz et al. (2014) argue that the sheer amount and complexity of today's data sources call for an elaborate analyzation of the data in order to be able get meaningful insights. The authors hold the opinion that the Semantic Web tackles at least a subset of the challenges with such data and endorse its suitability and robustness for data-intensive science. In particular, put focus on the benefit of using Semantic Technologies by introducing a number of dimensions in which the need for semantics increases. They call these dimensions diversity, synthesis and definiteness. The diversity dimension includes the increasing heterogeneity of data and domains, uncertainty and variety of data formats while the synthesis dimension needs semantics in terms of preparing data for analyzing it in a meaningful way. The definiteness dimension uses semantics to introduce logical consequences in terms of ontologies which represent a data provider's perspective explicitly.

There are also discussions on how to effectively share data, addressing challenging topics such as how to deal with unstructured data and overcome differences in data structures from multiple sources as by Jain, Persson, and Ceder (2016).

[31] Citrination
[32] Citrine Informatics
[33] MGI
[34] Materials Project

An example for sharing data and research findings among research groups is *QUANTUM ESPRESSO*[35])], a publicly available toolbox provided by Giannozzi et al. (2009) who use computations of density-functional theory and plane waves to simulate materials in terms of electronic structure. The toolbox allows the user-friendly analytical modeling of quantum simulations.

Pizzi (2018) also discusses an Open-Science platform for materials science as being reliant on not only tools for data generation, but also a common platform to combine the strengths of the tools and integrate them into an easily accessible collective system as well as a data management strategy to allow sharing code and data.

Curtarolo et al. (2012) argue that empirical information on scrystal structures and properties is key in the development of novel materials but that databases containing such knowledge are incomplete, which calls for the development of computationally derived repositories to fill the gaps. They introduce the aflowlib.org library which provides collected information about phase diagrams, structures of binary alloys, electronic structures of inorganic compounds as well as properties of alloys. The library has since been extended to be accessible through a programming interface Taylor et al. (2014) allowing the community to accelerate the computational development of materials by constructing high-level workflows using the data in the repository. To allow the reproducibility of results reported by the AFLOWLIB consortium and to ease the extension of the database with results of materials science community researchers, Calderon et al. (2015) introduced the AFLOW standard for the high-throughput construction of the database.

Another platform that promotes putting useful tools for computational materials science at the materials science community's disposal is the pymatgen library introduced by Ong et al. (2013) which deals with materials data representation and analyses as well as the establishment of collaborative platforms and which is used as a library in the open-source Python framework atomate Mathew et al. (2017) which provides tools for simulations and analyses and facilitates calculations of materials properties (semi-)automatically.

Yip (2007) introduces the concept of *Computational Materials* in their *handbook* as a new discipline of computational research. Following their deliberations, it can be argued that computational methods and in particular, machine learning algorithms are key to further advance research in materials science. The use of traditional machine learning methods is well suited to learn adequately generalizable functions, given the provided data is sufficient for this task. Unfortunately, in materials science this is usually not the case, as data is collected predominantly from conducted experiments, which generally does not suffice as it will never cover an adequately large area of the search space. Additional background knowledge, the previously mentioned semantic knowledge, is eventually required which includes fundamental knowledge about the control structures of materials science and which can be combined with machine learning methods. The functions mentioned above should not only explain the available data it was trained with, but should also allow making reliable statements about new data points, i.e. about the unexplored areas one is interested in. In particular, one

[35]QUANTUM ESPRESSO

cares about a function that gives information about which areas are promising with respect to the specified requirements and which will most probably not yield materials with the desired properties. In the following, the approach will be further detailed along with an explanation how the abovementioned challenges are intended to be tackled by incorporating semantic knowledge.

5.3 Conceptual Framework

In this work the focus lies on structural materials and examine a small set of metal alloys that are investigated thoroughly in literature. In particular, different alloys in the system Fe-C-Cr at this point are used, with Fe-C-Cr-Mn and some Al-Basis alloys being added in later stages.

The *material properties* one is interested in are common properties characterizing metal alloy features such as the hardness, yield strength, elongation and corrosion behavior of the respective alloy.

A sample in the context of this work may either be *microscopic* or *macroscopic*. A macroscopic (*macro*) sample can be used to obtain material properties directly because of its size. Macro samples can for example be created by using spray-forming techniques Ellendt, Uhlenwinkel, and Mädler (2014) or casting. A microscopic or *micro* sample is a spherical droplet of a few hundred μm in diameter. These samples are either generated in large quantities in one go using the pneumatic high-temperature droplet generator as described by Ellendt et al. (2016) or they are formed in a larger-scale sample using Laser Deep Alloying. They can be characterized with established techniques such as nanoindentation measurements where the results allow the prediction of the properties like Young's modulus or hardness of larger (macroscopic) samples as described by Ciftci et al. (2014). The development of reproducible high-throughput techniques for the short time sample characterization is vital for using the results in the microscopic level results to predict actual material properties of larger-scale samples. The idea is to use predictions of material properties based on short-time characterizations on micro samples and generate few sample points from the macro samples to validate the results. This of course requires the macro samples to be comparable with the micro samples in terms of their respective microstructure, i.e. an appropriate synthetizing and treatment needs to be found as stated by Ellendt and Mädler (2018).

high-throughput characterization ▷

The term descriptor is used to denote any kind of measurement or feature obtained from such characterization. They correlate with material properties so that a short-time characterization of a micro sample facilitates the prediction of material properties in macro-scale materials.

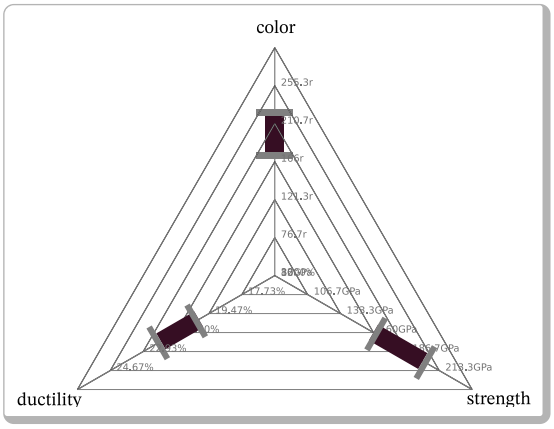
The materials are subject to different kinds of treatments which are called *processing steps*. A processing step can either be exclusively *descriptor-determining*, i.e. the treatment is only made to obtain descriptors of the sample or *coloring*, i.e. supposed to manipulate the sample's microstructure in a certain way. Examples for coloring steps are mechanical treatments such as milling, which changes the form of the material or thermal treatments such as heat treatments which change its internal structure. A de-

scriptor-determining process may also have a coloring effect on the material, yet only locally. Still, this process is still called descriptor-determining as this is its predominant purpose. In the most extreme case a sample is destroyed in the process as can happen for example with chemical treatments. Such a process will be called *destroying*.

In general, a sample undergoes multiple processing steps throughout its lifetime such that a *process chain* of both descriptor-determining and coloring steps is generated, possibly with a final destroying step. These process chains are examined with respect to the effects the single steps and their parameterizations have on the material's properties. These observations are then used to train models representing process-property relations and of course mappings of descriptors to actual material properties.

To use such models for inference, the user of the system defines characteristics, i.e. a set of property-value mappings the desired material should meet, called a *requirement profile*. This can be achieved using a graphical user interface that allows the selection of certain properties and assigning values or value ranges to them (see Figure 41). Using this profile, the system then searches the database to find a material that somehow matches this description either in all properties or using a predefined similarity measure that represents the contentment with the result. The process steps that led to this result are retrieved and used as initial hypotheses.

Requirement profile for the three properties *color*, *ductility* and *strength*. | *Figure 41*



This work builds upon two main pillars which is (1) the data driven inference and learning of probabilistic models based on experimental data and (2) the engineering of models representing semantic knowledge that can be learned from experience and literature. This work therefore consults additional sources of input data to find the beforementioned correlations and learn about the fundamental control structures behind the materials science. In particular, the focus lies on semantic knowledge that can be represented in a machine-readable fashion such as ontologies. An *ontology* generally consists of two parts, the TBox (also called taxonomy), comprising the terminological axioms defining categories of objects in a domain with their respective relations among each other and the ABox, containing the assertional axioms describing the world using definite instances (or individuals) of concepts in the TBox (Baader and Nutt 2003).

◁ *learn from external knowledge and experience*

WordNet[36] is a lexical database, structuring more than 117.000 concepts of the English language in terms of *is-a* relations, such that it can serve as an example for an upper ontology, limiting the possible relationships to be predominantly (but not exclusively) hierarchical.

The fundamental advantage of using ontologies is that one can generate hypotheses based on the similarities represented therein. In particular, one is able to find substitutions for materials or processes that are not available at the time or for some other reason not applicable. Section 5.3.3 will describe in more detail how semantic knowledge will be used in terms of ontologies.

The experimental data the models are trained with are assumed to contain the following information:

- the initial state of a material (i.e. its properties *before* conducting any experiments)
- the process steps that were conducted on the materials including their respective parameterization
- the final state of a materials (i.e. its properties *after* conducting the process steps)

The experiments are either descriptor-determining processes which exclusively analyze samples to identify their descriptors or 'coloring' processes which change a materials' microstructure through thermal, mechanical or thermo-mechanical action. Multiple processes can be conducted consecutively, forming a process chain associated with one or more property transformations of the treated material.

5.3.1 Problem Formulation

The purpose of MATCALO is to support materials scientists in their work of finding novel materials in an informed fashion to bypass time- and labor-intensive experiments.

To achieve this, a data set is required that contains process chains of which the exact outcome is known and which can be used to train probabilistic models that generalize well enough to make reliable statements about how single process steps and influencing factors drive the development of certain properties. For the formulation of the query a web interface allowing to define the desired characteristics of the new material will be provided. The input can be seen as a set of material properties each of which has a defined interval its value ranges in. As defined above, such properties can be something like the hardness or corrosion behavior. This input will be used to query the system and generate rated hypotheses, where a hypothesis in the context of MATCALO is a proposition about the processing chain a material has to undergo in order to develop a set of desired properties.

As a simple example, assume ther are only two processes available that can be conducted on a material. The first one is a thermal process, in which the material is heated at a given temperature. The second one is a mechanical process, indicating that the material is mechanically treated with some force. These processing steps have a certain

[36] WordNet

influence on the properties of the material. The first property is the *color* with values ranging from 40 to 250, where low values indicate a blue *color* and high values indicate a red one and which can be interpreted as a color gradient from blue to red. The second property is the *strength* that can take values from 10 to 20. Keep in mind that the output of a process may not only depend on the applied force or temperature which is given as a process parameter but also on the previous state of the material, i.e. which *color* or *strength* it had before conducting the treatment. Such dependencies can be modeled as additional 'indirect' process parameters. The models are trained for the two processes such that the system learned how the treatments influence the state of the material.

Looking for a way to produce a red and strong material, a query in MATCALO can be formulated to find out which chain of subsequent mechanical and thermal treatments will leave us with, say, a *color* value of > 200 and a *strength* value > 15.

Technically, MATCALO will trigger the compilation of a new process chain that is likely to produce the desired output. This includes the types of processing steps along with their respective parameterizations and the order in which they have to be conducted, as well as a measurement indicating the quality of the proposition.

More formally, $\mathcal{D}$, $\mathcal{P}$ and $\mathcal{A}$ are defined as sets of symbols for descriptors, properties and the arguments of process steps, respectively and introduce the set of feature symbols $\mathcal{F}$ as the union of descriptors and properties. As it is dealt with two different types of descriptors, $\mathcal{D}$ is divided into separate sets $\mathcal{D}_m$ and $\mathcal{D}_\mu$. Accordingly, the features are denoted as

$$\mathcal{F}_m = \mathcal{D}_m \cup \mathcal{P} \tag{93}$$

and

$$\mathcal{F}_\mu = \mathcal{D}_\mu \cup \mathcal{P}. \tag{94}$$

A *feature profile* φ is defined here as an exhaustive assignment of features to ranges of admissible values, i.e.

D

$$\varphi : \mathcal{F} \mapsto \overline{\mathbb{R}}^2, \tag{95}$$

where $\overline{\mathbb{R}}$ denotes the extended real number system $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$. Let $\mathcal{I}$ be the interval function, $I : \overline{\mathbb{R}}^2 \mapsto \mathbb{I}$, with $\mathbb{I} \subseteq \mathbb{R}$ being the set of all continuous real-valued intervals, and $I(a,b) = \{x \in \mathbb{R} | a \leq x \leq b; a, b \in \mathbb{R}\}$. A *colored state* σ of a material is an exhaustive assignment of features to concrete real values,

D

$$\sigma : \mathcal{F} \mapsto \mathbb{R}. \tag{96}$$

A colored state σ is said to *satisfy* a feature profile φ, in symbols $\varphi \vDash \sigma$, if and only if all values of S lie in the respective admissible values of φ:

D

$$\varphi \vDash \sigma := \begin{cases} \top & \text{if } \forall f \in \mathcal{F} : \sigma(f) \in \varphi(f) \\ \bot & \text{otherwise.} \end{cases} \tag{97}$$

A *requirement profile* ϱ is a feature profile for which the following holds:

$$\forall d., (d \in \mathcal{D} \Rightarrow \varrho(d) = \langle -\infty, +\infty \rangle). \tag{98}$$

i.e. a requirement profile postulates effective bounds only for material properties, but allows descriptor variables unconstrained. Let Φ denote the set of all feature profiles, Σ the set of all colored states and P the set of all requirement profiles. A single *process step* π can be viewed at as a function transforming a material's state into a new colored state

D

$$\pi_a : \Sigma \mapsto \Sigma, \tag{99}$$

which is controlled by the process' arguments $a \in \mathcal{A}$. Consequently, a *process chain* ζ of length N is defined as a composition of process steps

D

$$\zeta : \Sigma \mapsto \Sigma, \ \zeta(x) = \left(\pi_{a_N}^{(N)} \circ \cdots \circ \pi_{a_1}^{(1)}\right)(x). \tag{100}$$

Let $\Omega \subseteq \Sigma$ be the set of *original states* of materials ("Urform"). Given a requirement profile ϱ, the ultimate goal is to find a processing chain ζ and an original material $\omega \in \Omega$, such that the final material state satisfies the requirement profile:

$$\varrho \vDash \zeta(\omega). \tag{101}$$

handling uncertainty ▷

The manufacturing and processing of materials is subject to high uncertainty. This uncertainty originates mainly from noisy measuring techniques and limitations in the accuracy of controlling the manufacturing process. In addition, there is a lack of accurate and reliable analytical models that allow the exact prediction of the outcomes of precessing steps. It is therefore a necessity to take into account the uncertain nature of the problem and make it explicit in the models. As the acquisition of predictive models from data generated by novel high-throughput methods is intended, probabilistic regression models on the level of individual processing steps π are learnt,

$$P(S_{i+1}|A_i, S_i), \tag{102}$$

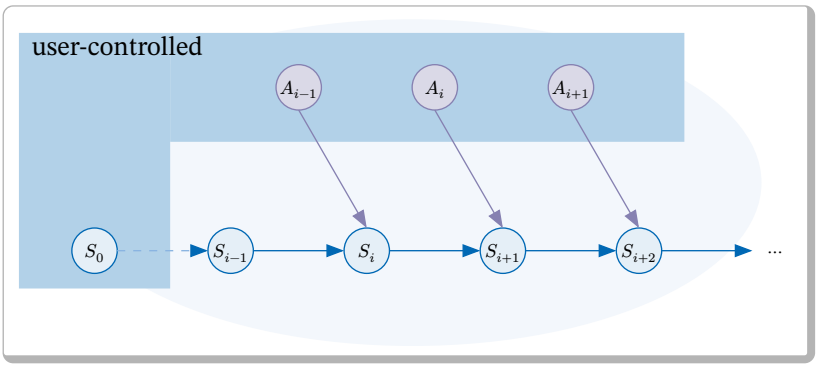
which represents the probability distribution over colored states S_{i+1} that are obtained from processing a material in state S_i with process parameters A_i (see Figure 42). Note that S_{i+1} and S_i here denote random variables whose domains are the space of all col-

ored states Σ. It is assumed that the resulting colored states are conditionally normally distributed, i.e.

$$S_{i+1} \mid A_i, S_i \sim \mathcal{N}(\mu(A_i, S_i), \Xi(A_i, S_i)), \tag{103}$$

i.e. for every pair $\langle A_i, S_i \rangle$, there is a mean vector μ and a covariance matrix Ξ that determines the normal distribution of S_{i+1}.

Schematic representation of a process chain including state variables S_i and actions A_i. | *Figure 42*



Material properties may not be measured directly but in terms of descriptors, therefore another function ψ is required, mapping a descriptor to an actual property:

D

$$\psi : \mathcal{D} \mapsto \mathcal{P} \tag{104}$$

Using the definitions above, one can now search for the most probable sequence of process steps along with their parameters that leads to a given requirement profile:

$$\underset{\pi^{(N)}_{a_N}, \ldots, \pi^{(1)}_{a_1}}{\arg\max} \quad P\left(\varphi \vDash \sigma \mid \left(\pi^{(N)}_{a_N} \circ \cdots \circ \pi^{(1)}_{a_1}\right)(\omega) = \sigma\right) \tag{105}$$

with φ being the requirement profile and ω the initial state of the material.

5.3.2 Hypothesis Generation

For the hypothesis generation multiple regression trees are trained each of which represents one coloring step. A regression tree is a decision tree in which the value to be predicted can be continuous. In general, decision trees are predictive models that are structured in a tree-like fashion. Every inner node, i.e. a node that is not a leaf node, represents an input variable that is connected to its child nodes through edges, each of which denotes a possible value (or value range) of that particular variable. Each leaf node represents the predicted value of the target variable given the values of the input variables along the path from the root node to this leaf node. In the modeling presented here, the feature nodes represent the parameterization of the coloring step and the material's property values before conducting the process. Accordingly, a path

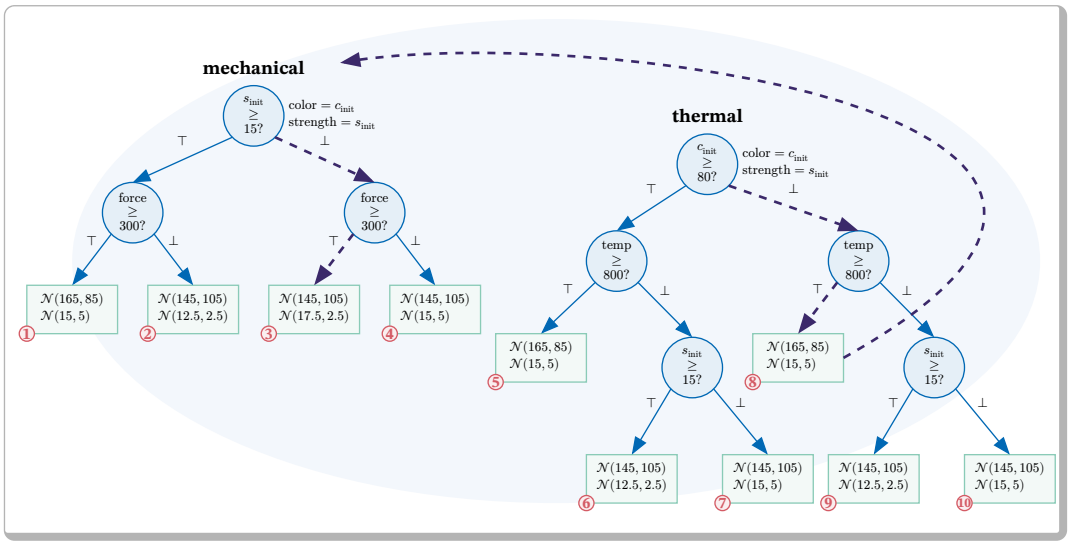
from the root node to a leaf node represents the transformation from one property set to another using the respective parameterized coloring. The goal is to find out what steps are to be performed to get to a certain set of properties or a particular *state* of a material. Therefore is it necessary to find a result in the *leaf* nodes of the trees that matches the query best and search backwards to the root to obtain the parameterization that lead to these properties, i.e. to look for the *result* of a certain action. In particular, a *reverse* decision tree inference is performed that results in a state of the material that has possibly been reached through one or more further coloring steps and therefore can itself be used as a query for further inference. An inference using this approach can therefore be understood as multiple reverse decision tree inferences being performed until a root node representing an initial coloring (manufacturing) step is reached. A pseudo-code representation of this approach is consolidated in Algorithm 9.

The example can be used with the two processing steps previously introduced in Section 5.3.1 to illustrate the process. Assume the models (i.e. regression trees) the system learned look like the ones shown in Figure 43.

Following the paths in the trees below from the respective root node to their leaf nodes the following rules can be derived:

- Applying high temperatures change the *color* of a material from blue to red, while
- high force either generates a red *color* if the material was already strong before or increases the $strength$ and
- low temperatures as well as low force treatments decrease the $strength$.

Decision tree example: Simple example for decision trees representing two different processing steps: (1) a *thermal* process which has only one parameter indicating the temperature of the heat treatment and (2) a *mechanical* process with one parameter *force*. In this small example, a material has only two properties; the *color* of the material can take values representing colors from red to blue, while the $strength$ value can be interpreted as *low* or *high*. The leaf nodes represent the multivariate distribution, which is denoted as two separate distributions for the *color* (upper) and $strength$ (lower), for simplicity. | *Figure 43*



Multiple Reverse Tree (MRT) Inference | *Algorithm 9*

MRT($\varrho, trees, t$):

Input: ϱ, a query
trees, a sequence of tree models
t, a sequence of hypotheses with confidence

Output: *hypotheses*, a sequence of tree models

```
1  hypotheses ← ∅
2  GENERATE-PATHS(∅, ϱ, trees, hypotheses)
3  return hypotheses
4
5  function GENERATE-PATHS(chain, ϱ, trees, results)
6      ϱ′ ← UPDATE-QUERY(ϱ, chain, trees)
7      candidates ← GENERATE-CANDIDATES(ϱ′, trees)
8
9      for each c in candidates do
10         chain′ ← MAKE-QUEUE(c, chain)
11         conf ← CALCULATE-CONFIDENCE(chain)
12         if conf < threshold then
13             continue
14         end if
15         if ϱ′ ⊨ chain′ then
16             results ← INSERT((conf, chain′), results)
17         else
18             GENERATE-PATHS(chain, ϱ′, trees, results)
19         end if
20     end for
21 end function
```

The leaf nodes in the figure are annotated with numbers, such that they can be referenced in the explanation of this example. Now a query is formulated that can be used to infer the necessary processing steps and parameters to generate a material with the desired properties. Assume that for some reason it is desirable to have a material that is red and strong, i.e. $\{\text{color} = 200, \text{strength} = 18\}$. To keep the example simple, another limitation is introduced by defining that the material in its original (unprocessed) state is blue and not very strong, i.e. $\{\text{color} = 210, \text{strength} = 53\}$. This will reduce the possible candidates to a manageable number. In a first step, all those leaf nodes, that comply with the defined query are identified, i.e. requirement profile. Each leaf

node represents a multivariate distribution over the value ranges of the properties. In this example, this is a Gaussian distribution but any other distribution representing the values of certain properties appropriately could be used. For reasons of simplicity two independent distributions in the leaf nodes are shown, each of which represents one of the two properties to facilitate the understanding of the reasoning in this example.

Using these distributions, the similarity of a certain node to a given requirement profile can be quantified by calculating the probability that any of the values in the intervals given this distribution are reached. In other words, the CDF as the integral over the PDF of the given distribution is determined.

In this example, that are all those particular nodes that not only have the mean of the *color* distribution at 165 and/or the mean of the strength distribution at 17.5, but also those ones that represent the *a priori* distribution with the means at 145 and 15 respectively. The a priori distributions can be interpreted as that the process does not affect the respective property, i.e. the initial values are left untouched. This would leave us with a list of candidates containing the nodes $1, 3, 4, 5, 7, 8$ and 10. Only nodes $2, 6$ and 9 can be ignored as they will most likely produce a material that fails to meet the criteria in at least one of the two properties.

However, for this example the focus lies on those nodes that are closest to match the requirement profile in terms of at least one of the properties, which in this case are the nodes $1, 3$ and 8. Following the path up to the respective root nodes now gives information about how this property value was achieved. In case of Node 3 high force would have to be applied for an initially weak material to achieve high strength, while for Node 1 and 8 a red *color* is achieved if either high temperatures to a previously blue material or high force to an already strong material is applied, which complies with the abovementioned observations.

In all three cases, however, only one of the defined property requirements are met so far. Furthermore, the additionally defined restriction that the original material was initially weak and blue, is also not fully satisfied for any of these nodes alone, therefore applying only one of the respective processes will not produce the desired result. Obviously, another process has to be conducted in order to manipulate the respective second property as well. In other words, whichever node is chosen as the 'starting point', another state is produced, that is subject to another query in order to find out how to achieve it.

To illustrate this, follow the path from Node 3 to the root of the tree. This node can only be reached by applying a high force (≥ 300) on a previously weak material ($s_{\mathrm{init}} < 15$). There is no information about the initial *color* of the material (c_{init}) and the distribution in the node ($\mathcal{N}(145, 105)$) indicates that this property can take any value of its value range, which can here be interpreted as it being left unchanged, i.e. whichever value this property had before conducting the force it will have the same value afterwards. In this case, starting with a blue material, following this path would end in a state where the material is strong but most likely still blue.

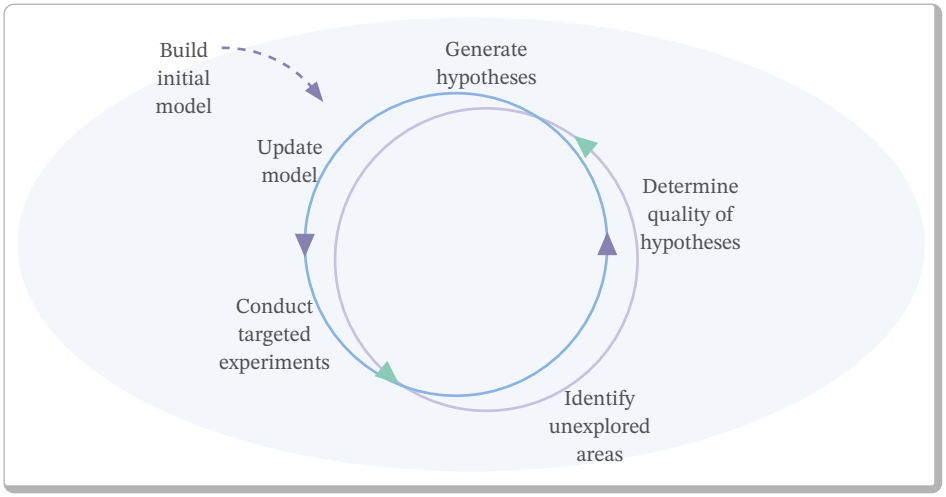
But how can the red *color* be achieved? One approach would be to try to find another process that leaves us with a red material that ideally does not affect the strength and

conduct this *prior* to the mechanical treatment. In other words, finding an output Node x among all the candidate leaves that tells us how to get there is necessary; therefore a a new query is defined in which all the preconditions of the currently considered path are required properties for the new search as the output of x represents the *input* of the process leading to Node 3.

This leaves us with nodes 1 and 8. Node 1, as a precondition, requires the material to be strong already, which the material initially does not satisfy, so this node is skipped for now and Node 8 is investigated. This node can be reached when the material was initially blue ($c_{\text{init}} < 80$) regardless of the strength (s_{init} unmentioned along the path and $\mathcal{N}(15, 5)$) and represents the desired red *color* output ($\mathcal{N}(165, 85)$). This node is therefore chosen as the precondition of the mechanical process can be fed to the heat treatment. To summarize, there is now a precondition of a blue and weak material (which is satisfied) and conducting the two processes in a chain will most likely produce a red high-strength material as required, i.e. now a state in which all the specified criteria are satisfied is reached.

There is yet to mention that even if this was identified as a possible hypothesis on how to generate the desired output, other solutions are still possible. Without going too much into detail, another possible solution would be to apply a high-force mechanical treatment on the material two times, which will increase the strength in the first treatment (Node 3) and then generate the red *color* in the second (Node 1).

Development loop: The initial models are trained with data from only few sampling points generates hypotheses that are evaluated in terms of the quality of the results and used to find weaknesses. These weaknesses are then tackled by using results of targeted high-throughput experiments in micro-scale to retrain and improve the models. | *Figure 44*



Solutions may contain loops of processes which keep on producing the same result which is why a penalty is used for long processing chains incorporating unnecessary (duplicate) steps and therefore define a strong preference for the shortest and simplest solutions.

By virtue of the large search space, there may be areas that are very well understood because they have been studied extensively and are well-described in literature. Queries

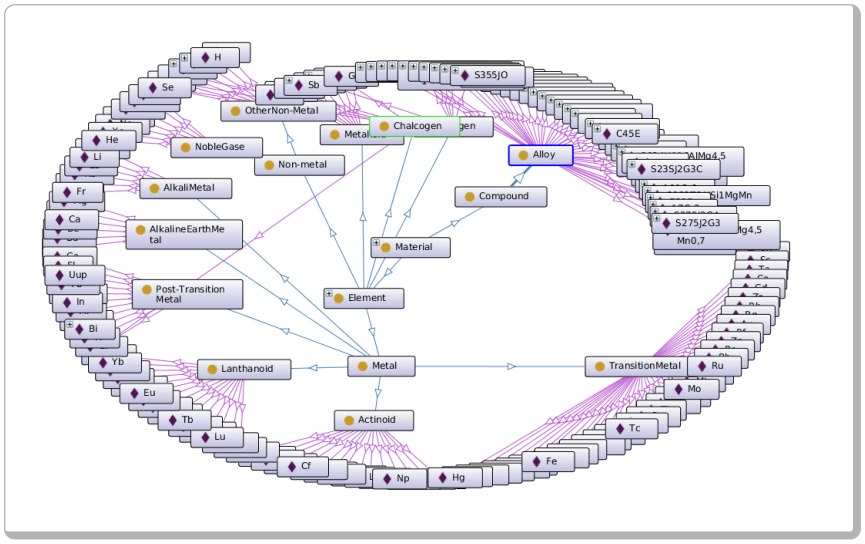
that lie within these areas will naturally produce reliable, higher-confident hypotheses. On the other hand there may be large fields that are left unexplored so far such that there are not enough data points to make precise statements. It is therefore crucial to augment the generated hypotheses with a measurement indicating the system's confidence in that proposition. A low confidence then allows to not only identify weak hypotheses, but also find areas that might be promising to explore further in terms of additional experiments to improve the trained models. Unexplored areas can be identified by investigating the standard deviations of the distributions in the leaf nodes.

The standard deviation allows to draw conclusions about the size of the value interval a property can range in following that particular process path. A large interval of values itself can be taken as evidence that the system was not able to make precise propositions and that more data in this specific value range is required. High-throughput methods for creating and testing micro samples allow to generate data with large variation in comparatively short time, which allows us to fill these gaps quicker than with conventional methods. By re-evaluating the trained models, identifying unexplored areas and generating targeted data on a regular basis allows to accomplish a closed loop in which the system is improved continuously (see Figure 44).

5.3.3 Semantic Representation

An ontology can be an excellent tool to represent the knowledge that is hidden in literature and regarded as 'commonly known' in the respective field of research but is hard to grasp and difficult to apply to complex data structures. Designing ontologies can be tricky, as there are many different ways to model relations between objects on a conceptual level. Depending on the chosen design and the type of question asked, performing inference on an ontology may be hard if not computationally infeasible.

Periodic table ontology: Small excerpt of the current state of the ontology comprising the taxonomic relations of the elements of the periodic table as well as a collection of steel-based alloys along with their respective properties. | *Figure 45*



Ontologies have been used widely in other research areas and are gaining interest in the field of materials science and in materials and catalyst informatics in particular. Takahashi, Miyazato, and Takahashi (2018) introduce them as a tool to potentially overcome difficulties that arise when using multiple databases from different researchers incorporating different terminologies and data types. The authors argue that ontologies can be used to add more semantics to scientific data by defining relationships between concepts that are typically overlooked and develop an ontology containing information from the periodic table that can be used to search for descriptors.

MATCALO uses ontologies to add another layer of knowledge. The used ontologies are represented using the W3C Web Ontology Language (OWL)[37] which is the most commonly used knowledge representation formalism. It is standardized, used within a large community and there exist multiple software programs and libraries to visualize, modify and query OWL ontologies.

◁ *external knowledge sources*

An individual in MATCALO is grounded in an ontology, i.e. it is identified by concepts in the ontology which gives it a globally unique and well-defined semantics that can be intercorrelated to other objects in the ontology - and therefore in all system components of MATCALO. Through this grounding in the ontology, it is now possible to determine a concepts' 'similarity' to other concepts and therefore allow for substitutions of similar materials or processes. To this end, the ontology is designed manually using the ontology editor *Protégé*[38] (Musen 2015), which provides visualizations for the class hierarchy, tools for importing data and a graphical user interface for the management of ontologies of different syntax types. It is structured purely hierarchical but will be extended to represent more sophisticated and informative relations between the entities as current research includes the learning of relations that can be represented in OWL such that eventually, the ontologies will be developed in a semi-automatic approach.

At its current state the ontology comprises mostly taxonomic relations of

- ▷ Materials (mostly based on the classifications in the periodic table)
- ▷ Properties (e.g. corrosion resistance, hardness)
- ▷ Treatments (i.e. processes such as electrochemical, mechanical, thermal treatments)

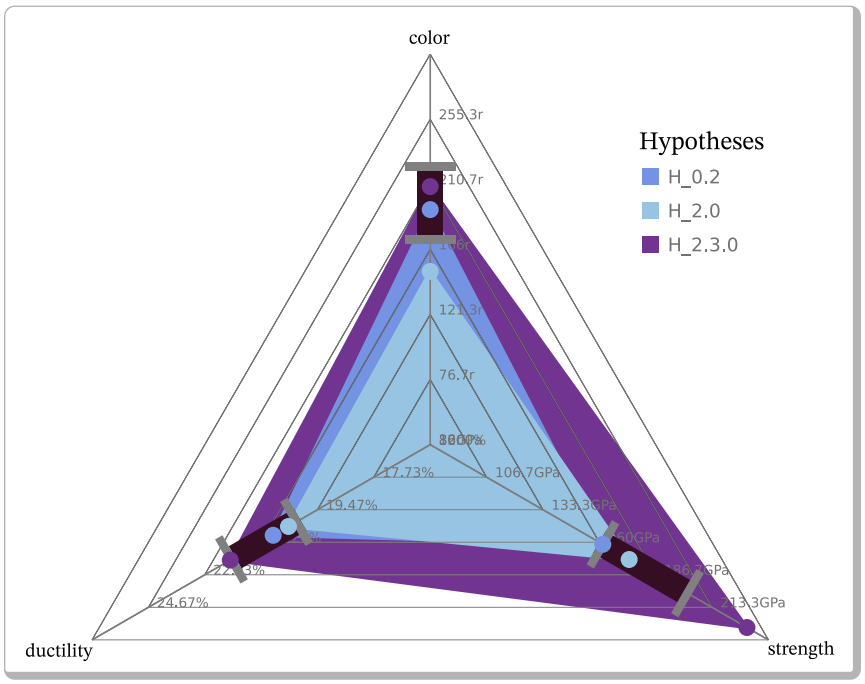
(see Figure 45). Future plans for the ontology include adding relations describing more sophisticated semantics. This is important as the similarity between elements is not suitably well represented by their respective proximity in the periodic table. There are rather much more complex relations between certain elements, element groups or alloys that play an important role when comparing them. It is also imaginable to represent physical conditions and material structure properties that possibly allow the deduction of the material's behavior based on the knowledge of a similar material. The knowledge about such relations is not easy to grasp as it is formed by the experience of proficient scientists and often fades into the background when it comes to data science and modeling. Still, a lot of those findings have been published and can therefore be found in various types of literature, be it papers, textbooks or online sources.

[37] OWL
[38] Protégé

5.3.4 System Architecture & Interface

The MATCALO system is written in the Python programming language and is compatible with Python version 3. Standard scientific programming libraries such as numpy, scipy and sklearn are used in multiple components of the system and are partly extended to provide the functionalities that are required for the MATCALO system such as the inference algorithm for the generation of the regression trees. The MATCALO code will be made available under the *MIT License*[39] on github. The licensed package will comprise the MATCALO source code, the synthetic data presented in this paper as well as a clear and concise documentation. The documentation as well as the modularly designed code which facilitates code reuse will allow users to integrate their own data and/or code. In future releases an API is planned to be added to provide access to the functionalities of MATCALO for the community. By making the source code publicly available and providing a web interface to benefit from this research the paradigm of open science is followed, which focuses on the transparency and accessibility of knowledge. Scientific communities are encouraged to share not only their findings in scientific publications but also provide tools and data to reproduce results and integrate with their own data.

Hypotheses radar chart: The generated hypotheses' results are loaded as datasets in a radar chart that corresponds to the requirement profile. The user can now visually investigate the quality of the hypotheses by comparing how close the datapoints match the required value intervals. | *Figure 46*



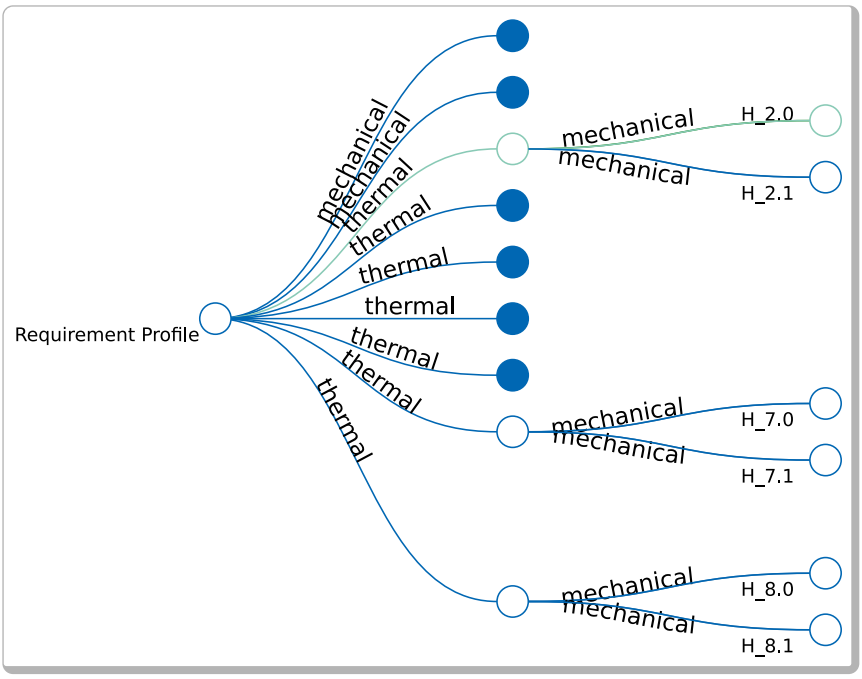
MATCALO can be accessed as a *web service*[40], of which a prototype is already publicly available. The web service is implemented using the *pyRAP*[41] framework which was

[39] MIT License
[40] MatCALO
[41] pyRAP

co-developed by Daniel Nyga and which allows building AJAX applications in Python, therefore easing the integration with the MATCALO system. In its current state the web service allows the user to select a dataset from a set of examples and define a requirement profile. This requirement profile is represented as a radar chart, of which each axis represents one property of interest (see Figure 41). Axes can be added or removed using the context menu. Each axis of the radar chart shows a red bar that can be stretched or shrinked by dragging the end points along the axis accordingly. The bar represents an interval of values for that particular property that the user defines as a requirement for the material. A complete requirement profile is therefore defined after adjusting the intervals for each property, creating a mapping from all properties of interest to intervals of accepted values.

After querying the system with the defined requirement profile, the generated hypotheses will be listed in a table and loaded into a radar chart corresponding to the requirement profile. Each colored polygon in the radar chart represents one hypothesis (see Figure 46). A hypothesis might match some of the criteria very well, but will fail on others (see $H\_2.0$ for property *color* or $H\_2.3.0$ for property $strength$). This visualization can be used to examine the quality of the hypotheses' results by visually inspecting if the single data points lie in between the respective interval defined for that particular property.

Hypotheses tree visualization: The tree visualization allows to investigate the generated hypotheses further. Each path from the root node to a leaf node represents a process chain producing the desired results. Hovering over the edges and nodes in the web service will show additional information about the respective process chain. Filled circles in the tree visualization denote that the branch has not been fully expanded yet. | *Figure 47*



After deciding which hypothesis might be the most promising, the user can visualize the process chain that leads to the predicted result of the hypothesis by generating a

tree as in Figure 47. Each hypothesis (i.e. process chain) corresponds to one path along the tree, each edge being one processing step as indicated by the appended text. The tree itself can be seen as a condensed version of the search tree that was built during the inference process, containing only the hypotheses in the inference results. The tree can also help to find the best solutions when some of the hypotheses seem to be of equal quality on the first sight. When two hypotheses have the same confidence and an equal or similarly good output, they might still differ in the length of the process chain that is to be conducted to achieve the result. The user is then advised to have a closer look at the chains and rate them using further metrics such as costs per processing steps, process time or applicability. The path to the selected hypothesis is highlighted in green ($H\_2.0$ in Figure 47). Additional information about (1) the exact hypothesis result, (2) the similarity of the hypothesis result to the requirement profile and (3) the parameters of the respective processing steps will be shown when hovering over the elements of the tree in the web service.

5.4 Experiments

An experiment was conducted designed to showcase the reverse tree inference generating more complex hypotheses to demonstrate the capabilities of the MATCALO system as a proof of concept. The experiment targets the capability of the algorithm of finding paths along *multiple* trees, therefore creating more complex process chains to produce a desired output.

The working hypothesis is governed by the assumption that there exist relationships between processing steps (and their parametrizations) and the resulting materials' properties. They are crucial for developing novel materials and this experiment showcases that the MATCALO system can find such relationships contained in (here: manually generated) data. This is to show the actual reverse-tree inference capabilities of the system which proves that the computational processes of the developed system are functioning.

The data for this experiment represents a setting using real processes for spherical steel alloy samples in comparison to the fictional running example in Section 5.3.2.

Two different trees are trained representing a deep rolling (cmp. Figure 90) and a thermal (cmp. Figure 91) treatment. While the deep rolling process is only guided by the *pressure* as a parameter to predict the three targets *deformation*, *density* and *dislocation density*, the thermal process predicts the targets *deformation*, *dislocation density* and *hardness* using *temperature* and *deformation* as inputs. The *deformation* is the ratio between the length and width of the sample, i.e. a alue approximating 1 represents no deformation, as the samples are sperical. The synthetic data contains relationships such as the finding that after exceeding a minimum degree of deformation, re-heating the sample over roughly 750°C causes recrystallization which reverts the foregone deformation. The order in which the processing steps are conducted may therefore play an important role if their respective results (partially) cancel out each other. In this case, the previous deformation of the material influences the results of the thermal

treatment. The discussion in Section 5.5 will show, which role these information play in the interpretation of the generated hypotheses.

5.5 Results and Discussion

Three different requirement profiles were chosen to highlight the capabilities of MATCALO as shown in Table 5. Each query contains the variables *density*, *deformation* and *hardness* with varying value intervals.

The first query produces three hypotheses, each of which requires a deep rolling process to be conducted first in order to achieve the desired density followed by a heating step that produces the desired hardness and may or may not override the foregoing deformation. Note that the two hypotheses *H_d2.h9* and *H_d2.h11* have a probability of 0, which means that their results would match the requirement profile according to their predicted outputs, but the succession of the steps are not executable in reality.

In particular, step *d2* which corresponds to the leaf node 2 in the deep rolling tree in Figure 90 and which is the first step for each of the three generated hypotheses will presumably cause a comparably obvious deformation (2.43). Depending on the deformation, however, the following heating process will produce different outputs as can be seen following the paths from the root node to the leaf nodes 9, 11 and 12 in Figure 91 which correspond to the steps *h9*, *h11* and *h12* of the hypotheses, respectively.

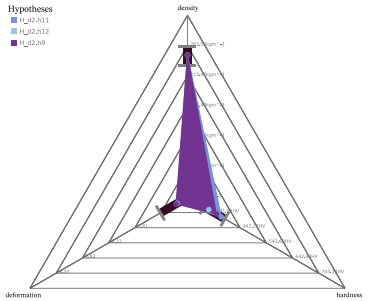
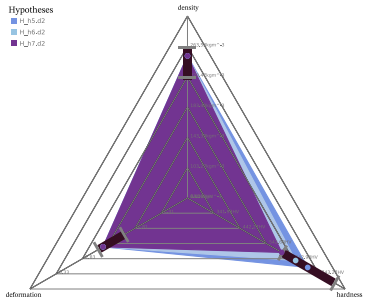
With a deformation value of 2.43 only leaf 12 can be reached, rendering the hypothesis *H_d2.h12* the only possible hypothesis for this query. The second query shows that the system only generates valid hypotheses, i.e. ones that

- produce a result containing each query variable from the requirement profile with a value that lays in the defined interval
- is executable in reality, i.e. each step is a valid successor of its predecessor such that the preconditions for the successor step do not collide with the results of the predecessor.

No combination of the deep rolling and heating steps with their possible parameterizations according to the trained trees render this query possible, therefore the system generates no hypotheses.

The third query is similar to the second one, but allows for a higher deformation of the material which can be facilitated in three different ways. Again, the system shows a strong preference for specific order of the processing steps (a heating step followed by a deep rolling step) because none of the parameterizations of the heating steps produces a deformation in the demanded interval which requires a deep rolling step to 'correct' the value. This example shows that the correct order of processing steps can be found not only found if the output of one step is preconditioned on a specific value of a variable that is an output of another step. In fact, the system also allows steps that violate one or more variables of the requirement profile at first if it finds compensating steps to restore the desired output and therefore can generate a valid hypothesis whose overall output matches the query.

Results of a selection of 3 example queries; each executed using a threshold of 0.0. The plots show how well the results of the respective hypotheses match the entered query. | *Table 5*

Query	Hypothesis	Result	Probability	Steps/Parameters	Plot
density: [235, 260] deformation: [0.8, 1.3] hardness: [300, 380]	H_d2.h9	density: 251 deformation: 1.02 hardness: 352 ddensity: $3.5e+09$	0.0	Deep Rolling: {pressure: $]300, \infty[$} Heating: {temperature: $]750, \infty[$, deformation: $]-\infty, 1.01[$}	
	H_d2.h11	density: 251 deformation: 1.0 hardness: 370.67 ddensity: $2.0e+09$	0.0	Deep Rolling: {pressure: $]300, \infty[$} Heating: {temperature: $]750, \infty[$, deformation: $]1.01, 2.01[$}	
	H_d2.h12	density: 251 deformation: 1.01 hardness: 323 ddensity: $3.2e+09$	0.37	Deep Rolling: {pressure: $]300, \infty[$} Heating: {temperature: $]750, \infty[$, deformation: $]2.01, \infty[$}	
density: [220, 260] deformation: [0.8, 1.0] hardness: [600, 800]	-	-	-	-	-
density: [220, 260] deformation: [2.0, 2.5] hardness: [600, 800]	H_h5.d2	density: 251 deformation: 2.43 hardness: 701 ddensity: $5.6e+11$	0.64	Heating: {temperature: $]-\infty, 750]$, deformation: $]1.01, 2.01[$} Deep Rolling: {pressure: $]300, \infty[$}	
	H_h6.d2	density: 251 deformation: 2.43 hardness: 655 ddensity: $5.6e+11$	0.39	Heating: {temperature: $]-\infty, 750]$, deformation: $]2.01, 3.05[$} Deep Rolling: {pressure: $]300, \infty[$}	
	H_h7.d2	density: 251 deformation: 1.02 hardness: 352 ddensity: $3.5e+09$	0.8	Heating: {temperature: $]-\infty, 750[$, deformation: $]3.05, \infty[$} Deep Rolling: {pressure: $]300, \infty[$}	

5.6 Conclusions

The experiments show that the MATCALO system is capable of finding valid processing plans incorporating knowledge about possible interacting effects of the process steps in the models generated from synthetic training data. Assuming such models exist using real data the wide impact the system can have once suitable model databases are built up is demonstrated. Experiments were conducted to generate larger databases. Evaluating the system using this real data will be subject to future work. In addition to that, the design, development and integration of ontologies representing rich semantics about physical coherences of materials, properties and processes to serve as additional source of knowledge in the MATCALO system are to be further investigated. A version of the MATCALO web interface allows scientists to upload files to test the system with their own data. The uploaded data is only be made available to others if it has been explicitly approved by the researchers who generated them. Currently, the original MATCALO system is not undergoing further development, as the CRC within which the framework was developed is not being continued. It can, however, be seen as the precursor to BAYROB due to its integration of the concept of utilizing tree structures for storing distributions of variables, which can be leveraged for reverse inference. The capability to anticipate the outcomes of specific actions and identify a sequence of actions that is most likely to result in a state aligning with a predefined goal specification is crucial, not only in the field of robotics but also across various scientific domains. While MATCALO was specifically designed for the materials science use case, it laid the groundwork for BAYROB. In BAYROB, more complex relationships can be represented, as the distributions are not bound to take a particular form. The incorporation of JPTs and their inherent powerful representation and inference tools has elevated the system to a higher level, introducing new possibilities in terms of applications and domains of use.

◁ *MATCALO as predecessor of BAYROB*

Chapter Six

EVALUATION

In this evaluation, it is showcased how the probabilistic hybrid models developed in BAYROB allow to ask any kind of question about the experience data the models have been trained with. The complex interplay between a robot's perception of the environment and its ability to make informed decisions is deemed crucial for autonomous systems. Leveraging the plenitude of experience data, the models aim to capture a coherent and adaptable framework for the representation of uncertainties inherent in real-world scenarios.

This evaluation is separated into four parts. The first part comprises a comprehensive exploration of how well the model translates experiential knowledge into meaningful representations of the robot's beliefs. Some of the inference schemata introduced in Section 2.5.1 will be revisited in this chapter by applying them on specific examples. The second part quantitatively evaluates different models (or model settings) based on the likelihood of test data. The third part makes use of the forward and back propagation of belief states over time, thereby paving the way for enhanced decision-making and navigation within dynamic and unpredictable environments. The fourth part introduces the web app that can be used to query single models or perform a plan refinement (search) on all models.

6.1 Model Stats

The three models - `turn`, `move_base`, and `perception` - employed in BAYROB were previously introduced in Section 4.2. Furthermore, the evaluation encompasses a fourth model, `pr2`, which was trained using real robot log data. Table 6 provides an overview of the models along with some of their characteristics, aiding in contextualizing the results.

The state space of the respective model is a lower-bound estimation based on the value ranges of the variables incorporated in the model. Since some variables are continu-

ous, their value ranges (and thus the state space) would be infinite, yet, for estimating a lower bound of the state space, the values are discretized, e.g. to $|b-a|$ values for large value ranges from $[a, b]$ (e.g. for X_{in}, Y_{in}, $Angle$) and $\frac{|b-a|}{0.1}$ for smaller value ranges from $[a, b]$ (e.g. Δ_{dir}, Δ_{pos}). As an example, the sate space of the `move_base` model is determined as follows:

The `move_base` model incorporates 7 variables, X_{in}, Y_{in}, $XDir_{in}$, $YDir_{in}$, Δ_{pos_x}, Δ_{pos_y}, and $Collided$. X_{in} and Y_{in} both can take values from $[0, 100]$, thus it is assumed that they can each take 100 different values. Similarly, $XDir_{in}$, $YDir_{in}$, Δ_{pos_x}, and Δ_{pos_y} can each take values from $[-1, 1]$, resulting in 20 different values each, assuming a step size of 0.1. The last variable, $Collided$ is Boolean, so it can take 2 different values. The state space is then calculated as:

$$100 \cdot 100 \cdot 20 \cdot 20 \cdot 20 \cdot 20 \cdot 2 = 1600000000. \tag{106}$$

Model statistics | *Table 6*

model	#variables	state space	#datapoints	#leaves	#total nodes
`turn`	5	57,600,000	35,000	106	211
`move_base`	7	1,600,000,000	2,533,680	74	147
`perception`	20	2,949,120,000,000	772	88	175
`pr2`	12	1,944,512,593,920	60,662	957	1913

It is worth noting that the state space sharply contrasts with the number of data points used to train the respective models. However, in many scenarios, most of the theoretically possible combinations of values are either nearly equivalent, e.g. as they are very close in terms of the position or facing direction. This is one reason why the data has been generated to reflect the scenarios that are considered relevant, the other being the fact that typically not enough data is available in real-world applications that represent every single world state imaginable. The calculation of the state space, however, helps understanding the overwhelmingly large range of possible worlds an autonomous agent has reason over when inferring the next (best) action.

6.2 Part I: Reproducing Ground Data with JPTs

Joint probability distributions allow asking questions about all variables incorporated. In BAYROB, the action models `move_base`, `perception` and `turn` represent artificially generated robot experience data about forward movements and rotations, as well as results of the robot's perception. The kinds of questions that can be asked therefore range from "What kinds of Actions have been executed?" over "Where was the agent standing when he detected milk?" to "Which actions failed most often?" and "What kinds of failures occurred?". All these questions can be further constrained to situations in which for example a specific object was detected, an action failed or some other object was involved. In the following, each model is examined by comparing the ground truth data filtered by given constraints with the probability distribution represented by

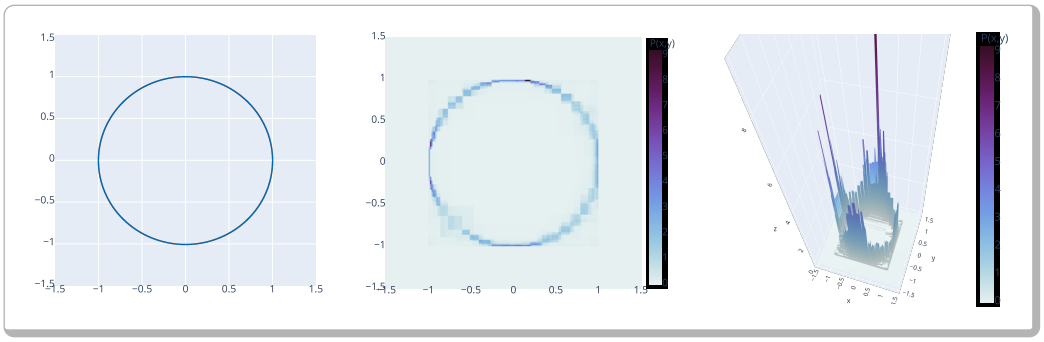
that same query. Additionally, another model called `pr2` is investigated the same way. This model represents real robot experience data (NEEMs) representing experiments in which the robot fetched milk.

6.2.1 Turn Data

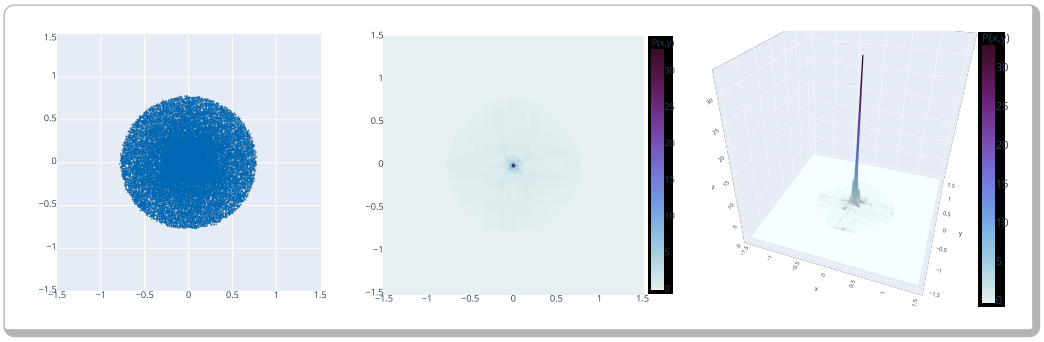
The `turn` model learned how the robot's facing direction changes when turning around a certain angle and is described by an initial facing direction, the parameter angle and the delta to the facing direction after the update. Figure 48 shows the unconstrained (apriori) marginal distributions of the initial and delta variables. The ground truth data on the left shows how the initial facing directions form a unit circle, since they reflect the inherent constraints imposed by the normalization process of the direction vector. Consequently, points within the unit circle uniformly represent all possible normalized directions. The marginal distribution of the variables $XDir_{in}, YDir_{in}$ of the learnt model in Figure 48 a) (middle and right) captures the ground data very well.

Comparison of data points (normalized ground truth) and distribution represented by the `turn` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the (conditioned) distributions, respectively. | *Figure 48*

Unconstrained distribution $P(XDir_{in}, YDir_{in})$ of the `turn` model | *a)*



Unconstrained distribution $P\left(\Delta_{dir_x}, \Delta_{dir_y}\right)$ of the `turn` model | *b)*



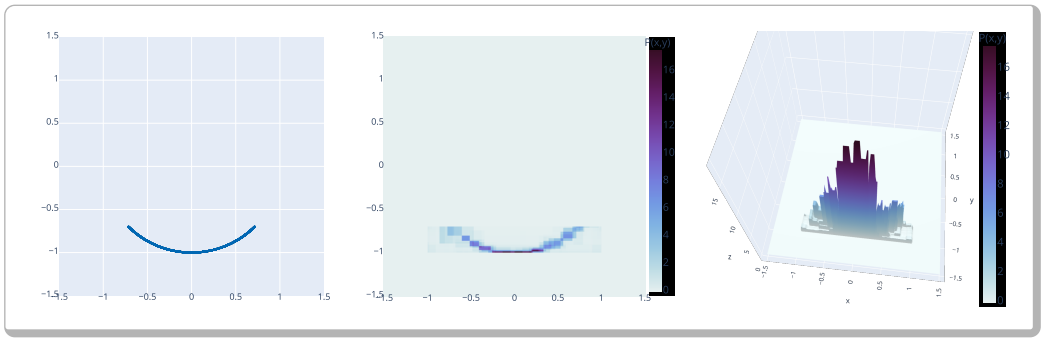
In contrast, the distribution of the deltas in Figure 48 b) (middle and right) forms a filled circle since both variables Δ_{dir_x} and Δ_{dir_y} can take values from 0 to $\sim$0.7, due to the agent's limited turn radius of 45 degrees, respectively. While the shape of the filled

circle is vaguely perceptible, yet present in the heatmap distribution, the 3D plot on the right reveals that the peak in the center around the 0-values of the variables causes the light colors in the circle's periphery. This observation corresponds to the increased density of datapoints in the center of the ground truth and the fact that high delta values for both variables at the same time typically happen rather rarely.

The ground truth represented on the left in Figure 49 is a subset of the original data, which can be guessed by the shape of its plot. When constraining the learnt models to satisfy given evidence, the resulting distributions may have very unique shapes, which need to be represented appropriately by the model to allow for reliable reasoning. Figure 49 a) shows the distribution of the initial facing direction, when constraining the $YDir_{in}$ variable to a certain value range. In combination with an unconstrained $XDir_{in}$ variable, this corresponds to facing the bottom in an arch shape. The corresponding delta distribution resembles a bowtie-like structure, as shown in Figure 49 b). This makes sense, since in combination with the limited range of motion of ~ -45 deg to ~ 45 deg, the motion deltas for the movement to the left and right form the two wings of the observed structure. The uncertainty in the model contributes further to the spread of the data points.

Comparison of data points (normalized ground truth) and distribution represented by the `turn` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the (conditioned) distributions, respectively. | *Figure 49*

Constrained $ydir_{in}$ variable of facing direction: $P\left(\Delta_{dir_x}, \Delta_{dir_y} \mid -1.3 \leq ydir_{in} \leq -0.7\right)$ | *a)*



Constrained $ydir_{in}$ variable of facing direction: $P\left(\Delta_{dir_x}, \Delta_{dir_y} \mid -1.3 \leq ydir_{in} \leq -0.7\right)$ | *b)*

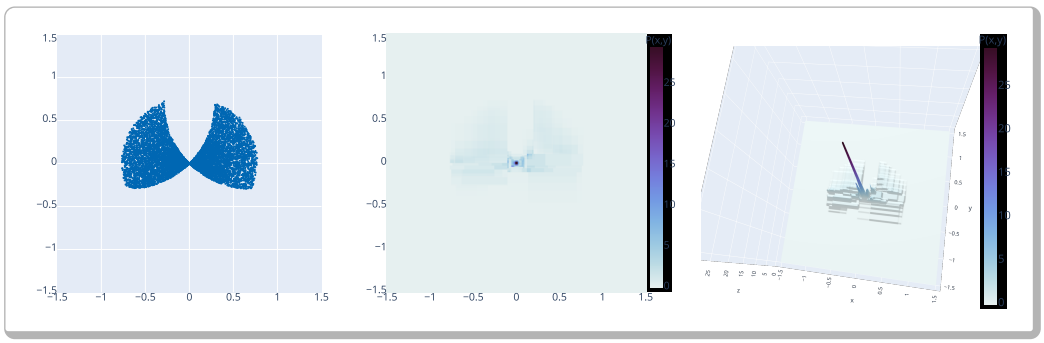


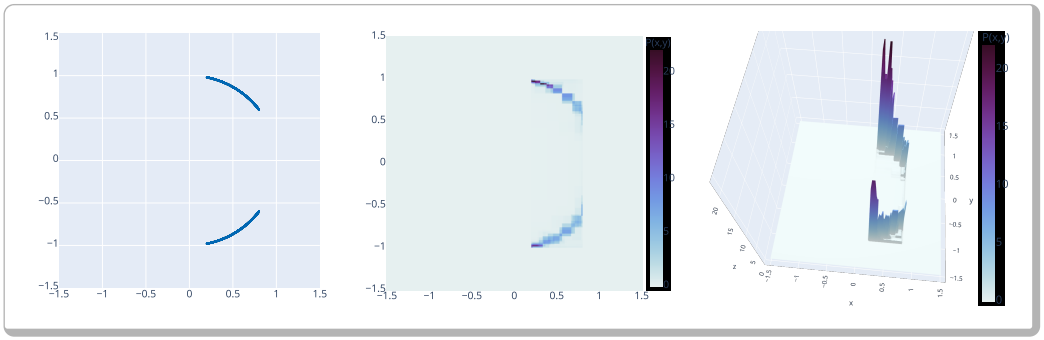
Figure 50 shows the resulting distributions with a constrained $XDir_{in}$ variable. Here, the value range is further restricted, such that there are two separate structures visible

instead of a single arch. Analogously to the delta distribution in Figure 49 b), Figure 50 b) shows the overlapping of two clusters, each representing possible orientations.

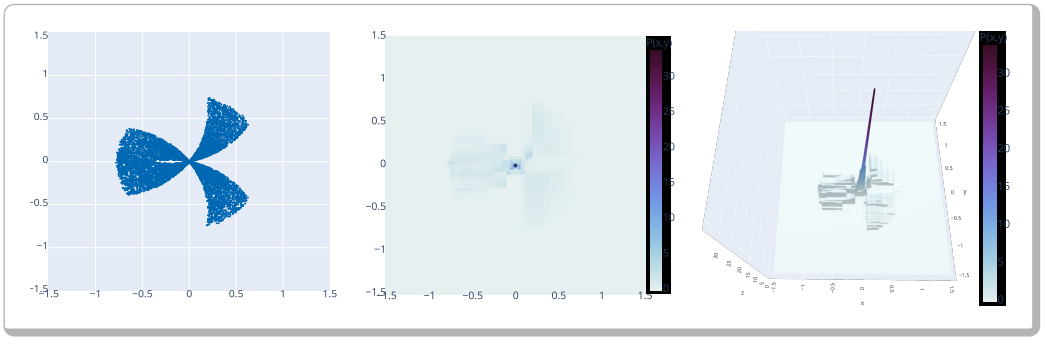
The plots of the delta distributions of these two examples show that arbitrary shapes of distributions can be learnt and accurately represented by the underlying framework.

Comparison of data points (normalized ground truth) and distribution represented by the `turn` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the (conditioned) distributions, respectively. | *Figure 50*

Constrained $xdir_{in}$ variable of facing direction: $P\left(\Delta_{dir_x}, \Delta_{dir_y} \mid 0.2 \leq xdir_{in} \leq 0.8\right)$ | *a)*



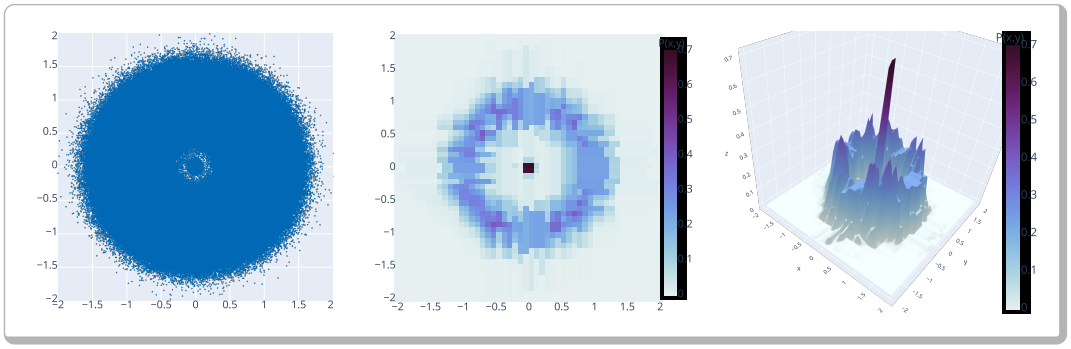
Constrained $xdir_{in}$ variable of facing direction: $P\left(\Delta_{dir_x}, \Delta_{dir_y} \mid 0.2 \leq xdir_{in} \leq 0.8\right)$ | *b)*



6.2.2 Move Data

Similar to the delta distributions in the turn model discussed earlier, Figure 51 illustrates the apriori distribution of position deltas for the `move_base` model. Given that the `move_base` action is defined with a step size of 1, one would anticipate observing a wide ring with a central dot representing collision points rather than a completely filled circle.

Unconstrained initial position or direction: The distribution $P\left(\Delta_{pos_x}, \Delta_{pos_y}\right)$ of the `move_base` model | *Figure 51*

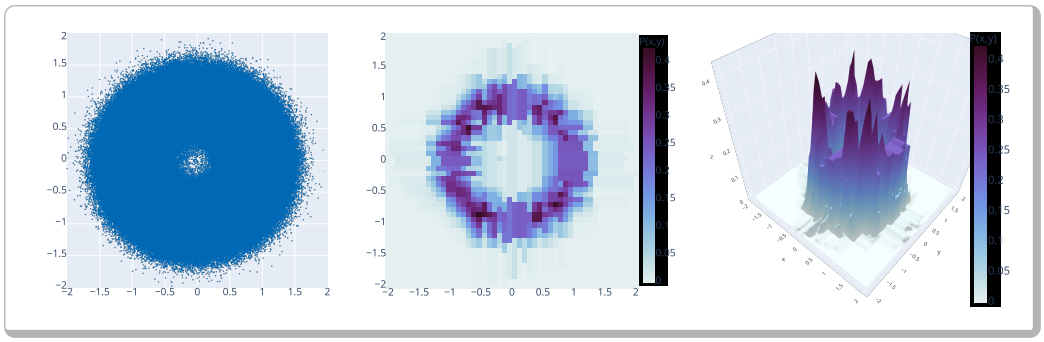


However, due to the introduction of Gaussian noise to the deltas when a collision occurs, their values are never precisely 0. Hence, a fuzzy dot is visible in the center.

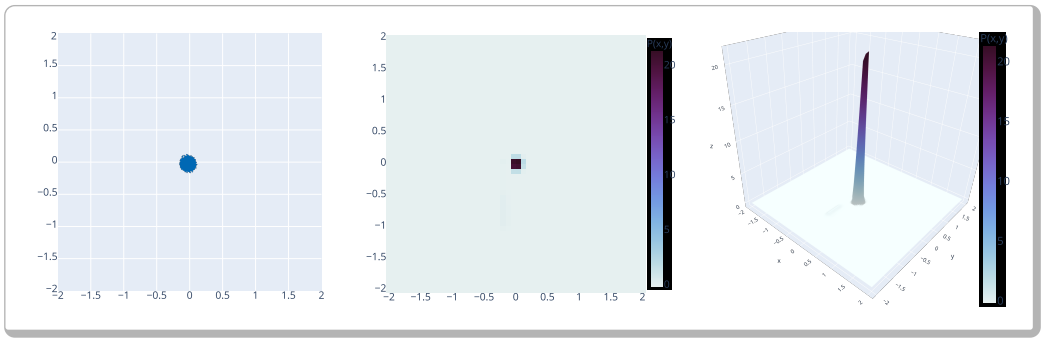
In Figures 52 a) and b), the ring and dot are separated into individual distributions by constraining the model to either no-collision or collision data. Once again, the distribution plots closely mirror the shapes of the ground truth plot, indicating that the learned model accurately represents the ground truth.

Comparison of data points (normalized ground truth) and distribution represented by the `move_base` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the (conditioned) distributions, respectively. | *Figure 52*

No-collision data: The distribution $P\left(\Delta_{pos_x}, \Delta_{pos_y} \mid \text{collided} = \bot\right)$ of the `move_base` model | *a)*

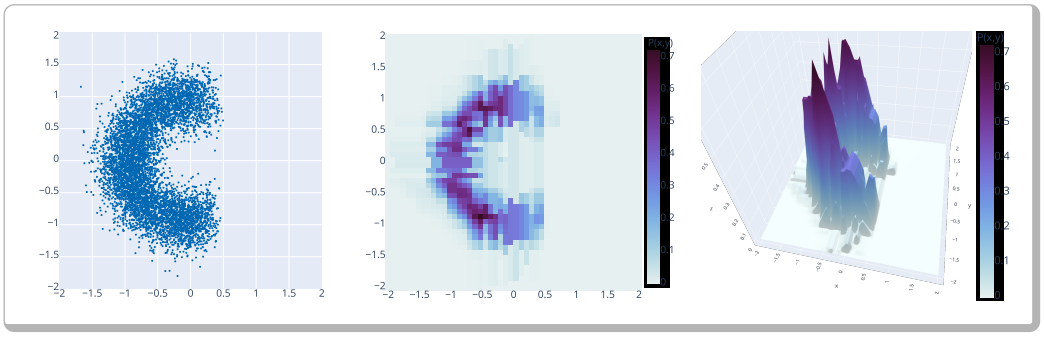


Collision data: The distribution $P\left(\Delta_{pos_x}, \Delta_{pos_y} \mid \text{collided} = \top\right)$ of the `move_base` model | *b)*

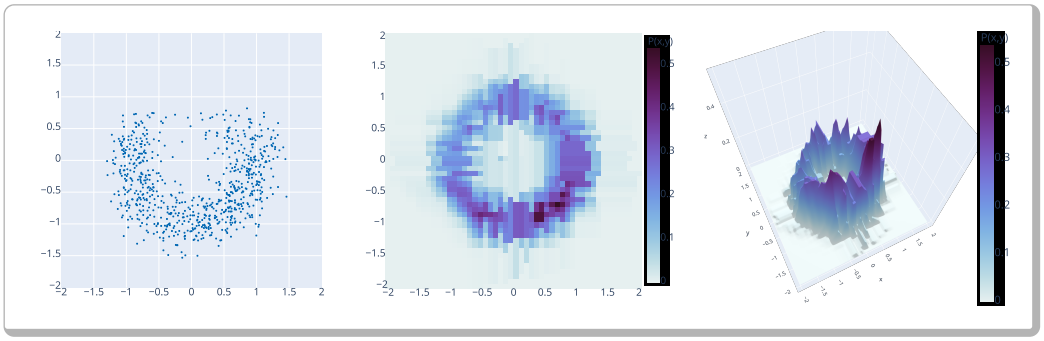


Comparison of data points (normalized ground truth) and distribution represented by the `move_base` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the (conditioned) distributions, respectively. | *Figure 53*

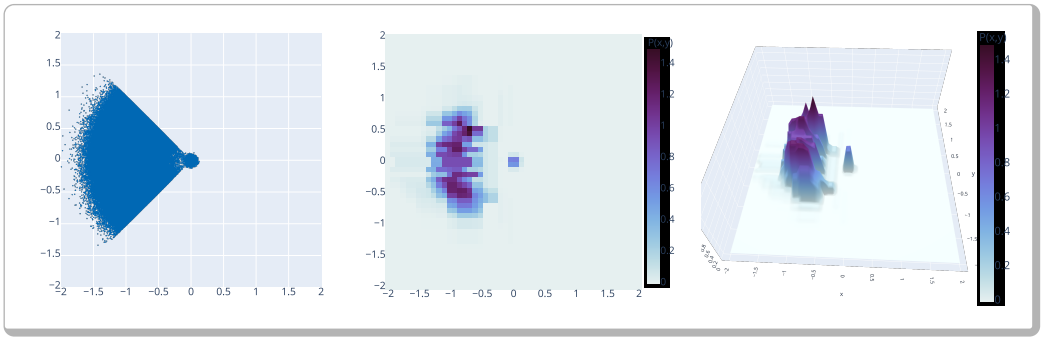
Position on the right wall of the kitchen: $P\left(\Delta_{pos_x}, \Delta_{pos_y} \mid \text{collided} = \bot, 99.5 \leq x_{in} \leq 100.5\right)$ and... | *a)*



on the upper left corner: $P\left(\Delta_{pos_x}, \Delta_{pos_y} \mid \text{collided} = \bot, 0 \leq xdir_{in} \leq 2, 98 \leq ydir_{in} \leq 100\right)$ | *b)*



Constrained $xdir_{in}$ variable of facing direction: $P\left(\Delta_{pos_x}, \Delta_{pos_y} \mid -1.3 \leq xdir_{in} \leq -0.7\right)$ | *c)*



This is particularly remarkable, since the distributions are comprised in a single model containing only 74 leaf nodes that are aggregated to an overall distribution when querying the model.

The images depicted in Figure 53 are restricted to either the position or facing direction, revealing specific patterns. In the upper row (Figure 53 a)), half circles are evident, as the X_{in} variable of the position is constrained to the far-right edge of the kitchen with a collision set to true ($\top$). Consequently, the delta positions are limited to depict movement to the left, as moving to the right would result in a collision with the kitchen wall.

The middle row (Figure 53 b)) exhibits a pronounced preference for the lower-right quarter of the circle, indicating movement in the lower-right direction. This is because the position was constrained to the upper-right corner of the kitchen, thereby restricting the possible movement directions accordingly.

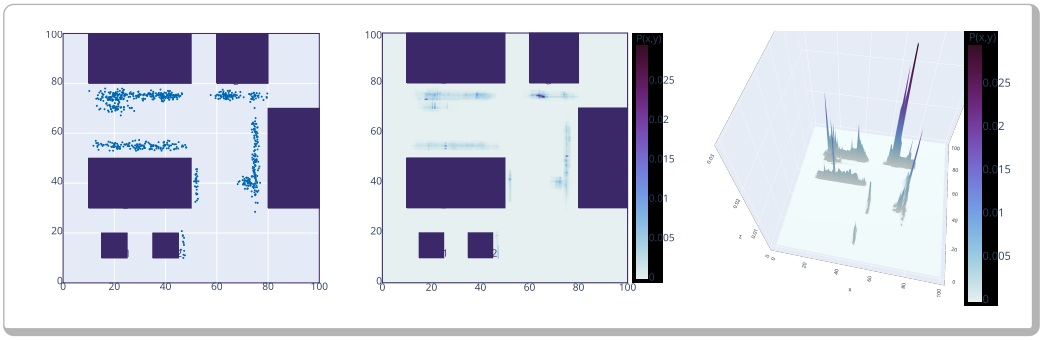
In contrast to the images discussed thus far, the lower row represents the relationship between the direction variables (as opposed to the position variables) and the position deltas. Without any constraints on the position or collision, a facing direction to the left strongly correlates with a movement to the left, as illustrated by the plots in Figure 53 c).

6.2.3 Perception Data

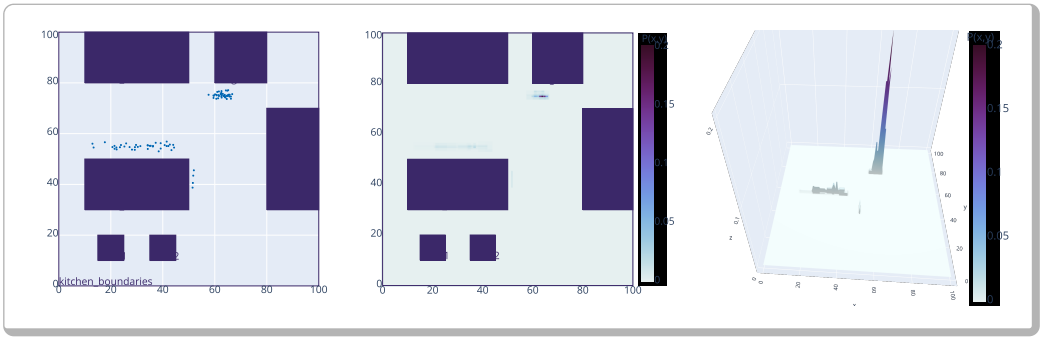
The `perception` model links the robot's state (position/facing direction) to the world state (position/visibility of objects, environment variables, and so on). This model therefore contains a lot of Boolean and multinomial variables that are directly linked to the (continuously numeric) state variables of the robot. Figure 54 a) shows the apriori marginal distribution of the position.

Comparison of data points (normalized ground truth) and distribution represented by the `perception` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the (conditioned) distributions, respectively. | *Figure 54*

$P(X_{in}, Y_{in})$: Where can I be located? | *a)*



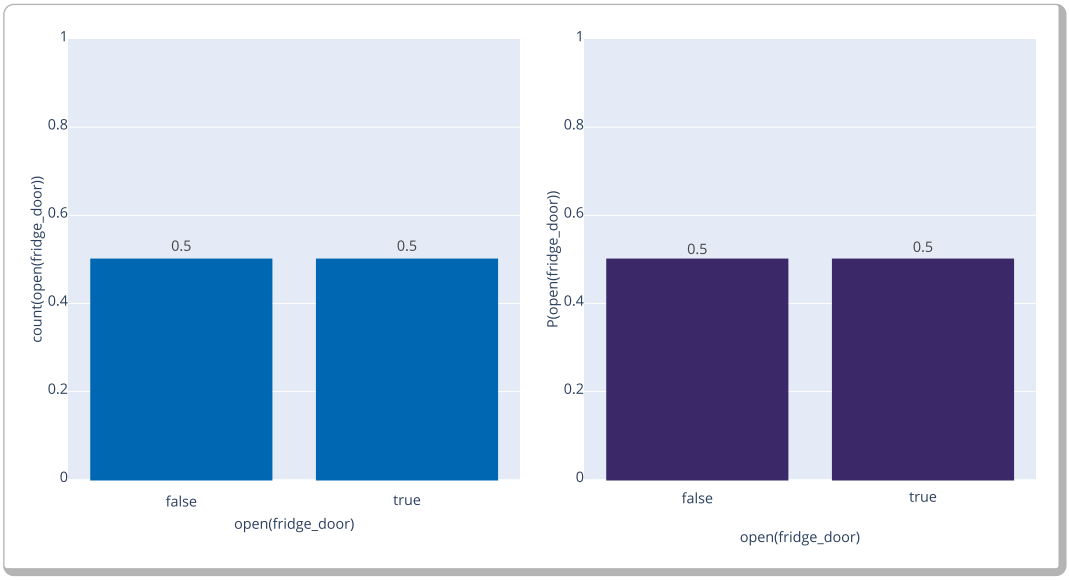
$P(X_{in}, Y_{in} \mid Detected(\text{milk}) = \top)$: I see milk. Where am I most likely standing? | *b)*



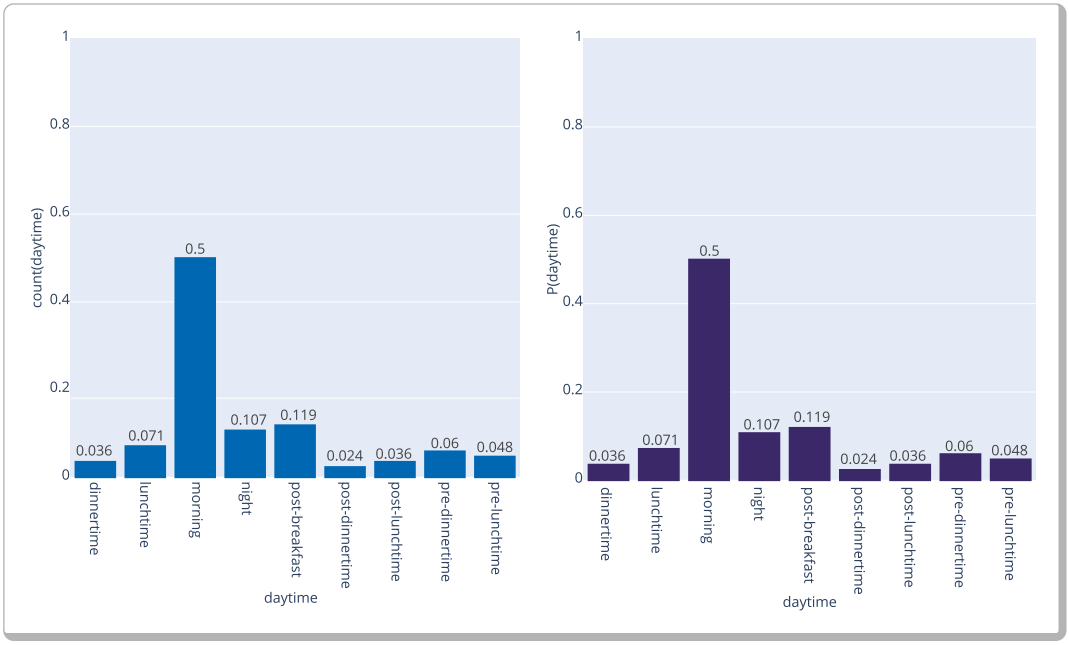
There are much less datapoints available compared to the `move_base` model since it is assumed that only for a few positions there are certain items visible and states observable. For all other positions no propositions can be made about the state of the furniture doors and drawers or about the presence of certain items. This drastically reduces the model size and forces the system to combine the three action models when aggregating the robot state.

Comparison of the (normalized) ground truth and distribution of the `perception` dataset with given evidence $Detected(\text{milk}) = \top$ | *Figure 55*

$P(Open(\text{fridge\_door}) \mid Detected(\text{milk}) = \top)$: I see milk. Is the fridge door open or closed? | *a)*



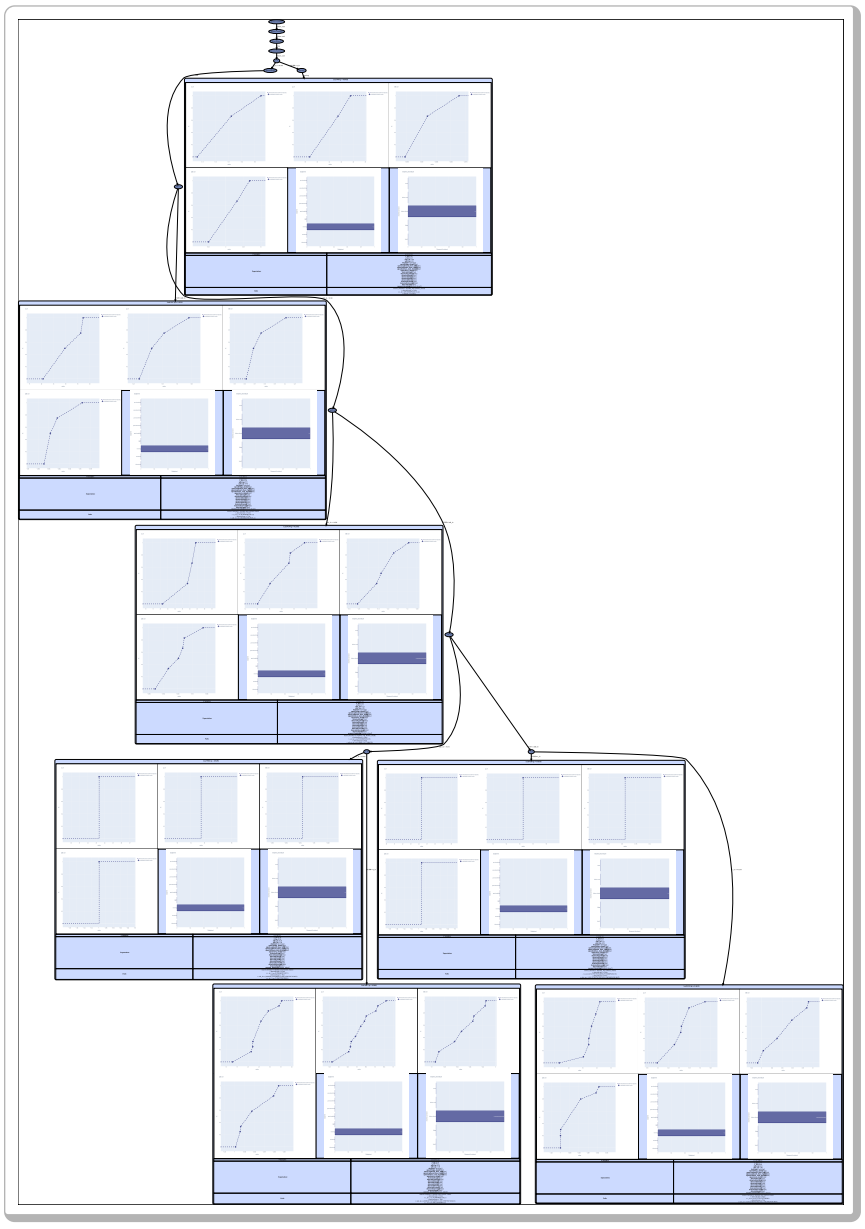
$P(Open(\text{fridge\_door}) \mid Detected(\text{milk}) = \top)$: I see milk. Is the fridge door open or closed? | *b)*



In Figure 54 b), positions near the fridge and along two edges of the kitchen island are displayed when querying for the most probable positions given the perception of milk. This aligns with the logical scenarios, as milk is either stored in the fridge when not in use or on the kitchen table as part of a breakfast table setting.

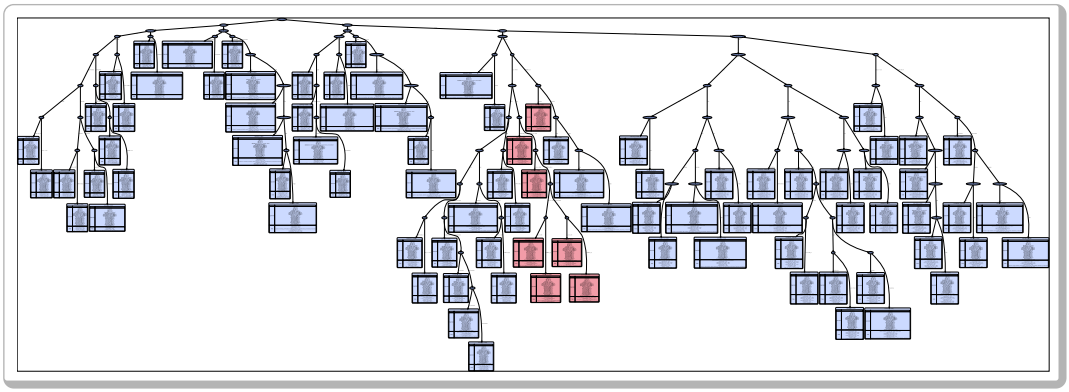
The distribution of the state of the fridge door in Figure 55 a) illustrates the correlation between an open door and positions around the fridge, while a closed door corresponds to positions around the kitchen table. This distribution mirrors the distribution of times of day in Figure 55 b), indicating that for approximately half of the instances where milk has been observed, it was morning (where it is typically found on the kitchen island), while the other half is distributed across other times of the day, where it is stored in the fridge.

The conditioned `perception` model $Detected(\text{milk}) = \top \wedge Daytime = \text{morning}$ | ***Figure 56***



The probability distributions are generated using the CONDITIONAL-JPT algorithm (Algorithm 3). When investigating a learnt tree model, one can actually trace certain regions in the tree that are responsible for handling the "special" cases such as walls and obstacles (`move_base`) or, in this case, the ones where milk is detected in the morning. This is particularly evident when the tree is conditioned on certain variables and the resulting conditional tree (Figure 56) is then compared to the original JPT (Figure 57) The conditional tree contains only the leaves consistent with the evidence, $Detected(\text{milk}) = \top \wedge Daytime = \text{morning}$. While the tree structure is typically a (modified) excerpt of the original tree, the distributions differ in that they were re-normalized, since the entire tree does not pose a proper probability distribution anymore after pruning nodes or even entire subtrees. The original positions of the leaves from the conditional tree are highlighted (red) in the original tree. Note that the value $Nearest\_Furniture = \text{kitchen\_island}$ in each of the nodes (as indicated by the respective lower right distribution plot in each leaf) which again shows the dependencies that have been learnt from the data.

The entire `perception` model. The highlighted nodes (red) are the ones that remain in the resulting tree when conditioned on $Detected(\text{milk}) = \top \wedge Daytime = \text{morning}$ | *Figure 57*



6.2.4 PR2 (NEEM) Data

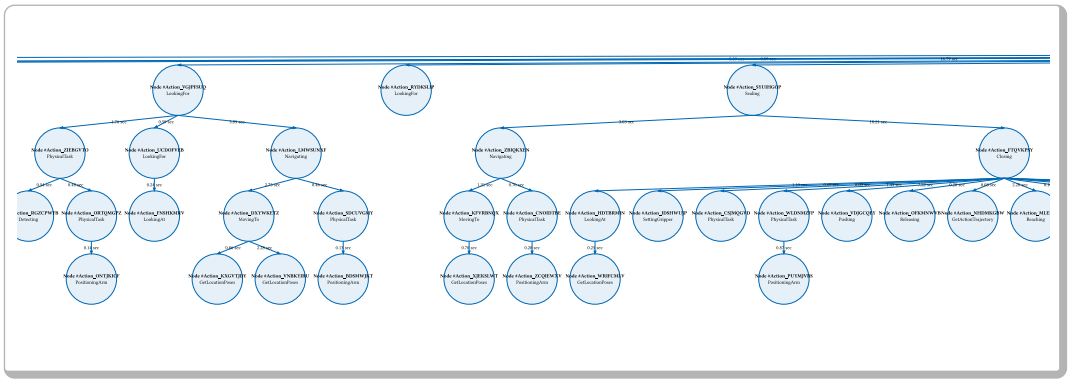
The pr2 dataset (by courtesy of Sebastian Koralewski) consists of 296 "fetch milk" experiments, each involving varying numbers of subactions, conducted by a PR2 robot in the laboratory kitchen of the *Institute for Artificial Intelligence (IAI)* in Bremen. The original dataset is provided in two files per experiment. One file contains tabular information about the actions performed (averaging over 200 actions), while the other includes additional pose data for some of the actions in the first file. In a preprocessing step, these pieces of information were merged to create a consolidated dataset comprising over 60,000 rows.

Each row in an experiment file represents the execution of a single subaction, which can be a higher-level action like fetching or placing, or lower-level actions such as positioning an arm. The actions are interconnected through a parent-relationship, meaning the entire experiment begins with one action that is further broken down into subtasks,

which, in turn, are composed of additional subtasks. Consequently, an experiment forms an action tree, and an excerpt of this structure is depicted in Figure 58.

The dataset includes the ID, type (cmp. Figure 60), and duration of each subaction, along with additional details such as the success or failure of its execution, the objects and body parts involved, timestamps (indicating the start and end of an action), and poses (3-dimensional position vectors along with a rotation quantile). However, the poses are only available for certain grasping and placing actions within each experiment, limiting the ability to track the robot's trajectories throughout the entire experiment.

An excerpt of the action tree for one fetch-milk experiment | *Figure 58*



Nevertheless, the dataset remains valuable for evaluating action models in BAYROB. It encompasses numeric robot log data, multinomial annotations, and boolean state variables that can be consolidated to construct a comprehensive model based on real robot experiment data.

Comparison of data points (normalized ground truth) and distribution represented by the `pr2` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the marginal apriori distribution $P(X_{in}, Y_{in})$ | *Figure 59*

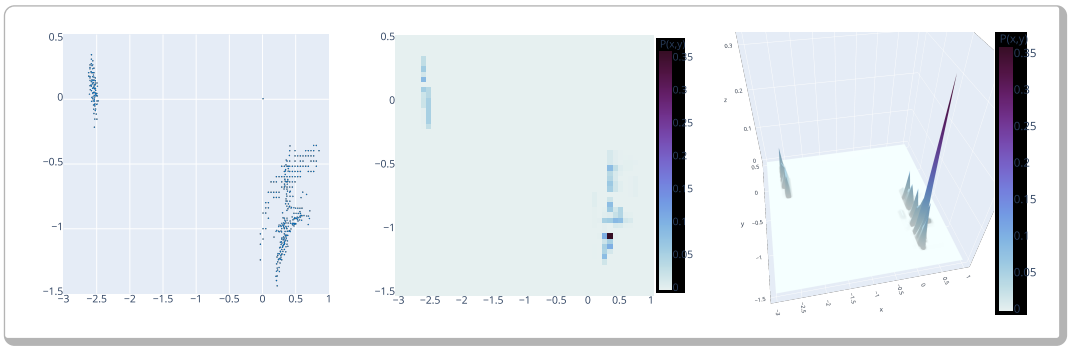
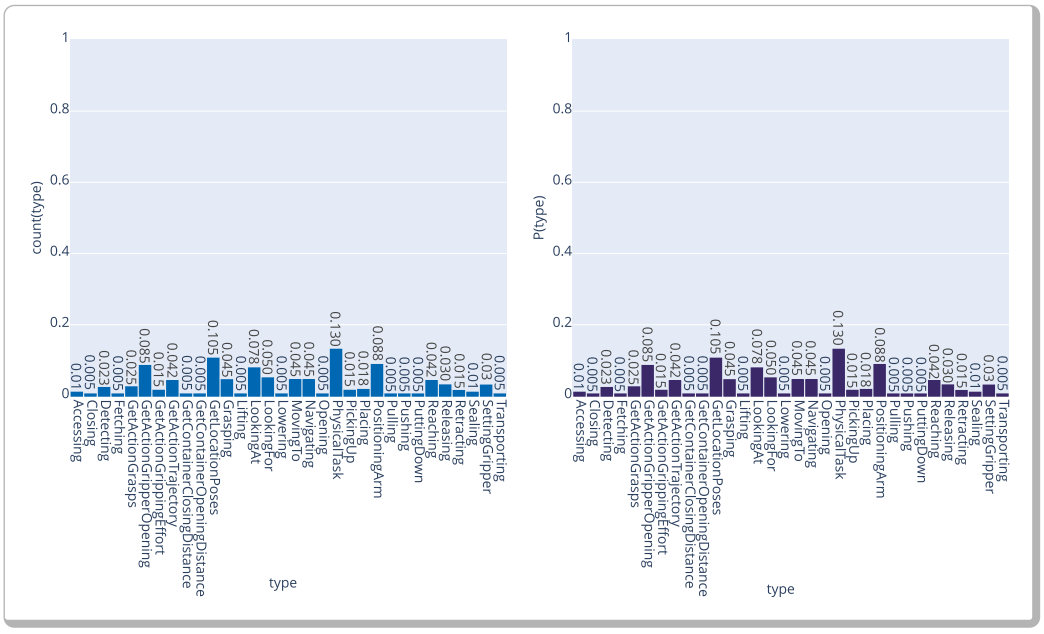


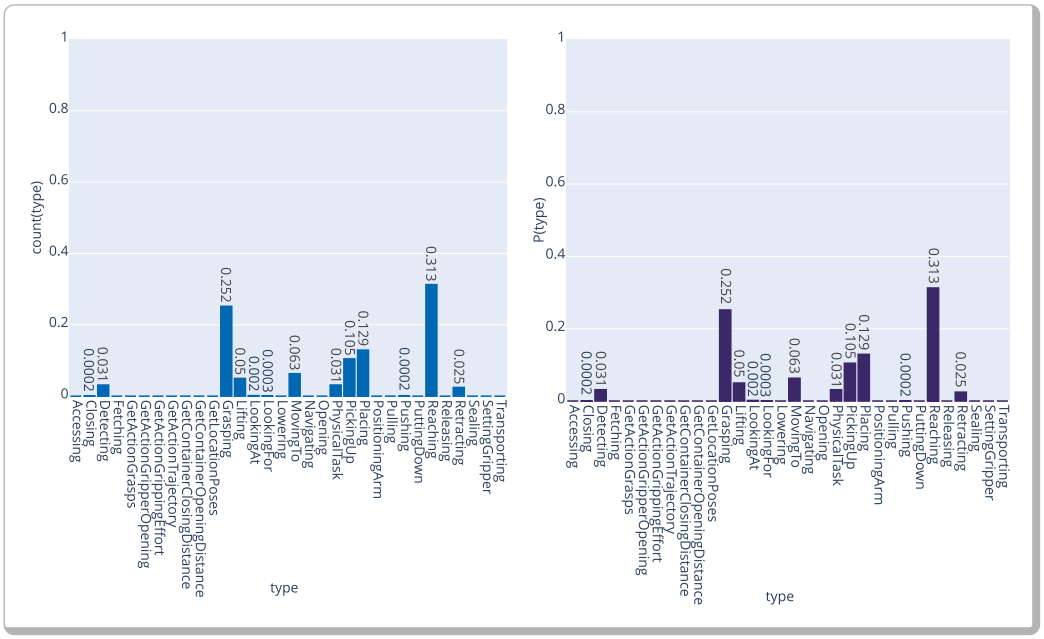
Figure 59 illustrates instances where position data is accessible, enabling their contextualization with succeeded or failed experiment subactions. The two discernible clusters of positions can be categorized into two action clusters: The upper-left cluster depicts positions associated with placing actions, whereas the lower-right cluster indicates positions where the agent was located during object grasping.

Comparison of the (normalized) ground truth and distribution represented by the `pr2` JPT | *Figure 60*

$P(Type)$: Which actions are present in the dataset? | *a)*



$P(Type \mid Success = \bot)$: Which actions have failed? | *b)*

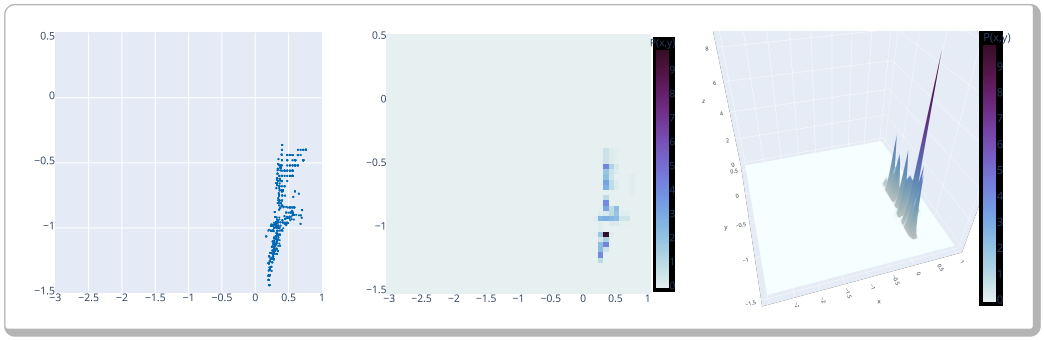


The variable *Type* represents the kind of action that has been executed. Figure 60 shows the distributions of this variable, either prior (Figure 60 a)) or conditioned on the action being successful (Figure 60 b)). The most frequently executed actions are lower-level actions such as arm positioning or retrieving location poses because these actions are typically sub-actions of higher-level, more abstract actions and are therefore called multiple times. The higher-level actions, on the other hand, are much more likely to

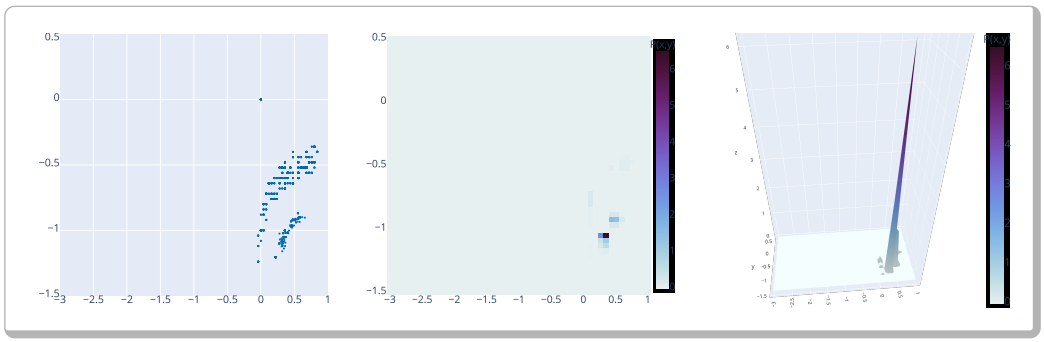
fail, typically, because they are more complex through their composition of multiple sub-tasks, each of which may be a source of failure.

Comparison of data points (normalized ground truth) and distribution represented by the `pr2` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the (conditioned) distributions, respectively. | *Figure 61*

$P(X_{in}, Y_{in} \mid Type = \text{grasping} \wedge Success = \top)$:
Where was I standing when my grasping action succeeded? | *a)*



$P(X_{in}, Y_{in} \mid Type = \text{grasping} \wedge Success = \bot)$:
Where was I standing when my grasping action failed? | *b)*

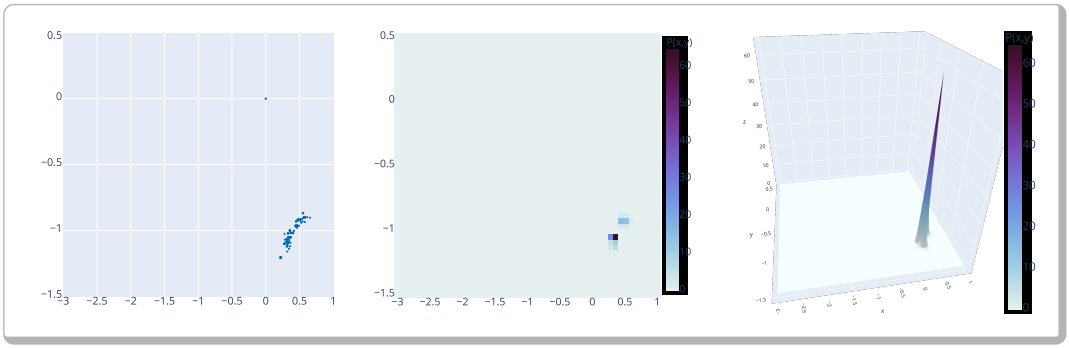


All placing actions represented by the upper-left cluster in Figure 59, where position data is available, were successful. On the contrary, the actions depicted in the lower-right cluster reveal patterns that highlight positions conducive to successful “grasping” actions and those that are not (cmp. Figure 61). However, for certain positions, both successful and unsuccessful actions are observed, necessitating further investigation to determine the parameterization for a successful experiment.

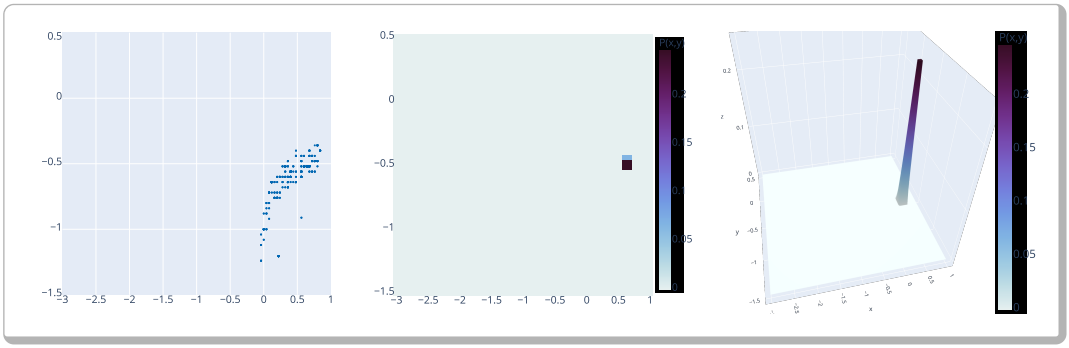
A starting point would be to look at the types of errors that occurred (Figure 63) and then link them to certain positions (Figure 62). The failure in the experiments shown in Figure 62 a) appears to be associated directly with the position. This inference is supported by the error type, which suggests that the object was out of reach, rendering it ungraspable. In Figures 62 b) and c), the second and third errors seem more related to hardware issues than position-related concerns. These errors point to low-level failures and a gripper problem with the robot. However, it is worth considering that the gripper error might still result from unfavorable positioning, preventing the agent from placing the object precisely in the center of the gripper.

Comparing positionings causing the three different failure types CRAMManipulationGoalNotReached, CRAMGripperClosedCompletely and CRAMManipulationLowLevelFailure | *Figure 62*

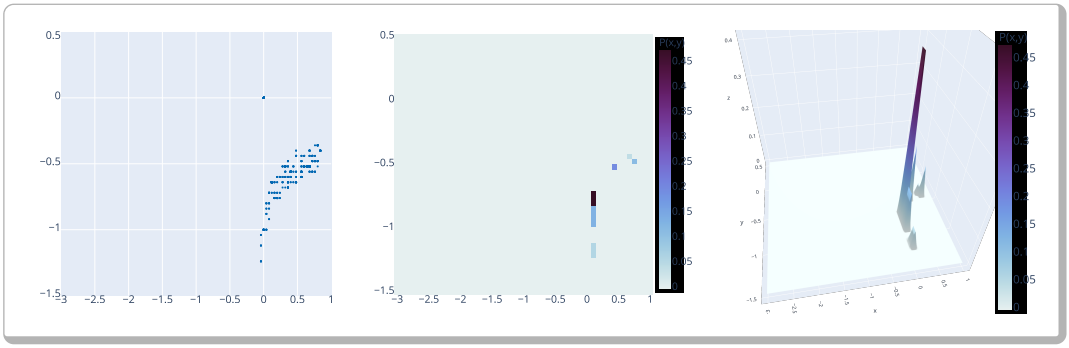
$P(X_{in}, Y_{in} \mid Failure = \text{CRAMManipulationGoalNotReached} \wedge Type = \text{grasping} \wedge Success = \bot)$ | *a)*



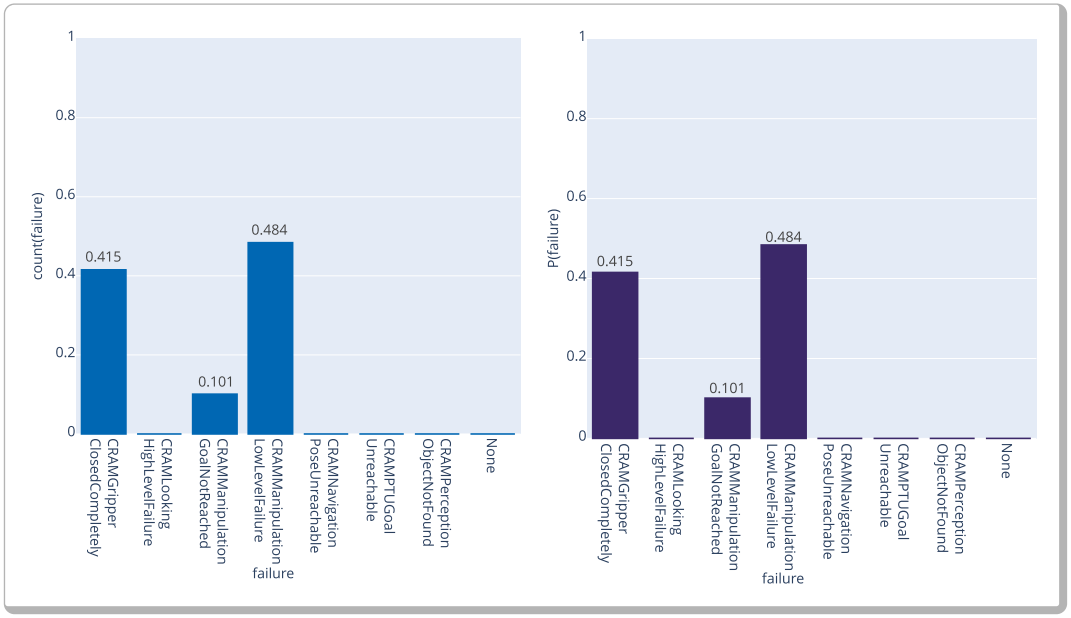
$P(X_{in}, Y_{in} \mid Failure = \text{CRAMGripperClosedCompletely} \wedge Type = \text{grasping} \wedge Success = \bot)$ | *b)*



$P(X_{in}, Y_{in} \mid Failure = \text{CRAMManipulationLowLevelFailure} \wedge Type = \text{grasping} \wedge Success = \bot)$ | *c)*



$P(Failure \mid Type = \text{grasping} \wedge Success = \bot)$:
Which types of errors occurred when my grasping actions failed? | *Figure 63*



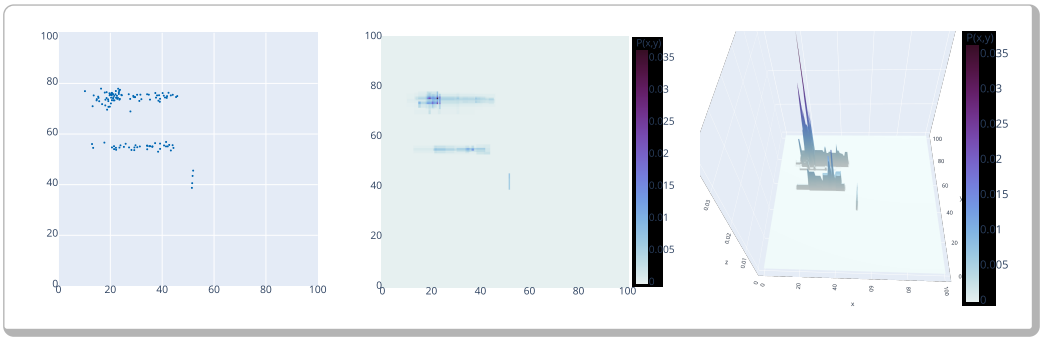
6.2.5 Inference Patterns

Probabilistic hybrid action models in BAYROB allow different inference patterns in queries, such as diganostic and causal reasoning or explaining away.

Diagnostic Inference · Figure 64 shows an example for a diagnostic inference pattern: Given the information (effect), that the agent currently perceives a bowl, what may most likely be the cause of it? The answer (or at least a part of it) is the marginal distribution of positions from which a bowl may be visible.

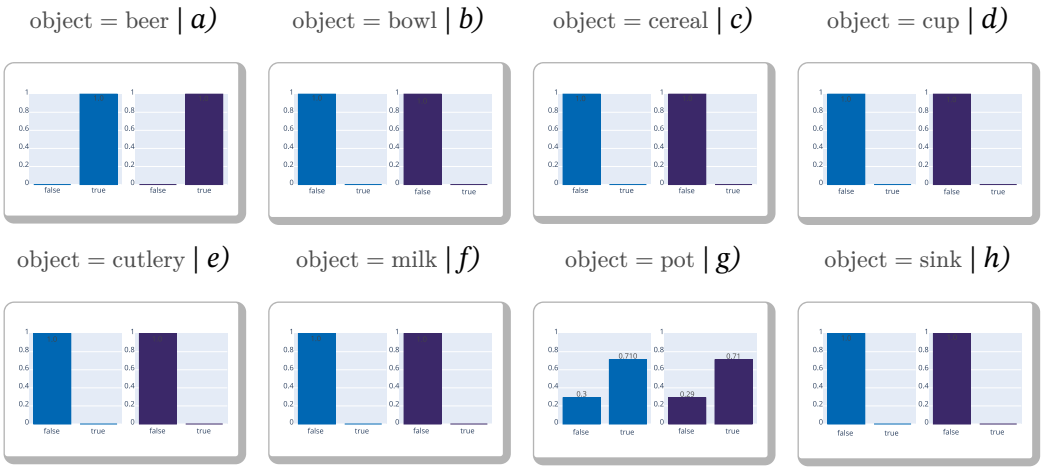
Comparison of data points (normalized ground truth) and distribution represented by the `perception` JPT: The left images represent the (filtered) dataset, the middle and right images the 2D and 3D renders of the (conditioned) distributions, respectively. | *Figure 64*

$P(X_{in}, Y_{in} \mid Detected(\text{bowl}) = \top)$:
I see a bowl. Where am I most likely standing? | *a)*



Other examples for diagnostic inferences that are possible in BAYROB are querying for certain positions given the current position and facing direction ("Where was I, before I made one step forward?") or querying for the state of some furniture's door or drawer given certain objects are visible ("I am near the kitchen unit and see cutlery, how is this possible?"). Note that the former requires backward inference since the effects for that query (i.e. the resulting positions after executing a `move_base` action) are modeled in terms of deltas, such that certain states have to be computed first.

$P(Detected(\langle \text{object} \rangle) \mid Nearest\_Furniture = \text{stove}, Open(\text{stove\_door}) = \top)$:
I am standing near the stove and the stove door is open. What do I see? | *Figure 65*



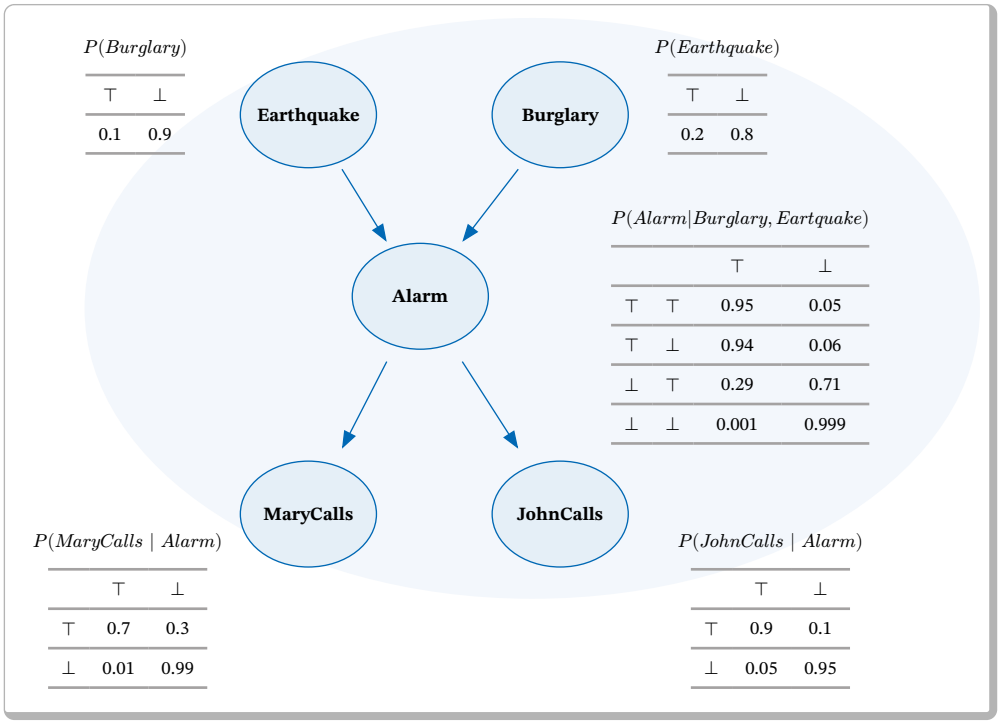
Causal Inference · BAYROB allows to predict effects of certain causes, e.g. when querying for a position the agent will most likely be located at after executing a `move_base` step. Another example is the inverse direction of the query from the diagnostic example: Querying for detected objects given the current position (or nearest furniture) and the state of a furniture door or drawer as evidence. This query translates to a question like "I am standing near the stove and the stove door is open. What can I expect to see?"

The results are shown in Figure 65. As expected, most items are still not visible to the agent, since only pots (Figure 65 g)) are stored in the stove cabinet. Depending on the time of day, however, the pot might be in use on the kitchen table, which is why there is still a chance it might not be visible standing near the stove. The beer (Figure 65 a)) poses an exception here, since it is located *on* the stove top, such that its visibility is not influenced by the state of the stove cabinet door.

Explaining away · When some event is observed, there may potentially two or more causes for this event to happen. As an example, take the well-known alarm scenario with 5 variables representing the events

- an earthquake occurred ($Earthquake$)
- a burglary took place ($Burglary$)
- an alarm went off ($Alarm$)
- Mary called ($MaryCalls$) and
- John called ($JohnCalls$).

The Bayes net for the alarm example | *Figure 66*



The BN with the CPDs is shown in Figure 66. A JPT learnt with the same data is available in Figure 89 in the appendix.

In a scenario where an alarm is triggered, there are two plausible explanations: either an earthquake occurred or a burglary took place. Both events are apriori considered unlikely, as indicated by their probabilities but in combination with an observed alarm, an earthquake seems to be the most probable cause. The distributions for these two scenarios are illustrated in Figure 67 a) and b).

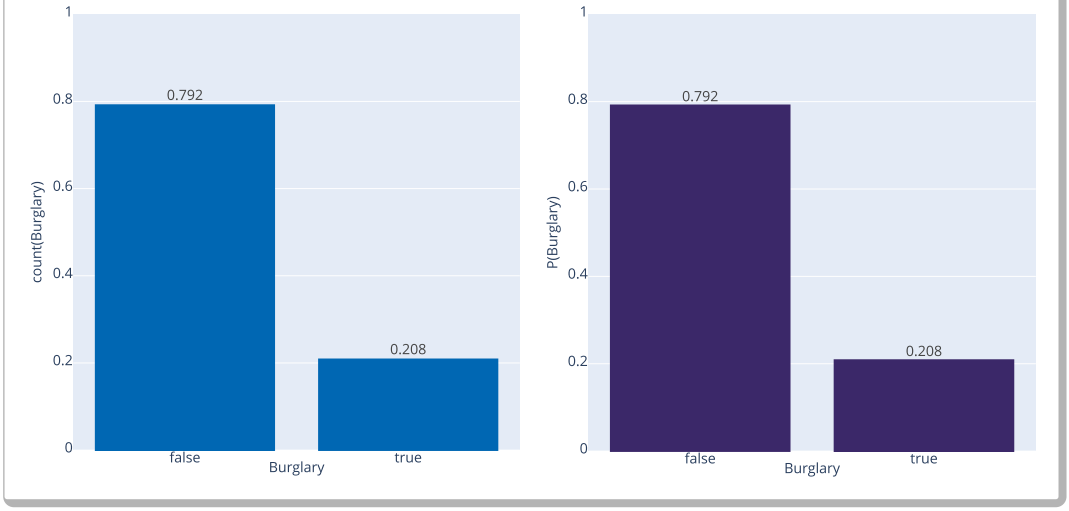
The alarm system is more responsive to the occurrence of an earthquake than to a burglary, as evident in the CPD of the *Alarm* node in the network. However, when one of the potential causes is observed, the distribution of the other cause changes accordingly. For example, upon observing that there was *no* earthquake, the probability of a burglary occurring suddenly rises from around 20% to over 96%. This shift occurs because the observation (or absence) of an earthquake influences the probability of a burglary, even though the two variables are initially independent. However, they become dependent through the observation of an alarm. The act of observing one cause *explains away* the other.

The phenomenon of explaining away is also evident in JPTs, as observed in BAYROB models. However, unlike artificially modeled conditional dependencies, strict independencies are typically not present in real-world data. This holds true for the generated data used in the `move_base`, `turn`, and `perception` models. The data was generated to resemble real-world scenarios rather than adhering to predefined distributions. Nevertheless, the effect of explaining away causes can still be observed and leveraged, for example, in identifying causes (or classes of causes) of errors.

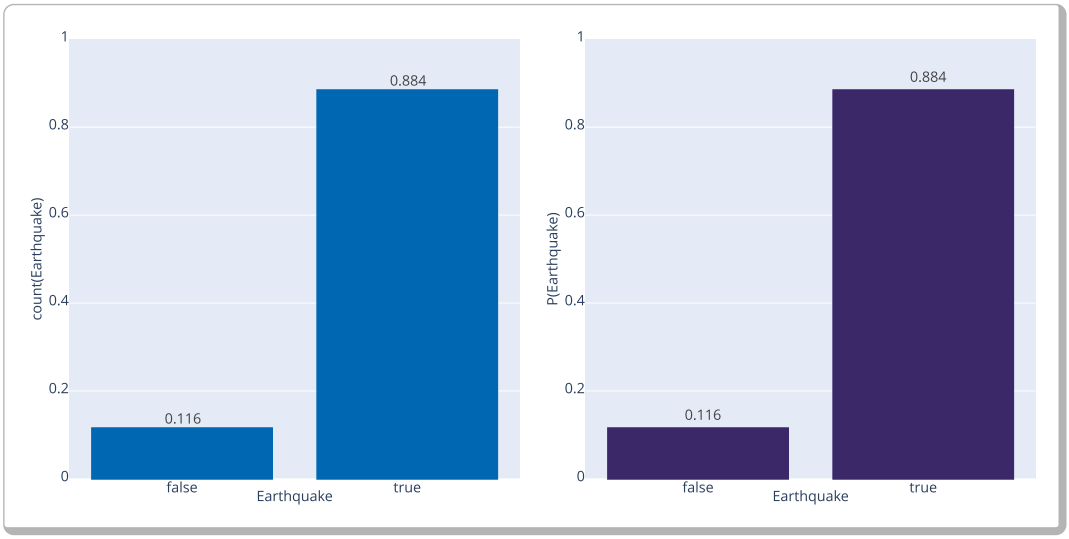
When faced with an undesirable variable state, such as an error due to items being invisible, it is crucial for an autonomous agent to reason about other environmental variables. This involves determining a) what led to the error and b) what actions can be taken to establish the desired variable state. Consider, for instance, an agent tasked with fetching milk, a precondition of which would be to detect it at first.

Alarm example for an explaining away inference in BAYROB | *Figure 67*

$P(Burglary \mid Alarm = \top)$:
An alarm went off. Was there a burglar in my house? | *a)*



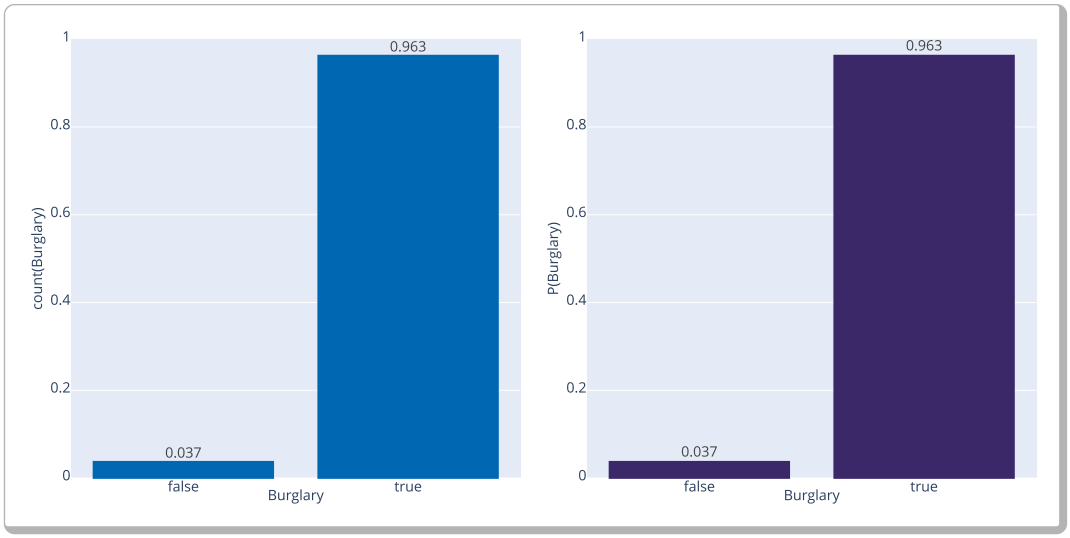
$P(Earthquake \mid Alarm = \top)$:
An alarm went off. Was the earth shaking? | *b)*



If the agent, in its current state, does not perceive milk, various reasons could account for this. Given the knowledge that milk is usually stored in the fridge, except during breakfast when it is placed on the kitchen island, the agent's inability to perceive milk could be attributed to different factors. For instance, if the milk is inside the fridge, it can, obviously, only be detected when the fridge is open to ensure an unobstructed view

of the milk carton. Therefore, the visibility of milk to the agent depends on its position, the time of day, and the state of the fridge door.

Alarm example for an explaining away inference in BAYROB | *Figure 68*



$P(Daytime \mid Detected(\text{milk}) = \bot \wedge 58 \leq x_{in} \leq 68 \wedge 70 \leq y_{in} \leq 80 \wedge Nearest\_Furniture = \text{fridge})$: I am located close to the fridge but I can't see milk. What daytime is it? | *Figure 69*

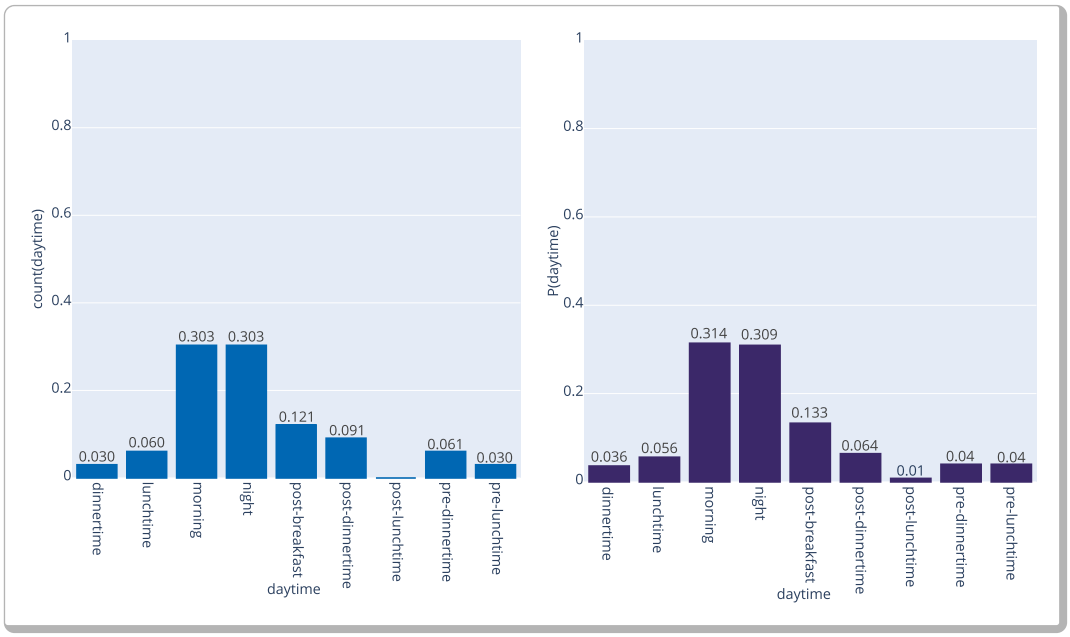
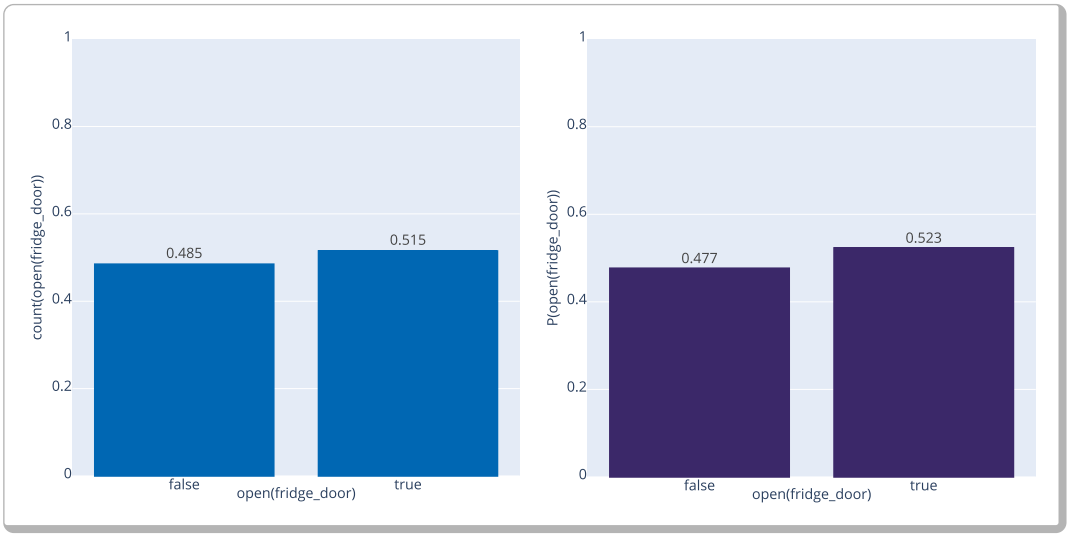


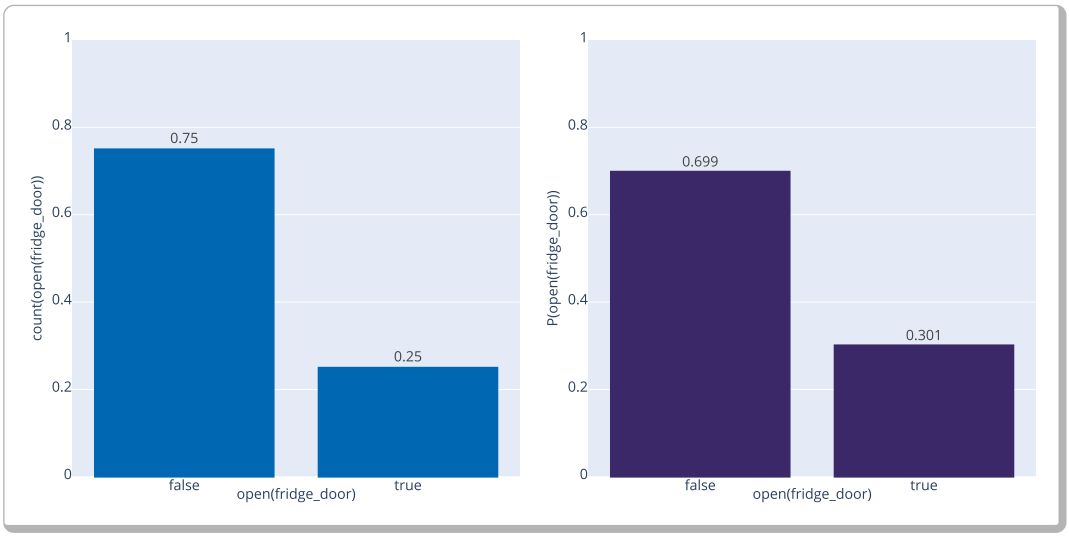
Figure 69 illustrates the distributions for the observation that the agent cannot see the milk but is certain to be in the right place (near the fridge). Examining the distribution of $Daytime$, the most probable values are morning and night, both with a probability of approximately 30%. The reasoning behind the milk being invisible in the morning is straightforward, as the milk is likely not in the fridge. However, regarding the night, this value makes sense only in conjunction with the second cause – the fridge door

(Figure 70), which is indecisive in itself, as it is almost equally probable that the door is open or closed.

$P(Open(\text{fridge\_door}) \mid Detected(\text{milk}) = \bot \wedge 58 \leq x_{in} \leq 68 \wedge 70 \leq y_{in} \leq 80 \wedge Nearest\_Furniture = \text{fridge})$:
I am located close to the fridge but I can't see milk. Is the fridge door open or closed? | *Figure 70*



$P(Open(\text{fridge\_door}) \mid Detected(\text{milk}) = \bot \wedge 58 \leq x_{in} \leq 68 \wedge 70 \leq y_{in} \leq 80 \wedge Nearest\_Furniture = \text{fridge} \wedge Daytime = \text{post-breakfast})$ | *Figure 71*



However, when one of the causes is observed, additional information about the other cause becomes available. Suppose the robot obtains information indicating that it is just after breakfast time. In this scenario, the probability of the variable $Open(\text{fridge\_door})$ being $\bot$ increases to about 70% (cmp. Figure 71). By observing the time of day, the agent acquires additional knowledge that the milk should indeed be inside the fridge at that moment. Since the agent cannot perceive the milk despite favorable conditions, the most likely explanation is that the fridge door is not open, thereby obstructing its view of the milk carton. Scenarios like this enable the agent to take action to establish

a situation that makes the success of the original task (fetch milk) more likely and in general, make more informed decisions.

6.3 Part II: Model settings

Employing JPTs for representing hybrid action models offers the advantage of avoiding assumptions about the shape of the distributions in advance. However, to optimize the results based on the tasks to be addressed with the models, certain design decisions are necessary. The following section assesses several design decisions by comparing their respective likelihoods, both per-variable and cumulatively, on a test dataset. In this evaluation, the original dataset created for the respective models is split (randomly) into a training set (90%) and a test set (10%). The action model is trained with each setting using the training set, and the test set is then employed to calculate the likelihoods.

6.3.1 `turn`

The turn dataset contains variables that can be semantically interpreted as features ($XDir_{in}, YDir_{in}$) and targets ($\Delta_{dir_x}, \Delta_{dir_y}$). Therefore, it is reasonable to explore whether addressing this distinction is beneficial when learning the model. Consequently, the first parameter to compare within the settings is the target. If target variables are omitted, the JPT is learned generatively, treating all variables equally. However, if targets are provided, the model is learned discriminatively, meaning the target variables are ignored when selecting splitting criteria for the tree nodes.

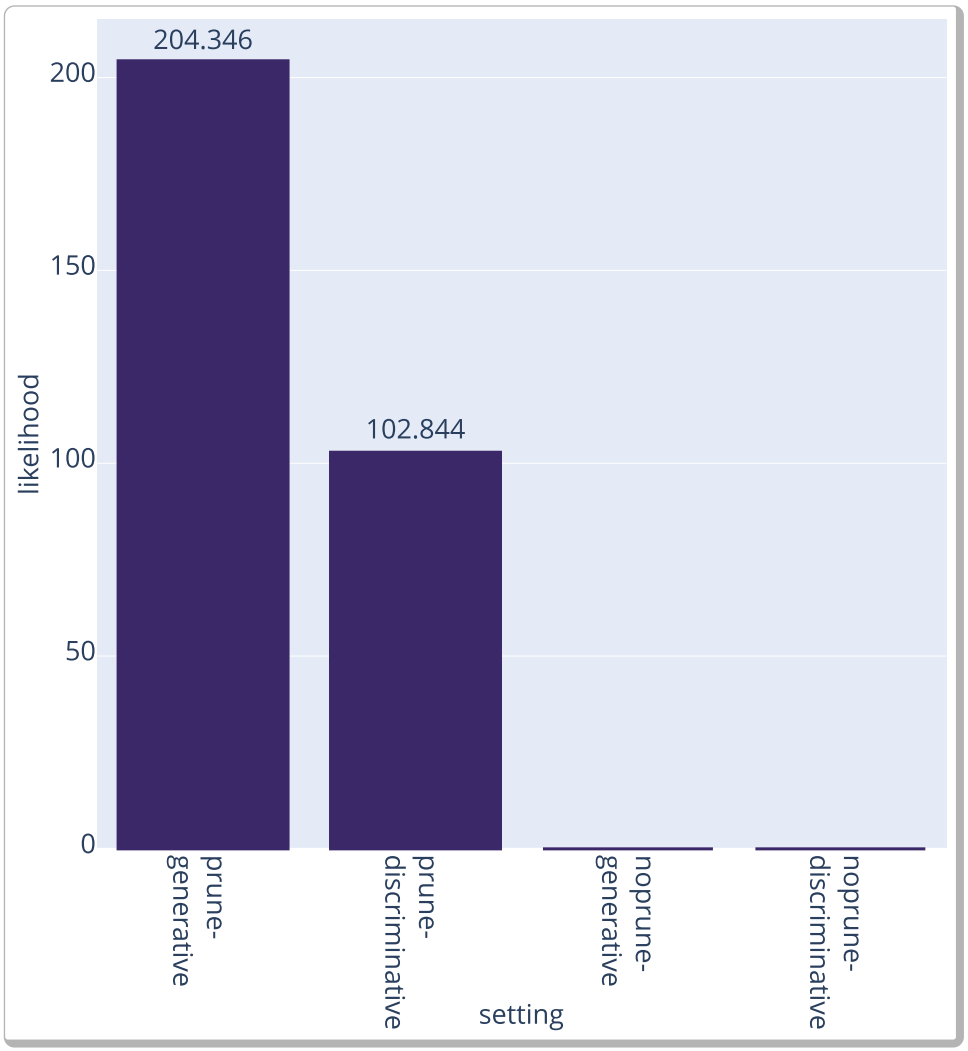
Description of the `turn` model settings | *Table 7*

setting	description
prune-generative	min_samples_leaf: 1 targets: None prune_or_split: $\top$
prune-discriminative	min_samples_leaf: 1 targets: $\{\Delta_{dir_x}, \Delta_{dir_y}\}$ prune_or_split: $\top$
noprune-generative	min_samples_leaf: 1 targets: None prune_or_split: $\bot$
noprune-discriminative	min_samples_leaf: 1 targets: $\{\Delta_{dir_x}, \Delta_{dir_y}\}$ prune_or_split: $\bot$

The second criterion under consideration is the prune_or_split parameter. When this parameter is included in the learning algorithm, a user-defined function influences the splitting behavior. If the function returns $\top$, a leaf node is created, preventing further splitting. Conversely, if the function returns $\bot$, the algorithm is not compelled to perform a split but will still consider other factors. In BAYROB, this function returns $\top$

when the distributions of the child nodes that would be created in the case of a split are too similar to their parent, specifically when the Jaccard similarity of the children and their parent exceeds a certain threshold.

The cumulated likelihood for each setting of the `turn` model | *Figure 72*



Combining these two criteria results in four different settings to compare, as depicted in Table 7. The parameter min_samples_leaf remains identical for all settings, indicating that at least one sample is required to create a leaf node (which is the default and least restrictive value for this parameter).

The per-variable results of the evaluation are presented in Table 8, while the cumulated likelihoods are illustrated through bar plots in Figure 72. A noticeable qualitative distinction emerges between the settings where pruning was applied and those where it was not employed. This is because, in cases where no pruning was enforced and no other constraining criterion, such as min_samples_leaf, was applied, the algorithm tended to overfit the model.

It generated one leaf per sample in the training set, rendering the model unsuitable for making predictions about unseen samples. The circumstance that continuous variables

with infinite domains are employed in the model, which makes it unlikely, that an exact copy of a training sample occurs in the test dataset, promotes the poor performance on the likelihoods.

The likelihoods per variable for each setting of the `turn` model | *Table 8*

		setting			
		prune-generative	**prune-discriminative**	**noprune-generative**	**noprune-discriminative**
variables	**xdir_in**	17.95	16.492	0.0	0.0
	ydir_in	18.182	8.506	0.0	0.0
	angle	0.104	0.105	0.0	0.0
	xdir_out	6.448	6.605	0.0	0.0
	ydir_out	7.407	4.49	0.0	0.0

The superiority in performance of the generative model over the discriminative one is remarkable as well, which justifies the choice of this setting (shaded in blue in Table 8) for the model employed in BAYROB.

6.3.2 `move_base`

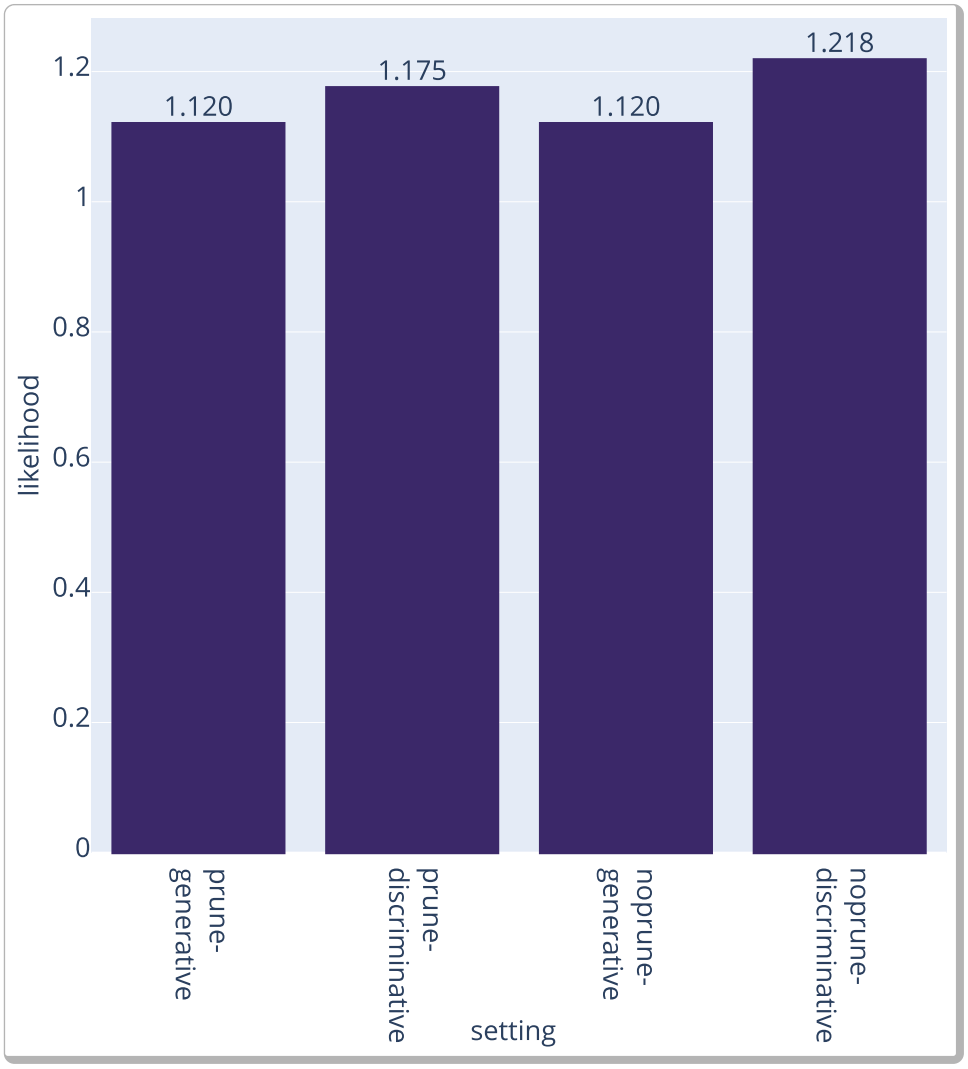
The `move_base` dataset shares similarities with the `turn` dataset, featuring variables that can be viewed as distinct feature ($X_{in}, Y_{in}, XDir_{in}, YDir_{in}$) and target ($\Delta_{pos_x}$, Δ_{pos_y}) variables. The settings under comparison therefore remain the same as before, with the addition of $\text{min\_samples\_leaf} = 0.01$ (equivalent to 1% of the data used for training) to mitigate overfitting (cmp. Table 9).

However, in practical scenarios, the *prune-generative* setting (highlighted in blue in Table 10) has demonstrated superior performance in distinguishing between free positions and obstacles, as well as determining unobstructed move directions.

Description of the `move_base` model settings | *Table 9*

setting	description
prune-generative	min_samples_leaf: 0.01 targets: None prune_or_split: $\top$
prune-discriminative	min_samples_leaf: 0.01 targets: $\{\Delta_{pos_x}, \Delta_{pos_y}, Collided\}$ prune_or_split: $\top$
noprune-generative	min_samples_leaf: 0.01 targets: None prune_or_split: $\bot$
noprune-discriminative	min_samples_leaf: 0.01 targets: $\{\Delta_{pos_x}, \Delta_{pos_y}, Collided\}$ prune_or_split: $\bot$

The cumulated likelihood for each setting of the `move_base` model | *Figure 73*



The likelihoods per variable for each setting of the `move_base` model | *Table 10*

		setting			
		prune-generative	**prune-discriminative**	**noprune-generative**	**noprune-discriminative**
variables	**x_in**	0.012	0.01	0.012	0.01
	y_in	0.011	0.01	0.011	0.01
	xdir_in	25.792	36.737	25.792	40.245
	ydir_in	27.929	41.843	27.929	44.117
	x_out	5.416	3.072	5.416	3.081
	y_out	4.228	2.478	4.228	2.51
	collided	0.998	0.939	0.998	0.939

As anticipated, the min_samples_leaf setting effectively prevented overfitting. It becomes apparent that the impact of the user-defined pruning function seems to be neg-

ligible, at least for this model, and the overall performance of all four settings does not exhibit significant differences.

This is evidenced by the query results in Figure 53, a), and b). The decision for the use in BAYROB is therefore made in favor of the more expressive generative model.

6.3.3 `perception`

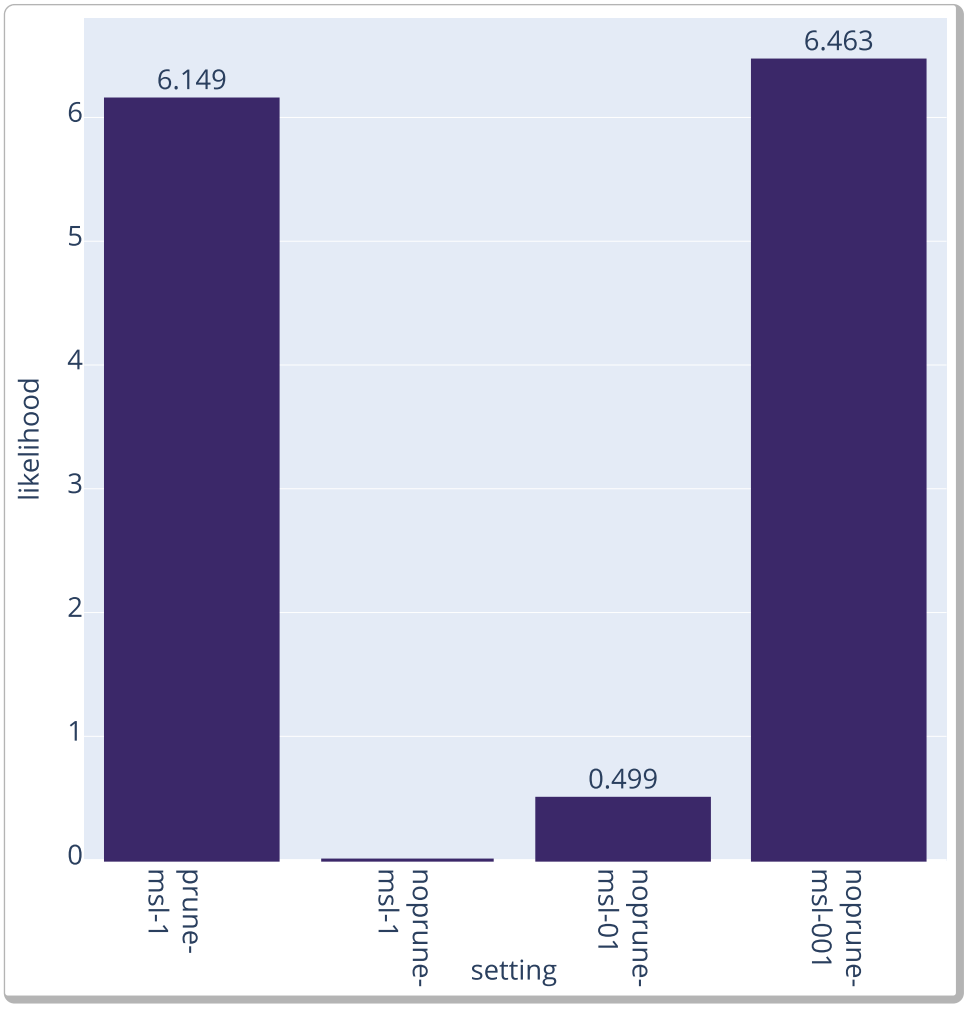
The `perception` model, in contrast to `turn` and `move_base` is the most diverse model in terms of variable types (mixed multinomial/Boolean and continuous) and semantics. None of the variables can clearly be identified as a distinctive feature or target variable, which is why this parameter is omitted in the settings compared in the evaluation of this model.

Due to the limited number of datapoints (772) for this model, the parameter min_samples_leaf is of higher interest here. Table 12 shows again the overfitting effect for setting *noprune-msl-1* when not constraining the model size. In particular, it becomes evident, that the continuous variables are the driving factor for the poor result, since they have 0 likelihood on the test data. The influence of the continuous variables on the overall result also shows in the other settings, even if less pronounced.

Description of the `perception` model settings | *Table 11*

setting	description
prune-msl-1	min_samples_leaf: 1 targets: None prune_or_split: $\top$
noprune-msl-1	min_samples_leaf: 1 targets: None prune_or_split: $\bot$
noprune-msl-01	min_samples_leaf: 0.1 targets: None prune_or_split: $\bot$
noprune-msl-001	min_samples_leaf: 0.01 targets: None prune_or_split: $\bot$

The cumulated likelihood for each setting of the `perception` model | *Figure 74*



The likelihoods per variable for each setting of the `perception` model | *Table 12*

variables	setting: prune-msl-1	noprune-msl-1	noprune-msl-01	noprune-msl-001
x_in	0.086	0.0	0.074	0.119
y_in	0.231	0.0	0.189	0.213
xdir_in	58.922	0.0	48.423	50.893
ydir_in	85.243	0.0	82.816	299.366
daytime	0.703	0.987	0.148	0.693
open(fridge_door)	0.914	0.949	0.851	0.919
open(cupboard_door_left)	0.98	1.0	0.819	0.948
open(cupboard_door_right)	1.0	1.0	0.976	0.997
open(kitchen_unit_drawer)	0.958	1.0	0.79	0.948
open(stove_door)	0.954	0.987	0.878	0.963
detected(cup)	0.994	1.0	0.746	0.977
detected(cutlery)	0.984	1.0	0.854	0.984
detected(bowl)	0.994	1.0	0.746	0.977
detected(sink)	1.0	1.0	1.0	1.0
detected(milk)	1.0	1.0	0.835	0.99
detected(beer)	1.0	1.0	0.88	0.999
detected(cereal)	0.99	1.0	0.797	0.98
detected(stovetop)	1.0	1.0	1.0	1.0
detected(pot)	0.942	1.0	0.792	0.929
nearest_furniture	1.0	1.0	0.98	1.0

The two settings *prune-msl-1* and *noprune-msl-001* both seem to perform comparably well. The learnt models are of comparable size, whilst one was limited in size through pruning based on the node similarity, while the other hard-cuts when a certain threshold of samples per leaf is reached. Since the model size is solely based on the more semantics-based pruning in *prune-msl-1*, this setting is chosen for use in BAYROB.

6.3.4 `pr2`

One could argue that the `pr2` model contains distinctive target variables ($\mathrm{success}$, $\mathrm{failure}$), since these are apparent effects of the execution of an action parameterized by others (type, $\mathrm{bodyPartsUsed}$, $t_{x/y/z}$, ...). However, with the goal of being able to pose any kind of query to the model, it is desirable to go for the more expressive learning strategy, which is *generative*. The evaluation of settings for this model is therefore not targeted towards comparing a generative vs. discriminative approach but rather different model sizes within a generative model. Compared to the `perception` model, the `pr2` dataset is rather large, which raises the question, how many of them are required per leaf to form preferably general yet expressive distributions. As mentioned before, the datasets is sparse w.r.t. pose data, i.e. the variables t_x and t_y are hardly ever filled with

meaningful values (t_z is 0 for all data points since the base of the robot is assumed to only move horizontally in its environment). Since JPTs cannot handle missing values, a default value of 0 is assumed for missing continuous values, "None" for missing multinomial values.

Description of the pr2 model settings | *Table 13*

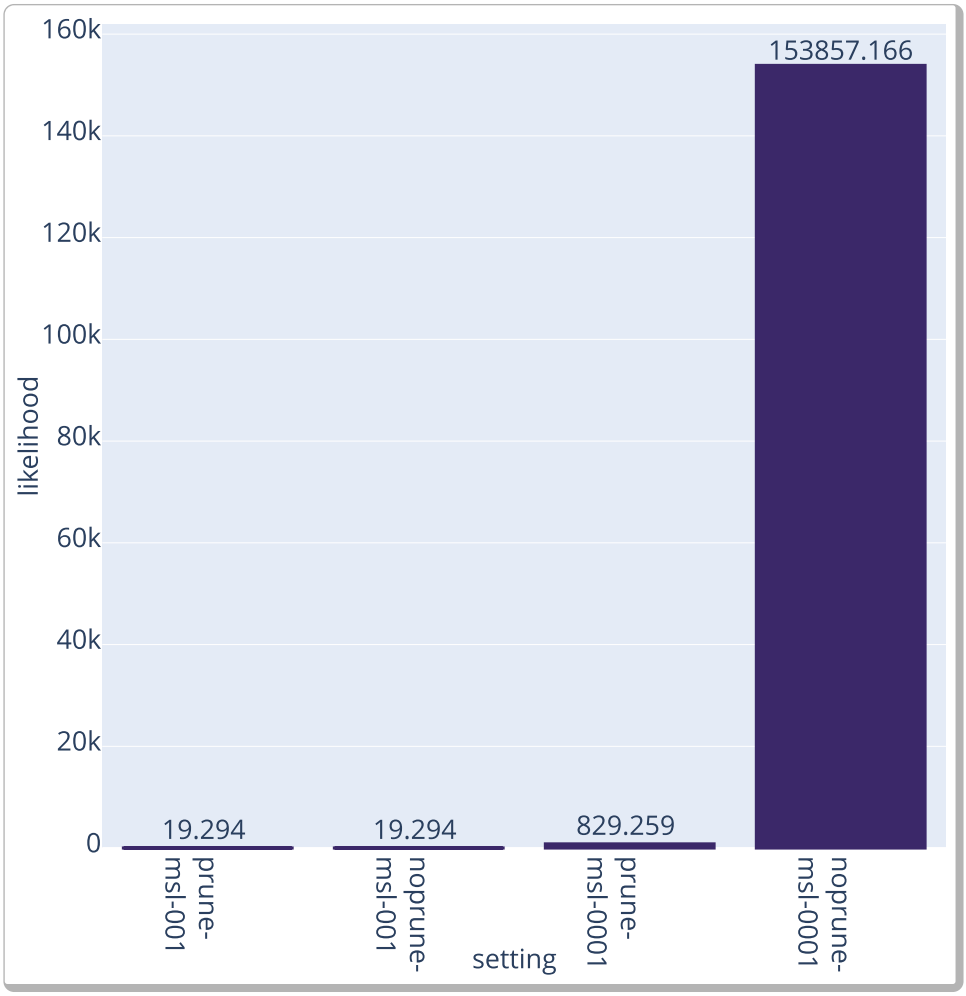
setting	description
prune-msl-001	min_samples_leaf: 0.01 targets: None prune_or_split: ⊤
noprune-msl-001	min_samples_leaf: .0.01 targets: None prune_or_split: ⊥
prune-msl-0001	min_samples_leaf: 0.001 targets: None prune_or_split: ⊤
noprune-msl-0001	min_samples_leaf: 0.001 targets: None prune_or_split: ⊥

Analogous to the perception evaluation, the settings differ in their value of min_samples_leaf and whether or not they use the user-defined pruning function. The results look somewhat surprising at first, since there are some outliers with unusually high values (cmp. variables $duration$, t_y, t_z, and $angle_z$ in Table 14).

The likelihoods per variable for each setting of the pr2 model | *Table 14*

		setting			
		prune-msl-001	**noprune-msl-001**	**prune-msl-0001**	**noprune-msl-0001**
variables	**type**	0.93	0.93	0.35	1.0
	duration	19.568	19.568	1.827	204.598
	success	0.984	0.984	0.996	1.0
	failure	0.974	0.974	0.996	1.0
	object_acted_on	0.972	0.972	0.996	0.999
	bodyPartsUsed	0.985	0.985	1.0	1.0
	arm	0.985	0.985	1.0	1.0
	t_x	0.948	0.948	1.036	1.354
	t_y	0.947	0.947	0.987	470.88
	t_z	4586305.46	4586305.46	2026190.604	425435.32
	information	0.987	0.987	0.996	1.0
	angle_z	0.938	0.938	3.628	4.179

The cumulated likelihood for each setting of the `pr2` model | *Figure 75*



The reason for this phenomenon lies in the abovementioned circumstance that for some columns in the dataset, many identical values occur, causing jumps in the PLFs. The slope in such jumps is ∞, which would cause further calculations to be indeterminate. Through the introduction of a scaling factor, the infinity values are limited to an upper bound, however, numeric side effects or errors are imminent. Since the appearance of large clusters of data points with equal values in certain variables in this case highly depends on how the training and test set are sampled from the overall dataset, multiple runs of the settings evaluation can yield very different results. However, in most cases, the abovementioned effect occurs to a greater or lesser extent.

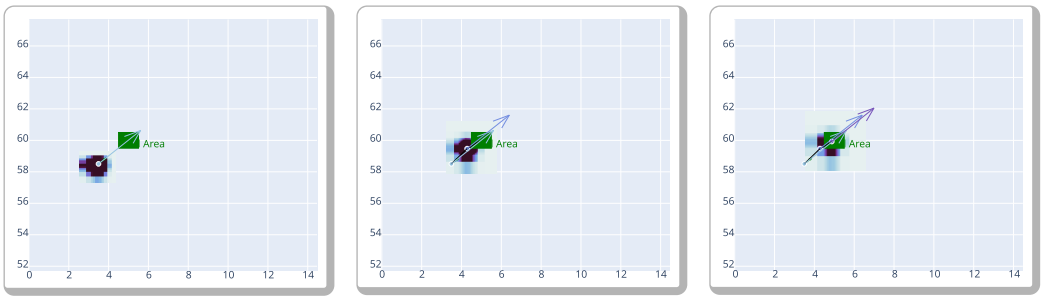
In practice, the *noprune-msl-0001* setting has proven the most promising configuration, however, the model still has its flaws due to the incomplete and inconsistent data situation. For further testing, it is required to collect further robot data from real experiments.

6.4 Part III: Plan Refinement

In Section 4.6, the idea of of using forward- and backward action updates in combination with an A*-like search to refine robot plans was introduced. This part of the evaluation will serve as a proof of concept for this appoach, in which the forward- and backward search are tested in example scenarios. Plan refinement in the context of this work is to be understood as the process of breaking down a superordinate, rather abstract higher-level task into multiple lower-level, robot-executable subtasks. Given a current position and a goal position, refining the plan "Go from position A to position B" would then be a sequence of multiple "Move 1 step forward" and "Turn *x* degrees" describing a path from position A to B. As a first example, an initial state with $X_{in} \sim \mathcal{N}(3.5, 0.05)$, $Y_{in} \sim \mathcal{N}(58.5, 0.05)$, $XDir_{in} \sim \mathcal{N}(0.7, 0.01)$ and $YDir_{in} \sim \mathcal{N}(0.7, 0.01)$ is assumed. All the variables that are not set explicitly are considered unknown, i.e. it is unknown, whether certain objects or funiture items are visible to the robot from its current position and whether doors and drawers are open or closed. The agent is tasked with finding a path to an arbitrary position within a rectangular area defined by the intervals $X_{in} \in [4.5, 5.5]$ and $Y_{in} \in [59.5, 60.5]$. As expected, since only a very small distance is to be overcome and the facing direction is already set towards the goal area, very few actions are required to reach the goal. In fact, only two `move_base` steps are necessary, as shown in Figure 76.

The forward search path example from an initial position to a position within the goal area (green rectangle) | *Figure 76*

Step 0 (initialization) | *a)* Step 1 | *b)* Step 2 | *c)*

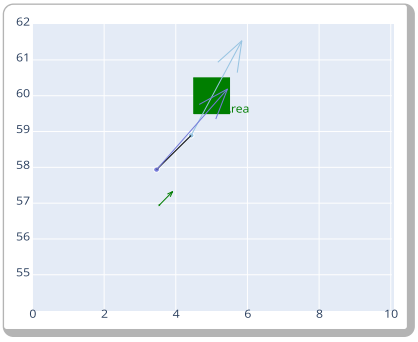


The backward search returns a slighly different result containing only one action step. This is not surprising, since in the forward search direction, the resulting path is generated by applying actions consecutively and updating the initial distribution with each step. The nature of the reverse direction search, however, does not allow that, such that each step along the path is generated from the in-distributions of its successor, which may lead to slightly different results. Moving the initial state a little bit further from the goal ($Y_{in} \sim \mathcal{N}(57, 0.05)$) then results in a path with more than one step. It is yet to be shown, that the forward execution of the path found by the backward search truly results in the desired goal state which places the agent in a position within the specified rectangular area. Since each action step corresponds to a particular leaf in an action

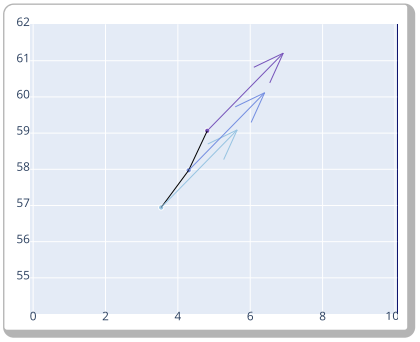
tree model, the consecutive update of the state through the forward application of the leaf will result in the most probable (still prospected) agent state.

Comparison of the found path vs. its forward execution of the previous example using backward search | *Figure 77*

The refined plan generated by the backward search algorithm | *a)*



The prospected result of the forward execution of the refined plan | *b)*



The path found by the backward search as well as the result of the forward execution of the found path are depicted in Figure 77. While there are minor differences in the first action step, the results (i.e. the second state) are fairly similar. Note that the node representing the final state is not placed within the rectangular goal area (Figure 77 a). This is due to the positions of the nodes being set at the expectation values of the respective position variables. The entire distribution may therefore still share a large enough area with the goal to be considered a goal state by the algorithm.

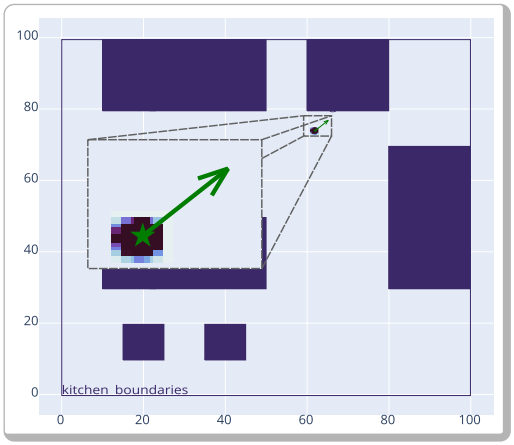
Much more interesting is the plan refinement when it comes to tasks that require *semantic* refinements. Probabilistic action models in BAYROB in combination with a search algorithm are capable of refining underspecified tasks such as “Detect milk”. From the `perception` model, the agent can infer that milk can be detected when certain criteria are met. Figure 38 in Section 4.6.1 show a subset of these criteria, i.e. predecessor candidates for the goal specification $Detected(milk) = \top$. These criteria limit the agent’s position and facing directions from which it can detect milk, as well as the time of day and the state of the fridge door. Given an initial state with $X_{in} \sim \mathcal{N}(62, 0.1)$, $Y_{in} \sim \mathcal{N}(74, 0.1)$, $XDir_{in} \sim \mathcal{N}(0.3, 0.01)$ and $YDir_{in} \sim \mathcal{N}(0.9, 0.01)$ then has to again refine the task to get from the current location to one of the locations of the predecessor candidates, as shown in the first example. The resulting refined plan for the “Detect milk” task is shown in Figures 78 a) - d).

Please note that plan refinement incurs a substantial computational cost, primarily due to the computationally intensive nature of distribution convolutions involved in the belief state update. The backward search, in particular, necessitates multiple convolutions for each leaf in each search iteration, in the worst-case scenario. To manage this complexity, various measures have been implemented to keep these operations at a reasonable level. For instance, after a distribution convolution, the resulting distribution is approximated to maintain a manageable number of PLFs. While the approximation

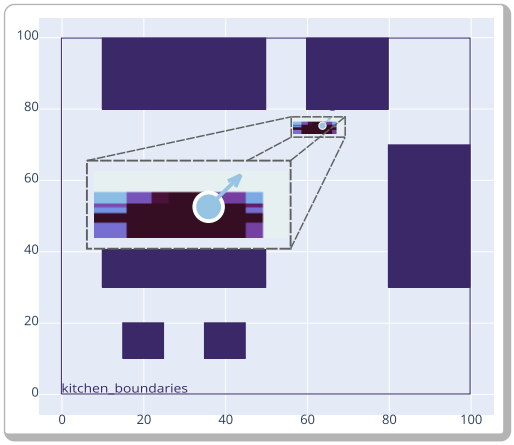
process requires additional computational power, the effort proves worthwhile when the resulting, less complex distribution is reused for convolution in subsequent steps.

Plan refinement of the task "Detect milk": $Detected(milk) = \top$ | *Figure 78*

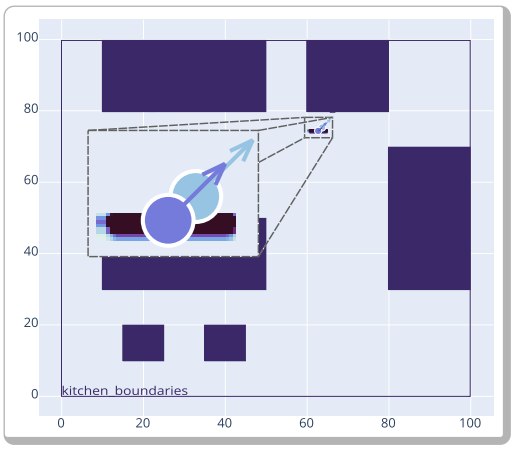
Step 0 (initialization): "I am somewhere here" | *a)*



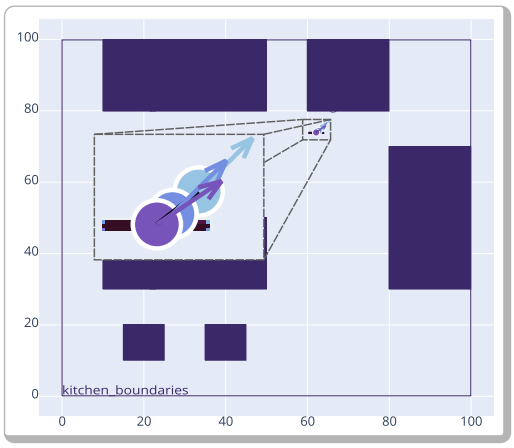
Step 1: "How do I detect milk?" | *b)*



Step 2: "How do I get to the position from the candidate found in b?" | *c)*



Step 3: "How do I get to the position from the candidate found in c?" | *d)*



Given the computational complexity and the expansive nature of the search space (refer to the aforementioned approximations) in the running example, it may be necessary to expand thousands of nodes in the search tree, depending on the complexity of the query. The completion of this task could take several hours, contingent on the underlying hardware. As a reference point, the task completion depicted in Figure 76 took $\sim$ 5 minutes and expanded 24 nodes, while the one in Figure 78 took 4 minutes and expanded 29 nodes on an Intel Core i7, 1.9 GHz Quad-Core Processor with 16GB RAM, even though the resulting action sequences only contained a very small number of steps.

Example of a CRAM plan generated from search result in Figure 78 | *Code 3*

```
defplan path(goal_location, robot_location)
    perform (an action
                (type move_base)
                (min_x_in 60.43)
                (max_x_in 63.75)
                (min_y_in 73.96)
                (max_y_in 73.97)
                (min_xdir_in 0.74)
                (max_xdir_in 0.87)
                (min_ydir_in 0.52)
                (max_ydir_in 0.7))
    perform (an action
                (type move_base)
                (min_x_in 60.89)
                (max_x_in 64.45)
                (min_y_in 74.28)
                (max_y_in 74.51)
                (min_xdir_in 0.65)
                (max_xdir_in 0.70)
                (min_ydir_in 0.71)
                (max_ydir_in 0.76))
    perform (an action
                (type perception)
                (min_x_in 61.59)
                (max_x_in 65.73)
                (min_y_in 75.28)
                (max_y_in 75.35)
                (min_xdir_in 0.68)
                (max_xdir_in 0.77)
                (min_ydir_in 0.82)
                (max_ydir_in 0.82))
```

The found action sequence in Figure 78 forms the base for instructing an autonomous agent how to perform a task. The action designators as well as their respective parameterization can be extracted from the distributions and fed to the agent in the form of a CRAM plan. For this example, the resulting plan is shown in Code 3. The generation of a CRAM plan can be executed automatically from a BAYROB search result.

6.5 Part IV: BAYROB and BAYROB Web

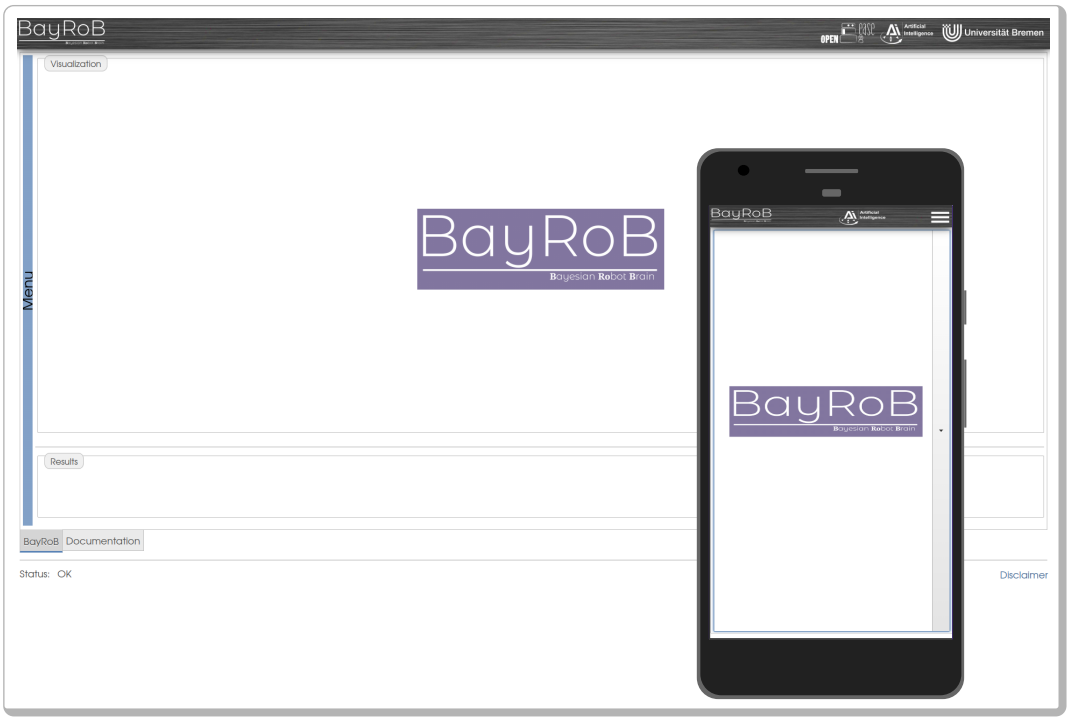
The BAYROB system will be released as an open-source Python implementation on GitHub[42]. This release aims to make the insights from this work accessible to the scientific community. The system not only allows for the exploration of the models outlined in this thesis but also facilitates learning and integration of new models into the system.

Furthermore, the web tool BAYROB web (Figure 79), developed as part of the BAYROB framework, provides a comprehensive exploration of each action model within the system. Users can not only replicate the evaluation results presented in this chapter, ensuring the reproducibility of outcomes but also expore action descriptions and engage

[42]GitHub, a developer platform

in plan refinement, employing both forward- and backward inference methodologies introduced earlier.

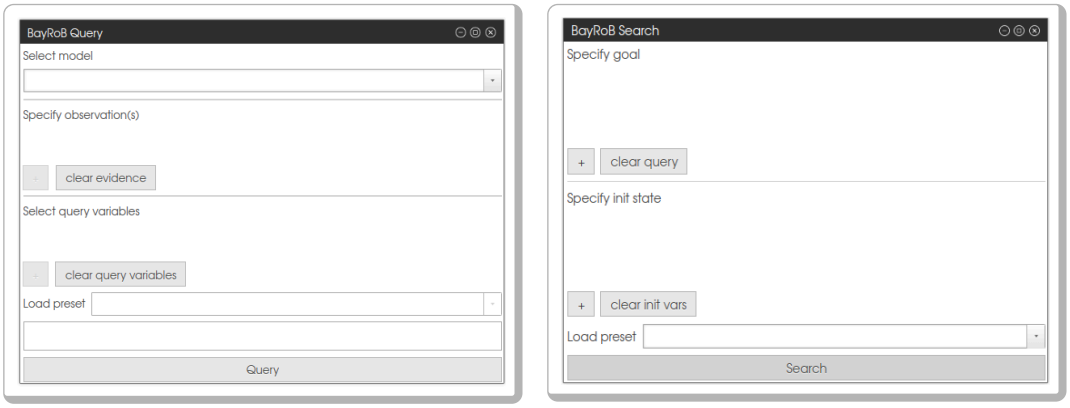
The BAYROB web app | *Figure 79*



The evident benefits encompass the app's capacity to expose significant aspects of the BAYROB system's functionalities to the scientific community. This includes features like action prospection, informed decision-making, and plan refinement. The web app facilitates two types of requests: the *model query* and the *search*.

The query and search options in the BAYROB web app | *Figure 80*

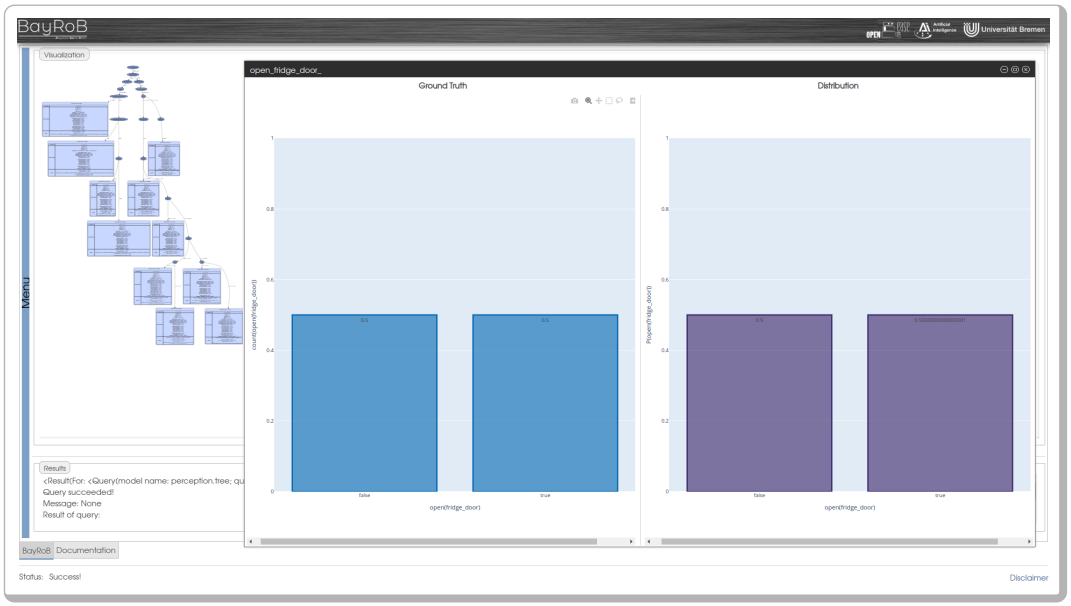
Query | *a)* Search | *b)*



The query function, depicted in Figure 80 a), enables users to define a set of observations (evidence) and select one or more variables for querying in a specific action model. Upon clicking the *Query* button, the system processes the request, triggering internal

inference and visual processing, showcasing ground truth and distribution plots for the designated query variables. Users can also request the visualization of the conditional tree resulting from the query. While the query modeling assumes a certain understanding of model characteristics, users receive assistance through predefined queries and suggestions for valid values when selecting an evidence variable. A text field in the lower section of the BAYROB *Query* window offers a preview of the sent request to the model.

The result of a query in BAYROB shows separate windows for the distribution of each requested variable. The conditinal tree can be visualized in the background (optional) | *Figure 81*



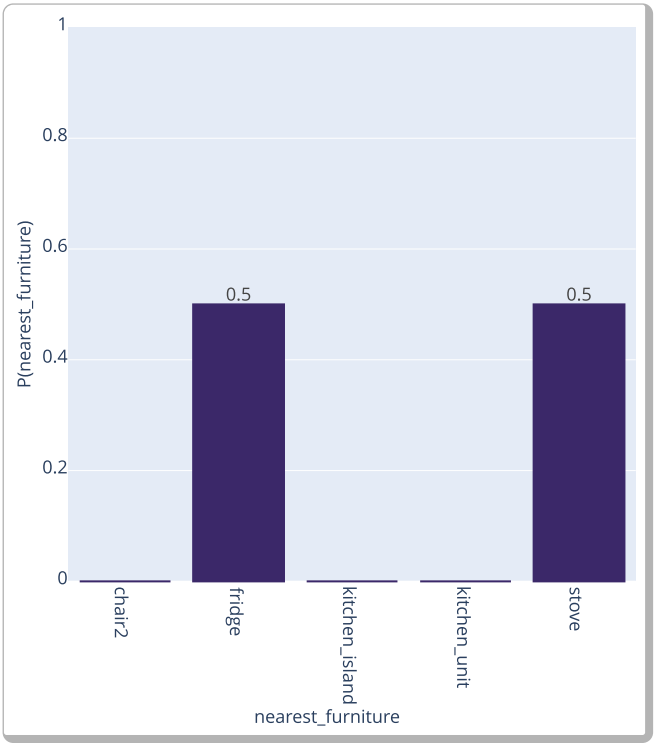
The execution of the search, illustrated in Figure 80 b), activates the plan refinement pipeline, internally incorporating an A*-like search of predecessor/successor steps for a specified query. Users can define a goal and an initial state by choosing variables within the models of the BAYROB system's running example (`turn`, `move_base`, and `perception`) and specifying allowed value ranges. For numeric variables, these specifications may include a range (interval) of values or a value complemented with a tolerance.

A value v accompanied by a tolerance t will be converted into a Gaussian distribution $\mathcal{N}(\mu = v, \sigma = t)$. When dealing with a range of values, the transformation will result in a uniform distribution that encompasses the specified values. For multinomial and Boolean variables, the provided value(s) can be a singular value or a set of values separated by commas. This set will be transformed into a distribution that uniformly allocates the probability mass across the specified values, assigning a probability of 0 to all other potential values within the variable's domain. To illustrate, consider the distribution of the specification of the variable $Nearest\_Furniture$ with values {"fridge", "stove"}, depicted in Figure 82.

Again, to learn how the system works, users can choose a predefined specification from the dropdown menu. Upon clicking the *Search* button, the plan refinement process will

be triggered. In other words, the BAYROB system will attempt to discover a sequence of actions that transforms the specified initial state into the desired goal state. It is important to note that plan refinement may take a considerable amount of time, potentially exceeding the session lifetime of the web app due to the involvement of complex computations. Thus, users are advised to exercise caution and consider restricting the search to smaller examples, as suggested by the presets.

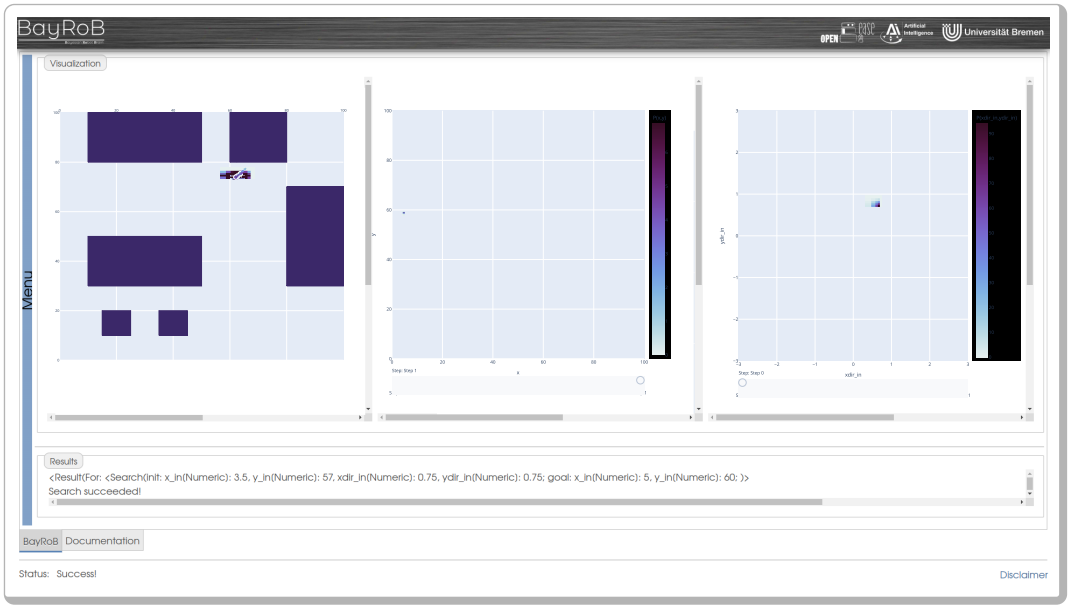
The "uniform" distribution over the allowed values for the variable *Nearest_Furniture* | *Figure 82*



The result of a successful search includes a) a plot of the found path b) an animation of the distribution updates of the position variables and c) an animation of the distribution update of the direction variables. The plots will be displayed in the main window of the application, along with additional information about the results in the lower third of the main window.

Essentially, the BAYROB web app encourages researchers to explore probabilistic modeling and decision-making within the domain of BAYROB. It functions as a tool for investigation, providing a means to deepen understanding of the system's internal mechanisms in the context of this research. The app will be complemented by documentation that includes a user manual and an API specification for the BAYROB framework.

The result of a search in BAYROB shows separate windows for the distribution of each requested variable. The conditinal tree can be visualized in the background (optional) | *Figure 83*



chapter seven

Related Work

In this chapter, a comprehensive examination of related works is presented, specifically addressing two interconnected areas. Firstly, the foundational research is explored with a primary focus on the related work concerning learning and knowledge representation as well as the utilized formalism of JPTs. The aim of this section is to establish the theoretical foundations of the approach, investigating the research that has contributed to the development and understanding of JPTs. Concurrently, the investigation extends to the broader context of research relating to the BayRoB system as a whole. The diverse body of research surrounding and complementing the BayRoB framework is reviewed.

The awareness that robots need real understanding to solve tasks to a level that allows them to draw conclusions about what subtasks can be identified and how to solve the tasks without consulting further human assistance is crucial to equip the agent with essential knowledge. There are numerous approaches to build large-scale knowledge databases by learning structured data from text like *Never Ending Language Learning (NELL)*[43] (Mitchell et al. 2018), *YAGO* (Suchanek, Kasneci, and Weikum 2007), *Freebase* (Bollacker et al. 2008) and Google's *Knowledge Graph* (Pujara et al. 2013), or from other web content like Google's *Knowledge Vault* (Dong et al. 2014), to only name a few. The information extraction algorithms of these systems employ different (often cascaded) machine learning techniques, to transform unstructured data into machine-readable and -interpretable formats that can then be analyzed and used to derive new knowledge. They vary from rule-based or statistical information extraction techniques to named-entity recognition, from (semi-)supervised learning like bootstrapping, to clustering, classification, reinforcement learning and deep learning.

◁ *knowledge bases*

There also exist knowledge bases for *robots*, for example the one introduced by Sijs and Fletcher (2021), who adopt a hypergraph model to maintain a hierarchical knowledge base that allows real-time updates from the robot's sensors. The knowledge processing system *KnowRob* (Tenorth and Beetz 2013) and its successor, the knowledge representation and reasoning framework *KnowRob2* (Beetz et al. 2018), allow a robotic agent to

[43] NELL

acquire open-ended manipulation skills and competence, while enabling it to reason about how to perform manipulation tasks and simultaneously obtain commonsense knowledge. For a more detailed comparison of multiple robotic knowledge bases the reader is referred to Thosar et al. (2018) who evaluate the systems by means of the criteria *knowledge acquisition*, *knowledge representation*, and *knowledge processing*.

knowledge representation, learning & reasoning ▷

In Kaelbling and Lozano-Pérez (2013), a belief state representation involves a full joint Gaussian distribution over object poses. This is possible, as the domain described comprises comparatively few objects, whose locations are discretized. The authors pursue to find approximate solutions to exceedingly complex *Partially Observable MDP (POMDP)* problems through a process of planning within belief space, employing streamlined models, and iteratively adjusting the plan as needed. A characteristic of POMDPs is that they can effectively represent the value functions in a *finite* world with *finite* state, action and measurement spaces by incorporating piecewise linear functions. For real-world scenarios, however, they are typically not considered to suffice, because of the huge complexity of the belief spaces.

The domain of probabilistic learning and reasoning is a vital and actively explored area within AI, with numerous studies in recent years focusing on improving the tractability of probabilistic models. Prior research, exemplified by Poon and Domingos (2011), introduced SPNs as deep architectures for representing probability distributions. Unlike SPNs, JPTs operate independently of a predefined dependency model prior to learning and facilitate explainable reasoning across symbolic and continuous variable domains.

In contrast to PCs (Shah et al. 2021), JPTs autonomously learn their dependency model from data and can articulate distributions without imposing any model assumptions. Traditional Probabilistic PGMs typically require a fixed dependency model to be known before parameter learning, thereby introducing an additional NP-hard problem when structuring the model. However, in the case of JPTs, the dependency model is acquired automatically from the data, eliminating the need to specify a model in advance.

Peharz et al. (2020) identify a significant hurdle faced by probabilistic circuits, particularly their scalability compared to other deep generative models. Since research typically focuses on the flexibility and expressiveness of the developed models, comparatively little attention is paid when it comes to tractable modeling. There is often no investigation and formal proof of the set of inference tasks that can reliably be addressed within the models. By proposing *Einsum Networks (EiNets)*, a novel implementation design for PCs is introduced, which significantly enhances speed and memory usage by integrating numerous arithmetic operations into a unified einsum operation.

Since the typical approach to learning PCs often involves employing greedy and time-intensive algorithms, Di Mauro et al. (2021) present a novel unified method for effectively learning PCs with various structural characteristics. Specifically, the concept of extremely randomized *extremely randomized Probabilistic Circuits (XPCs)* is proposed, which are PCs featuring a random structure based on random logical constraints-based conditioning. XPCs satisfy the conditions for smoothness and decomposability and can be designed to be decomposable as well. Their superiority over other density estimators is validated on standard benchmarks for density estimation tasks.

Similar to JPTs, cutset networks (Rahman, Kothalkar, and Gogate 2014) employ recursive partitioning, but accommodate arbitrary BNs within the partitions. Nevertheless, cutset networks are constrained by the complexity of the Bayesian networks in the leaves and lack the capacity to generally represent continuous variables. JPTs can thus be viewed as expansions of both SPNs and PCs that efficiently learn probability distributions over hybrid domains without requiring pre-defined model structures.

Bishop (1994) introduces *Mixture-Density Networks (MDNs)*, which integrate conventional neural networks with mixture density models. MDNs offer a comprehensive framework for modeling conditional probability densities of the output variables, thus facilitating multi-valued mappings. Analogously to JPTs, MDNs are mixture models representing distributions over symbolic and subsymbolic variables. However, unlike MDNs, JPTs are generative models that represent joint distributions over all variables instead of conditional probability densities of the output variables, thereby limiting MDNs to be discriminative.

Moreover, JPTs leverage the advantages of transparent tree structures that are easily interpretable, provide comprehensible decisions, and enable efficient learning and reasoning processes. Bishop and Svensén (2012) employ variational inference with *Hierarchical Mixture of Experts (HME)* to offer a Bayesian approach to modeling. This approach is applicable to solving multi-dimensional regression tasks and is anticipated to be effective for binary classification as well. However, the HME often converge to local optima that may not necessarily be optimal, a challenge addressed by the authors through multiple runs to select the best solutions. Nonetheless, this approach might be impractical for large datasets. For a more comprehensive exploration of mixture-of-expert models, Yuksel, Wilson, and Gader (2012) can provide additional insights.

Beetz and Grosskreutz (2005) discuss a framework for modeling and predicting concurrent behavior in autonomous mobile robots using *Probabilistic Hybrid Action Models (PHAMs)*. PHAMs enable the representation of continuous feedback control processes, non-deterministic effects, interference modes, and exogenous events. The authors focus on integrating AI planning with robotic control systems, emphasizing the need for more realistic models of robot behavior and physical effects - aiming to improve the adaptability, efficiency, and reliability of autonomous robots especially in complex and dynamic environments. PHAMs are designed to predict robot behavior with high probability, aiding in plan revision and decision-making during execution. The authors propose the use of PHAMs for *Concurrent Reactive Plans (CRPs)*, which involve the simultaneous execution of multiple control processes that interact with each other and the environment, and online plan revision, highlighting their potential to enhance the performance of autonomous service robots. The authors also discuss the challenges of rule learning and the extraction of projection rules from experience. These rules guide the prediction process by encoding patterns of behavior observed in actual executions. The challenge lies in developing mechanisms to automatically extract these rules from learned plans and execution traces. PHAMs enable the prediction of robot behavior during plan execution. By analyzing the projected behavior, the system can identify potential flaws or undesired outcomes and revise the plan in real-time. This prediction-based plan revision enhances the robot's ability to adapt to changing conditions and optimize its actions improving the efficiency and reliability of autonomous robot con-

◁ *autonomous agents*

trol. The strength of PHAMs lies in capturing the physical effects and temporal structure of robot behavior, particularly in complex and uncertain environments.

When dealing with autonomous robotic systems, one has to concern oneself with large domains in open world scenarios, handling hybrid domains and the informed decision making under uncertainty. These topics are key to successfully developing autonomous agents that can confidently act in real-world scenarios and are therefore heavily investigated in the scientific community. Numerous works investigate planning under uncertainty (Ha, Driess, and Toussaint 2020; Enachescu et al. 2021; Blythe 2001; Karlsson 2001; Indelman, Carlone, and Dellaert 2014; Jain and Niekum 2018). Very prominent are the works by Kaelbling, Littman, and Cassandra (1998), who present an approach to handle larger problems by using function-approximation techniques and employing simulation to focus the approximations on the frequently visited regions of the belief space. The authors present the prospect of incorporating hidden Markov models to learn POMDPs to allow the application of the approach presented in their work.

The approach presented in this thesis does not align with classical planning strategies, where the objective is typically to determine a sequence of actions to achieve a predetermined goal. An example is the *Stanford Research Institute Problem Solver (STRIPS)*, first introduced by Fikes and Nilsson (1971).

McDermott (1992) argues that, rather than classical planning, which typically employs logical formalisms, a different framework is required. This framework should not only incorporate programming language semantics and the ability to anticipate abstract intentions prior to determining their realization or feasibility, but also integrate probability and control theory.

classical planning vs. concurrent reactive plans ▷

BAYROB operates within the framework of concurrent reactive plans, which involve defining object and designators and adjusting them dynamically based on contextual factors and experience during runtime. This allows for the attainment of desired outcomes within specific environments by adapting actions in response to changing conditions. This approach incorporates the fundamental concept of using experiential knowledge to *understand* a task, which is essential to improve decision-making. In a similar context, Laird et al. (2017) propose *interactive task learning* where an agent collects experience by naturally interacting with a human instructor, allowing it to learn what a task *is*.

This approach followed in this work is inspired by works of Beetz and Grosskreutz (2005) mentioned earlier, and aligns with Beetz (2002). While Nyga et al. (2018) introduced a probabilistic framework that generates robot-executable action sequences from vague natural-language instructions, BAYROB uses experiential knowledge to provide the necessary information for augmenting underspecified instructions. However, both approaches aim at solving the problem probabilistically to account for the uncertainty grounded in real-world environments.

Developing control programs that can adapt to complex and dynamic environments to enable the handling of complex decision making processes is a recurring subject in many research propositions. Fedrizzi (2011) captures this idea to find what he calls *Action-Related Places (ARP LACEs)*, i.e. robot base positions that are most likely ideal for the successful execution of grasping tasks. The author argues, that certain abilities

are necessary for a robot to robustly act in unconstrained environments, including handling uncertainty, dynamically adapt to changing environments and make adequate decision in unclear situations and that pre-programmed solutions do not solve these issues. BAYROB extends this approach to the entire belief state, which is represented by hybrid probability distributions over action and object designators.

◁ *cognition*

Tenorth and Beetz (2012) present an approach to augment underspecified (natural-language) task descriptions with infomation about what is missing in the instruction and where to find relevant pieces knowledge. The framework can also anticipate the effects of action executions of everyday activities, such as those in cooking recipes. Their system integrates different knowledge sources and allows to reason over them to fill missing information in incomplete instructions. Beßler, Koralewski, and Beetz (2018) extend the KNOWROB system (Tenorth and Beetz 2013) by representing actions using force dynamic events. The models are used to acquire episodic memories, which link performed actions with sub-symbolic data. The episodic memories can be used as experiential data in future experiments or by other robots. While the inference in both works is performed using static (prolog) rules (leveraging additional ontological knowledge) and is therefore grounded in logical reasoning, BAYROB allows probabilistic reasoning through the incorporation of probability distributions over actions and objects.

chapter eight

Conclusions

This work has proposed BayRoB, a probabilistic framework incorporating hybrid action models to support autonomous agents in making informed, context-based decisions under uncertainty. While robot control programs are typically hybrid, yet rather static state machines, BayRoB superimposes them with probability theory over joint probability distributions, which has a huge impact on the decision making capabilities of the robots, since all sorts of queries are imaginable. In particular, this applies to the agent's competence in its anticipation skills, which are key to choosing the best actions in the given situation.

Developed as part of the EASE research focus, the BayRoB framework employs a probability-based approach to refine action models, aligning seamlessly with the NEEMs principles for informed decision-making. At the heart of BayRoB lies the concept of action cores, representing potentially higher-level action descriptions. The system introduces a subsymbolic component into these cores, crafting hybrid action models that elevate decision-making processes. Through a formalism capable of representing and computing complex probability distributions in hybrid domains, BayRoB stands as a notable advancement in probabilistic modeling.

JPTs were introduced as tree-based representations that allow the compact representation and efficient learning of and reasoning about joint probability distributions. As opposed to PGMs, JPTs support learning and reasoning in hybrid domains, i.e. they allow to coalesce numeric and symbolic variables in one single model in a sound and consistent way. A key feature of JPTs is that the dependencies among variables are represented in a tree structure, whose leaves maintain univariate distributions over all variables. For numeric variables, quantile-parameterized distributions are proposed, which can be regarded as model-free representations of arbitrary numeric distributions. The experiments show that JPTs are able to accurately learn and represent complex interactions between many variables, while keeping interpretability and scalability. It can be argued that the challenges that are addressed with JPTs will be key in making the transition from narrow, specialized expert systems to hybrid, high performance AI systems

and in pushing todays computer systems towards more generally intelligent and more robust and reliable decision making.

Within an exemplary kitchen environment and using a selection of action designators, the BAYROB methodology, including algorithms and plan refinement strategies were showcased. The core of this approach lies in the representation of belief states as joint probability distributions and the concomitant update strategies. The update of the continuous distributions in the belief state follows the natural understanding of *adding* and *subtracting* values and thus performs a convolution of the distributions to emulate this behavior. The execution of multiple actions then entails the consecutive computation of belief state updates, each performed on the result of the respective preceding one. The observation that, with an increasing number of action executions and their consequent belief state updates, the belief state distributions tend to increase in terms of complexity as well as uncertainty is addressed and means to tackle these issues are presented.

The forward update of the belief state corresponds to the anticipation of the outcome of an action, that is, how the execution of that particular action affects the robot and which environmental changes are to be expected. On the one hand, this knowledge can be used to prepare the agent for the situation *after* the action execution and provide essential information about what to do next. On the other hand, the anticipation skills can be used to *prevent* certain situations, e.g. the one in which the action execution fails or produces an undesirable outcome. A pancake might still end up on the floor, even though the flipping action itself was carried out successfully, due to an unfavorable positioning of the pan causing the pancake to slip. By computing the action with the highest probability of producing a desired outcome, the prediction capabilities can be used to select a favorable action and parameterization in the first place. The backward action update therefore corresponds to the search of a state and an action parameterization that leads to a desired outcome.

Both update types are employed when searching for a sequence of actions that help the autonomous agent in successfully performing a given task. In general, the search is used to reach a certain goal, which can be interpreted as an underspecified action plan that is to be refined. The approach is developed to increase the autonomy in robotic agents but is not limited to this application. On the contrary, the methodology can be used in many applications where experiential knowledge is available (and machine interpretable) and can be used to reason over actions and resulting environment changes. Apart from the robotic domain, this work presented the predecessor of the BAYROB system in the context of material design applications which is similar in its challenges since the goal is to develop novel materials with a given set of properties in the vast range of possible material combinations, properties, and processing steps.

The evaluation has shown promising results for a wide range of queries and query patterns, showcasing that the BAYROB system is indeed capable of not only accurately learning and representing coherences in actions, action parameters and environment characteristics but also reliably reasoning over arbitrary matters of the domain.

The BAYROB framework will be publicly available as an open source implementation accompanied by a browser-based web application allowing the user to investigate BAYROB's reasoning process.

The approach of integrating probabilistic hybrid models into a framework, as exemplified by BAYROB, where learning occurs through experiential data, holds the potential to fundamentally improve the decision-making processes of autonomous agents. A pivotal aspect of this transformation lies in the integration of full joint probability distributions, encompassing both aspects of the world and the agent itself. This integration is key in facilitating informed decision-making grounded in experience. By incorporating JPTs as efficient models that proficiently learn, represent, and reason over various facets of the agent and its environment, it becomes feasible to equip autonomous robots with cognitive abilities. This empowerment enables them to accurately anticipate the outcomes of actions based on the context, providing the agent with the essential tools to make well-informed decisions when selecting the optimal actions and parameters for its task.

I believe that the presented approach holds significant promise in equipping robots with the necessary cognitive capabilities to autonomously competently solve tasks in real-world environments. To the best of my awareness, this approach is the first that incorporates joint hybrid probability distributions of arbitrary shape learnt from comprehensive experiential robot data. This integration leverages the complete spectrum of inferential power for more effective decision-making.

In its current state, the system is limited to the action models presented in this work, primarily learnt from artificially generated data. While the system has already shown that it can handle real robot data and reason about error types that allow to detect classes of errors in certain situations, future prospects include the further automatization of the learning process. This implies the automated identification of object and action designators from experiential data and the creation of the respective probabilistic hybrid action models. Furthermore, BAYROB currently does not emply a general procedure to solve the problem of spurious correlations, and, to the best of my knowledge, no reliable method exists so far. Future developments will therefore rely on action models, that necessitate manual engineering to a certain extent. ◁ *outlook*

I am excited to see how the system will perform on more mature datasets that provide the necessary information allowing the refinement of more sophisticated action plans - and the successful validation through execution on a real robot.

Bibliography

Agrawal, Ankit, and Alok Choudhary. 2016. “Perspective: materials informatics and big data: realization of the ``fourth paradigm'' of science in materials science”. *APL Materials* 4 (5): 53208–9. https://doi.org/https://doi.org/10.1063/1.4946894

Aldridge, Sarah, Stephen Watt, Michael A Quail, Tim Rayner, Margus Lukk, Michael F Bimson, Daniel Gaffney, and Duncan T Odom. 2013. “AHT-Chip-Seq: A Completely Automated Robotic Protocol for High-Throughput Chromatin Immunoprecipitation”. *Genome Biology* 14: 1–12

Ashmore, Rob, Radu Calinescu, and Colin Paterson. 2021. “Assuring the Machine Learning Lifecycle: Desiderata, Methods, And Challenges”. *ACM Computing Surveys (CSUR)* 54 (5): 1–39

Baader, Franz, and Werner Nutt. 2003. “Basic Description Logics.”. In *Description Logic Handbook*, 43–95

Beetz, Michael. 2002. “Plan Representation for Robotic Agents.”. In *AIPS*, 223–32

Beetz, Michael, and Henrik Grosskreutz. 2005. “Probabilistic Hybrid Action Models for Predicting Concurrent Percept-driven Robot Behavior”. *Journal of Artificial Intelligence Research* 24: 799–849

Beetz, Michael, and Daniel Nyga. 2023. “Knowledge Representation and Reasoning”. In *Robotics Goes MOOC: Knowledge*. Springer International Publishing

Beetz, Michael, Daniel Beßler, Andrei Haidu, Mihai Pomarlan, Asil Kaan Bozcuoğlu, and Georg Bartels. 2018. “Know Rob 2.0 — a 2nd Generation Knowledge Processing Framework for Cognition-Enabled Robotic Agents”. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 0:512–19. https://doi.org/10.1109/ICRA.2018.8460964

Besag, Julian. 1975. “Statistical Analysis of Non-Lattice Data”. *Journal of the Royal Statistical Society: Series D (The Statistician)* 24 (3): 179–95

Beßler, Daniel, Sebastian Koralewski, and Michael Beetz. 2018. “Knowledge Representation for Cognition-and Learning-Enabled Robot Manipulation.”. In *Cogrob@ KR*, 11–19

Bhattacharyya, A. 1946. “On a Measure of Divergence between Two Multinomial Populations”. *Sankhyā: The Indian Journal of Statistics (1933-1960)* 7 (4): 401–6. http://www.jstor.org/stable/25047882

Bishop, Christopher M. 1994. “Mixture Density Networks”

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Vol. 4. Springer

Bishop, Christopher M, and Markus Svensén. 2012. “Bayesian Hierarchical Mixtures of Experts”. *Arxiv Preprint Arxiv:1212.2447*

Blythe, Jim. 2001. “An Overview of Planning under Uncertainty”. *Artificial Intelligence Today: Recent Trends and Developments*, 85–110

Bollacker, Kurt, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. “Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge”. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, 1247–50. SIGMOD '08. Vancouver, Canada: Association for Computing Machinery. https://doi.org/10.1145/1376616.1376746

Brachman, R.J., and H.J. Levesque. 1985. *Readings in Knowledge Representation*. Morgan Kaufmann Readings Series. M. Kaufmann Publishers. https://books.google.de/books?id=Jn1QAAAAMAAJ

Breiman, Leo, Jerome H. Friedman, Richard Olshen, and Charles Stone. 1984. “Cart”. *Classification and Regression Trees*

Calderon, Camilo E., Jose J. Plata, Cormac Toher, Corey Oses, Ohad Levy, Marco Fornari, Amir Natan, et al. 2015. “The AFLOW standard for high-throughput materials science calculations”. *Computational Materials Science* 108: 233–38. https://doi.org/https://doi.org/10.1016/j.commatsci.2015.07.019

Casher, Omer, and Henry S Rzepa. 2006. “SemanticEye: a semantic web application to rationalize and enhance chemical electronic publishing”. *Journal of Chemical Information and Modeling* 46 (6): 2396–2411

Chan, Lupe SH, Amanda MY Chu, and Mike KP So. 2023. “A Moving-Window Bayesian Network Model for Assessing Systemic Risk in Financial Markets”. *Plos One* 18 (1): e279888

Chater, Nick, Joshua B. Tenenbaum, and Alan Yuille. 2006. “Probabilistic Models of Cognition: Conceptual Foundations”. *Trends in Cognitive Sciences* 10 (7): 287–91

Choi, Y, Antonio Vergari, and Guy Van den Broeck. 2020. “Probabilistic Circuits: A Unifying Framework for Tractable Probabilistic Models”. *UCLA. URL: Http://starai. Cs. Ucla. Edu/papers/probcirc20. Pdf*

Ciftci, N., N. Ellendt, R. von Bargen, H. Henein, L. Mädler, and V. Uhlenwinkel. 2014. “Atomization and characterization of a glass forming alloy (Fe0.6-Co0.4)0.75B0.2Si0.0596Nb4”. *Journal of Non-Crystalline Solids*, 36–42. https://doi.org/https://doi.org/10.1016/j.jnoncrysol.2014.03.023

Curtarolo, Stefano, Wahyu Setyawan, Shidong Wang, Junkai Xue, Kesong Yang, Richard H. Taylor, Lance J. Nelson, et al. 2012. “AFLOWLIB.ORG: A distributed materials properties repository from high-throughput ab initio calculations”. *Computational Materials Science* 58: 227–35. https://doi.org/https://doi.org/10.1016/j.commatsci.2012.02.002

Dang, Meihua, Antonio Vergari, and Guy Broeck. 2020. “Strudel: Learning Structured-Decomposable Probabilistic Circuits”. In *International Conference on Probabilistic Graphical Models*, 137–48

Deza, Michel-Marie, and Elena Deza. 2006. *Dictionary of Distances*. Elsevier

Dong, Xin, Evgeniy Gabrilovich, Geremy Heitz, Wilko Horn, Ni Lao, Kevin Murphy, Thomas Strohmann, Shaohua Sun, and Wei Zhang. 2014. “Knowledge Vault: A Web-Scale Approach to Probabilistic Knowledge Fusion”. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 601–10. KDD '14. New York, New York, USA: Association for Computing Machinery. https://doi.org/10.1145/2623330.2623623

Dua, Dheeru, and Casey Graff. 2017. “UCI Machine Learning Repository”. 2017. http://archive.ics.uci.edu/ml

Ellendt, N, and L Mädler. 2018. “High-Throughput Exploration of Evolutionary Structural Materials”. *HTM Journal of Heat Treatment and Materials* 73 (1): 3–12. https://doi.org/10.3139/105.110345

Ellendt, N, N Ciftci, C Goodreau, V Uhlenwinkel, and L Madler. 2016. “Solidification of single droplets under combined cooling conditions”. *IOP Conference Series: Materials Science and Engineering* 117 (1): 12057–58. http://stacks.iop.org/1757-899X/117/i=1/a=012057

Ellendt, N, V Uhlenwinkel, and L Mädler. 2014. “High yield spray forming of small diameter tubes using pressure-gas-atomization”. *Materialwissenschaft Und Werkstofftechnik* 45 (8): 699–707. https://doi.org/10.1002/mawe.201400306

Enachescu, Vince, Paul Schrater, Stefan Schaal, and Vassilios Christopoulos. 2021. “Action Planning and Control under Uncertainty Emerge Through a Desirability-Driven Competition between Parallel Encoding Motor Plans”. *Plos Computational Biology* 17 (10): e1009429

Engel, Uwe. 2020. “Blick in Die Zukunft Wie Künstliche Intelligenz Das Leben Verändern Wird Ergebnisse Eines Umfrageprojekts in Der Wissenschaft, Politik Und Bevölkerung Der Freien Hansestadt Bremen”. https://doi.org/10.13140/RG.2.2.14478.92489

Fedrizzi, Andreas. 2011. “Action-Related Places for Mobile Manipulation”

Fikes, Richard E, and Nils J Nilsson. 1971. “STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving”. *Artificial Intelligence* 2 (3–4): 189–208

Finucan, H. M. 1964. “The Mode of a Multinomial Distribution”. *Biometrika* 51 (3/4): 513–17. http://www.jstor.org/stable/2334165

Friedman, Jerome H. 1991. “Multivariate Adaptive Regression Splines”. *Ann. Statist*

Friston, Karl. 2012. “The History of the Future of the Bayesian Brain”. *Neuroimage* 62 (2): 1230–33. https://doi.org/https://doi.org/10.1016/j.neuroimage.2011.10.004

Gaultois, Michael W, Anton O Oliynyk, Arthur Mar, Taylor D Sparks, Gregory J Mulholland, and Bryce Meredig. 2016. “Web-based machine learning models for real-time screening of thermoelectric materials properties”

Gaultois, Michael W, Taylor D Sparks, Christopher KH Borg, Ram Seshadri, William D Bonificio, and David R Clarke. 2013. “Data-driven review of thermoelectric materials: performance and resource considerations”. *Chemistry of Materials* 25 (15): 2911–20

Gauthaml, BP, R Kumarl, S Bothraz, G Mohapatral, N Kulkarnil, and KA Padmanabhan. 2011. “More Efficient ICME Through Materials Informatics and Process Modeling”. In *Proceedings of the 1st World Congress on Integrated Computational Materials Engineering (ICME)*, 35–36

Gens, Robert, and Pedro Domingos. 2012. “Discriminative Learning of Sum-Product Networks”. *Advances in Neural Information Processing Systems* 25: 3239–47

Gens, Robert, and Pedro Domingos. 2013. “Learning the Structure of Sum-Product Networks”. In *International Conference on Machine Learning*, 873–80

Giannozzi, Paolo, Stefano Baroni, Nicola Bonini, Matteo Calandra, Roberto Car, Carlo Cavazzoni, Davide Ceresoli, et al. 2009. “QUANTUM ESPRESSO: a modular and open-source software project for quantum simulations of materials”. *Journal of Physics: Condensed Matter* 21 (39): 395502–3. http://stacks.iop.org/0953-8984/21/i=39/a=395502

Gil, Yolanda, and Bart Selman. 2019. “A 20-Year Community Roadmap for Artificial Intelligence Research in the Us”

Goebel, Randy, Ajay Chander, Katharina Holzinger, Freddy Lecue, Zeynep Akata, Simone Stumpf, Peter Kieseberg, and Andreas Holzinger. 2018. “Explainable AI: The New 42?”. In *Machine Learning and Knowledge Extraction*, edited by Andreas Holzinger, Peter Kieseberg, A Min Tjoa, and Edgar Weippl, 295–303. Cham: Springer International Publishing

Griffiths, Thomas L., Charles Kemp, and Joshua B. Tenenbaum. 2023. “Bayesian Models of Cognition”. In *The Cambridge Handbook of Computational Cognitive Sciences*, edited by RonEditor Sun, 2nd ed., 80–138. Cambridge Handbooks in Psychology. Cambridge University Press. https://doi.org/10.1017/9781108755610.006

Ha, Jung-Su, Danny Driess, and Marc Toussaint. 2020. “A Probabilistic Framework for Constrained Manipulations and Task and Motion Planning under Uncertainty”. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, 6745–51

Hellinger, Ernst. 1909. “Neue Begründung Der Theorie Quadratischer Formen Von Unendlichvielen Veränderlichen.”. *Journal Für Die Reine Und Angewandte Mathematik* 1909 (136): 210–71

Hidalgo-Carvajal, Diego, Hanzhi Chen, Gemma C. Bettelani, Jaesug Jung, Melissa Zavaglia, Laura Busse, Abdeldjallil Naceri, Stefan Leutenegger, and Sami Haddadin. 2023. “Anthropomorphic Grasping with Neural Object Shape Completion”. *IEEE Robotics and Automation Letters* 8 (12): 8034–41. https://doi.org/10.1109/LRA.2023.3322086

Iberraken, Dimia, and Lounis Adouane. 2022. “An Evasive Strategy for Safe Autonomous Navigation Using Bayesian Networks and CMA-Es”. In , 589–604. https://doi.org/10.1007/978-3-030-97672-9_53

Indelman, Vadim, Luca Carlone, and Frank Dellaert. 2014. “Planning under Uncertainty in the Continuous Domain: A Generalized Belief Space Approach”. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 6763–70

Jain, Ajinkya, and Scott Niekum. 2018. “Efficient Hierarchical Robot Motion Planning under Uncertainty and Hybrid Dynamics”. In *Conference on Robot Learning*, 757–66

Jain, Anubhav, Shyue Ping Ong, Geoffroy Hautier, Wei Chen, William Davidson Richards, Stephen Dacek, Shreyas Cholia, et al. 2013. “Commentary: The Materials Project: A materials genome approach to accelerating materials innovation”. *Apl Materials* 1 (1): 11002–3

Jain, Anubhav, Kristin A Persson, and Gerbrand Ceder. 2016. “Research Update: The materials genome initiative: Data sharing and the impact of collaborative ab initio databases”. *APL Materials* 4 (5): 53102–3

Jain, Dominik. 2012. “Probabilistic Cognition for Technical Systems: Statistical Relational Models for High-Level Knowledge Representation, Learning and Reasoning”. http://mediatum.ub.tum.de/node?id=1096684&change_language=en

Janowicz, Krzysztof, Frank Van Harmelen, James A Hendler, and Pascal Hitzler. 2014. “Why the data train needs semantic rails”. *AI Magazine*

Jeannerod, Marc. 2001. “Neural Simulation of Action: A Unifying Mechanism for Motor Cognition”. *Neuroimage* 14 (1): S103–S109. https://doi.org/https://doi.org/10.1006/nimg.2001.0832

Jost, Lou. 2006. “Entropy and Diversity”. *Oikos* 113 (2): 363–75

Kaelbling, Leslie Pack, and Tomás Lozano-Pérez. 2013. “Integrated Task and Motion Planning in Belief Space”. *The International Journal of Robotics Research* 32 (9–10): 1194–1227. https://doi.org/10.1177/0278364913484072

Kaelbling, Leslie Pack, Michael L. Littman, and Anthony R. Cassandra. 1998. “Planning and Acting in Partially Observable Stochastic Domains”. *Artificial Intelligence* 101 (1): 99–134. https://doi.org/https://doi.org/10.1016/S0004-3702(98)00023-X

Kahneman, Daniel. 2017. *Thinking, Fast and Slow*

Karlsson, Lars. 2001. “Conditional Progressive Planning under Uncertainty”. In *IJCAI*, 431–38

Kasneci, Enkelejda, Kathrin Sessler, Stefan Küchemann, Maria Bannert, Daryna Dementieva, Frank Fischer, Urs Gasser, et al. 2023. “Chatgpt for Good? On Opportunities and Challenges of Large Language Models for Education”. *Learning and Individual Differences* 103: 102274–75. https://doi.org/https://doi.org/10.1016/j.lindif.2023.102274

Keelin, Thomas W., and Bradford W. Powley. 2011. “Quantile-Parameterized Distributions”. *Decision Analysis* 8 (3): 206–19. https://doi.org/10.1287/deca.1110.0213

Keeney, Stephen. 2011. “Use of Robotics in High-Throughput DNA Sequencing”. *PCR Mutation Detection Protocols*, 227–37

King, Michael R, and ChatGPT. 2023. “A Conversation on Artificial Intelligence, Chatbots, And Plagiarism in Higher Education”. *Cellular and Molecular Bioengineering* 16 (1): 1–2

Knill, David C., and Alexandre Pouget. 2004. “The Bayesian Brain: The Role of Uncertainty in Neural Coding and Computation”. *Trends in Neurosciences* 27 (12)

Koller, D., and N. Friedman. 2009. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive Computation and Machine Learning. MIT Press. https://books.google.de/books?id=7dzpHCHzNQ4C

Koubaa, Anis, Wadii Boulila, Lahouari Ghouti, Ayyub Alzahem, and Shahid Latif. 2023. “Exploring Chatgpt Capabilities and Limitations: A Critical Review of the NLP Game Changer”. *Preprints*, March. https://doi.org/10.20944/preprints202303.0438.v1

Kullback, S. 1959. *Information Theory and Statistics*. Wiley Publication in Mathematical Statistics. Wiley. https://books.google.de/books?id=XeRQAAAAMAAJ

Laghaee, Aroosha, Chris Malcolm, John Hallam, and Peter Ghazal. 2005. “Artificial Intelligence and Robotics in High Throughput Post-Genomics”. *Drug Discovery Today* 10 (18): 1253–59. https://doi.org/https://doi.org/10.1016/S1359-6446(05)03581-6

Laird, John E., Kevin Gluck, John Anderson, Kenneth D. Forbus, Odest Chadwicke Jenkins, Christian Lebiere, Dario Salvucci, et al. 2017. “Interactive Task Learning”. *IEEE Intelligent Systems* 32 (4): 6–21. https://doi.org/10.1109/MIS.2017.3121552

LeCun, Yann, and Corinna Cortes. 2010. “MNIST Handwritten Digit Database”. http://yann.lecun.com/exdb/mnist/

Loh, Wei-Yin. 2014. “Fifty Years of Classification and Regression Trees”. *International Statistical Review* 82 (3): 329–48

Marcus, Gary, and Ernest Davis. 2019. *Rebooting AI – Building Artificial Intelligence we can trust*. Pantheon

Mathew, Kiran, Joseph H. Montoya, Alireza Faghaninia, Shyam Dwarakanath, Muratahan Aykol, Hanmei Tang, Iek-heng Chu, et al. 2017. “Atomate: A high-level interface to generate, execute, and analyze computational materials science workflows”. *Computational Materials Science* 139: 140–52. https://doi.org/https://doi.org/10.1016/j.commatsci.2017.07.030

Mauro, Nicola Di, Gennaro Gala, Marco Iannotta, and Teresa M.A. Basile. 2021. “Random Probabilistic Circuits”. In *Proceedings of the Thirty-Seventh Conference on Uncertainty in Artificial Intelligence*, edited by Cassio de Campos and Marloes H. Maathuis, 161:1682–91. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v161/di-mauro21a.html

McCarthy, John. 1959. “Programs with Common Sense”. London

McDermott, Drew. 1992. “Robot Planning”. *AI Magazine* 13 (2): 55–56. https://doi.org/10.1609/aimag.v13i2.992

McLachlan, Scott, Kudakwashe Dube, Graham A Hitman, Norman E Fenton, and Evangelia Kyrimi. 2020. “Bayesian Networks in Healthcare: Distribution by Medical Condition”. *Artificial Intelligence in Medicine* 107: 101912–13. https://doi.org/https://doi.org/10.1016/j.artmed.2020.101912

Michael, Sam, Douglas Auld, Carleen Klumpp, Ajit Jadhav, Wei Zheng, Natasha Thorne, Christopher P Austin, James Inglese, and Anton Simeonov. 2008. “A Robotic Platform for Quantitative High-Throughput Screening”. *Assay and Drug Development Technologies* 6 (5): 637–57

Michalos, George, Panagiotis Karagiannis, Nikos Dimitropoulos, Dionisis Andronas, and Sotiris Makris. 2022. “Human Robot Collaboration in Industrial Environments”. In *The 21st Century Industrial Robot: When Tools Become Collaborators*, edited by Maria Isabel Aldinhas Ferreira and Sarah R. Fletcher, 17–39. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-78513-0_2

Mitash, Chaitanya, Fan Wang, Shiyang Lu, Vikedo Terhuja, Tyler Garaas, Felipe Polido, and Manikantan Nambi. 2023. “Armbench: An Object-Centric Benchmark Dataset for Robotic Manipulation”. In *ICRA 2023*. https://www.amazon.science/publications/armbench-an-object-centric-benchmark-dataset-for-robotic-manipulation

Mitchell, T., W. Cohen, E. Hruschka, P. Talukdar, B. Yang, J. Betteridge, A. Carlson, et al. 2018. “Never-Ending Learning”. *Commun. ACM* 61 (5): 103–15. https://doi.org/10.1145/3191513

Moravec, Hans. 1988. *Mind Children: The Future of Robot and Human Intelligence*. Harvard University Press

Murphy, Kevin P. 2022. *Probabilistic Machine Learning: An Introduction*. MIT Press

Musen, Mark A. 2015. “The ProtÉGÉ Project: A Look Back and a Look Forward”. *AI Matters* 1 (4): 4–12. https://doi.org/10.1145/2757001.2757003

Ng, Andrew, and Michael Jordan. 2001. “On Discriminative Vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes”. In *Advances in Neural Information Processing Systems*, edited by T. Dietterich, S. Becker, and Z. Ghahramani, 14:. MIT Press. https://proceedings.neurips.cc/paper_files/paper/2001/file/7b7a53e239400a13bd6be6c91c4f6c4e-Paper.pdf

Nyga, Daniel. 2017. “Interpretation of Natural-Language Robot Instructions: Probabilistic Knowledge Representation, Learning, And Reasoning”. https://media.suub.uni-bremen.de/handle/elib/1215

Nyga, Daniel, Subhro Roy, Rohan Paul, Daehyung Park, Mihai Pomarlan, Michael Beetz, and Nicholas Roy. 2018. “Grounding Robot Plans from Natural Language Instructions with Incomplete World Knowledge”. In *2nd Conference on Robot Learning (Corl 2018)*. http://proceedings.mlr.press/v87/nyga18a/nyga18a.pdf

Nørskov, Jens K., and Thomas Bligaard. 2013. “The Catalyst Genome”. *Angewandte Chemie International Edition* 52 (3): 776–77. https://doi.org/10.1002/anie.201208487

Ong, Shyue Ping, William Davidson Richards, Anubhav Jain, Geoffroy Hautier, Michael Kocher, Shreyas Cholia, Dan Gunter, Vincent L. Chevrier, Kristin A. Persson, and Gerbrand Ceder. 2013. “Python Materials Genomics (pymatgen): A robust, open-source python library for materials analysis”. *Computational Materials Science* 68: 314–19. https://doi.org/https://doi.org/10.1016/j.commatsci.2012.10.028

Park, James D, and Adnan Darwiche. 2004. “Complexity Results and Approximation Strategies for MAP Explanations”. *Journal of Artificial Intelligence Research* 21: 101–33

Pearson, Karl. 1900. “X. On the Criterion That a Given System of Deviations from the Probable in the Case of a Correlated System of Variables Is Such That It Can Be Reasonably Supposed to Have Arisen from Random Sampling”. *The London, Edinburgh, And Dublin Philosophical Magazine and Journal of Science* 50 (302): 157–75. https://doi.org/10.1080/14786440009463897

Peharz, Robert, Steven Lang, Antonio Vergari, Karl Stelzner, Alejandro Molina, Martin Trapp, Guy Van Den Broeck, Kristian Kersting, and Zoubin Ghahramani. 2020. “Einsum Networks: Fast and Scalable Learning of Tractable Probabilistic Circuits”. In *Proceedings of the 37th International Conference on Machine Learning*, edited by Hal Daumé III and Aarti Singh, 119:7563–74. Proceedings of Machine Learning Research. PMLR. https://proceedings.mlr.press/v119/peharz20a.html

Picklum, Mareike, and Michael Beetz. 2019. “MatCALO: Knowledge-enabled machine learning in materials science”. *Computational Materials Science* 163: 50–62. https://doi.org/https://doi.org/10.1016/j.commatsci.2019.03.005

Picklum, Mareike, Daniel Nyga, Tom Schierenbeck, and Michael Beetz. 2023. “Joint Probability Trees”. 2023. https://arxiv.org/abs/2302.07167

Pizzi, Giovanni. 2018. “Open-Science Platform for Computational Materials Science: AiiDA and the Materials Cloud”. In *Handbook of Materials Modeling : Methods: Theory and Modeling*, edited by Wanda Andreoni and Sidney Yip, 1–24. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-42913-7_64-1

Plebani, Mario. 2023. *Clinical Chemistry and Laboratory Medicine (CCLM)* 61 (7): 1131–32. https://doi.org/doi:10.1515/cclm-2023-0387

Poon, Hoifung, and Pedro Domingos. 2011. “Sum-Product Networks: A New Deep Architecture”. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 689–90

Pujara, Jay, Hui Miao, Lise Getoor, and William Cohen. 2013. “Knowledge Graph Identification”. In *The Semantic Web – ISWC 2013*, edited by Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Josiane Xavier Parreira, Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, 542–57. Berlin, Heidelberg: Springer Berlin Heidelberg

Quinlan, J Ross. 2014. *C4. 5: Programs for Machine Learning*. Elsevier

Quinlan, J. Ross. 1986. “Induction of Decision Trees”. *Machine Learning* 1 (1): 81–106

Quinlan, Ross. 1993. “C4. 5”. *Programs for Machine Learning*

Rahman, Tahrima, Prasanna Kothalkar, and Vibhav Gogate. 2014. “Cutset Networks: A Simple, Tractable, And Scalable Approach for Improving the Accuracy of Chow-Liu Trees”. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 630–45

Rajan, Krishna. 2012. “Materials informatics”. *Materials Today* 15 (11): 470–71

Rajan, Krishna. 2015. “Materials informatics: The materials “gene” and big data”. *Annual Review of Materials Research* 45: 153–69

Richardson, Elad, Yuval Alaluf, Or Patashnik, Yotam Nitzan, Yaniv Azar, Stav Shapiro, and Daniel Cohen-Or. 2021. “Encoding in Style: A Stylegan Encoder for Image-to-Image Translation”. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2287–96

Rodgers, John R., and David Cebon. 2006. “Materials Informatics”. *MRS Bulletin* 31 (12): 975–80. https://doi.org/10.1557/mrs2006.223

Rooshenas, Amirmohammad, and Daniel Lowd. 2014. “Learning Sum-Product Networks with Direct and Indirect Variable Interactions”. In *International Conference on Machine Learning*, 710–18

Russell, Stuart J, and Peter Norvig. 2010. *Artificial Intelligence – a Modern Approach (3. Internat. Ed.)*. Pearson Education, Inc.

Schickinger, Thomas, and Angelika Steger. 2002. *Diskrete Strukturen 2: Wahrscheinlichkeitstheorie Und Statistik*. Springer-Verlag

Schneider, Gisbert. 2018. “Automating Drug Discovery”. *Nature Reviews Drug Discovery* 17 (2): 97–113

Seshadri, Ram, and Taylor D Sparks. 2016. “Perspective: Interactive material property databases through aggregation of literature data”. *APL Materials* 4 (5): 53206–7

Shah, Nimish, Laura Isabel Galindez Olascoaga, Shirui Zhao, Wannes Meert, and Marian Verhelst. 2021. “PIU: A 248gops/w Stream-Based Processor for Irregular Probabilistic Inference Networks Using Precision-Scalable Posit Arithmetic in 28nm”. In *2021 IEEE International Solid- State Circuits Conference (ISSCC)*, 64:150–52. https://doi.org/10.1109/ISSCC42613.2021.9366061

Shannon, C. E. 1948. “A Mathematical Theory of Communication”. *The Bell System Technical Journal* 27 (3): 379–423. https://doi.org/10.1002/j.1538-7305.1948.tb01338.x

Shapiro, Stuart. 2009. “Knowledge Representation and Reasoning Logics for Artificial Intelligence”

Sijs, Joris, and James Fletcher. 2021. “A Knowledge Base for Robots to Model the Real-World as a Hypergraph”. In *2021 Fifth IEEE International Conference on Robotic Computing (IRC)*, 0:119–20. https://doi.org/10.1109/IRC52146.2021.00027

SPARC. 2014. “Strategic Research Agenda for Robotics in Europe 2014–2020”. *IEEE Robot. Autom. Mag* 24: 171–72. https://old.eu-robotics.net/sparc/upload/topic_groups/SRA2020_SPARC.pdf

SPARC. 2016. “Robotics 2020 Multi-Annual Roadmap for Robotics in Europe”. *SPARC Robotics, EU-Robotics AISBL, The Hague, The Netherlands* 5: 2018–19. https://old.eu-robotics.net/cms/upload/topic_groups/H2020_Robotics_Multi-Annual_Roadmap_ICT-2017B.pdf

Suchanek, Fabian M., Gjergji Kasneci, and Gerhard Weikum. 2007. “Yago: A Core of Semantic Knowledge”. In *Proceedings of the 16th International Conference on World Wide Web*, 697–706. WWW '07. Banff, Alberta, Canada: Association for Computing Machinery. https://doi.org/10.1145/1242572.1242667

Szpunar, Karl K, R Nathan Spreng, and Daniel L Schacter. 2014. “A Taxonomy of Prospection: Introducing an Organizational Framework for Future-Oriented Cognition”. *Proceedings of the National Academy of Sciences* 111 (52): 18414–21

Takahashi, Keisuke, and Yuzuru Tanaka. 2016. “Materials informatics: a journey towards material design and synthesis”. *Dalton Transactions* 45 (26): 10497–99

Takahashi, Keisuke, Lauren Takahashi, Itsuki Miyazato, Jun Fujima, Yuzuru Tanaka, Takeaki Uno, Hiroko Satoh, et al. 2019. “The Rise of Catalyst Informatics: Towards Catalyst Genomics”. *Chemcatchem* 0 (0). https://doi.org/10.1002/cctc.201801956

Takahashi, Lauren, Itsuki Miyazato, and Keisuke Takahashi. 2018. “Redesigning the Materials and Catalysts Database Construction Process Using Ontologies”. *Journal of Chemical Information and Modeling* 58 (9): 1742–54. https://doi.org/10.1021/acs.jcim.8b00165

Taylor, Kieron R, Robert J Gledhill, Jonathan W Essex, Jeremy G Frey, Stephen W Harris, and Dave C De Roure. 2006. “Bringing chemical data onto the semantic web”. *Journal of Chemical Information and Modeling* 46 (3): 939–52

Taylor, Richard H., Frisco Rose, Cormac Toher, Ohad Levy, Kesong Yang, Marco Buongiorno Nardelli, and Stefano Curtarolo. 2014. “A RESTful API for exchanging materials data in the AFLOWLIB.org consortium”. *Computational Materials Science* 93: 178–92. https://doi.org/https://doi.org/10.1016/j.commatsci.2014.05.014

Tegally, Houriiyah, James Emmanuel San, Jennifer Giandhari, and Tulio de Oliveira. 2020. “Unlocking the Efficiency of Genomics Laboratories with Robotic Liquid-Handling”. *BMC Genomics* 21 (1): 1–15

Tenenbaum, Joshua B., Charles Kemp, Thomas L. Griffiths, and Noah D. Goodman. 2011. “How to Grow a Mind: Statistics, Structure, And Abstraction”. *Science* 331 (6022): 1279–85. https://doi.org/10.1126/science.1192788

Tenorth, Moritz, and Michael Beetz. 2012. “A Unified Representation for Reasoning About Robot Actions, Processes, And Their Effects on Objects”. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 1351–58

Tenorth, Moritz, and Michael Beetz. 2013. “Knowrob: A Knowledge Processing Infrastructure for Cognition-Enabled Robots”. *The International Journal of Robotics Research* 32 (5): 566–90. https://doi.org/10.1177/0278364913481635

Thosar, Madhura, Sebastian Zug, Alpha Mary Skaria, and Akshay Jain. 2018. “A Review of Knowledge Bases for Service Robots in Household Environments.”. In *AIC*, 98–110

Vanschoren, Joaquin, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. 2013. “Openml: Networked Science in Machine Learning”. *SIGKDD Explorations* 15 (2): 49–60. https://doi.org/10.1145/2641190.2641198

Vergari, Antonio, YooJung Choi, Anji Liu, Stefano Teso, and Guy Van den Broeck. 2021. “A Compositional Atlas of Tractable Circuit Operations for Probabilistic Inference”. *Advances in Neural Information Processing Systems* 34: 13189–201

Vernon, David. 2014. *Artificial Cognitive Systems: A Primer*. MIT Press

Vernon, David, Michael Beetz, and Giulio Sandini. 2015. “Prospection in Cognition: The Case for Joint Episodic-Procedural Memory in Cognitive Robotics”. *Frontiers in Robotics and AI* 2: 19–20

Vidal-Naquet, Michel, and Shimon Ullman. 2003. “Object Recognition with Informative Features and Linear Classification.”. In *ICCV*, 3:281–82

Villani, Cédric. 2009. “The Wasserstein Distances”. In *Optimal Transport: Old and New*, 93–111. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-71050-9_6

Voigt, Florian, Lars Johannsmeier, and Sami Haddadin. 2020. “Multi-Level Structure Vs. End-to-End-Learning in High-Performance Tactile Robotic Manipulation.”. In *Corl*, 2306–16

Walsh, Aron. 2015. “Inorganic materials: The quest for new functionality”. *Nature Chemistry* 7 (4): 274–75

Wang, Bing, Paul A Dobosh, Stuart Chalk, Mirek Sopek, and Neil S Ostlund. 2016. “Computational Chemistry Data Management Platform Based on the Semantic Web”. *The Journal of Physical Chemistry a* 121 (1): 298–307

Williams, Daniel George. 2018. “The Mind as a Predictive Modelling Engine: Generative Models, Structural Similarity, And Mental Representation”

Yip, S. 2007. *Handbook of Materials Modeling*. Handbook of Materials Modeling. Springer Science & Business Media. https://books.google.de/books?id=0sJaJ5juL9EC

Yuksel, Seniha Esen, Joseph N Wilson, and Paul D Gader. 2012. “Twenty Years of Mixture of Experts”. *IEEE Transactions on Neural Networks and Learning Systems* 23 (8): 1177–93

Zhang, Lungan, Liangxiao Jiang, and Chaoqun Li. 2019. “A Discriminative Model Selection Approach and Its Application to Text Classification”. *Neural Computing and Applications* 31: 1173–87

Čech, Eduard, and M Katětov. 1969. “Chapter II: General Metric Spaces”. *Point Sets*, 42–91

Listing I

GLOSSARY

action core action cores are symbols representing (higher-level) actions that can be decomposed into smaller subactions. In the context of this work, examples for action cores could be *move forward* or *turn* · 8, 10

BAYROB Bayesian Robot Brain (BAYROB) is a fully operational system combining joint probability distributions to generate action models from a robot's experience data. The code is open-source and can be found at https://github.com/mareikep/bayrob-dev (Accessed: Sep 12th, 2023) · 8, 9, 15, 16, 17, 18, 33, 34, 37, 48, 71, 85, 87, 94, 95, 96, 99, 107, 108, 109, 111, 112, 115, 139, 141, 142, 152, 156, 157, 158, 159, 160, 166, 168, 172, 174, 175, 176, 177, 178, 179, 182, 183, 185, 186, 187

belief state a belief state in general can be viewed at as a set of statements about the robot itself as well as its environment (the *world*) at a certain point in time t that the robot believes to be true. In the context of BayRoB such a belief state is represented by means of a joint probability distribution over variables representing the agent's and its environment's current condition. · · 9, 10

CCC The CCC, affiliated with the CRA, is dedicated to advancing innovative and impactful computing research that addresses both national and global challenges. It operates as a responsive, visionary organization representing the computing research community, with a strong commitment to diversity, equity, and inclusivity. CCC serves as a catalyst, bringing together leaders from academia, industry, and government to articulate and promote compelling research visions. https://cra.org/ccc/ (Accessed: Sept 11th, 2023) · · · · · · · · · · · · · · · · · · 5

CRA The CRA is a prominent organization with member organizations across North America, including academic departments, industry labs, government agencies, and professional societies like AAAI, ACM, and IEEE Computer Society. Its mission is to promote and advance computing research by collaborating with industry, government, and academia, representing the computing research community, informing policymakers, and advocating for diversity and social responsibility in computing research. CRA plays a vital role in supporting and uniting the computing community, advocating for research funding, mentoring aspiring researchers, and fostering innovation in the field. https://cra.org/ (Accessed: Sept 11th, 2023) · · · · · 5

CRC CRCs are long-term university-based research institutions, established for up to 12 years, in which researchers work together within a multidisciplinary research programme. https://www.dfg.de/en/research_funding/programmes/coordinated_programmes/collaborative_research_centres/index.html (Accessed: Jan 20th, 2023) · · · · · · · · · · · · · 6, 139

descriptor We use the term descriptors to denote any kind of measurement or feature obtained from short-time characterizations on micro samples. They correlate with material properties so that a short-time characterization of a micro sample facilitates the prediction of material properties in macro- scale materials. · 122

EASE an interdisciplinary research center to develop cognitive robots with the skills to execute everyday activities. The focus lies on conducting *open research* and sharing knowledge and data. https://ease-crc.org/ (Accessed: Jan 13th, 2023) · · · 6, 8, 185

KnowRob Knowledge processing for robots. functions as a knowledge processing system merging methods for knowledge representation and reasoning with approaches for acquiring and anchoring knowledge within a physical system. It serves as a unified semantic framework capable of integrating information from various sources. https://knowrob.org/knowrob (Accessed: Mar 19th, 2024) · · · · · · · · · · · · · · · · · · · 183

MAR The MAR (for Robotics in Europe) complements the Strategic Research Agenda (SRA) by offering more in-depth technical and market insights. It serves as a detailed technical guide, highlighting anticipated progress and medium-term research and innovation objectives within the robotics community. The MAR is updated annually to align with evolving priorities, technologies, and strategic developments in European research, development, and innovation. https://old.eu-robotics.net/cms/upload/topic_groups/H2020_Robotics_

Multi-Annual_Roadmap_ICT-2017B.pdf (Accessed: Sept 5th, 2023) 4

MatCALO The MatCALO system is an implementation of an intelligent, assistive system that is capable of supporting materials scientists in their work by generating hypotheses on how to process certain materials to obtain desired properties. 17, 116, 117, 124, 125, 133, 134, 135, 136, 137, 139

MGI The Materials Genome Initiative is a collaborative effort involving multiple federal agencies aimed at expediting the discovery, production, and utilization of advanced materials, achieving results in half the time and at a significantly reduced expense compared to conventional approaches. This initiative establishes guidelines, allocates resources, and builds the necessary infrastructure to assist American institutions in embracing approaches that accelerate the development of materials. https://www.mgi.gov (Accessed: Sep 12th, 2023) · · · · · · · · 120

OWL OWL forms the standard for a Semantic Web language which can be used to represent complex knowledge about things and relations between things https://www.w3.org/OWL/ (Accessed: Aug 1st, 2022) · · · · · · · · · · · · · · · · · 133

sample A *sample* in the context of this work may either be *microscopic* or *macroscopic*. A macroscopic (*macro*) sample can be used to obtain material properties directly because of its size. Macro samples can for example be created by using spray-forming or casting. A microscopic or *micro* sample is a spherical droplet of a few hundred μm in diameter. These samples are either generated in large quantities in one go using the pneumatic high-temperature droplet generator as described by (Ellendt, Ciftci, Goodreau, Uhlenwinkel, and Madler 2016) or they are formed in a larger-scale sample using Laser Deep Alloying. · · 122

SPARC The *SPARC Partnership for Robotics in Europe* is a civilian-funded robotics innovation programme and a partnership between the European Commission and the European industry and academia. https://old.eu-robotics.net/sparc/ (Accessed: Sept 5th, 2023) · 4

SRA The SRA (for Robotics in Europe) offers a comprehensive strategic perspective for the field of robotics. Additionally, it serves as a means to acquaint non-experts, policymakers, entrepreneurs, and industries interested in the robotics sector with the European robotics community. This document aims to provide insight into the current state and future prospects of robotics. https://old.eu-robotics.net/sparc/upload/topic_groups/SRA2020_
SPARC.pdf (Accessed: Sept 5th, 2023) · · · · · · · · · 4

Listing II

Acronyms

AI Artificial Intelligence 1, 3, 4, 5, 6, 7, 12, 13, 14, 69, 180, 181, 185

API Application Programming Interface 120, 134, 177

ARP LACEs Action-Related Places 182

BNs Bayesian Network . 25, 37, 38, 39, 40, 41, 42, 43, 45, 47, 56, 158, 181

CART Classification and Regression Trees 53, 59, 60, 61, 65, 66, 67, 68

CDF Cumulative Distribution Function .26, 31, 36, 61, 62, 63, 64, 69, 130

CPD Conditional Probability Distribution . . . 38, 39, 40, 158

CPL CRAM Plan Language 11

CRAM Cognitive Robot Abstract Machine 99, 101, 174

CRPs Concurrent Reactive Plans 181

DAG Directed Acyclic Graph 38, 48

DT Decision Tree 52, 53

EiNets Einsum Networks 180

EM Expectation Maximization 46, 61

FG Factor Graph 44

ChatGPT Generative Pre-trained Transformer 13, 14

HME Hierarchical Mixture of Experts 181

ID3 Iterative Dichotomiser 3 52

JPTs Joint Probability Tree · 16, 17, 18, 55, 56, 57, 59, 60, 61, 64, 65, 66, 67, 68, 69, 71, 85, 87, 88, 90, 91, 103, 139, 151, 158, 162, 169, 179, 180, 181, 185, 187

KB Knowledge Base · 15, 19, 24

KR&R Knowledge Representation and Reasoning · 19, 20

MAE Mean Absolute Error · 66, 67, 68

MAP Maximum A Posteriori · 25, 48

MARS Multivariate Adaptive Regression Spline · 63

MB Markov Blanket · 43

MCMC Markov Chain Monte Carlo · 44

MDNs Mixture-Density Network · 181

ML Machine Learning · 7, 25, 65, 68, 69

MLE Maximum Likelihood Estimate · 50

MLNs Markov Logic Network · 7

MNs Markov Network · 25, 37, 42, 43, 44, 45

MPE Most Probable Explanation · 25, 69, 111

MRFs Markov Random Field · 42, 47, 56

MRT Multiple Reverse Tree · 129

MSE Mean Squared error · 52, 60, 63, 64

NEEMs Narrative-enabled Episodic Memory · 6, 7, 8, 143, 185, 13

NELL Never Ending Language Learning · 179

NLP Natural Language Processing · 4, 15

OLS Ordinary Least Squares · 66

PCs Probabilistic Circuits · 17, 46, 47, 48, 56, 180, 181

PDF Probability Density Function · 26, 30, 31, 34, 47, 61, 62, 64, 95, 96, 130

PEAMs Pragmatic Everyday Activity Manifold · 7

PGMs Probabilistic Graphical Model 17, 37, 45, 46, 47, 56, 61, 69, 180, 185

PHAMs Probabilistic Hybrid Action Models · 181, 182

PL Propositional Logic · 23

PLFs Piecewise Linear Function · 62, 63, 64, 103, 170, 172

PMF Probability Mass Function · 26, 28, 29, 32

POMDPs Partially Observable MDP · · · · · · · · · 180, 182

PPF Percent Point Function · · · · · · · · · · · · 64

QPD quantile-Parameterized Distribution · · · · · · · 62, 64

RT Regression Tree · · · · · · · · · · · · · 52, 53, 66

SPNs Sum-Product Network · · · · · · · 45, 46, 47, 180, 181

STRIPS Stanford Research Institute Problem Solver · · · · · 182

XPCs extremely randomized Probabilistic Circuits · · · · · 180

Listing III

Links

KUKA https://www.kuka.com/en-US (Accessed: Aug 29th 2023) · · 12

KUKA & BMW https://www.kuka.com/en-de/industries/solutions-database/2017/06/solution-systems-bmw-dingolfing (Accessed: Aug 29th 2023) · · 13

MGI https://www.mgi.gov/ (Accessed: Sep 12th, 2023) · · · · 120

MIT License https://opensource.org/license/mit/ (Accessed: Sep 12, 2023) · 134

MatCALO http://matcalo.open-ease.org/ (Accessed: Aug 28th, 2019) · 134

Materials Project https://materialsproject.org (Accessed: Aug 28th, 2019) · · 120

Microsoft Bing https://www.bing.com/?cc=en (Accessed: Aug 29th 2023) · · 1

NELL http://rtw.ml.cmu.edu/rtw/ (Accessed: Aug 31st 2023) · · 179

New York Times on ChatGPT https://www.nytimes.com/2022/12/05/technology/chatgpt-ai-twitter.html (Accessed: Dec 5th 2022) · · 13

OWL https://www.w3.org/OWL/ (Accessed: Sep 12th, 2023) · · 133

OpenAI https://openai.com/ (Accessed: Sep 12th 2023) · · · · · 13

Protégé https://protege.stanford.edu/ (Accessed: Sep 12th, 2023) · · 133

QUANTUM ESPRESSO https://www.quantum-espresso.org/ (Accessed: Sep 12th, 2023) 121

Socratic https://socratic.org/ (Accessed: Aug 29th 2023) · · · · · 1

The Guardian on ChatGPT https://www.theguardian.com/technology/2022/dec/05/what-is-ai-chatbot-phenomenon-chatgpt-and-could-it-replace-humans (Accessed: Dec 5th 2022) · · 13

UCSB Thermoelectrics Database http://www.mrl.ucsb.edu:8080/datamine/thermoelectric.jsp (Accessed: Aug 28th, 2019) · · 119

Utah Battery Database http://tomcat.eng.utah.edu/sparks/battery.jsp (Accessed: Aug 28th, 2019) · · 119

Wikimedia Commons https://commons.wikimedia.org/wiki/File:Pixel_XL_(2016).svg (Accessed: Mar 8th, 2024) · · 13

WordNet https://wordnet.princeton.edu/ (Accessed: Sep 12th, 2023) · 124

Wysa https://www.wysa.com/ (Accessed: Aug 29th 2023) · · · · 1

Youper https://www.youper.ai/ (Accessed: Aug 29th 2023) · · · · 1

pyRAP https://pyrap.org/ (Accessed: Aug 28th, 2019) · · · · · 134

pyjpt https://pypi.org/project/pyjpt/ (Accessed: Mar 19th 2024) · · 16

Listing IV

SYMBOLS

$\mathcal{A}$	*a set of symbols for arguments of process steps* · · · · · 125
$\mathcal{D}$	*a set of symbols for descriptors* – in the context of MATCALO · 125
$\mathcal{D}$	*a dataset* – in the context of JPTs · · · · · · · · 60, 64
$\mathcal{D}_m$	*a set of macro descriptors* · · · · · · · · · · · · 125
$\mathcal{D}_\mu$	*a set of μ descriptors* · · · · · · · · · · · · · 125
$\tilde{\mathcal{D}}$	*a subset of a dataset that serves as training data* – in the context of JPTs · · · · · · · · · · · · · · · · 62, 63, 64
$\mathcal{F}$	*a set of feature symbols* – the union of the sets of properties and descriptors, i.e. $\mathcal{P} \cup \mathcal{D}$ · · · · · · · · · · · · · 125
$\mathcal{I}$	*a continuous, real-valued interval* – in the context of MATCALO 125
$\mathcal{N}$	*the normal distribution $\mathcal{N}(\mu, \Xi)$* – determined by a mean vector μ and a covariance matrix Ξ · · · · · · · · · · 127
$\mathcal{P}$	*a set of symbols for properties* · · · · · · · · · · 125
$\mathcal{U}$	*the uniform distribution $\mathcal{U}(X)$* – over a random variable X · 60
$\mathcal{X}$	*the set of all possible worlds* – all possible complete variable assignments · · · · · · · · · · · · · · · · · · 57
λ	*a leaf of a tree T* · · · · · · · · · · · · · · · 58, 59
Λ	*the set of leaves of a tree T* · · · · · · · · · · · · · 58
ω	*an original state* – a material in its Urform · · · · 126, 127
Ω	*the set of all original states* · · · · · · · · · · · 126

φ *a feature profile* – an exhaustive assignment of features to ranges of admissible values · · · · · · · · · · 125, 126, 127

Φ *the set of all feature profiles* · · · · · · · · · · · 126

π *a process step* – a function transforming a material's state into a new colored state (controlled by the process' arguments $a \in \mathcal{A}$) · 126

ψ *a function mapping a descriptor to a property* · · · · · 127

ϱ *a requirement profile* – a feature profile which postulates effective bounds only for material properties, but allows descriptor variables unconstrained · · · · · · · · · · · · 126

P *the set of all requirement profiles* · · · · · · · · · 126

σ *a colored state* – an exhaustive assignment of features to real values · · · · · · · · · · · · · · · · · · · 126

Σ *the set of all colored states* · · · · · · · · · · · 126

ϑ *a confidence level* · · · · · · · · · · · · · · · 64

dom *domain* · · · · · · · · · · · · · · · · · · · 57

f *a Piecewise Linear Function (PLF)* – a function defined on a finite number of intervals, each of which has a linear function f_i attached. · · · · · · · · · · · · · · · · · · · 62

F *the Cumulative Distribution Function (CDF) of the desired probability distribution* · · · · · · · · · · · · · · 62, 64

H *entropy* – the entropy of a distribution over a symbolic random variable X is $H(P(X))$ · · · · · · · · · · · · · 60

I *an impurity improvement over a dataset D_i when split of data is performed on variable X_i: $I(\mathcal{D}, X_i)$* – in the context of JPTs ·60

P *probability* – a posterior $P(Q|E)$ or prior $P(Q)$ probability · 55

$P_{\mathcal{D},X_i}$ *distribution over X_j* – induced by the data set $\mathcal{D}$ when split at variable X_i: $P_{\mathcal{D},X_i}(X_j)$ · · · · · · · · · · · · · 60

T *a tree structure* · · · · · · · · · · · · · · · · 58, 61

X_{num} *the set of numeric variables in X* · · · · · · · · · 60

X_{sym} *the set of symbolic variables in X* · · · · · · · · · 60

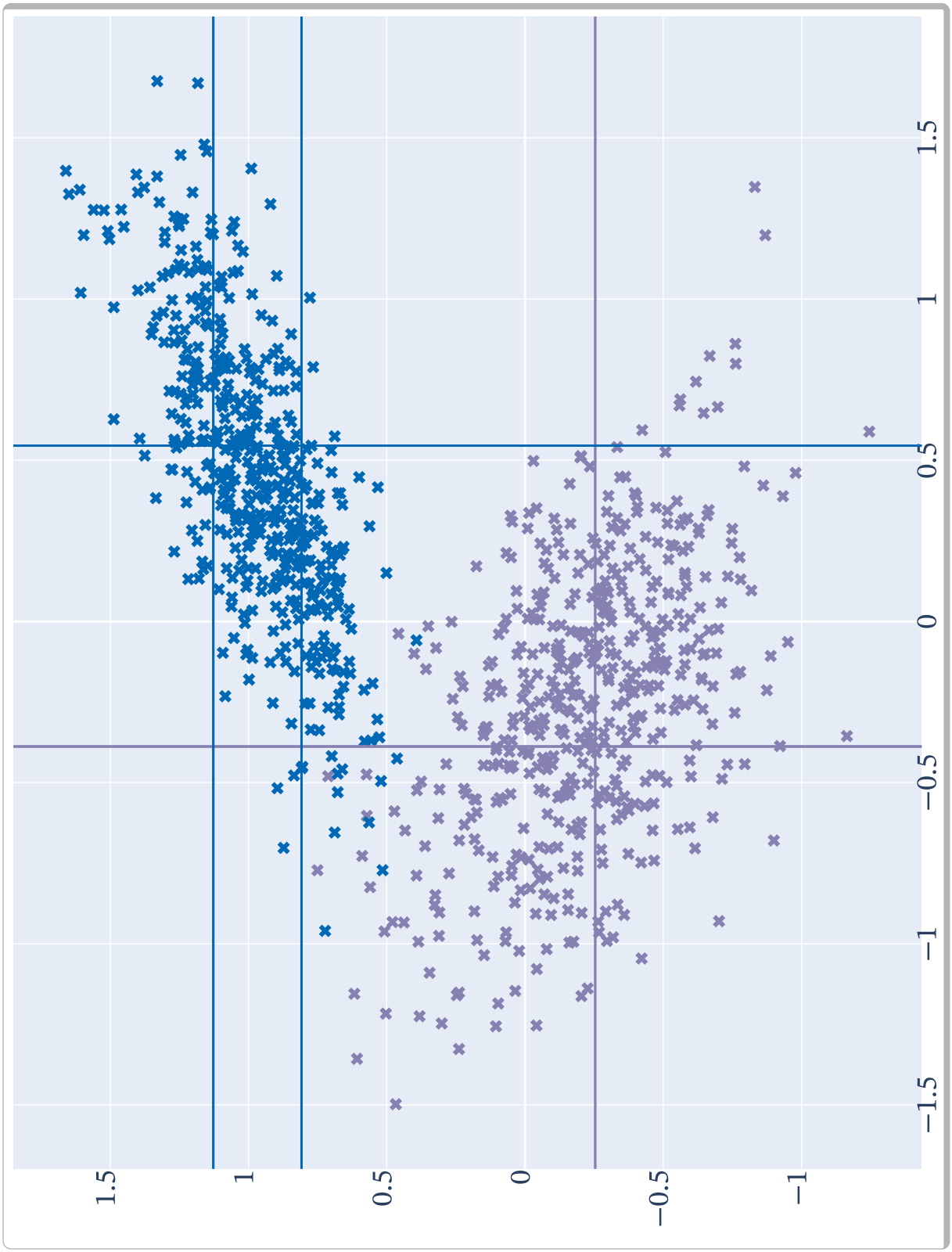
X *a vector of random variables X_i* · · · · · 57, 58, 59, 60, 64

ζ *a process chain* – a composition of process steps · · · · 126

JOINT PROBABILITY TREES

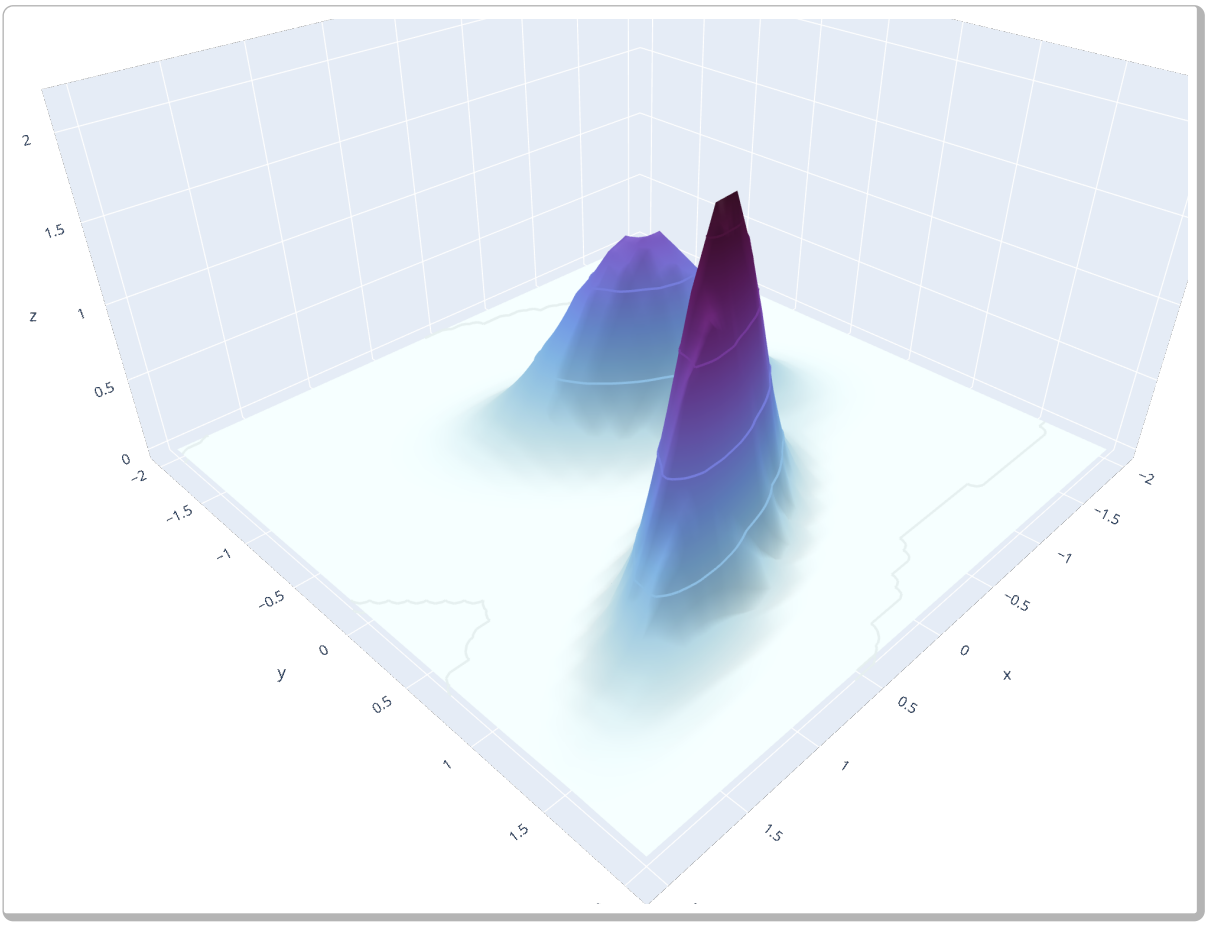
A.1 Trees

A.1.1 JPT Example - Tree

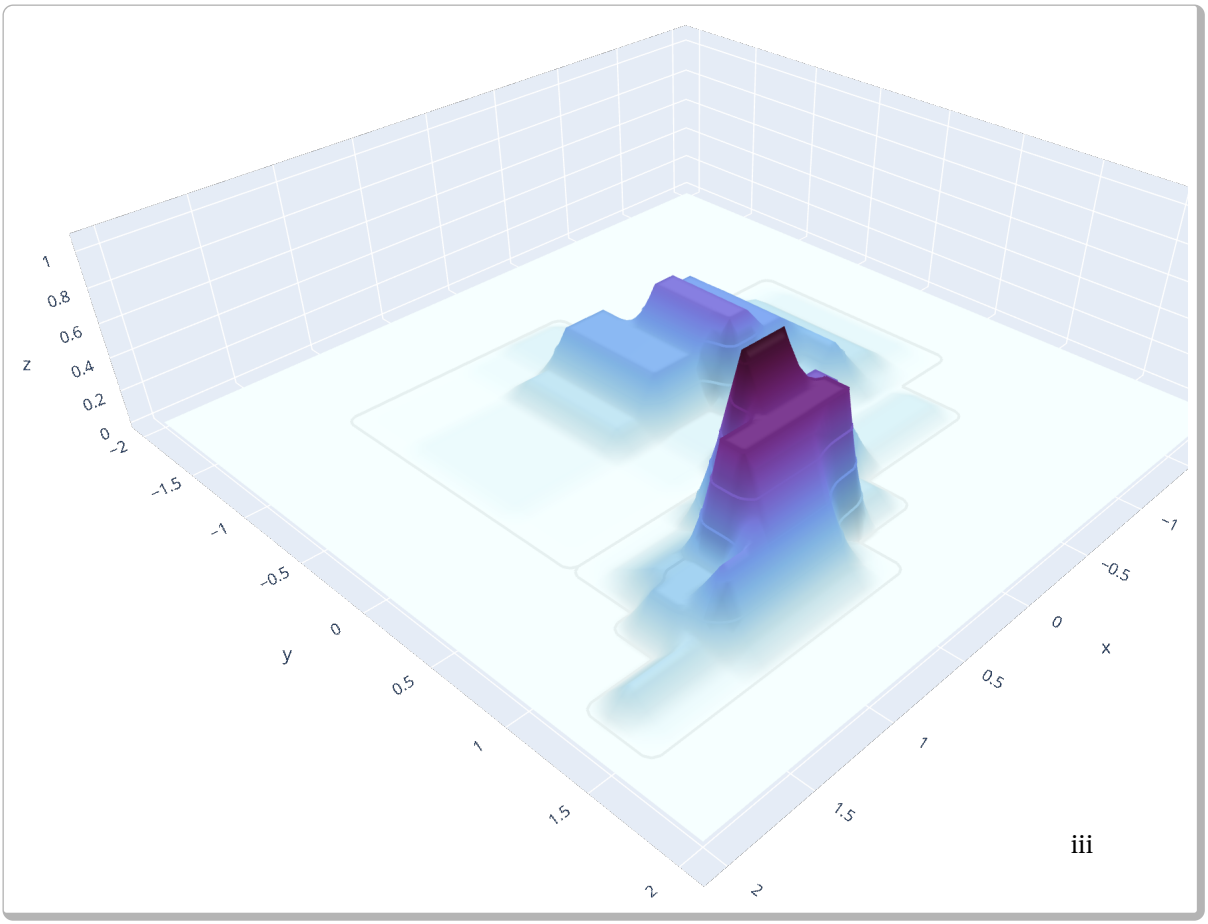


The scatterplot of the underlying toy data set in Section 3.2.3 | *Figure 84*

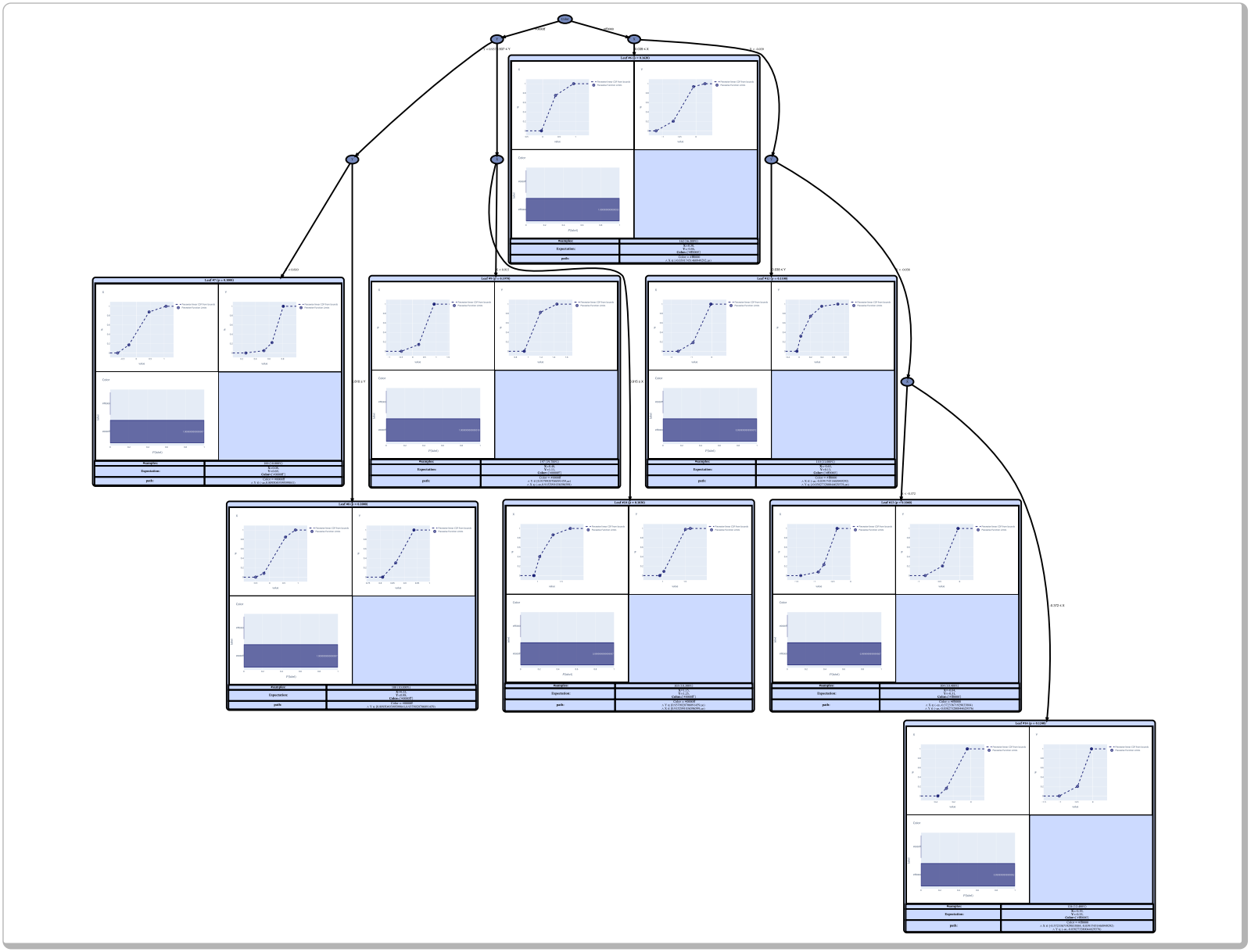
The ground truth distribution of the toy data set in Section 3.2.3. | *Figure 85*



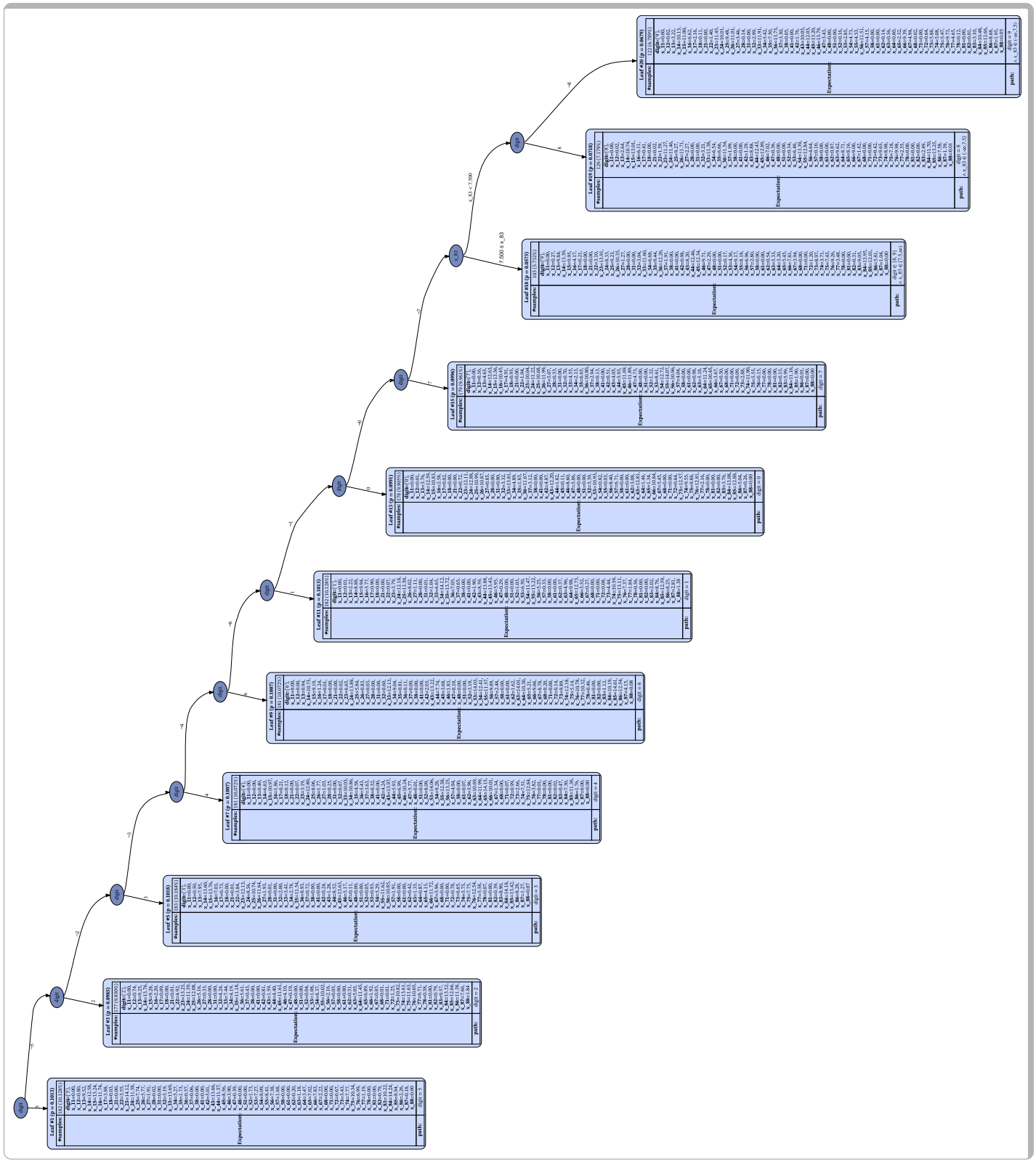
The plot of the marginal joint distribution $P(X, Y)$ of the toy data set in Section 3.2.3 | *Figure 86*



The JPT structure learnt using the toy data set in Section 3.2.3 | *Figure 87*

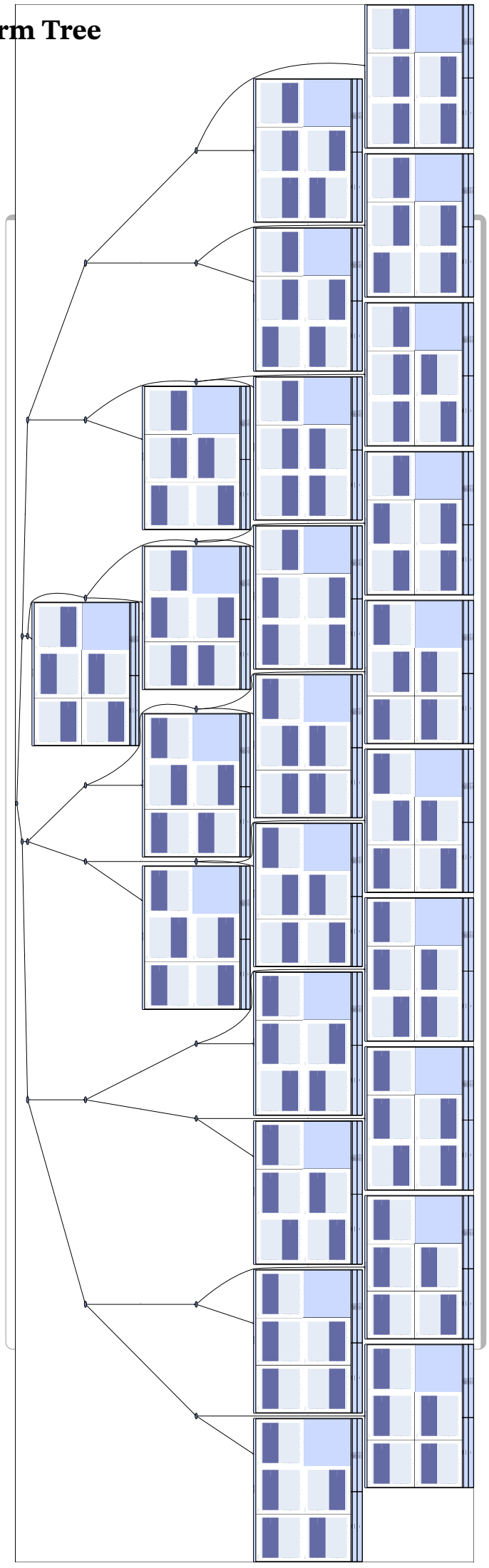


A.1.2 MNIST Tree



The JPT structure learnt using the MNIST data set | *Figure 88*

A.1.3 Alarm Tree



The JPT structure learnt using the alarm data set | *Figure 89*

A.2 Tables

A.2.1 Empirical Evaluation

Results of the evaluation of JPTs on eight benchmark data sets of the UCI machine learning repository (-Dua and Graff 2017) for different hyperparameters. *Min samples per leaf* means that at least the respective percentage of data points available for training must be represented by any leaf of the tree. *Min samples per leaf*=90% thus results in a JPT with only one leaf, ie a set of independent prior distributions over all variables considered. *0-likelihood test samples* determines the percentage of test samples with 0 likelihood. This may happen to samples lying outside the convex hull of the training data, where 0 probability mass is assigned by the CDF-LEARN algorithm. | *Table 15*

Dataset	Min samples per leaf	Model Size	Average Train Log-Likelihood	Average Test Log-Likelihood	0-likelihood Test Samples
IRIS Dataset	90%	23	−5.45	−5.63	-
Examples: 150	40%	46	−3.54	−3.33	-
Variables: 5	20%	94	−2.16	−2.66	20%
	10%	184	−1.37	−1.91	27%
	5%	353	−0.18	−1.20	67%
	1%	679	6.11	-	100%
Adult Dataset	90%	1472557	−55.46	−54.98	49%
Examples: 150	40%	2945118	−53.79	−53.07	58%
Variables: 5	20%	5890233	−52.96	−52.45	64%
	10%	11780477	−51.17	−50.18	70%
	5%	22088384	−50.13	−50.25	81%
	1%	94243769	−41.57	−42.96	93%
Dry Bean	90%	476858	−17.49	−16.98	95%
Dataset	40%	953709	−12.13	−13.23	95%
Examples: 150	20%	1430565	−9.8	−10.68	95%
Variables: 5	10%	3814841	−4.91	−6.75	95%
	5%	7629662	−2.31	−4.11	95%
	1%	36717612	2.41	0.85	96%
Wine Dataset	90%	1550	−18.97	−19.83	56%
Examples: 150	40%	3101	−17.06	−18.69	83%
Variables: 5	20%	6197	−13.42	−14.9	94%
	10%	12417	−10.98	−12.46	94%
	5%	24824	−6.74	-	100%
	1%	207028	41.22	-	100%
Wine Quality	90%	67	−9.50	−9.80	-
Dataset	40%	138	−8.10	−8.34	1%
Examples: 150	20%	204	−7.60	−7.68	1%
Variables: 5	10%	479	−6.48	−6.57	2%
	5%	1072	−5.67	−5.85	4%
	1%	5307	−3.08	−3.82	12%

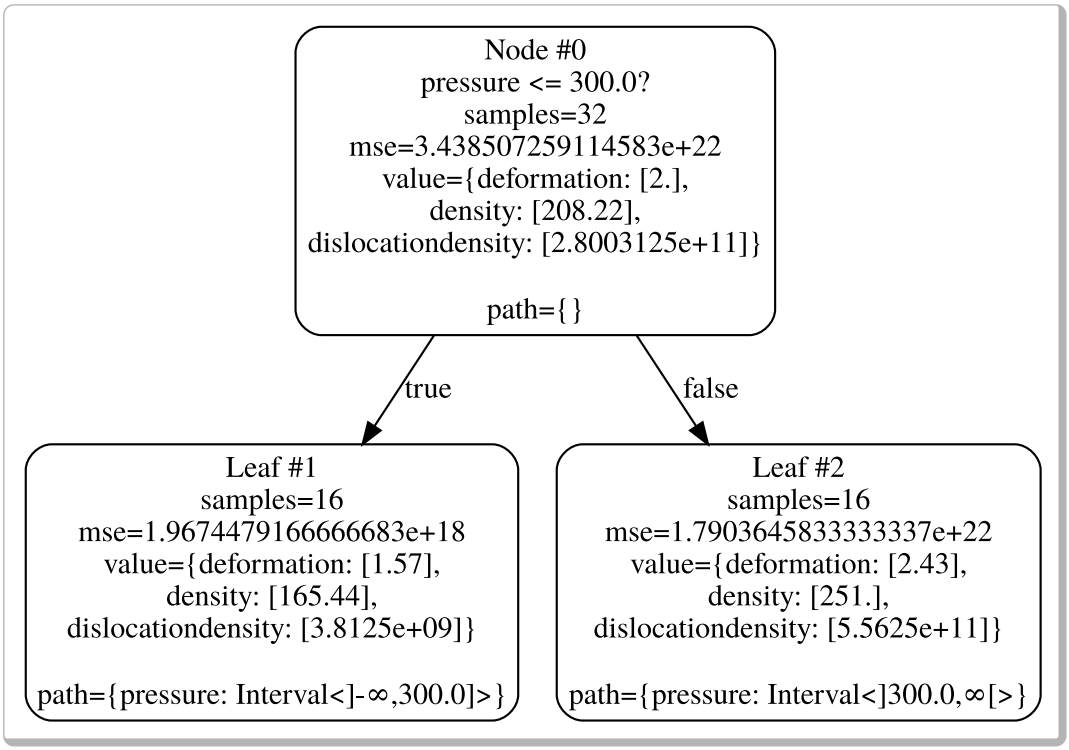
Bank and Marketing Dataset Examples: 150 Variables: 5	90%	116433	−34.60	−34.64	7%
	40%	232866	−34.50	−34.54	8%
	20%	465732	−34.31	−34.32	12%
	10%	815031	−33.00	−32.77	20%
	5%	1630062	−32.43	−32.08	32%
	1%	7335279	−30.46	−29.45	73%
Car Evaluation Dataset Examples: 150 Variables: 5	90%	21	−6.89	−7.03	-
	40%	21	−6.89	−7.03	-
	20%	63	−6.52	−6.62	-
	10%	147	−6.48	−6.61	-
	5%	231	−6.45	−6.59	-
	1%	1249	−6.38	−6.66	-
Abalone Dataset Examples: 150 Variables: 5	90%	64	0.09	−0.04	-
	40%	134	3.65	3.66	-
	20%	208	5.34	5.11	-
	10%	482	8.14	8.05	2%
	5%	1034	9.32	9.28	3%
	1%	5502	11.34	10.74	18%

Appendix B

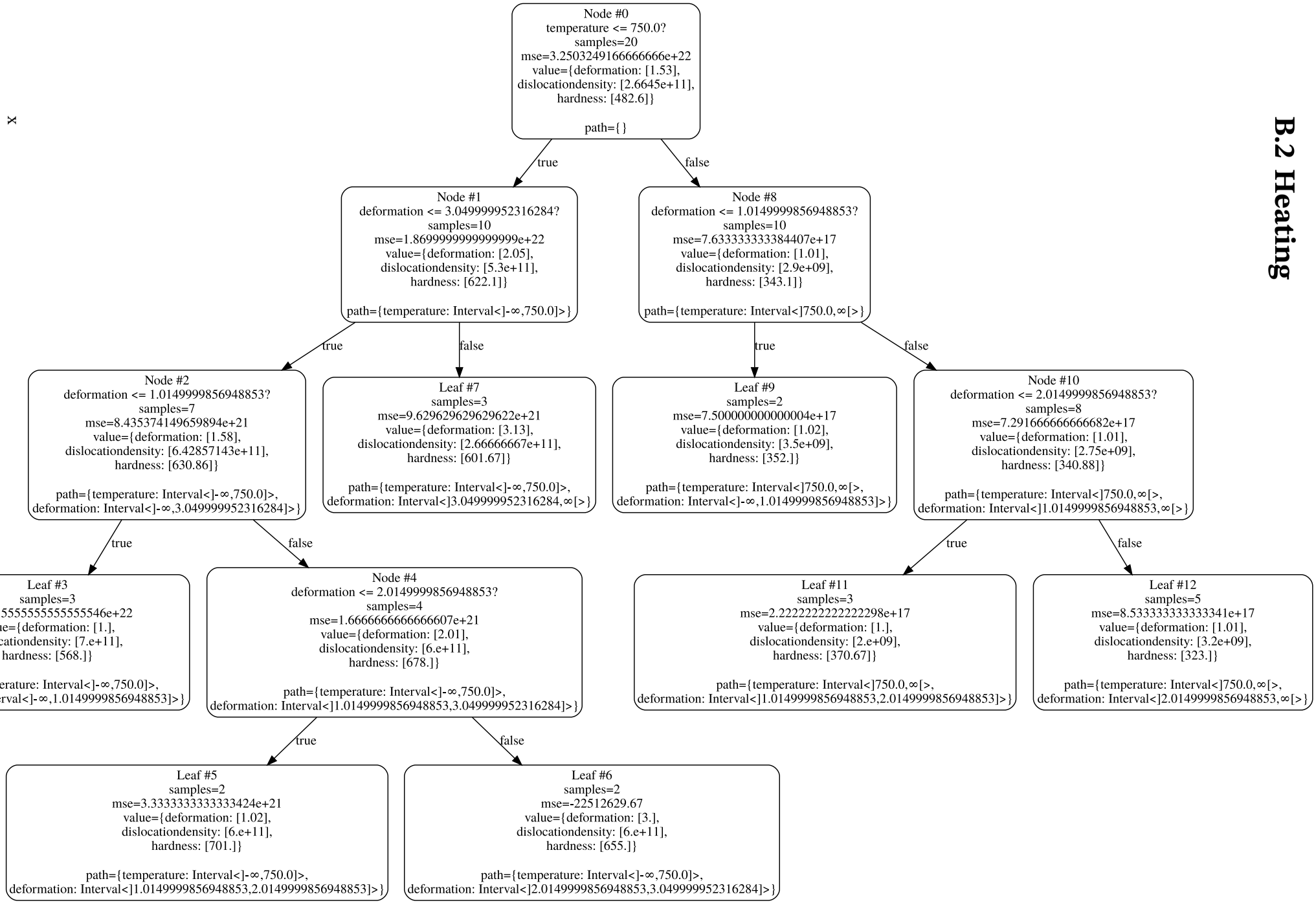
MatCALO

B.1 Deeprolling

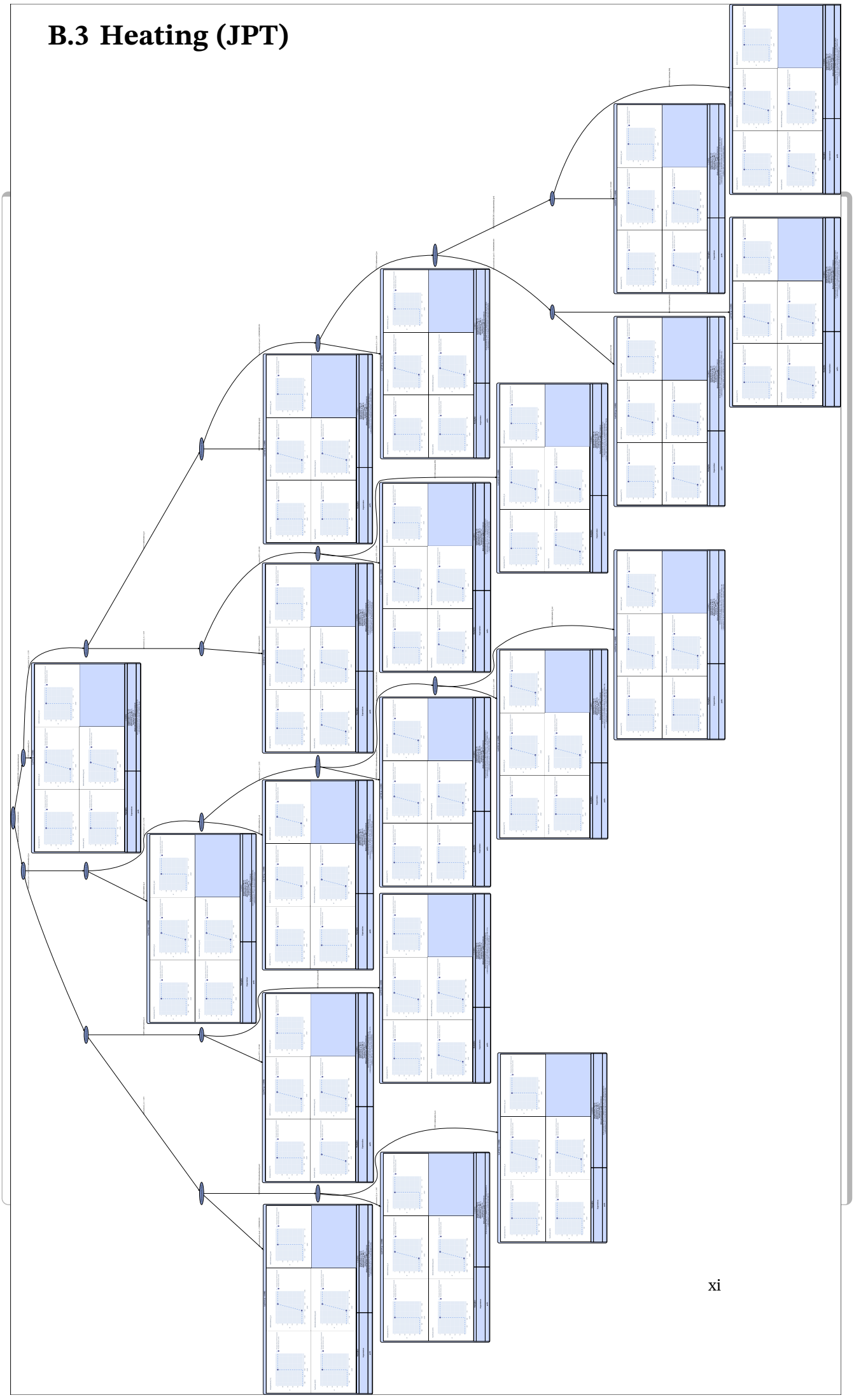
The generated tree for the deep rolling process. | *Figure 90*



The generated tree for the heating process. | *Figure 91*

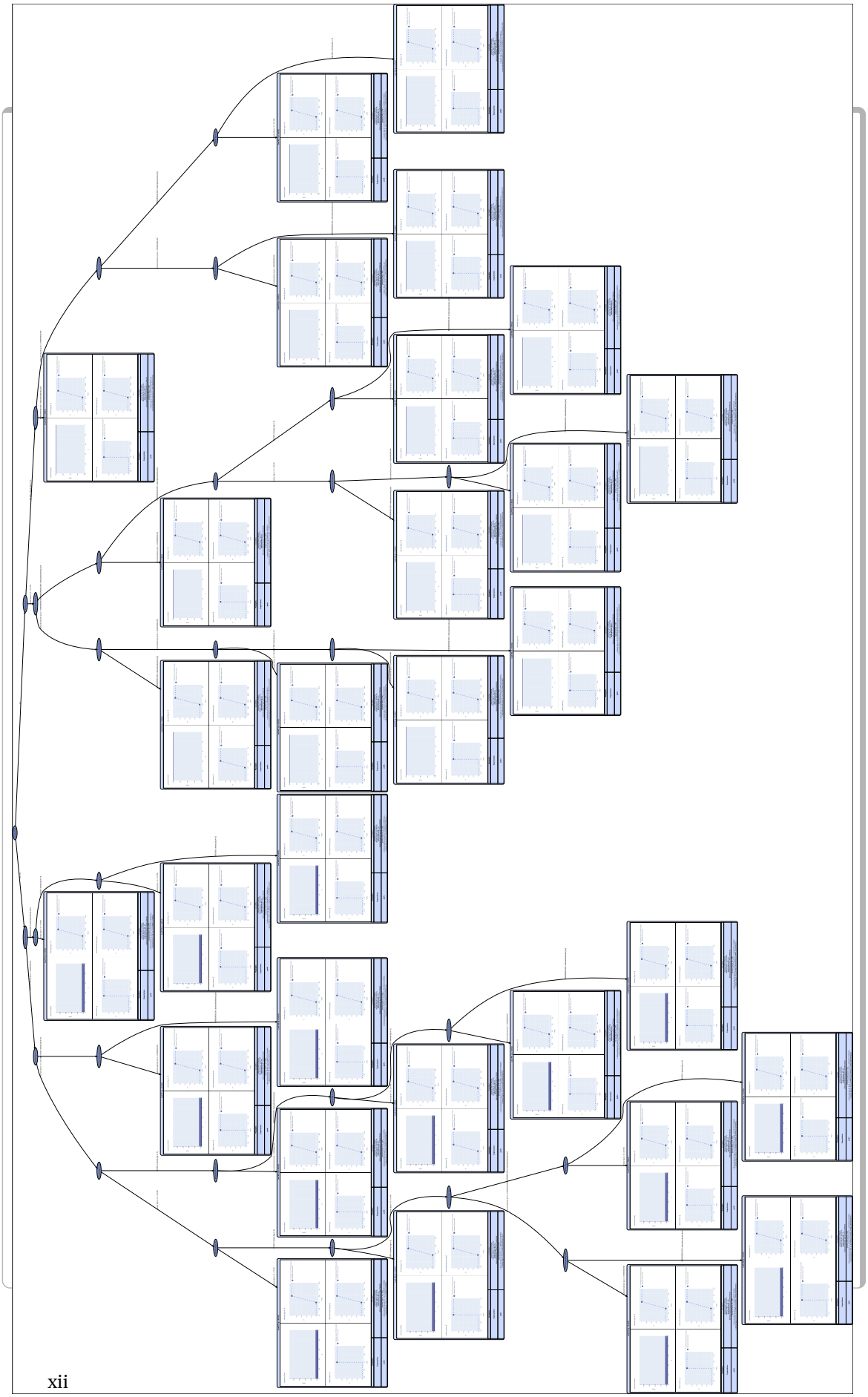


B.3 Heating (JPT)



The generated JPT for the heating process. | *Figure 92*

B.4 Deeprolling (JPT)



The generated tree for the deep rolling process. | *Figure 93*

Appendix C

PROVENANCE

C.1 Media Sources

- Figure 1 was created with the assistance of *DALL·E 2*[44]
- Figure 9 was adapted from an image used in the Lecture "Foundations of Artificial Intelligence" held annually in the summer term of the University of Bremen. The original creator is unknown.
- Figures 15, 16, 17, 18 as well as Tables 1, 2, 3 and 15 in Chapter 3 and Section A.2.1 of the Appendix are (adapted) versions of tables and images in Picklum, Nyga, Schierenbeck, and Beetz (2023)
- Figures 40, 41, 42, 43, 44, 45, 46, 47 as well as Table 5 in Chapter 5 were taken (and partly recreated to match the document style) from Picklum and Beetz (2019)
- Figure 79 contains elements taken from Wikimedia Commons[45]

C.2 Datasets

- The datasets for the `move_base`, `turn`, and `perception` models are generated and owned by the author.
- The dataset for the `pr2` model (by courtesy of Sebastian Koralewski) was generated from the raw experience data (NEEMs) logs collected when conducting pick-and-place experiments using a PR2 robot in the laboratory kitchen of the *Institute for Artificial Intelligence (IAI)* in Bremen.

[44] DALL·E 2
[45] Wikimedia Commons