Scalable Structure Learning, Inference, and Analysis with Probabilistic Programs

by

Feras Ahmad Khaled Saad

S.B., Massachusetts Institute of Technology (2016)
M.Eng., Massachusetts Institute of Technology (2016)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2022

Author . . . .
Department of Electrical Engineering and Computer Science
August 26, 2022

Certified by . . . .
Vikash K. Mansinghka
Principal Research Scientist
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

Scalable Structure Learning, Inference, and Analysis with Probabilistic Programs

by

Feras Ahmad Khaled Saad

Submitted to the Department of Electrical Engineering and Computer Science
on August 26, 2022, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

How can we automate and scale up the processes of learning accurate probabilistic models of complex data and obtaining principled solutions to probabilistic inference and analysis queries? This thesis presents efficient techniques for addressing these fundamental challenges grounded in probabilistic programming, that is, by representing probabilistic models as computer programs in specialized programming languages. First, I introduce scalable methods for real-time synthesis of probabilistic programs in domain-specific data modeling languages, by performing Bayesian structure learning over hierarchies of symbolic program representations. These methods let us automatically discover accurate and interpretable models in a variety of settings, including cross-sectional data, relational data, and univariate and multivariate time series data; as well as models whose structures are generated by probabilistic context-free grammars. Second, I describe SPPL, a probabilistic programming language that integrates knowledge compilation and symbolic analysis to compute sound exact answers to many Bayesian inference queries about both hand-written and machine-synthesized probabilistic programs. Third, I present fast algorithms for analyzing statistical properties of probabilistic programs in cases where exact inference is intractable. These algorithms operate entirely through black-box computational interfaces to probabilistic programs and solve challenging problems such as estimating bounds on the information flow between arbitrary sets of program variables and testing the convergence of sampling-based algorithms for approximate posterior inference. A large collection of empirical evaluations establish that, taken together, these techniques can outperform multiple state-of-the-art systems across diverse real-world data science problems, which include adapting to extreme novelty in streaming time series data; imputing and forecasting sparse multivariate flu rates; discovering commonsense clusters in relational and temporal macroeconomic data; generating synthetic satellite records with realistic orbital physics; finding information-theoretically optimal medical tests for liver disease and diabetes; and verifying the fairness of machine learning classifiers.

Thesis Supervisor: Vikash K. Mansinghka
Title: Principal Research Scientist

ٱلْحَمْدُ لِلَّهِ رَبِّ ٱلْعَالَمِينَ

Acknowledgements

I am fortunate to have benefited from many advisers, colleagues, and friends during my time at MIT.

My advisor, Vikash Mansinghka, has provided exceptional mentorship that greatly advanced my personal and professional development. His persistence in light of my frequent objections guided us through many joyful research journeys that would have otherwise never materialized. I thank Vikash for helping me establish essential foundations during my early years in the group and for making prudent interventions every now and then as I began to draw out an independent research program.

Collaborating with Martin Rinard shaped my graduate career in several ways. Martin Rinard is a fountain of rare knowledge and unconventional wisdom. His strenuous Socratic interrogations helped us together distill the durable ideas at the intellectual core of our own research endeavors. I am grateful to Martin for strengthening the quality of my research and academic writing and for many memorable conversations about the wilderness in the wider world beyond academia and MIT.

Joshua Tenenbaum and Armando Solar-Lezama served on my thesis committee and stepped in with crucial support when it was most needed. Several themes around Bayesian structure learning for automated model discovery in Part I of this thesis build directly on arguments that Josh has been making for the better part of two decades now. Armando's graduate programming languages course formally introduced me to the field and enabled me to integrate PL as a pillar in my research approach. I wish to also thank Nick Roy, who gave an expansive end-of-semester speech that had all the ingredients to encourage a curious but hesitant undergraduate to join graduate school.

The wonderful environment at the MIT Probabilistic Computing Project made working in lab a pleasure. Michael Chang deserves special recognition for originally connecting me with the group. Ulrich Scheachtle was a fun collaborator and formidable debate opponent; our discussions about science, politics, and culture brought life to the otherwise mundane routine of running experiments or solving DARPA challenge problems. Taylor Campbell and Alexey Radul patiently fielded my barrage of questions about unfamiliar software systems and taught me many tricks of the software engineering trade. Marco Cusumano-Towner was a brilliant research colleague and a dependable friend. Two chapters in this thesis would have remained unpublished research notes without his encouragement. Marco developed the Gen probabilistic programming system, which is used in this thesis to implement the online time series learner in Chapter 2 (improving on an earlier version developed with Ulrich) and to implement the meta-programs for statistical estimation and testing in Chapters 8 and 9. Cameron Freer and Nate Ackerman derived the proof of Proposition 9.21 in Chapter 9, theoretically establishing a mathematical conjecture obtained from large-scale computer simulations. Cameron also made central contributions to other joint work on random variate generation that is not included in this thesis. Alex Lew provided input that improved several of my research papers; his methodological approach to listening and collaborative reasoning is an asset to those who work around him. Amanda Brower and Rachel Paiste were very helpful in managing administrative and organizational matters. Christina Curlette, Leonardo Casarsa, Jonathan Rees, Veronica Weiner, and Andrew Bolton courageously used prototype software from my research in their own work, filing many bug reports and usability issues along the way. Finally, I must acknowledge all the colleagues whose shared an office with me, for tolerating the near-freezing room temperatures and my ear-shattering mechanical keyboard.

During my visits to CMU, I received a substantial amount of feedback from several people, including Jan Hoffmann, Matt Fredrikson, Limin Jia, Frank Pfenning, Bob Harper, Marijn Huele, Umut Acar, Guy Blelloch, Pravesh Kothari, Weina Wang, and Srini Seshan. The madPL group—Aws Albarghouthi, Thomas Reps, Loris D'Antoni, and Somesh Ja—along with Remzi Arpaci-Dusseau were wonderful hosts at UW-Madison. I am thankful for all these valuable interactions and am confident that they will serve me well in the years to come.

Everything I have managed to achieve is a direct result of the support and sacrifices of my family. To you, my gratitude is eternal.

THIS PAGE INTENTIONALLY LEFT BLANK

Contents

List of Figures

List of Tables

THIS PAGE INTENTIONALLY LEFT BLANK

List of Listings

List of Algorithms

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 1

Introduction

> "When it is evening, you say, 'It will be fair weather, for the sky is red.' And in the morning, 'It will be stormy today, for the sky is red and threatening.' You know how to interpret the appearance of the sky, but you cannot interpret the signs of the times."
>
> Matthew 16:2–3 (ESV)

The problem of inferring knowledge that enables prediction and interpretation of complex empirical phenomena is central to many studies in the social, physical, and natural sciences. Consider, for example, a time series, where the value of a noisily measured variable such as daily rainfall is evolving over time. A useful model of this data might reveal the underlying temporal structures—such as linear trends, periodic components, or changepoints—that explain the variation in the observed data and help predict future data. These problems are far from new: well before the advent of modern science, ancient civilizations attempted to solve these problems using techniques such as astrology, physiognomy, and other forms of divination. While such techniques have largely given way to scientific and computerized approaches, the foundational problems of "discovering" and "querying" models of complex data are two recurring themes that appear under various guises across research disciplines, for example:

- In artificial intelligence—as "knowledge acquisition" of a rule base [Marcus, 1988] and "automated reasoning" in an inference engine [Portoraro, 2021] within an expert system.
- In statistics—as "structure learning" [Heckerman, 1999] and "latent variable inference" [Cowell, 1999] within a family of probabilistic models.
- In cognitive science—as "inductive learning" [Smith, 1989] and "deductive reasoning" [Johnson-Laird, 1999] within an intuitive theory.
- In programming languages—as "program synthesis" [Gulwani et al., 2017] and "program analysis" [Nielson et al., 1999] within a class of computer programs.

This thesis is concerned with two primary research questions that tie these themes together:

How can we build systems that automatically discover probabilistic models of noisy empirical data?
How can we build systems that return principled answers to queries about probabilistic models?

Our research questions are centered around "probabilistic models" because the data generating processes we are interested in discovering and reasoning about are, as with most empirical real-world phenomena, inherently uncertain and produce noisy observed data.

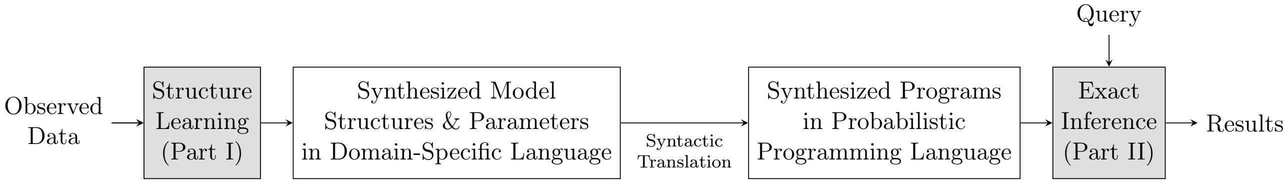

Query
Observed Data
Structure Learning (Part I)
Synthesized Model Structures & Parameters in Domain-Specific Language
Syntactic Translation
Synthesized Programs in Probabilistic Programming Language
Exact Inference (Part II)
Results

Figure 1.1: Overview of probabilistic structure learning and inference in probabilistic programs.

The main contributions of this thesis are given in three parts.

- Part I shows how to automatically learn the structure and parameters of probabilistic models given complex observational data in multiple real-world domains, by using approximate Bayesian inference over symbolic expressions in probabilistic domain-specific data modeling languages.
- Part II shows how to efficiently obtain sound exact solutions to Bayesian inference queries about machine-synthesized and hand-crafted probabilistic models, by representing models as probabilistic programs and leveraging specialized compilers and symbolic analyses to compute answers.
- Part III shows how to accurately solve high-dimensional statistical estimation and testing problems in probabilistic programs, by performing dynamic analyses of execution traces obtained via black-box computational interfaces for stochastic simulation.

Figure 1.1 shows that automatically discovered models obtained using the structure learning techniques in Part I can be translated into probabilistic programs for which many Bayesian inference queries are tractable to solve exactly using the symbolic analysis techniques in Part II. The methods in Part III connect with both of these areas. First, they can be used to test the convergence of approximate Bayesian inference algorithms for structure learning. Second, they can deliver approximate solutions to estimation queries about probabilistic programs that are fundamentally intractable to solve exactly.

Why Automatically Learn Probabilistic Models? It is often said that traditional statistics operates best in the “small data” regime [Lazer et al., 2014, Broderick, 2014, Faraway and Augustin, 2018] whereas machine learning shines in “big data” [Al-Jarrah et al., 2015, Zhou et al., 2017]. The line of demarcation between “small data” and “big data” is, of course, elusive at best, as these terms mean different things to different people. Faraway and Augustin [2018] suggest the following contrast: “Big data deals with the large, observational, and machine analyzed. Small data results from the experimental or intentionally collected data of a human scale, where the focus is on causation and understanding rather than prediction”. Whereas a typical small data problem is associated with a strong domain theory of how the observed data was generated, in big data problems the sheer number of variables and observations and tangled interactions between them make it too unwieldy for a human to manually build a model that accurately represents the data generating process. While pattern recognition approaches for classification and regression work well in many applications, they are not able to predict structure that can be used to obtain a systematic understanding of the data, as grimly embodied by the ***Tough Luck*** outcome in the visual user guide of the world’s most popular machine learning software library [Pedregosa et al., 2011]. Moreover, conventional unsupervised learning methods such as principal component analysis or biclustering assume a fixed form of structure, which imposes strong limits on the type of theories that can be discovered from data and the ability to adapt to novel observations. The so-called “Parable of Google Flu” [Lazer et al., 2014]—in which a sophisticated flu tracking system developed by Google was vastly over-predicting the proportion of doctor visits for influenza and missed the peak of the 2013 flu season by 140 percent—illustrate the pitfalls of forgoing a structured understanding of the data in favor of high capacity prediction algorithms. It should be

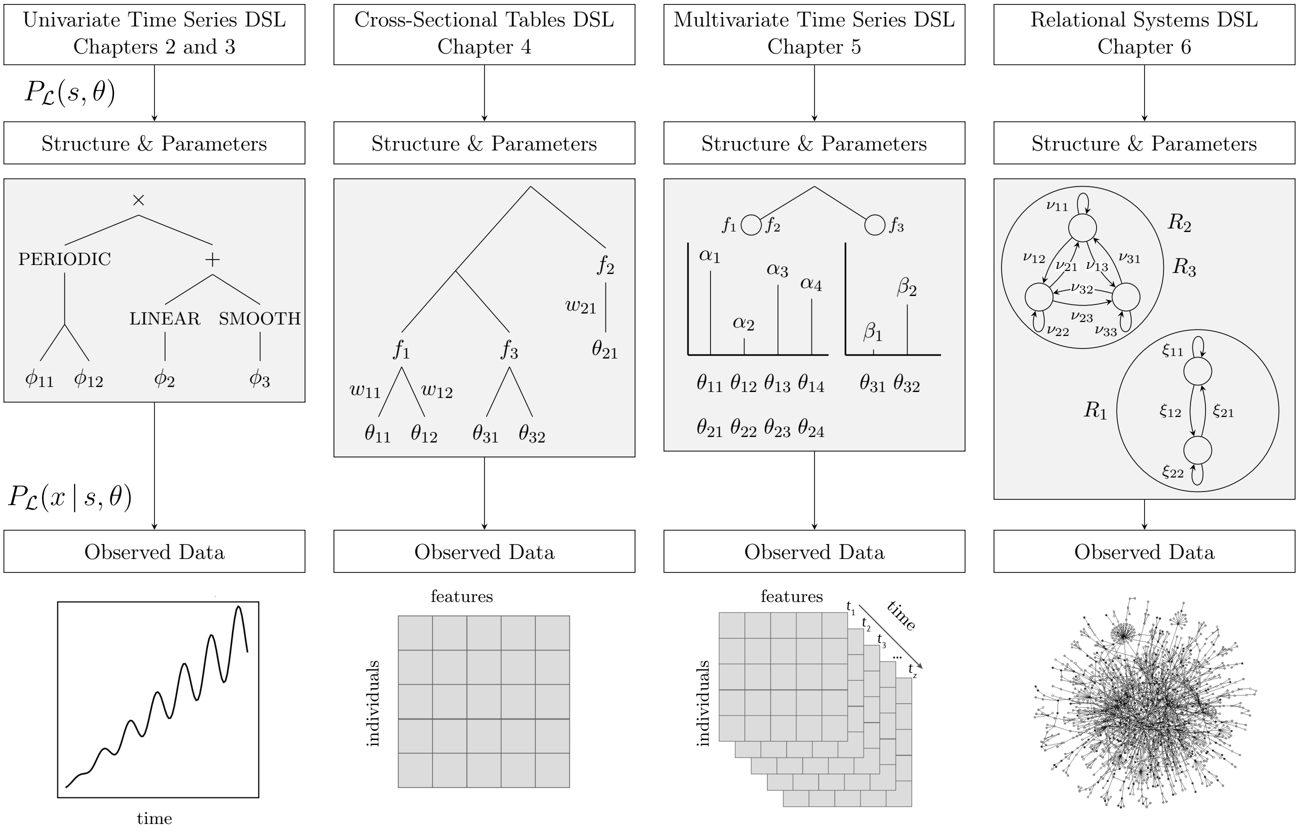


Figure 1.2: Hierarchical Bayesian framework for synthesizing probabilistic generative model structures and parameters in four example domains from Part I. Unlike traditional techniques that fit the parameters of a fixed model structure, the proposed approach is to instead define domain-specific data modeling languages $\mathcal{L}$ (top row) whose constituent expressions form a large family of model structures and parameters. Each domain-specific language $\mathcal{L}$ is associated with a prior distribution $P_{\mathcal{L}}(s, \theta)$ over latent model structure s and parameters θ (middle row), which together induce a probabilistic generative model $P_{\mathcal{L}}(x \mid s, \theta)$ over datasets x in the domain (bottom row). For a given dataset x, the structure learning problem is to obtain an ensemble of plausible model structures and parameters that explain the data by generating samples from the posterior distribution $P_{\mathcal{L}}(s, \theta \mid x) \propto P_{\mathcal{L}}(s, \theta) P_{\mathcal{L}}(x \mid s, \theta)$. These posterior samples are used for two purposes: first, obtaining qualitative insights, by syntactic analysis of the discovered model structures and parameters; second, solving prediction problems, by translating domain-specific expressions into probabilistic programs in SPPL (Chapter 7) which provides fast and automated machinery for computing provably sound exact solutions to Bayesian inference queries.

unsurprising to those familiar with the arguments of Hand [2006] that a simple model which projected lagged two-week data performed better than Google's system [Goel et al., 2010].

It is thus our motivation to develop automatic model discovery systems that go beyond pattern recognition by using observational data to synthesize probabilistic model structures and parameters that make accurate predictions while supporting explanation and understanding. These model discovery systems explore spaces of structured theories to find good explanations of the observed data. Each theory is formally represented as a symbolic expression in a probabilistic domain-specific data modeling language, which defines a family of model structures and parameters for explaining datasets within a domain. To enable understanding, the discovered models are designed to retain interpretable semantics, answer not one but an open-ended set of queries, and produce well-calibrated results, i.e., when producing inaccurate predictions, they report broad error bars to reflect lack of confidence.

Bayesian Inference The approach to automatic model discovery pursued in this thesis is firmly rooted in Bayesian inference, which gives a rational framework for updating beliefs about latent variables in generative models given observed data [Savage, 1954]. A full account of the breadth of the Bayesian paradigm across disciplines is impossible to summarize succinctly. In brief, Bayesian inference has enabled decades of fruitful progress in many research fields that include artificial intelligence [Pearl, 1988], information theory [MacKay, 2003], statistics [Gelman et al., 2014], robotics [Thrun et al., 2005], machine learning [Murphy, 2012, Ghahramani, 2015], epistemology [Landes, 2021], human cognition [Griffiths et al., 2008], and the bioinformatics [Lesaffre and Lawson, 2012].

Bayesian modeling specifies a universe of possible explanations of some observed data x through a space Θ of hypotheses, sometimes called parameters or latent variables. Each hypothesis $\theta \in \Theta$ is associated with a prior probability denoted $P(\theta)$ as well as a probability distribution $P(x \,|\, \theta)$ over datasets $x \in X$ that can be observed. The joint probability distribution $P(\theta, x) ::= P(x \,|\, \theta)P(\theta)$ over latent variables and data is referred to as a generative model, or simply a model. Given a specific dataset x, the Bayes rule updates the prior probability $P(\theta)$ to posterior probabilities $P(\theta \,|\, x)$ according to

$$P(\theta \,|\, x) ::= \frac{P(x \,|\, \theta)P(\theta)}{\sum_{\theta' \in \Theta} P(x \,|\, \theta')P(\theta')} \propto P(\theta, x) \qquad\qquad (\theta \in \Theta). \tag{1.1}$$

Eq. (1.1) quantifies how plausible each hypothesis θ is given the data x, which is a combination of its prior probability before observing x, as measured by $P(\theta)$, and the degree to which it explains x, as measured by $P(x \,|\, \theta)$. The Bayesian inference problem is then to "compute" the posterior distribution (1.1), which usually means either deriving a computational process that generates a random hypothesis θ with probability $P(\theta \,|\, x)$ or deriving an algorithm that returns the real number $P(\theta \,|\, x)$ for any θ.

A central contribution of this thesis is to show that Bayesian inference gives a practical and scalable approach not only to solving inference problems within a fixed model, but also to discovering plausible model structures in real-world data science applications where the appropriate model to use is itself uncertain. This contribution builds on a line of research in the cognitive science literature that uses Bayesian inference over structured spaces to explain and predict human behavior [Tenenbaum et al., 2011, Lake et al., 2017, Ullman and Tenenbaum, 2020] toward engineering more commonsense machine intelligence. The Bayesian paradigm invites the view that discovering models from observed data and solving queries about latent variables within a fixed model (box on Page 15) are both instances of Bayesian inference that operate at different levels of abstraction. A prior distribution $P(s)$ is introduced over unknown model structure $s \in S$, which itself is a latent variable at the highest level of abstraction that induces generative models $P(\theta, x \,|\, s)$. The overall model is then $P(s, \theta, x) = P(s)P(\theta \,|\, s)P(x \,|\, s, \theta)$, and the inference problem is to generate samples from $P(s, \theta \,|\, x)$. This problem is referred to as "Bayesian synthesis" throughout the thesis, as the goal is to "synthesize" a collection $\{(s_i, \theta_i)\}$ of model structures and parameters that together form explanations of the observed data. Figure 1.2 shows hierarchical Bayesian models for structure learning in four representative domains covered in Part I.

Probabilistic Programing Languages The centrality of Bayesian inference has brought about substantial interest in probabilistic programming [Russell and Norvig, 2021, Chapter 18; van de Meent et al., 2021], an emergent research field that tightly integrates probability into the design and implementation of programming languages. Probabilistic programs provide specialized modeling languages for expressing rich probabilistic models and reusable computational machinery for solving inference problems. This approach lets us integrate central ideas from computer science—such as language design, semantics, compilers, symbolic execution, and dynamic analysis—to more effectively express models and perform inference. Recent years have seen a surge in applications of probabilistic programming in areas such as computer vision [Kulkarni et al., 2015, Gothoskar et al., 2021], programmable networks [Gehr et al., 2018], high-energy physics [Baydin et al., 2019], and bioinformatics [Merrell and Gitter, 2020].

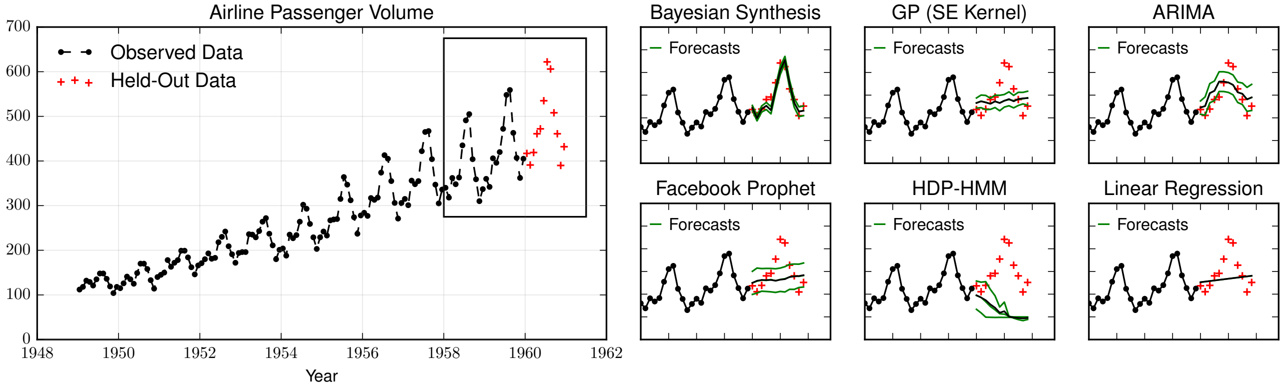


Figure 1.3: Bayesian synthesis techniques for learning models of univariate time series discover detailed temporal patterns in the observed data and produce substantially more accurate forecasts as compared to widely used methods from statistics and machine learning. Discovering models for univariate time series data is the subject of Chapter 2.

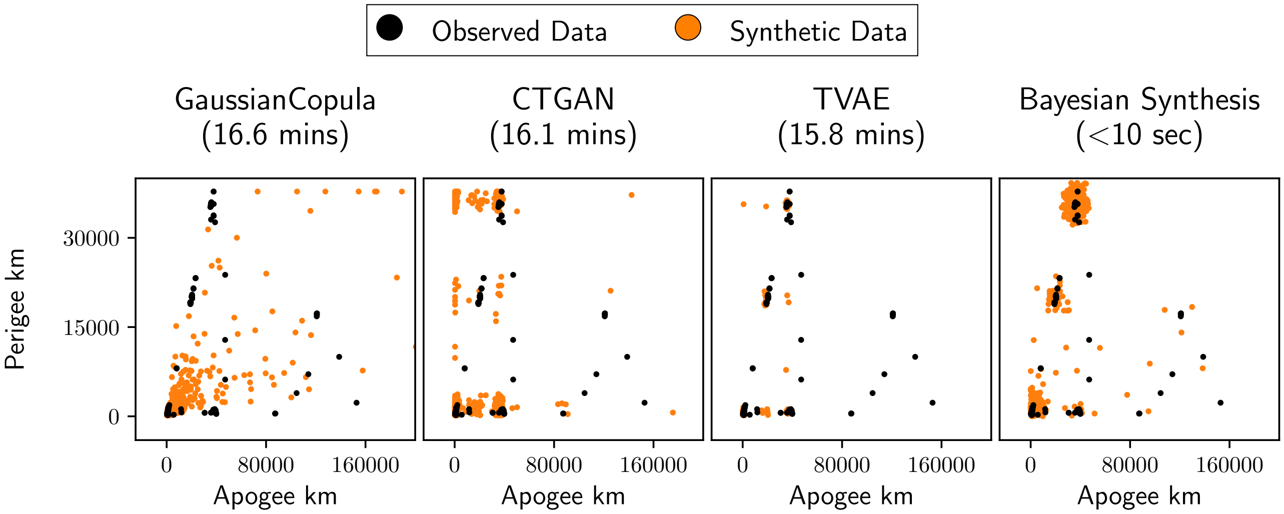


Figure 1.4: Bayesian synthesis techniques for learning models of cross-sectional data tables produce more realistic synthetic data as compared to statistical baselines such as Gaussian copulas and deep learning baselines such as conditional tabular generative adversarial networks (CTGAN) and tabular variational autoencoders (TVAE)—using 100x less runtime. The plots above show a pair of variables describing the orbital physics of satellites from a dataset of 21 variables and 1167 satellites, where observed data is in black and synthetic data in orange. Synthetic data from models discovered by Bayesian synthesis closely matches the marginal distribution of this pair of variables, whereas the baseline methods produce synthetic data in artificial regimes where no true data exists or fail to produce synthetic data in regimes where true data exists. Discovering models for cross-sectional data tables is the subject of Chapter 4.

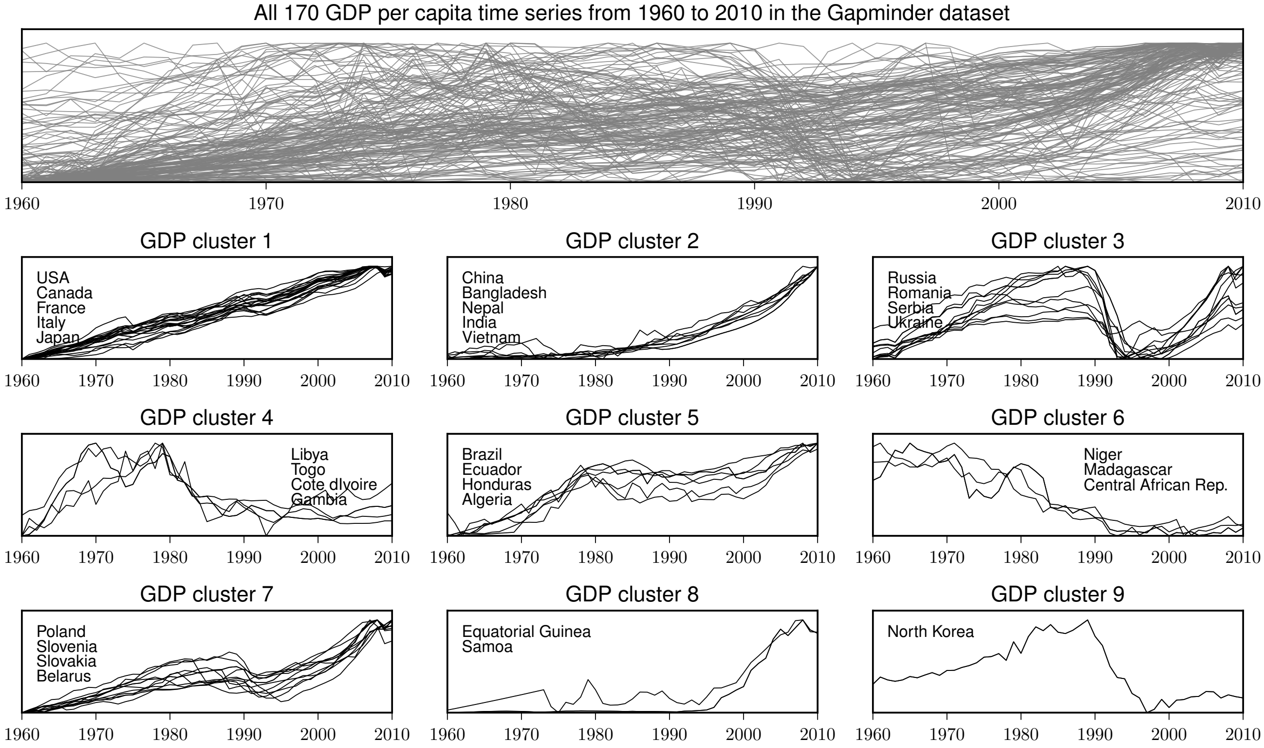


Figure 1.5: Bayesian synthesis techniques for learning models of multivariate time series discover interpretable groups of countries whose economic indicators—in this case the GDP per capita—evolved through similar temporal processes over 50 years, which include linear trends, exponential growth, and changepoints. Discovering models for multivariate time series data is the subject of Chapter 5.

How can we realize the broad promise of probabilistic programming for more automated and scalable probabilistic approaches to analyzing structured data? There are certainly many challenges that practitioners face when leveraging the probabilistic modeling and inference toolkit. A main challenge is the expertise in statistical modeling and programming needed to build suitable probabilistic models [Henley et al., 2020]. Automatic model discovery can reduce these burdens, provided that the learning techniques scale well in real-world problems and deliver accurate and interpretable solutions. Another challenge is the lack of accuracy, soundness, or runtime stability in existing inference engines, largely due to their lack of specialization. Third, the lack of abstraction in the usual approach of representing models as ad-hoc data structures in languages such as C or Python requires custom query solvers to be implemented over and over, which simply does not scale as a reliable engineering methodology.

The approach to structure learning and inference in probabilistic programs shown in Figure 1.1 goes beyond existing probabilistic programming languages such as BUGS [Gilks et al., 1994], Stan [Carpenter et al., 2017], Gen [Cusumano-Towner et al., 2019], or Pyro [Bingham et al., 2019], which do not contain automatic model discovery systems. That said, probabilistic programming has a central role to play in both components of Figure 1.1. First, the process of synthesizing model structures and parameters in domain-specific languages can itself be implemented using probabilistic programming, as illustrated in Saad et al. [2019a] using the Venture language [Mansinghka et al., 2014] and in Cusumano-Towner et al. [2020] using the Gen language [Cusumano-Towner, 2020]. Second, the learned models are translated from domain-specific languages into probabilistic programming languages with reusable and optimized inference machinery. This thesis introduces SPPL in Chapter 7, a probabilistic programming language that delivers sound, exact, and fully-automated solutions for all domain-specific models from Part I.

1.1 The Path to Scalability

The idea of using Bayesian inference gngnbghbbhbvglearn the structure of generative models builds on a line of research in the cognitive science and probabilistic machine learning literature, which is surveyed in Section 3.6. Even though there are many benefits to pursuing a coherent probabilistic approach to structure learning, a recurring challenge that has impeded the broader adoption of these methods is the lack of a design and engineering methodology that is both mathematically principled and practically scalable. Similarly, while many probabilistic programming systems provide a universal or "Turing-complete" language for expressing arbitrarily sophisticated probabilistic models, the question of how to design abstractions that enable tractable solutions remains an open problem. This thesis uses several techniques that improve the scalability of structure learning and inference in probabilistic programs, where "scaling" refers not only to the number of variables and observations, but also to the range of problem domains that can be analyzed and to the types of queries that can be efficiently solved.

1.1.1 Specialization via Probabilistic Domain-Specific Languages

Specialization in this context means carefully restricting the class of models under consideration for a given modeling problem and the class of queries that can be asked. Lack of specialization is a fundamental reason that research endeavors such as learning the structure of arbitrary Bayesian networks, searching for neural network architectures, or synthesizing programs in Turing-complete programming languages have largely failed to scale to meaningful real-world applications. The route pursued in Part I is to design probabilistic domain-specific data modeling languages with (i) simple modeling primitives and composition operators, for defining model structures with interpretable surface syntax; (ii) nonparametric probabilistic semantics, so that the model structures are flexible enough to express a wide range of data patterns; and (iii) efficient algorithms for solving a broad class of Bayesian inference queries about observed data, such as computing marginal and conditional probabilities. Each DSL presented in Part I is specialized in all these ways. In Part II, the SPPL system from Chapter 7 is also specialized in that it syntactically rules out a large class of probabilistic models and Bayesian inference queries for which obtaining exact solutions is fundamentally intractable.

1.1.2 Flexible Model Families via Nonparametric Bayesian Priors

Traditional modeling approaches in statistics and machine learning are based on a fixed model structure, which imposes limits on the type of structure that can be learned from data and the ability to adapt to novel data. These approaches also provide no built-in support for improving their own structure in the presence of severe model misspecification. The probabilistic DSLs introduced in Part I instead define entire families of many model structures by using Bayesian nonparametric distributions, reviewed in forthcoming chapters, which compactly specify rich priors over infinite-dimensional spaces of functions, partitions, and other combinatorial structures. Bayesian nonparametric distributions are used in two different ways. First, they are used to specify prior distributions over expressions in the DSL. Second, they are used to specify prior distributions over datasets that a given DSL expression defines. The resulting models have unbounded size, in the sense that their internal structure can automatically adapt as new and novel observations become available (Figure 2.8). Within these DSLs, inference over the unknown model structure and parameters is performed via approximate posterior sampling—as opposed to greedy optimization methods such as cross-validation or variational learning—for quantifying the inherent uncertainty about the latent data generating process.

1.1.3 Exploiting Sparsity via Independence Discovery

Each probabilistic DSL in Part I includes modeling constructs for expressing independence relationships, that is, groups of variables or observations that can be modeled completely separately from one another. Exploiting probabilistic independencies can enable large improvements in scalability. Consider 10 variables each with 10 outcomes—a joint probability distribution that assumes all these variables are dependent requires 10 billion parameters to be fully specified; if the variables factorize into two separate groups then 200,000 parameters are needed; and if all 10 variables are mutually independent then only 100 parameters suffice. The Bayesian nonparametric priors over DSL expressions allow structural independencies to be inferred from observations, provided that there is sufficient evidence to suggest that such independencies are likely to exist. These inductive biases often lead to more concise and interpretable explanations of the data as compared to assuming full dependence (Figures 5.4 and 6.3).

1.1.4 Online Structure Learning via Sequential Monte Carlo

Computing the posterior distribution (1.1) is intractable for all but the simplest of problems, which means that approximate algorithms for Bayesian structure learning are needed. Most approaches either use simulation-based algorithms such as Markov chain Monte Carlo (MCMC), which provide sound approximations but converge slowly, or abandon sound approximations and instead adopt heuristics such as greedy search. Chapter 3 shows how to design more scalable learning algorithms using sound techniques known as “resample-move sequential Monte Carlo” (SMC), which can perform Bayesian structure learning in real time and greatly improve upon MCMC (Figure 2.11). SMC-based structure learning composes well with the Bayesian nonparametric priors over DSL expressions: an initial batch of observations is used to synthesize a starting set of plausible model structures which are incrementally refined and adapted as more data is observed (Figures 2.5 and 4.6).

1.1.5 Exact Bayesian Inference via Compilers and Symbolic Program Analysis

While supervised learning methods are trained to solve only a single query, a generative model obtained by structure learning can be used to solve an open-ended set of queries, such as simulating forecasts, computing probabilities, quantifying information flow, or finding probable outliers. However, it quickly becomes impractical to develop, optimize, and prove the correctness of custom inference algorithms for solving each new query within a DSL, let alone many queries across many DSLs. Chapter 7 develops the SPPL probabilistic programming language, which provides provably sound and exact answers to many probabilistic inference queries about programs expressed in the language. SPPL can also serve as a unified probabilistic program synthesis target language for the probabilistic DSLs presented in Part I, which enables substantial reuse of its inference machinery across multiple models, datasets, and queries.

1.1.6 Statistical Estimation and Testing via Dynamic Program Analysis

Many queries about probabilistic programs are fundamentally intractable to solve exactly. One example is estimating conditional mutual information among groups of program variables, which SPPL either syntactically rules out (for numerical variables) or can only solve exactly by enumeration over an exponentially large state space (for discrete variables). Another example is testing the convergence of sampling-based algorithms for approximate posterior inference such as MCMC or SMC used for Bayesian structure learning. Chapters 8 and 9 introduce fast and theoretically principled methods, implemented as meta-programs in the Gen probabilistic programming language (Listings 8.1–8.4 and 9.1), for solving two such inference problems: estimating information measures between random variables and testing convergence of approximate sampling algorithms. These chapters show how to design statistical inference procedures with good frequentist properties by simulating probabilistic programs using black-box interfaces and performing dynamic analyses of the weighted traces.

1.2 Outline and Contributions

This thesis is comprised of three main parts that integrate the themes for engineering scalable systems described in Section 1.1.

Part I describes scalable structure learning techniques for synthesizing generative models of complex observational data in multiple real-world domains.

Chapter 2 shows how to learn accurate and interpretable models for univariate time series data.

Chapter 3 formalizes the problem of Bayesian structure learning for synthesizing probabilistic programs in domain-specific data modeling languages; presents sound synthesis algorithms for approximate posterior inference; and establishes a class of DSLs generated by context-free grammars where sound Bayesian synthesis can be fully automated end-to-end.

Chapters 4–6 present structure learning methods for synthesizing generative models of cross-sectional data tables, multivariate time series, and relational data; and uses them to solve challenging tasks in exploratory, inferential, and predictive data analytics.

Part II addresses the problem of efficiently computing exact solutions to Bayesian inference queries by leveraging probabilistic program compilers and symbolic analysis techniques.

Chapter 7 presents the SPPL probabilistic programming language that delivers exact answers to a range of Bayesian inference queries. SPPL returns provably sound answers, is fully automated, and has substantially more efficient and predictable performance as compared to state-of-the-art solvers. The evaluations show runtime gains of up to 10^5x on challenging benchmark problems from the literature. Each DSL from Part I admits a formal translation into SPPL syntax, which also allows the language to serve as a unified synthesis target with reusable machinery for fast probabilistic inference.

Part III shows how to solve challenging statistical estimation and testing problems via dynamic analyses of execution traces obtained by simulating probabilistic programs.

Chapter 8 presents interval estimators of entropy and many related information-theoretic quantities for arbitrary sets of variables in a probabilistic program. These estimators apply to substantially more general queries than traditional "model-based" estimators, scale better than "model-free" nonparametric estimators, and solve challenging queries in optimal experimental design.

Chapter 9 presents a family of goodness-of-fit tests for high-dimensional discrete data structures to assess how well observed an dataset matches the distribution defined by a probabilistic program. A central application of these tests is in diagnosing the convergence of simulation-based algorithms for approximate posterior inference, such as those used for structure learning in Part I.

Chapter 10 offers concluding remarks and discusses directions for future work.

Evaluations A large number of empirical results establish that the proposed approach to structure learning and inference can deliver substantial improvements over widely used baselines across diverse data science problems. Three representative examples are shown in Figures 1.3–1.5. Further evaluations include adapting to extreme novelty in streaming time series data (Figure 2.8); imputing and forecasting sparse multivariate flu rates (Tables 5.1 and 5.2 and Figures 5.7 and 5.8); finding commonsense structure in temporal and relational macroeconomic data (Figures 5.5 and 6.6); generating synthetic satellite records with realistic orbital physics (Figure 4.7); assessing the convergence of MCMC samplers for Dirichlet process mixtures and Ising models (Figures 9.5 and 9.6) ranking medical tests for liver disease and diabetes by their value of information about a patient's latent illnesses (Figures 8.5 and 8.8 and Table 8.1); and verifying the fairness of machine learning classifiers (Table 7.2).

1.3 Scope

- While the general principles and methods developed in this thesis are applicable to several problem domains, the main focus is on "structured" data science problems, which include time series, cross-sectional data tables, and relational systems, as opposed to "unstructured" data such as videos or text documents. Within structured data problems, the evaluations cover breadth across many domains and datasets rather than depth within a given domain.

- No attempt is made to automatically learn entire probabilistic domain-specific languages from data, using, for example, yet another level of abstraction in the hierarchical Bayesian framework from Figure 1.2. In particular, all the DSLs in Part I have been manually designed and then used repeatedly to solve structure learning and inference problems across many datasets and queries. While learning DSLs from data is conceptually appealing, there are many bottlenecks in scaling such approaches to online structure learning and inference for real-world data science problems.

- While all the DSLs in Part I have interpretable generative semantics and internally specify causal relationships between model variables, they should be understood as phenomenological models: their internal structures do not necessarily reflect the true causal structure in the underlying data generating process. Discovering causal relationships from purely observational data is generally impossible without stronger assumptions on the data generating process or experimenter-driven interventions during the data collection phase, which are both beyond the scope of this thesis.

- To avoid excessive formalisms that do not result in essential theorems, several technical details are omitted. Example omissions include the formal syntax of the underlying probabilistic domain-specific languages for multivariate time series in Chapter 5 and relational systems in Chapter 6, as well as formalisms of the program translators from each DSL in Part I to the source syntax of the SPPL probabilistic programming language in Chapter 7.

1.4 Software

Reference implementations of the methods described in this thesis are available online.

1. Chapter 2: DSL for univariate time series
   https://doi.org/10.1145/3291623
   https://github.com/probcomp/pldi2019-gen-experiments

2. Chapter 4: DSL for cross-sectional data
   https://doi.org/10.1145/3291623
   https://github.com/probcomp/cgpm

3. Chapter 5: DSL for multivariate time series
   https://github.com/probcomp/trcrpm

4. Chapter 6: DSL for relational data
   https://github.com/probcomp/hierarchical-irm

5. Chapter 7: Sum-Product Probabilistic Language
   https://github.com/probcomp/sppl

6. Chapter 8: Estimators of entropy and information
   https://proceedings.mlr.press/v151/saad22a/saad22a-supp.zip

1.5 Publications

This thesis is based on research presented in the following publications.

[Saad et al., 2022] Feras A. Saad, Marco Cusumano-Towner, and Vikash K. Mansinghka. Estimators of entropy and information via inference in probabilistic models. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 5604–5621. PMLR, 2022

[Saad and Mansinghka, 2021] Feras A. Saad and Vikash K. Mansinghka. Hierarchical infinite relational model. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1067–1077. PMLR, 2021

[Saad et al., 2021] Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. SPPL: Probabilistic programming with fast exact symbolic inference. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Design and Implementation*, pages 804–819. Association for Computing Machinery, 2021. doi:10.1145/3453483.3454078

[Saad et al., 2019b] Feras A. Saad, Cameron E. Freer, Nathanael L. Ackerman, and Vikash K. Mansinghka. A family of exact goodness-of-fit tests for high-dimensional discrete distributions. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 1640–1649. PMLR, 2019b

[Saad et al., 2019a] Feras A. Saad, Marco F. Cusumano-Towner, Ulrich Schaechtle, Martin C. Rinard, and Vikash K. Mansinghka. Bayesian synthesis of probabilistic programs for automatic data modeling. *Proceedings of the ACM on Programming Languages*, 3(POPL):37.1–37.32, 2019a. doi:10.1145/3290350

[Saad and Mansinghka, 2018] Feras A. Saad and Vikash K. Mansinghka. Temporally-reweighted Chinese restaurant process mixtures for clustering, imputing, and forecasting multivariate time series. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 755–764. PMLR, 2018

Part I

Probabilistic Structure Learning via Approximate Bayesian Inference

Chapter 2

Synthesizing Models for Univariate Time Series

> It is difficult to make predictions, especially about the future.
>
> Danish Proverb

Consider the data in Figure 2.1, which shows the number of miles traveled by paying airline passengers in the United States between 2009 and 2020, according to the U.S. Bureau of Transportation Statistics. Several patterns can be identified from a close inspection of the time series, including:

1. There is an overall increasing linear trend.
2. There is a yearly peak in airline travel during the summer months.
3. There is a yearly dip in airline travel during the first two months of the year.
4. The peak-to-peak amplitude between the winter and summer is increasing over time.
5. There is a single smaller peak in the spring, before the summer peak.
6. There are two smaller peaks in the fall, after the summer peak.

Even without constructing a statistical model of the data, humans are can rapidly identify interpretable structure of this form. This chapter shows how to automatically learn statistical models for time series that reflect interpretable patterns in the observed data and generate accurate forecasts for future data. Section 2.1 reviews Gaussian process models. Section 2.2 describe a domain-specific data modeling language based on Gaussian processes that can be used to express a rich class of time series

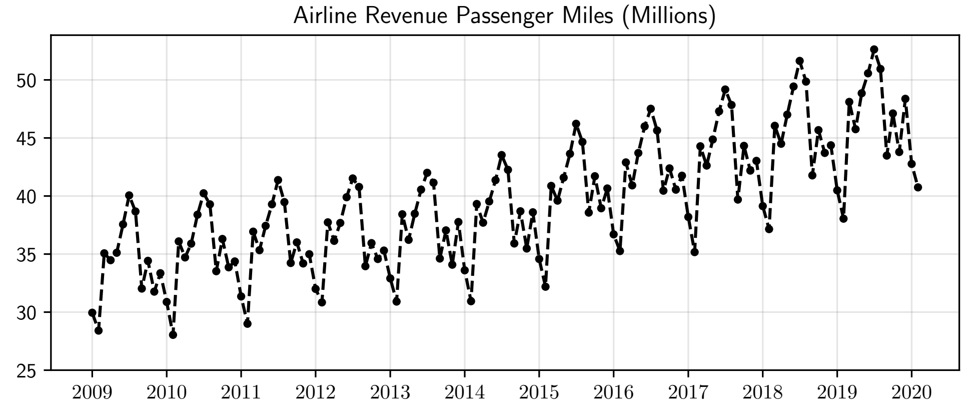


Figure 2.1: Monthly airline revenue passenger miles in the United States from Jan 2009 to Feb 2020.

structures. Section 2.3 presents a tutorial of how to use Bayesian inference to synthesize expressions in this DSL given an observed data stream. Section 2.4 quantitatively benchmarks these automatic data modeling techniques on multiple real-world econometric time series and shows that they outperform widely used statistical baselines.

2.1 Background: Gaussian Processes

Definition 2.1. Let $\mathbb{T}$ be an arbitrary index set. A collection $X ::= \{X(t) \mid t \in \mathbb{T}\}$ of random variables is said to be a *Gaussian process* if for any length n tuple $\mathbf{t} = [t_1, \dots, t_n]$ of distinct indexes, the random vector $X(\mathbf{t}) ::= [X(t_1), \dots, X(t_n)]$ has a joint Gaussian distribution. «

Remark 2.2. A Gaussian process is completely specified by its *mean function* $m : \mathbb{T} \to \mathbb{R}$ and *covariance function* $k : \mathbb{T} \times \mathbb{T} \to \mathbb{R}_{\geq 0}$, where for all $t, t' \in \mathbb{T}$,

$$m(t) = \mathbb{E}\left[X(t)\right] \tag{2.1}$$

$$k(t,t') = \operatorname{cov}(X(t), X(t')) = \mathbb{E}\left[(X(t) - m(t))(X(t') - m(t'))\right]. \tag{2.2}$$

The joint distribution of $X(\mathbf{t})$ is thus

$$\begin{bmatrix} X(t_1) \\ \vdots \\ X(t_n) \end{bmatrix} \sim \text{MultivariteNormal}\left(\begin{bmatrix} m(t_1) \\ \vdots \\ m(t_n) \end{bmatrix}, \begin{bmatrix} k(t_1,t_1) & \dots & k(t_1,t_n) \\ \vdots & \ddots & \vdots \\ k(t_n,t_1) & \dots & k(t_n,t_n) \end{bmatrix} \right). \tag{2.3}$$

«

The notation $X \sim \text{GP}(m,k)$ indicates that X is a Gaussian process with mean m and covariance k. The shorthand $m(\mathbf{t}) ::= [m(t_1), \dots, m(t_n)]$ denotes the mean vector. Further, if $\mathbf{t}' ::= [t'_1, \dots, t'_m]$ is a set of m indexes, then $k(\mathbf{t}, \mathbf{t}')$ denotes the $n \times m$ covariance matrix whose ij entry is $k(t_i, t'_j)$. The covariance matrix in Eq. (2.3) is then precisely $k(\mathbf{t}, \mathbf{t})$. This chapter focuses on random processes X that are one-dimensional continuous time series, so that the index set $\mathbb{T} = \mathbb{R}$.

Prior and Posterior Densities Equipped with the mean vector $m(\mathbf{t})$ and covariance matrix $k(\mathbf{t},\mathbf{t})$, the joint probability density of $X(\mathbf{t})$ evaluated at a realization $x(\mathbf{t}) ::= [x(t_1), \dots, x(t_n)] \in \mathbb{R}^n$ is

$$\log p(x(\mathbf{t})) = -\frac{1}{2}\left[[x(\mathbf{t}) - m(\mathbf{t})]^\top k(\mathbf{t},\mathbf{t})^{-1}[x(\mathbf{t}) - m(\mathbf{t})] - \log\left(\det(k(\mathbf{t},\mathbf{t}))\right) - n\log\left(2\pi\right) \right]. \tag{2.4}$$

As multivariate Gaussians are closed under conditioning, given observations $x(\mathbf{t})$, the posterior distribution of $X(\mathbf{t}')$ at new time points $\mathbf{t}'$ is also a multivariate Gaussian

$$X(\mathbf{t}') \mid \{X(\mathbf{t}) = x(\mathbf{t})\} \sim \text{MultivariateNormal}(m^{\text{post}}(\mathbf{t}'), k^{\text{post}}(\mathbf{t}', \mathbf{t}')) \tag{2.5}$$

$$m^{\text{post}}(\mathbf{t}') ::= k(\mathbf{t}', \mathbf{t})k(\mathbf{t},\mathbf{t})^{-1}x(\mathbf{t}) \tag{2.6}$$

$$k^{\text{post}}(\mathbf{t}', \mathbf{t}') ::= k(\mathbf{t}', \mathbf{t}') - k(\mathbf{t}', \mathbf{t})k(\mathbf{t},\mathbf{t})^{-1}k(\mathbf{t}, \mathbf{t}'). \tag{2.7}$$

The posterior joint density $p(x(\mathbf{t}') \mid x(\mathbf{t}))$ is then readily computed as in Eq. (2.4) by using the expressions for the conditional mean (2.6) and conditional covariance (2.7).

Observation Noise Suppose that $Z \sim \text{GP}(m,k)$ is a Gaussian process. A new Gaussian process X can be formed by setting $X(t) ::= Z(t) + \gamma(t)$ for all $t \in \mathbb{R}$, where $\gamma(t) \sim \text{Normal}(0, \epsilon)$ are i.i.d. Gaussian innovations. Then X is said to be a Gaussian process with output noise ϵ, denoted $X \sim \text{NoisyGP}(m,k,\epsilon)$. As any linear combination of independent Gaussian random variables is itself Gaussian, it can be seen that $X \sim \text{GP}(m, k')$ where $k'(t,t') = k(t,t') + \mathbf{1}[t = t']\epsilon$, for all $t, t' \in \mathbb{R}$.

2.2 Gaussian Process DSL for Modeling Univariate Time Series

Gaussian process are powerful statistical models. The covariance function dictates the structure of the time series X and provides the inductive bias that lets Gaussian processes capture complex patterns across many different datasets. Listing 2.1 shows a domain-specific modeling language $\mathcal{L}$ of symbolic expressions that describe the noise level ϵ and covariance expression K of a zero-mean noisy Gaussian process. There are four primitive covariance structures $\{\texttt{Constant}, \texttt{Smooth}, \texttt{Linear}, \texttt{Periodic}\}$ and three composition operators $\{\texttt{*}, \texttt{+}, \texttt{ChangePoint}\}$ that together produce an open set of more complex time series structures—several additions to this basic language are possible, as discussed in Duvenaud [2014, Chapter 2]. Listing 2.2 shows the semantics of the covariance expressions K, which define functions $k(t, t')$ that take a pair of time points and return the covariance (2.2) between $X(t)$ and $X(t')$. Table 2.1 explains the meaning of each numeric parameter in the covariance expressions.

Example 2.3. Figure 2.2 shows several realizations of noisy Gaussian process $X \sim \text{NoisyGP}(0, k, \epsilon)$ evaluated at 100 time points spaced linearly between $t = 0$ and $t = 10$, for various covariance functions $k ::= [\![K]\!]$ and noise levels ϵ. The resulting time series have the following structure:

Figure 2.2a $K =$ `Linear(1)`
Random straight lines whose time intercept is always 1.

Figure 2.2b $K =$ `Smooth(3.4)`
Random smooth functions.

Figure 2.2c $K =$ `Periodic(60, 3) * Linear(0.1)`
Random periodic waves whose amplitude increases over time, with an overall upward or downward linear trend whose time intercept is always 0.1.

Figure 2.2d $K =$ `ChangePoint(5, 0.001, Constant(2), Linear(6))`
Random functions whose behavior changes rapidly at time $t = 5$. For $t < 5$, the data is a random constant, whose value is distributed as Gaussian with mean zero and variance 2. For $t > 5$, the data is a random straight line with time intercept of 6.

≪

2.3 Online Learning of Time Series Structure from Data

In traditional time series modeling, practitioners manually construct a covariance structure for a given dataset using a combination of statistical expertise, heuristics, and domain knowledge [Littell et al., 2000, Kincaid, 2005]. This section shows how to automatically discover the covariance structure for a time series. In particular, given a time series such as the airline data in Figure 2.1, the goal is to synthesize expressions $E =$ `NoisyGP(0,` K`,` ϵ`)` in the DSL from Listing 2.1 such that the choice of K and ϵ capture patterns in the observed data and produce accurate forecasts for future data.

2.3.1 Formulation as a Bayesian Inference Problem

The proposed approach to synthesizing DSL expressions uses probabilistic inference over the unknown covariance and noise level of a Gaussian process time series model. Following the Bayesian approach of modeling unknown quantities in the DSL expression `NoisyGP(0,` K`,` ϵ`)` as random variables, for any set of $n > 0$ distinct time points $\mathbf{t} = [t_1, \ldots, t_n]$ the generative model is

$$
\begin{aligned}
&E ::= \texttt{NoisyGP(0, } K\texttt{, } \epsilon\texttt{)}\\
&K ::= \texttt{Constant(}\varphi\texttt{)} \mid \texttt{Linear(}\theta\texttt{)} \mid \texttt{Smooth(}\varphi\texttt{)} \mid \texttt{Periodic(}\varphi_1\texttt{,}\varphi_2\texttt{)}\\
&\quad \mid K_1 \texttt{ * } K_2 \mid K_1 \texttt{ + } K_2 \mid \texttt{ChangePoint(}\theta\texttt{,}\varphi\texttt{,}K_1\texttt{,}K_2\texttt{)}\\
&\epsilon \in \mathbb{R}_{>0} \qquad \theta \in \mathbb{R} \qquad \varphi \in \mathbb{R}_{>0}
\end{aligned}
$$

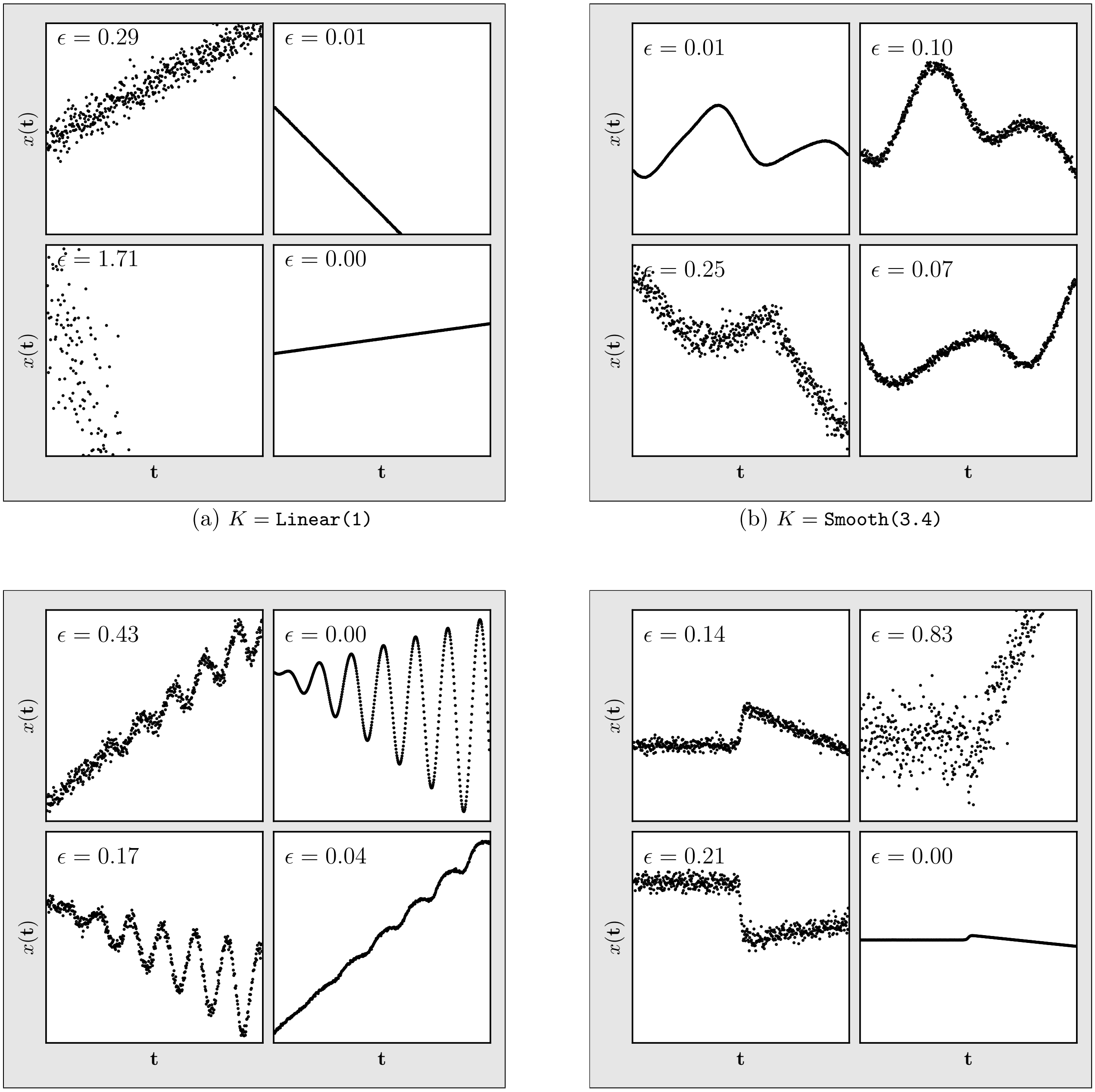
Listing 2.1: Context-free grammar defining a domain-specific language of time series structures.

$$
\begin{aligned}
[\![\texttt{Constant(}\varphi\texttt{)}]\!] &::= \lambda t.\lambda t'.\varphi\\
[\![\texttt{Linear(}\theta\texttt{)}]\!] &::= \lambda t.\lambda t'.(t-\theta)(t'-\theta)\\
[\![\texttt{Smooth(}\varphi\texttt{)}]\!] &::= \lambda t.\lambda t'.\exp(-(t-t')^2/2\varphi)\\
[\![\texttt{Periodic(}\varphi_1\texttt{,}\varphi_2\texttt{)}]\!] &::= \lambda t.\lambda t'.\exp(-\sin^2(2\pi/\varphi_2)|t-t'|)/\varphi_1)\\
[\![\texttt{(}K_1 \texttt{ * } K_2\texttt{)}]\!] &::= \lambda t.\lambda t'.\, [\![K_1]\!]\,(t,t') \times [\![K_2]\!]\,(t,t')\\
[\![\texttt{(}K_1 \texttt{ + } K_2\texttt{)}]\!] &::= \lambda t.\lambda t'.\, [\![K_1]\!]\,(t,t') + [\![K_2]\!]\,(t,t')\\
[\![\texttt{ChangePoint(}\theta\texttt{,}\varphi\texttt{,}K_1\texttt{,}K_2\texttt{)}]\!] &::= \lambda t.\lambda t'.\mathbf{let} \quad f \quad = \lambda x.(1+\tanh((\theta-x)/\varphi))/2\\
&\qquad\qquad\qquad u_1 \quad = f(t) \times [\![E_1]\!]\,(t,t') \times f(t')\\
&\qquad\qquad\qquad u_2 \quad = (1-f(t)) \times [\![E_2]\!]\,(t,t') \times (1-f(t'))\\
&\qquad\qquad\quad \mathbf{in} \quad u_1 + u_2
\end{aligned}
$$

Listing 2.2: Semantics of covariance expressions K in the time series structure DSL from Listing 2.1.

Table 2.1: Description of parameters in the time series structure DSL from Listing 2.1.

K	Parameter	Description
`Constant`	θ	Variance of constant process around 0
`Linear`	θ	Time intercept of linear process, i.e., $X(\theta) = 0$ with probability 1
`Smooth`	φ	Length scale of stationary smooth process
`Periodic`	φ_1	Length scale of periodic process
	φ_2	Frequency of periodic process
`ChangePoint`	θ	Time location of changepoint
	φ	How rapidly change occurs



(a) $K =$ `Linear(1)`

(b) $K =$ `Smooth(3.4)`

(c) $K =$ `Periodic(60, 3) * Linear(0.1)`

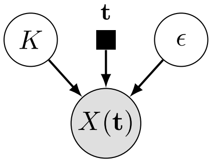
(d) $K =$ `ChangePoint(5, 0.001,Constant(2), Linear(6))`

Figure 2.2: Samples of Gaussian process time series for various covariance expressions K and noise ϵ.

(I1) Sample a noise level $\epsilon \sim P(\epsilon)$.

(I2) Sample a covariance expression $K \sim P(K)$.

(I3) Sample time series data $X(\mathbf{t}) \sim \text{NoisyGP}(0, [\![K]\!], \epsilon)$.



The graphical model representation of the joint distribution $P(\epsilon, K, X(\mathbf{t}))$ is shown above, where latent variables are white nodes, observed variables are shaded nodes, and fixed inputs are solid squares. Figure 2.3 shows the combinatorial space of expressions K and Figure 2.4 shows a broad prior distribution $P(K)$ over the space of expressions. This prior encodes a state of ignorance about the time series structure while penalizing large, complex expressions that can overfit the data. If ϵ and K are fixed, then probability density of a realization $\{X(\mathbf{t}) = x(\mathbf{t})\}$ follows Eq. (2.4):

$$P(x(\mathbf{t}) \mid K, \epsilon) = \exp\left(-\frac{1}{2}\left[x(\mathbf{t})^\top W^{-1} x(\mathbf{t}) - \log\left(\det(W)\right) - n \log\left(2\pi\right)\right]\right), \tag{2.8}$$

where

$$W ::= \begin{bmatrix} [\![K]\!](t_1, t_1) + \epsilon & \dots & [\![K]\!](t_1, t_n) \\ \vdots & \ddots & \vdots \\ [\![K]\!](t_n, t_1) & \dots & [\![K]\!](t_n, t_n) + \epsilon \end{bmatrix} \tag{2.9}$$

is the $n \times n$ covariance matrix encoded by (ϵ, K). As these terms are not known, the Bayes rule is used to update their prior probabilities to posterior probabilities conditioned on the observation:

$$P(\epsilon, K \mid x(\mathbf{t})) ::= \frac{P(\epsilon, K, x(\mathbf{t}))}{P(x(\mathbf{t}))} = \frac{P(x(\mathbf{t}) \mid K, \epsilon) P(K) P(\epsilon)}{P(x(\mathbf{t}))}. \tag{2.10}$$

The denominator of Eq. (2.10) is known as the *marginal likelihood* of $x(\mathbf{t})$, which is obtained by integrating out (K, ϵ) over the joint distribution,

$$P(x(\mathbf{t})) ::= \int P(\epsilon, K, x(\mathbf{t}))\, \mathrm{d}\epsilon\, \mathrm{d}K. \tag{2.11}$$

Problem 2.4 (Informal)**.** *Given a time series data stream $x(\mathbf{t}) = [x(t_1), \dots, x(t_n)]$, generate a set*

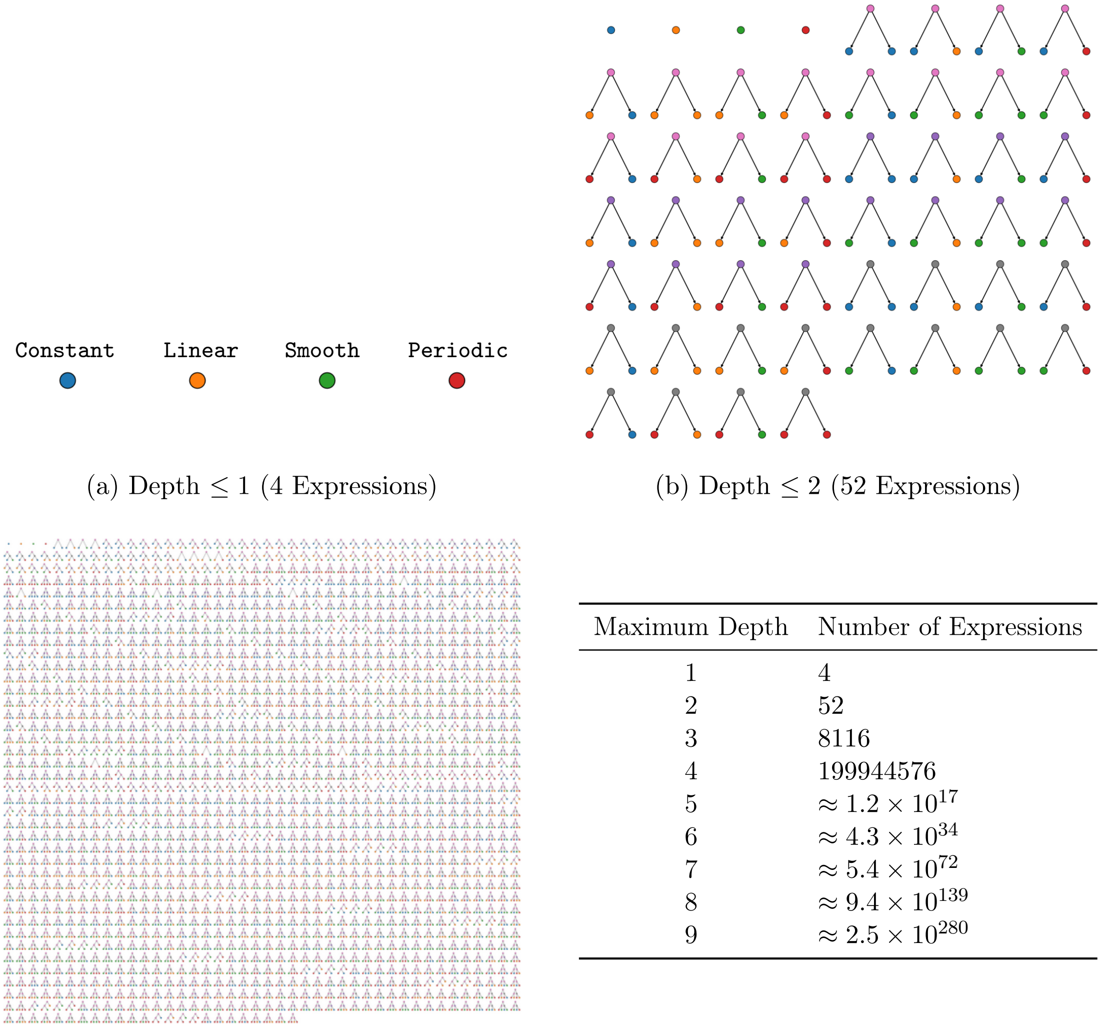
$$\{(\epsilon_i^{(j)}, K_i^{(j)}) \mid i = 1, \dots, M\} \sim^{\text{i.i.d.}} P(\epsilon, K \mid x(\mathbf{t}_{1:j})) \tag{2.12}$$

of $M \geq 1$ posterior samples of (ϵ, K) given observations at $\mathbf{t}_{1:j} ::= [t_1, \dots, t_j]$, for each $j = 1, \dots, n$. «

Problem 2.4 is informal, as there are several technical details that need to be specified for the setup to be well defined, which include:

- How can a valid probability distribution $P(K)$ over covariance expressions be constructed?
- In what sense does $P(\epsilon, K \mid x(\mathbf{t}))$ in Eq. (2.10) induce a probability distribution over (ϵ, K)?
- How is the integral over (ϵ, K) in Eq. (2.11) defined?
- Assuming the integral in Eq. (2.11) is well defined, under what conditions is it a finite number?

All these questions will be answered in Chapter 3, which presents a formal description of Bayesian inference over probabilistic domain-specific data modeling languages, establishes sufficient conditions for the problem to be well defined, and derives sound approximation algorithms for sampling from the sequence of posteriors (2.12). For now, it is assumed that the model (I1)–(I3) defines a valid prior distribution over $(\epsilon, K, X(\mathbf{t}))$ and, conditioned on an observation $\{X(\mathbf{t}) = x(\mathbf{t})\}$, induces a well-defined posterior distribution (2.10) over (ϵ, K) from which it is possible to draw (approximate) samples.



(a) Depth ≤ 1 (4 Expressions)

(b) Depth ≤ 2 (52 Expressions)

(c) Depth ≤ 3 (8116 Expressions)

Maximum Depth	Number of Expressions
1	4
2	52
3	8116
4	199944576
5	$\approx 1.2 \times 10^{17}$
6	$\approx 4.3 \times 10^{34}$
7	$\approx 5.4 \times 10^{72}$
8	$\approx 9.4 \times 10^{139}$
9	$\approx 2.5 \times 10^{280}$

Figure 2.3: Number of covariance expressions, excluding numeric parameters, by depth of parse tree.

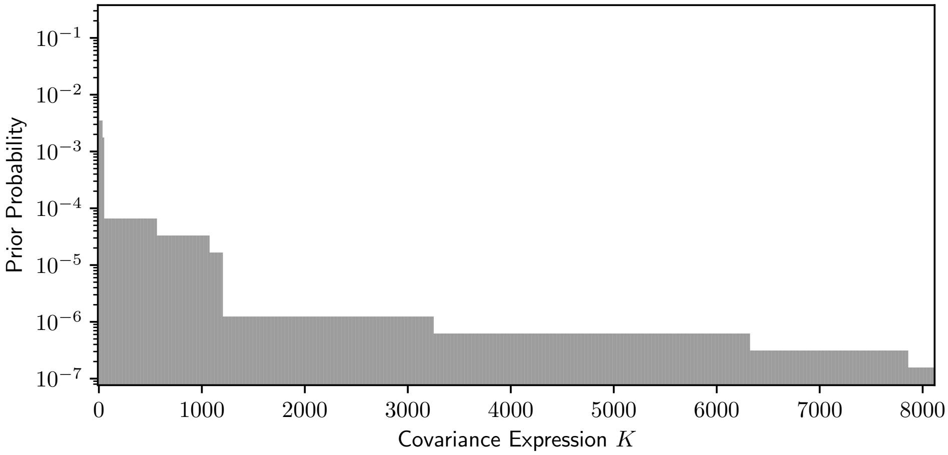
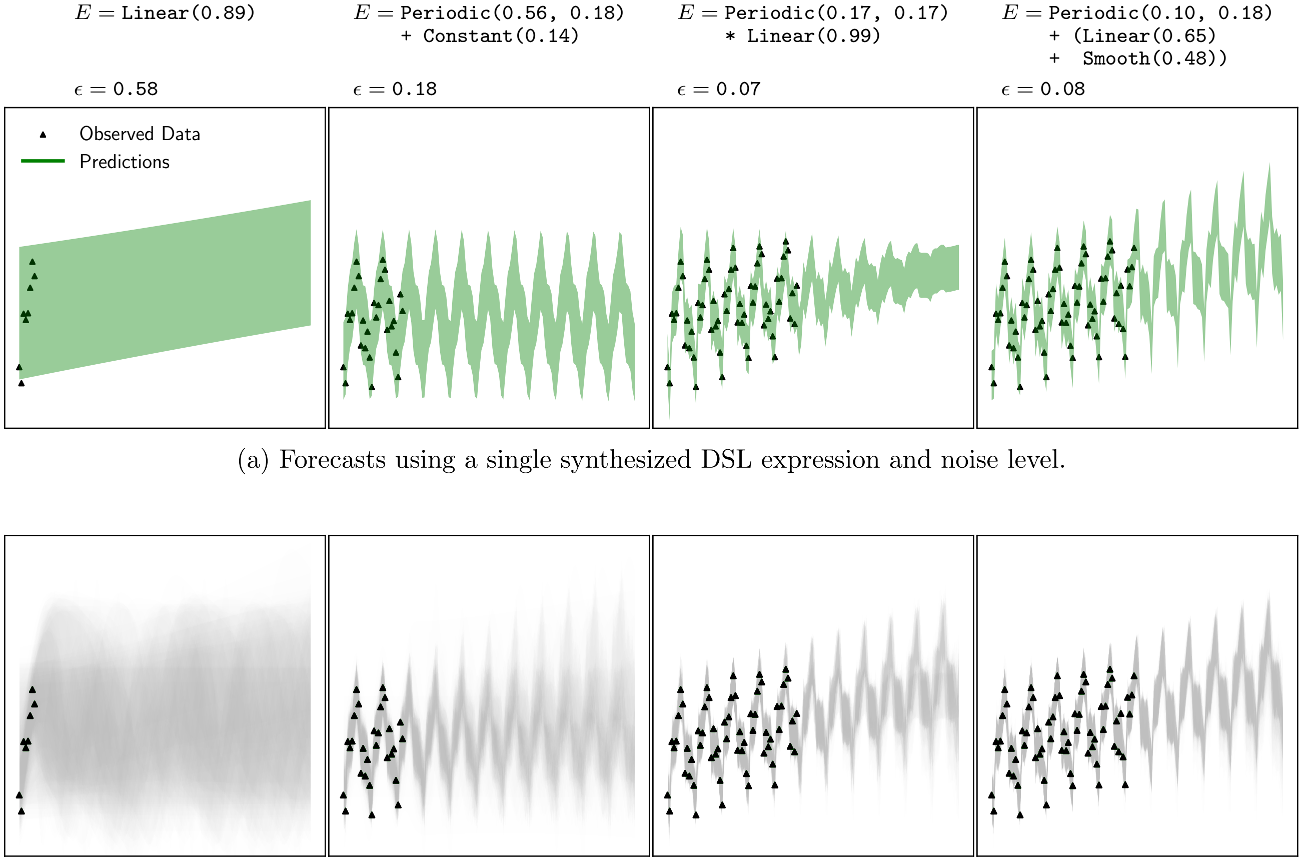


Figure 2.4: Broad prior distribution over covariance expressions. The horizontal axis shows the first 8116 expressions in the DSL from Figures 2.3a–2.3c, ordered by decreasing probability.



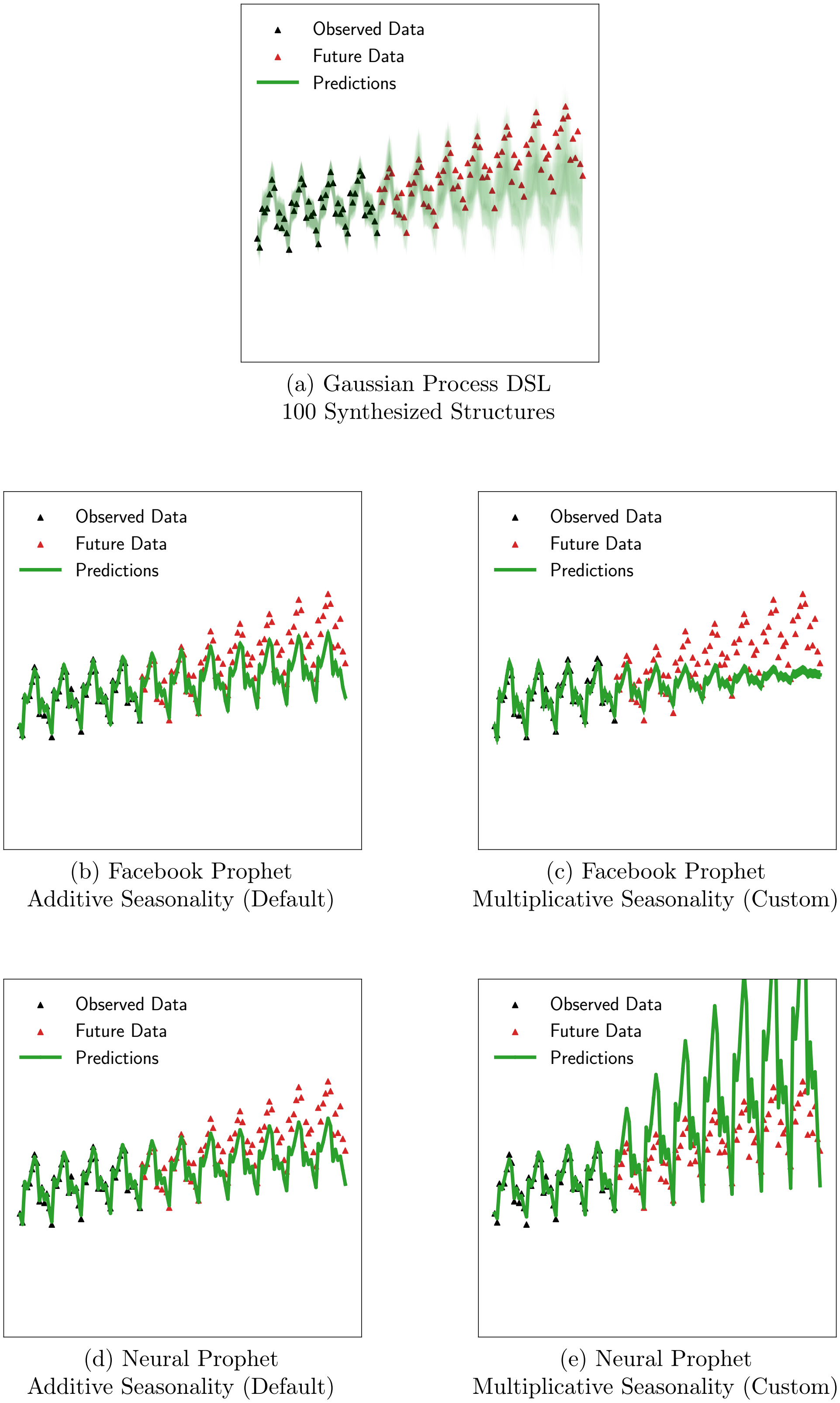
(a) Forecasts using a single synthesized DSL expression and noise level.

(b) Forecasts using a stochastic ensemble of 100 synthesized DSL expressions and noise levels.

Figure 2.5: Online time series structure learning and forecasting for airline passenger data from Figure 2.1. The green regions show 95% prediction intervals.

2.3.2 Online Structure Learning for Airline Data

Learning a Single Structure Figure 2.5a shows posterior samples of the covariance expression K and noise level ϵ (using $M = 1$ in Problem 2.4) for the airline data in Figure 2.1 given data up to times $\mathbf{t}_{1:j}$ ($j = 8, 28, 60, 66$). The green regions represent the 95% prediction interval around the posterior mean (2.6) of the predictive distribution (2.5) at a set of probe points $\mathbf{t}'$. The samples of (ϵ, K) are obtained using the sequential Monte Carlo learning algorithm described in Section 3.3, with a single particle. In the first panel, which includes only eight data points, the data is explained using a linear covariance and large noise level. In the second panel, after a full period is observed, a periodic signal is detected along with a constant offset. In the third panel, the upward trend is modeled using a product of a periodic and linear function, however the amplitude is incorrectly inferred to decrease over time. In the fourth panel, which includes an six additional data points, the inferred covariance is a sum of periodic, linear, and smooth components, which together produce predictions that accurately reflect the patterns in the data. The plots in Figure 2.5a reflect a shortcoming with inferring only a single covariance expression and noise level: while the predictions in the first three panels are plausible given the data observed so far, they do not reflect the inherent inferential uncertainty over (ϵ, K) which leads to inaccurate forecasts. The interpretable surface syntax of the synthesized model structures makes it particularly easy to understand the reason for these inaccurate predictions.



(a) Gaussian Process DSL
100 Synthesized Structures

(b) Facebook Prophet
Additive Seasonality (Default)

(c) Facebook Prophet
Multiplicative Seasonality (Custom)

(d) Neural Prophet
Additive Seasonality (Default)

(e) Neural Prophet
Multiplicative Seasonality (Custom)

Figure 2.6: Comparison of airline passenger forecasts using (a) an ensemble of 100 synthesized programs in the Gaussian process DSL; (b)–(c) Facebook Prophet; and (d)–(e) Neural Prophet. All methods return results in less than 2 seconds of computation.

Learning Multiple Structures Figure 2.5b shows the same experiment as in Figure 2.5a, except with an ensemble of $M = 100$ posterior samples of noise levels and covariance expressions. The green regions now show an overlay of 100 prediction intervals, one for each (ϵ_i, K_i) in the ensemble. In the first panel, there is broad uncertainty which includes a mix of periodic, linear, and smooth structures, as opposed to the single linear structure in the corresponding panel from Figure 2.5a. In the second panel, the periodic structure is captured as before, except that there are now structures in the ensemble that also hypothesize an increase in amplitude over time. The ensembles in the third and forth panels contain similar hypotheses. However, the predictions in the fourth panel are less noisy as the six additional data points are consistent with the inferred structure so far, thereby reducing the uncertainty in the later forecasts. By synthesizing an ensemble of $M = 100$ DSL expressions in Figure 2.5b, the system maintains a collection of plausible explanations for the data that together deliver more robust predictions as compared to the $M = 1$ case in Figure 2.5a.

Comparison to Time Series Baselines To assess the quality of the forecasts from the proposed technique, predictions were compared to those obtained from two machine learning baselines: Facebook Prophet [Taylor and Letham, 2018] and Neural Prophet [Triebe et al., 2021]. Facebook Prophet is a widely used statistical forecasting package that formulates time series modeling as a curve fitting problem by using three components that depend only on time: trend, seasonality, and holiday effects. Neural Prophet extends Facebook Prophet with deep learning methods for modeling autoregressive effects and exogenous covariates, and is shown in Triebe et al. [2021] to outperform Facebook Prophet across many datasets. As with the proposed method for time series structure discovery, the two Prophet baselines are designed to automatically learn models for a wide range of time series patterns.

Figure 2.6 shows a comparison of the forecasts on the airline data given the first $j = 50$ observations. Figure 2.6a shows forecasts obtained from an ensemble of 100 synthesized expressions in the Gaussian process DSL. The held-out future data (red dots) lie cleanly within the 95% forecast intervals. In contrast, both Facebook Prophet in Figure 2.6b and Neural Prophet in Figure 2.6d model the data using additive seasonality by default, which leads to a constant amplitude for the periodic component and forecasts that are too low and insufficiently uncertain to cover the true data. Figures 2.6c and 2.6e show forecasts when manually specifying multiplicative seasonality, which means the amplitude of the periodic component is no longer constant. However, Facebook Prophet in Figure 2.6c infers a multiplicative factor that is too small and Neural Prophet in Figure 2.6e infers a multiplicative factor that is too large, which lead to forecasts that are too small and too large, respectively, and have poorly calibrated uncertainties. In all four cases (Figures 2.6b–2.6e) the baselines fail to generate accurate forecasts, as they only specify a single time series structure and parameter setting, whereas the Bayesian ensemble of 100 DSL expressions in Figure 2.6a reflects posterior uncertainty in the time series structure, parameters, and forecasts. While these comparisons focus on the automated modeling capabilities of Facebook Prophet and Neural Prophet, these systems also support an "analyst-in-the-loop" workflow that lets users further customize the learned structure for more accuracy, at the expense of the modeling expertise and iterative testing that are needed by the user to make the appropriate modeling changes.

2.3.3 Online Adaption to Extreme Novelty

The model forecasts in Figure 2.5 indicate that the inferred linear trend with seasonal variations in the airline data will repeat indefinitely. While this hypothesis is plausible based on previously observed airline data, many real-world econometric time series are influenced by unpredictable external shocks that cause a sudden change in the underlying structure [Olaberria, 2010]. Consider Figure 2.7, which shows the same airline passenger volume as in Figure 2.1 over a longer time horizon that now includes a crash in demand on March 2020 caused by the onset of the global COVID-19 pandemic. In absence of broader information such as macroeconomic indicators, text analysis of news articles, or public

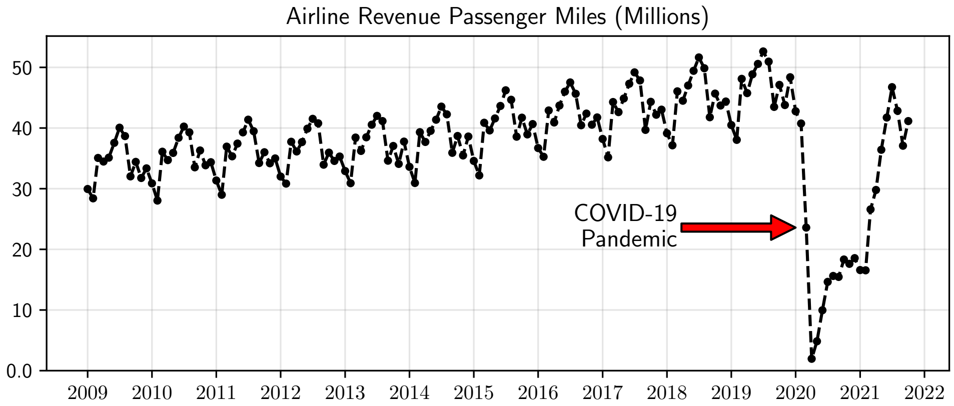
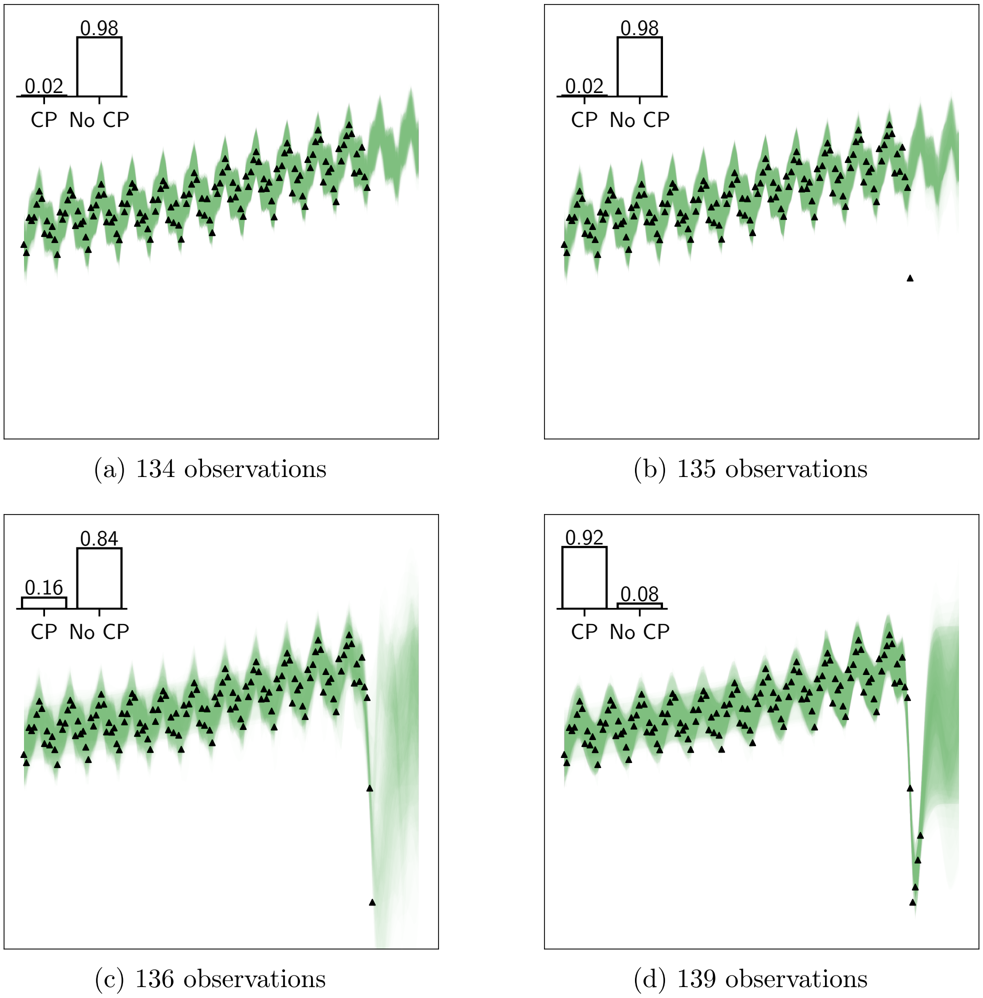


Figure 2.7: Monthly airline revenue passenger miles in the United States from Jan 2009 to Oct 2021.



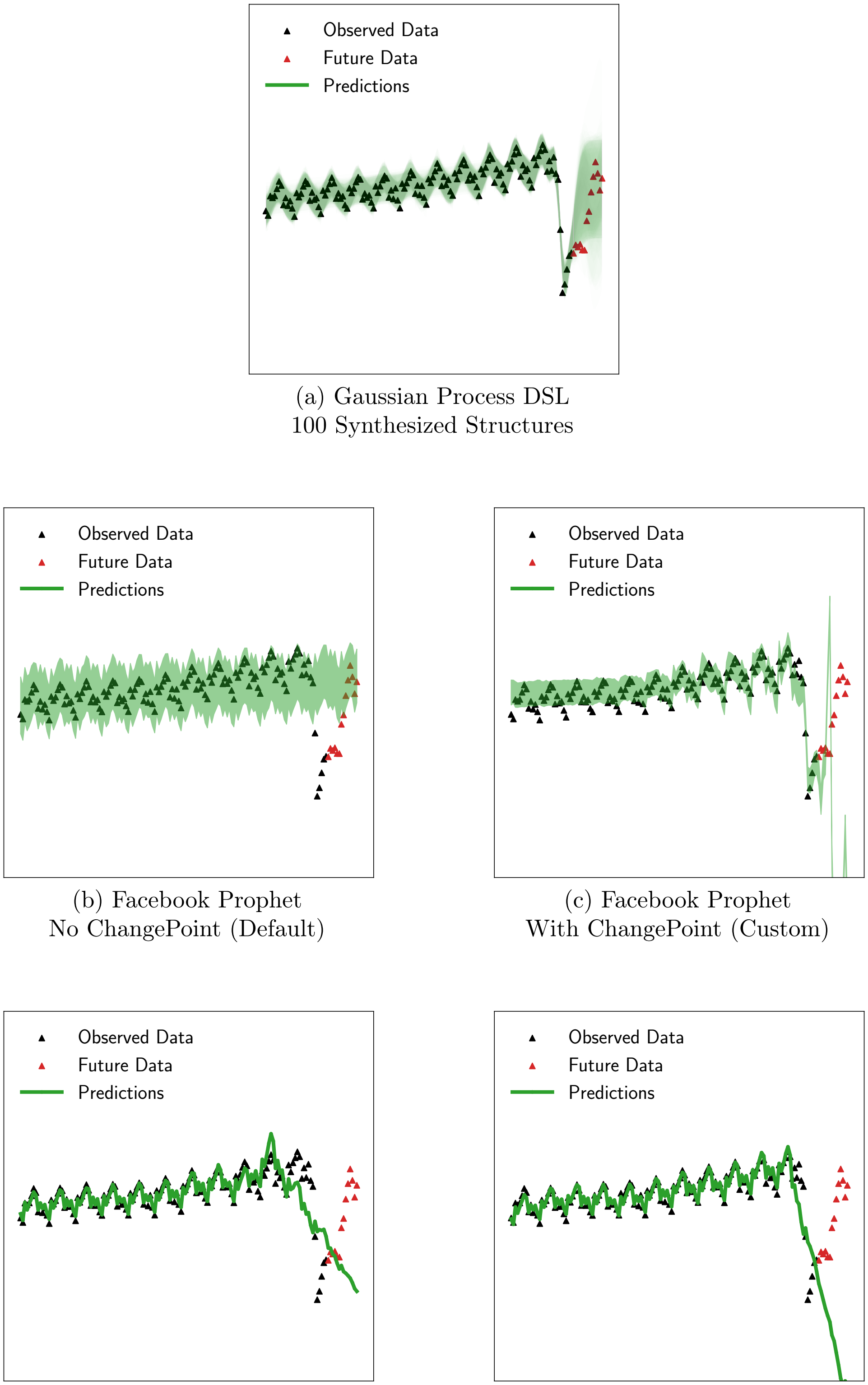
(a) 134 observations

(b) 135 observations

(c) 136 observations

(d) 139 observations

Figure 2.8: Online time series structure learning and forecasting for airline passenger data from Figure 2.7, which includes the crash in demand due to the global COVID-19 pandemic. In each panel, the bar chart in the top left shows the approximate posterior probability of a changepoint (CP), computed across the ensemble of 100 synthesized DSL expressions.



(a) Gaussian Process DSL
100 Synthesized Structures

(b) Facebook Prophet
No ChangePoint (Default)

(c) Facebook Prophet
With ChangePoint (Custom)

(d) Neural Prophet
No ChangePoint (Default)

(e) Neural Prophet
With ChangePoint (Custom)

Figure 2.9: Comparison of airline passenger forecasts with novelty using (a) an ensemble of 100 synthesized programs in the Gaussian process DSL; (b)–(c) Facebook Prophet; and (d)–(e) Neural Prophet.

health statistics, in the lead up to March 2020, a crash of this severity is impossible to predict from historical airline data alone. However, online structure learning can dynamically adapt the expressions $\{(\epsilon_i^{(j)}, K_i^{(j)})\}_{i=1}^{M}$ in the ensemble as novel data is encountered at times $j \geq$ March 2020. Using the Gaussian process DSL in Listing 2.1, the crash can be modeled using a `ChangePoint(`θ`, `φ`, `K_1`, `K_2`)` covariance expression, where θ is the time location of the crash, φ dictates how rapidly the change occurs, and K_1 and K_2 are the covariance structures before and after the crash, respectively.

Figure 2.8a shows the model forecasts given data up to February 2020, where the approximate posterior probability of a changepoint is 2% (i.e., two of the 100 synthesized DSL expressions contain a `ChangePoint` subexpression). The locations of the changepoint in these two expressions both lie beyond the latest observed data point, representing a small probability of future novelty. Figure 2.8b shows the first novel data point, which causes no change in the changepoint probabilities but a higher noise level in the forecasts as compared to Figure 2.8a. Figure 2.8c contains a novel data point that lies far in the tails of the predictive distribution, which causes the changepoint probability to rise to 16% and noisier forecasts in both the pre-crash and post-crash regimes. Figure 2.8d shows the forecasts after observing five novel data points, where the changepoint probability is now 92% and the prediction intervals form a broad band over the range of previously observed values.

Figure 2.9 compares these forecasts to those obtained from the Facebook Prophet and Neural Prophet baselines, which both support automatic changepoint detection and piecewise time series models. Figure 2.9a shows the results using the ensemble of 100 synthesized Gaussian process models, where the held-out data again lies well within the prediction interval. Figures 2.9b and 2.9d show forecasts from Facebook and Neural Prophet, which both fail to automatically detect the March 2020 changepoint and produce highly inaccurate forecasts. Figures 2.9c and 2.9e show forecasts from the Prophet baselines when a changepoint has been manually specified. Despite encoding the correct structure, both baselines incorrectly predict that the airline demand will spiral towards zero. It should also be noted that the Prophet baselines cannot handle streaming data and must be retrained from scratch each time a new observation is available, whereas online Bayesian structure learning leverages sequential Monte Carlo inference (Section 3.3.2) for incorporating new observations as they become available in real-time.

2.4 Evaluation

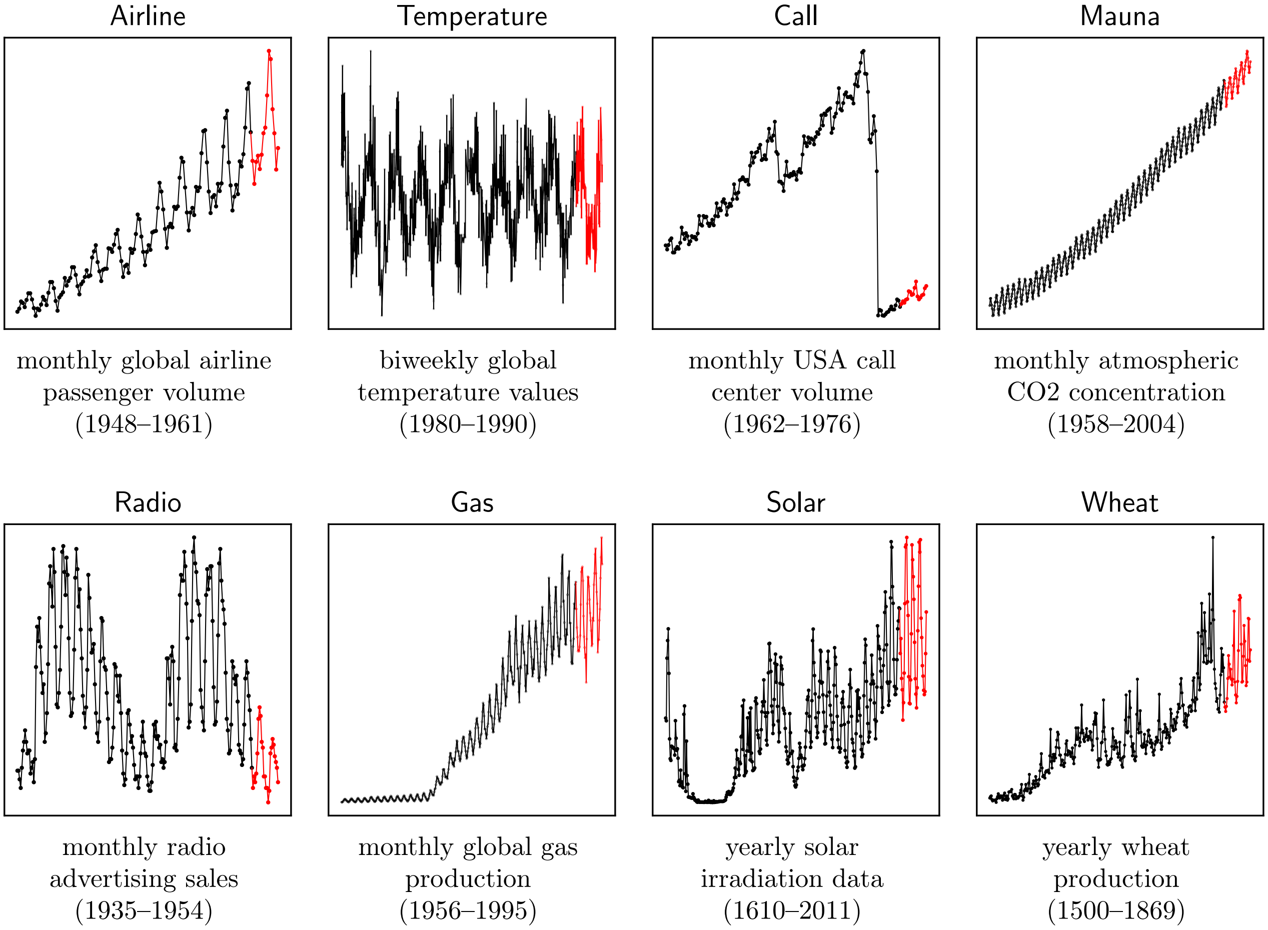
Bayesian synthesis in the Gaussian process time series DSL was evaluated on a benchmark set of real-world econometric time series that reflect a range temporal structures. Figure 2.10 shows a plot and description of each of the eight time series in the benchmark set, which are adapted from the online repository of Lloyd [2014]. Section 2.4.1 compares forecasting accuracy on the held-out data in Figure 2.10 to multiple baselines and shows that Bayesian synthesis produces time series models with more accurate forecasts in six out of the eight datasets. Section 2.4.2 verifies the plausibility of the posterior inferences about the presence of absence of various temporal structures such linearity, periodicity, and changepoints. Section 2.4.3 presents runtime versus accuracy measurements using the Markov chain Monte Carlo synthesis algorithm described in Section 3.3.1 and resample-move Sequential Monte Carlo algorithm described in Section 3.3.2, where the latter algorithm delivers up to two orders of magnitude improvements in scalability. All experiments were implemented in the Gen probabilistic programming system [Cusumano-Towner et al., 2019].

2.4.1 Prediction Accuracy

Table 2.2 shows a comparison of the standardized root mean squared forecasting error of held-out data for the eight benchmark time series and five baselines methods for time series forecasting. Baselines that meet three criteria were selected: (i) they have open source and reusable implementations; (ii) they are widely cited in the literature; and (iii) they do not require significant manual tuning of structure or

Table 2.2: Standardized root mean squared forecasting error of held-out data for benchmark time series.

	Airline	Temperature	Call	Mauna	Radio	Gas	Solar	Wheat
Bayesian Synthesis	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	**1.0**	1.47	1.50
Gaussian Process (Squared Exponential)	2.01	1.70	4.26	1.54	2.03	1.45	1.63	1.37
Auto-Regressive Integrated Moving Average	1.32	1.85	2.44	1.09	2.08	2.00	**1.0**	1.41
Facebook Prophet	1.83	2.00	5.61	1.23	3.09	1.69	1.73	1.29
Hierarchical-DP Hidden Markov Model	4.61	1.77	2.26	14.77	1.19	—	3.49	1.89
Linear Regression	1.79	1.30	6.23	2.19	2.73	1.33	1.57	**1.0**



monthly global airline passenger volume (1948–1961)

biweekly global temperature values (1980–1990)

monthly USA call center volume (1962–1976)

monthly atmospheric CO2 concentration (1958–2004)

monthly radio advertising sales (1935–1954)

monthly global gas production (1956–1995)

yearly solar irradiation data (1610–2011)

yearly wheat production (1500–1869)

Figure 2.10: Eight econometric time series. Observed data is shown in black and held-out data in red.

hyperparameters on a per-dataset basis. Baselines include Gaussian process with a squared exponential covariance function; Facebook Prophet [Taylor and Letham, 2018]; autoregressive integrated moving average (ARIMA) [Hyndman and Khandakar, 2008]; the hierarchical Dirichlet process hidden semi-Markov Model (HDP-HSMM) [Johnson and Willsky, 2013]; and simple linear regression. In Table 2.2, bold entries have a statistically significant smallest error. Bayesian synthesis produces more accurate predictions for six of the eight benchmarks and is competitive with other techniques on the Solar and Wheat benchmarks. For these two time series, there is substantial novelty in the held-out data which no method was able to accurately predict from the held-in data alone. Figure 1.3 from Chapter 1 shows example forecasts from all the baselines for the airline data, demonstrating that the synthesized DSL programs most accurately capture the underlying structure in the data.

Table 2.3: Approximate posterior probabilities of various temporal structures in benchmark time series.

	Airline	Temperature	Call	Mauna	Radio	Gas	Solar	Wheat
Linear Structure	95%	16%	93%	98%	6%	85%	35%	65%
Periodic Structure	95%	92%	97%	93%	93%	76%	77%	68%
Changepoint Structure	22%	4%	90%	13%	23%	76%	31%	21%

2.4.2 Extracting Qualitative Structure

Each dataset in Figure 2.10 has different temporal structure: for example, the Temperature data is characterized by a yearly periodic structure and the Call Center data has linear plus periodic structure until 1973 followed by a sharp drop at the changepoint. Table 2.3 shows the posterior inferences about the qualitative structure underlying each dataset, where percentages indicate the fraction of the $M = 100$ synthesized expressions in the ensemble that contain a `Linear`, `Periodic`, or `ChangePoint` subexpression. In the first six time series, the probabilities reflect relatively high certainty about the absence or presence of each structure (probabilities in the range $[0, 25]$ and $[75, 100]$). In contrast, the uncertainty about the structure is highest (probabilities in the range $[25, 75]$) in the Solar and Wheat data, where the structure is less apparent and the forecast errors from Table 2.2 are the highest.

2.4.3 Runtime versus Accuracy

Comparison of MCMC and SMC Figure 2.11 shows profiles of the predictive likelihood on held-out data versus the synthesis runtime (wall clock seconds) for the eight econometric time series from Figure 2.10. Two strategies for generating the approximate posterior samples $\{(\epsilon_i, K_i) \mid i =, \dots, M\} \sim P(\epsilon, K \mid x(\mathbf{t}))$ are compared: Markov chain Monte Carlo (Section 3.3.1) and resample-move sequential Monte Carlo (Section 3.3.1). Since both these strategies are asymptotically exact—i.e., they return arbitrary accurate approximations to the true posterior distribution in the limit of computation—the long-term accuracy should converge to the same number. However, for finite computational budget, SMC delivers 1x–1000x improvement in runtime needed to achieve a given accuracy level. SMC provides the most improvement over MCMC when the underlying structure is apparent from the initial observations in the time series, which makes it possible for SMC to rapidly discover models before observing the entire dataset. In contrast, MCMC recomputes the entire data likelihood at each iteration, which is wasteful in cases where a short prefix of the data is sufficient to infer the structure. Second, the adaptive resampling step in SMC makes it more robust to the local minima that MCMC chains become trapped in for an extended period of time, which are the "flat" portions of the MCMC trajectories in Figure 2.11. Section 3.5.1 provides a complexity analysis for the runtime scaling of these algorithms and discusses sparse Gaussian process approximations for reducing the $O(n^3)$ cost needed to obtain the inverse and determinant of the $n \times n$ covariance matrix in Eq. (2.8).

Speedups via Incremental Computation The measurements in Figure 2.11 can be further optimized by memoizing covariance terms in Listing 2.2 when recomputing the data likelihood (2.8). For example, suppose the current expression $K ::=$ `(`K_1 `*` K_2`)` is a sum of two covariances, and an MCMC or SMC rejuvenation step proposes a new expression $K' ::=$ `(`K_1 `*` K_2'`)`. As K_1 is unchanged and K_2 has changed to K_2', there is a duplicate expression that can be cached:

$$[\![K]\!](t, t') = [\![K_1]\!](t, t') + [\![K_2]\!](t, t'), \qquad [\![K']\!](t, t') = \underbrace{[\![K_1']\!](t, t')}_{\text{cached}} + [\![K_2']\!](t, t'). \tag{2.13}$$

That is, the matrix $[\![K_1]\!](t_i, t_j)$ (for each $1 \le i \le j \le n$) need not be recomputed when computing the new covariance matrix $[\![K_1]\!]$. The Gen probabilistic programming language used to implement the exper-

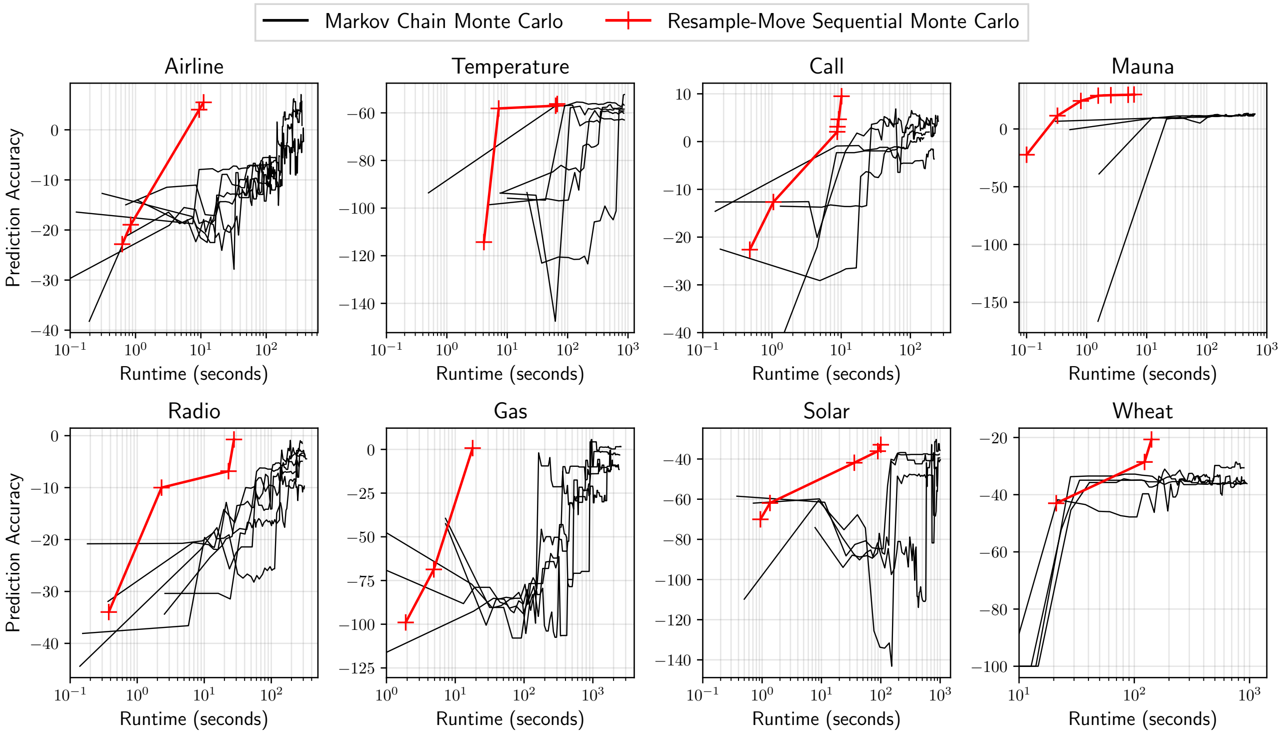


Figure 2.11: Runtime (x-axis, logarithmic scale) versus prediction accuracy (y-axis) measured as the predictive likelihood of held-out data for the econometric time series in Figure 2.10. The black lines show measurements from five independent runs of Markov Chain Monte Carlo (Algorithm 3.1) with 1500 iterations, and the red lines show measurements from independent runs of resample-move sequential Monte Carlo (Algorithm 3.2) whose runtime is a function of the number of particles, number of rejuvenation steps, and batch size.

iments in this section provides affordances for automatically caching recursive computations [Cusumano-Towner, 2020, Section 5.3]. Table 2.4 shows the lines of code and runtime of one MCMC step for the airline data in Figure 2.10 as originally reported in Cusumano-Towner et al. [2019], using four different implementations implementations: Gen with caching in the static modeling language; a native implementation in the Julia programming language [Bezanson et al., 2017]; Gen without caching in the dynamic modeling language; and the Venture probabilistic programming language [Schaechtle et al., 2017, Mansinghka et al., 2018]. Automatic caching in Gen delivers around 1.8x speedup over native Julia (which does not implement automatic caching), 2.5x speedup over Gen without caching, and 108x speedup over Venture. This improvement, however, comes at a ∼70% increase in the lines of code as compared to the Gen implementation without caching, which reflects the common trade-off between performance optimization and implementation complexity.

Table 2.4: Performance comparison of one MCMC step over covariance expression K.

	Lines of Code	Caching	Runtime (ms per step)
Gen Static Modeling Language	174	Yes	2.57 (±0.09)
Julia Native Implementation	193	No	4.73 (±0.45)
Gen Dynamic Modeling Language	102	No	6.21 (±0.94)
Venture	70	No	279 (±31)

Chapter 3

Synthesizing Probabilistic Programs in Domain-Specific Modeling Languages

> In the earliest days, symbolic manipulation and abstract languages for knowledge representation and reasoning were considered the heart of intelligence. And I think in many ways they are still the best idea that anybody has ever had about how intelligence works in computational terms.
>
> Joshua B. Tenenbaum

Chapter 2 showed an informal tutorial of using Bayesian inference to synthesize probabilistic programs in specialized data modeling languages for time series, which help automate the process of learning interpretable and accurate models from data. This chapter formally describes the general "Bayesian synthesis" framework illustrated in Figure 1.2 from Chapter 1. Section 3.1 introduces probabilistic domain-specific languages for representing families of generative models. Each expression in the DSL is assigned two denotational semantics: one for the prior probability distribution of the expression and another for probability distribution over datasets that it induces. Section 3.2 formalizes the problem of Bayesian synthesis and identifies sufficient conditions needed for this problem to well defined. Section 3.3 describes a class of Markov chain Monte Carlo (MCMC) and sequential Monte Carlo (SMC) algorithm templates that deliver sound approximate solutions to the Bayesian synthesis problem. Section 3.4 introduces a family of probabilistic DSLs that are generated by context-free grammars, outlines a generic algorithm that implements both the MCMC and SMC algorithm templates for this DSL family, and proves that it satisfies the preconditions for sound inference, i.e., the solutions converge asymptotically to the posterior distribution over DSL expressions given the data. Section 3.5 revisits the univariate time series DSL from Chapter 2, verifying that it meets the conditions needed for sound synthesis. Section 3.6 discusses related work.

3.1 Probabilistic Domain-Specific Modeling Languages

Definition 3.1. A *domain-specific data modeling language* is comprised of:

- A countable set S called the *structure space*.
- An indexed family $\{\Theta_s, s \in A\}$ of sets, called the *parameter spaces*.
- A set T called the *input space*.
- An indexed family $\{X_t, t \in T\}$ of sets, called the *data spaces*. «

Definition 3.2. The set of *expressions* of a domain-specific data modeling language D is

$$\mathcal{L}(D) ::= \{(s, \theta) \mid s \in S, \theta \in \Theta_s\}. \tag{3.1}$$

For any expression $E ::= (s, \theta) \in \mathcal{L}(D)$, the symbol s denotes the *structure* of E and θ denotes the *parameter* of E. When D is clear from context, $\mathcal{L}(D)$ is written as $\mathcal{L}$. «

Definition 3.3. The set of *observations* of a domain-specific data modeling language D is

$$\mathcal{X}(D) ::= \{(t, x) \mid t \in T, x \in X_t\}. \tag{3.2}$$

For any observation $O ::= (t, x) \in \mathcal{X}(D)$, the symbol t denotes the *input* of O and x denotes the *output* of O. When D is clear from context, $\mathcal{X}(D)$ is written as $\mathcal{X}$. «

Definition 3.4. A *probabilistic domain-specific data modeling language*, or simply a probabilistic DSL, is a domain-specific data modeling language equipped with:

- For each $s \in S$, a sigma-algebra $\mathcal{F}_s$ over Θ_s and a sigma-finite base measure λ_s over $(\Theta_s, \mathcal{F}_s)$.
- For each $t \in T$, a sigma-algebra $\mathcal{F}_t$ over X_t and a sigma-finite base measure λ_t over $(X_t, \mathcal{F}_t)$.
- A semantic function Prior : $\mathcal{L} \to \mathbb{R}_{\geq 0}$.
- A semantic function Likelihood : $\mathcal{L} \to \mathcal{X} \to \mathbb{R}_{\geq 0}$. «

Definition 3.5. Following Fremlin [2009, 214L], for a probabilistic DSL D, the "disjoint-sum" sigma-algebra $\Sigma_{\mathcal{L}}$ over $\mathcal{L}$ has events of the form $\{(s, \theta) \mid s \in A, \theta \in A_s\}$ for some $A \subset S$ and $A_s \in \mathcal{F}_s$ for each $s \in A$. A generic event in $\Sigma_{\mathcal{L}}$ is written $(A, \{A_s, s \in A\})$ for short. «

To be well defined, the semantic functions in Definition 3.4 must satisfy four technical conditions.

Condition 3.6 (Normalized Prior)**.** The Prior semantic function induces a probability distribution over $(\mathcal{L}, \Sigma_{\mathcal{L}})$, in the sense that

$$\sum_{s \in S} \left[\int_{\theta \in \Theta_s} \text{Prior} \, [\![(s, \theta)]\!] \, \lambda_s(\mathrm{d}\theta) \right] = 1. \tag{3.3}$$

«

Condition 3.7 (Normalized Likelihood)**.** For each $t \in T$ and $E \in \mathcal{L}$, the function defined by the rule $x \mapsto \text{Likelihood} \, [\![E]\!] \, (t, x)$ induces a probability distribution over $(X_t, \mathcal{F}_t)$, in the sense that

$$\int_{x \in X_t} \text{Likelihood} \, [\![E]\!] \, (t, x) \; \lambda_t(\mathrm{d}x) = 1. \tag{3.4}$$

«

Condition 3.8 (Positive Likelihood)**.** For each $t \in T$ and $x \in X$, there exists a Prior positive measure set $A \subset \mathcal{L}$ such that for all $E \in A$, Likelihood $[\![E]\!] \, (t, x) > 0$. «

Condition 3.9 (Bounded Likelihood)**.** For each $t \in T$ and $x \in X$, the function defined by the rule $E \mapsto \text{Likelihood} \, [\![E]\!] \, (t, x)$ is measurable and essentially bounded, i.e., there exists a finite positive constant $c_{tx}^{\max}$ such that the set of expressions $\{E \in \mathcal{L} \mid \text{Likelihood} \, [\![E]\!] \, (t, x) > c_{tx}^{\max}\}$ is a subset of a Prior measure zero set. «

Remark 3.10. If Condition 3.6 holds, then the Prior probability of event $(A, \{A_s, s \in A\}) \in \Sigma_{\mathcal{L}}$ is

$$\sum_{s \in A} \left[\int_{\theta \in A_s} \text{Prior} \, [\![(s, \theta)]\!] \, \lambda_s(\mathrm{d}\theta) \right]. \tag{3.5}$$

«

The following proposition is a straightforward consequence of the fact that the expectation of a positive essentially bounded random variable lies between zero and its maximum value.

Proposition 3.11. *If Conditions 3.6–3.9 hold, then for each $t \in T$ and $x \in X$, the marginal likelihood c_{tx} is positive and finite, that is*

$$0 < c_{tx} ::= . \sum_{s \in S} \left[\int_{\theta \in \Theta_s} \text{Likelihood} \left[\!\left[(s, \theta) \right]\!\right] (t, x) \cdot \text{Prior} \left[\!\left[(s, \theta) \right]\!\right] \; \lambda_s(\mathrm{d}\theta) \right] < c_{tx}^{\max} < \infty. \tag{3.6}$$

«

When the semantic functions satisfy the above conditions, they induce a new semantic function Post : $\mathcal{L} \to \mathcal{X} \to \mathbb{R}_{\geq 0}$, called the posterior:

$$\text{Post} \left[\!\left[E \right]\!\right] (t, x) ::= \text{Likelihood} \left[\!\left[E \right]\!\right] (t, x) \cdot \text{Prior} \left[\!\left[E \right]\!\right] / c_{tx}. \tag{3.7}$$

Integrating Eq. (3.7) over $E = (s, \theta) \in \mathcal{L}$ establishes the following proposition.

Proposition 3.12. *If Conditions 3.6–3.9 hold, then for any observation $(t, x) \in \mathcal{X}$, the function defined by the rule $E \mapsto$ Post $\left[\!\left[E \right]\!\right] (t, x)$ induces a probability distribution over $(\mathcal{L}, \Sigma_{\mathcal{L}})$, such that the probability of any event $(A, \{A_s, s \in A\}) \in \Sigma_{\mathcal{L}}$ is given by*

$$\sum_{s \in A} \left[\int_{\theta \in A_s} \text{Post} \left[\!\left[(s, \theta) \right]\!\right] (t, x) \lambda_s(\mathrm{d}\theta) \right]. \tag{3.8}$$

«

3.2 Bayesian Synthesis in Probabilistic DSLs

The Bayesian synthesis problem can now be stated.

Problem 3.13 (Bayesian Synthesis). *Let D be a probabilistic domain-specific modeling language as in Definition 3.4, whose* Prior *and* Likelihood *semantics satisfy Conditions 3.6–3.9. Given an observation $O \in \mathcal{X}(D)$, generate samples of expressions $E \in \mathcal{L}$ from the posterior distribution over $(\mathcal{L}, \Sigma_{\mathcal{L}})$ defined by* Post $\left[\!\left[\cdot \right]\!\right] (O)$ *in Eq.* (3.7). «

Figure 3.1 shows the workflow of Bayesian synthesis, which is next described in more detail.

3.2.1 Bayesian Synthesis via Sound Approximate Inference

In Figure 3.1, the input to Bayesian synthesis is a probabilistic DSL D and observation $O \in \mathcal{X}$. The goal is to synthesize an ensemble $\{E_1, \dots, E_M\}$ of $M > 0$ DSL programs from the posterior Eq. (3.8), which collectively represent a set of likely structures and parameters that observation O. Because generating exact posterior samples is intractable in most settings, approximate sampling is used instead. The approximate sampling techniques described in Section 3.3 are *sound* in the sense that they produce DSL programs from a distribution that becomes arbitrarily close to the target distribution (3.8) given enough computational effort.

3.2.2 Querying Synthesized Probabilistic Programs

To discover structure in the observed data and make future predictions, the synthesized DSL programs are analyzed by solving *queries*. A query $\phi : \mathcal{L} \to \mathcal{Q}$ is a mapping from DSL programs to a space $\mathcal{Q}$ that defines the set of allowable results. Recalling that each expression $E = (s, \theta) \in \mathcal{L}$ represents the structure s and parameters θ for a family of statistical models, there are two classes of queries:

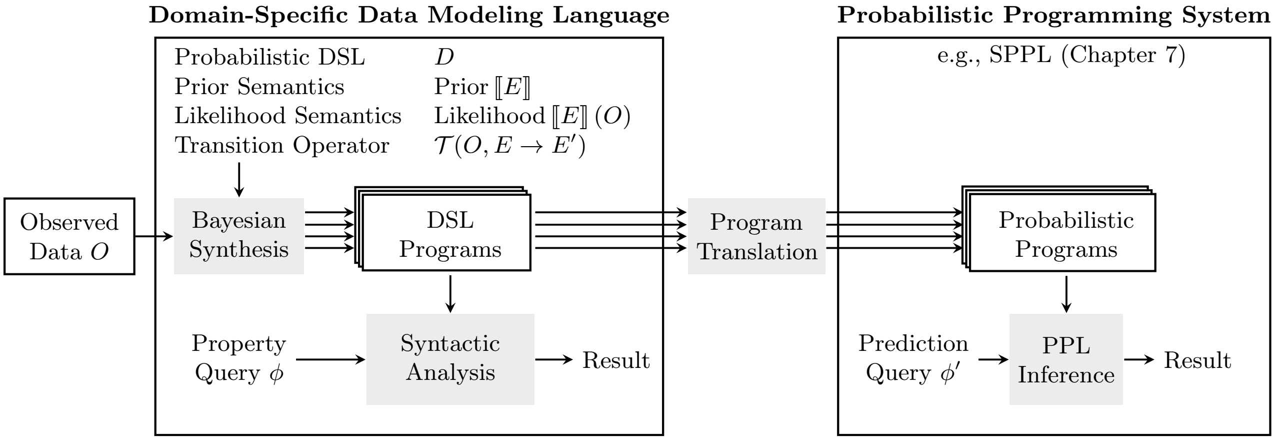


Figure 3.1: Components of Bayesian synthesis of probabilistic programs for automatic data modeling.

(i) *Property Queries.* This class of queries can be solved using simple syntactic analyses on the synthesized DSL expression to extract interpretable patterns. A predicate query has the signature $\phi : \mathcal{L} \to \{0, 1\}$, where $\phi [\![E]\!] = 1$ if E satisfies the predicate and 0 otherwise. Using the ensemble of M synthesized DSL expressions, an approximation to the posterior probability that $\phi [\![E]\!] = 1$ is given by

$$\Pr\{\phi [\![E]\!] = 1 \mid O\} \approx \frac{1}{M} \sum_{i=1}^{M} \phi [\![E_i]\!] . \tag{3.9}$$

The probabilities in Table 2.3 are solutions to property queries about the probability that a given temporal pattern is present.

(ii) *Prediction Queries.* This class of queries includes simulating new data or computing both marginal and conditional probabilities of events involving observations $(t, x) \in \mathcal{X}$. Rather than develop custom solvers for each DSL, prediction queries solved by first applying a syntactic operation Translate : $\mathcal{L} \to \mathcal{L}'$ that converts domain-specific programs $E \in \mathcal{L}$ into probabilistic programs $E' \in \mathcal{L}'$ in a probabilistic programming language. The Sum-Product Probabilistic Language (SPPL) described in Chapter 7 can serve as the unified target language $\mathcal{L}'$ for all probabilistic DSLs in Part I. Chapter 7 formalizes the SPPL modeling syntax and the class of queries that it can solve. Translating DSL expressions into SPPL programs enables substantial reuse of general-purpose inference machinery to automatically obtain exact results to queries, irrespective of the original DSL from which the SPPL program was translated. Using the ensemble of M synthesized DSL expressions, an approximation to the posterior probability that $\phi [\![E]\!] \in Q$ for some subset $Q \subset \mathcal{Q}$ is given by

$$\Pr\{\phi [\![E]\!] \in Q \mid O\} \approx \frac{1}{M} \sum_{i=1}^{M} \mathbf{1}\big[\phi' [\![\text{Translate} [\![E_i]\!]]\!] \in Q\big], \tag{3.10}$$

where ϕ' is the query in SPPL syntax. The 95% prediction intervals in the top row of Figure 2.5 are solutions to a prediction query.

Algorithm 3.1 Markov chain Monte Carlo algorithm for Bayesian synthesis.

```
Require: observation $O ::= (t, x) \in \mathcal{X}$, number of iterations $n \geq 1$
Ensure: sample from $\text{ApproxPost}^{(n)}_{E_0} [\![\cdot]\!] (O)$ defined in Eqs. (3.15)–(3.16), for some $E_0 \in \mathcal{L}$ with
    Likelihood $[\![E_0]\!] (O) > 0$
1: procedure BAYESIAN-SYNTHESIS-MCMC($O, n$)
2:     do
3:         $E_0 \sim$ GENERATE-EXPRESSION-FROM-PRIOR()    ▷ generate $E_0$ with probability Prior $[\![E]\!]$
4:     while EVALUATE-LIKELIHOOD($O, E_0$) $= 0$
5:     for $i = 1 \dots n$ do                              ▷ run $n$ sampling iterations
6:         $E_i \sim$ GENERATE-NEW-EXPRESSION($O, E_{i-1}$)
7:     return $E_1, \dots, E_n$
```

3.3 Algorithms for Bayesian Synthesis

Solving Problem 3.13 exactly is computationally intractable most settings. Sections 3.3.1 and 3.3.2 describe algorithm templates for randomly sampling expressions $E \in \mathcal{L}$ with probability that approximates Post $[\![E]\!] (O)$ using Markov chain Monte Carlo (MCMC) and sequential Monte Carlo (SMC), respectively. These algorithms are sound in the sense that they are guaranteed converge to the posterior distribution in the limit of computation, thereby producing arbitrarily accurate approximations to queries using estimators of the form (3.9) and (3.10). Both algorithm templates require three procedures (P1)–(P3), shown below, for synthesis in a given probabilistic DSL. The procedure (P3), which transitions the current DSL expression, is often the main design challenge in engineering synthesis algorithms. Section 3.4 describes a generic and sound implementation of (P3) that applies to any probabilistic DSL generated by a context-free grammar.

(P1)	GENERATE-EXPRESSION-FROM-PRIOR()	sample $E \in \mathcal{L}$ with probability Prior $[\![E]\!]$
(P2)	EVALUATE-LIKELIHOOD(O, E)	evaluate Likelihood $[\![E]\!] (O)$, for $E \in \mathcal{L}$, $O \in \mathcal{X}$
(P3)	GENERATE-NEW-EXPRESSION(O, E)	sample E' from E with probability $\mathcal{T}(O, E \to E')$

Remark 3.14. Let D be a probabilistic DSL. To avoid excessive measure-theoretic proofs, this section assumes that for each structure $s \in S$, the parameter space $\Theta_s = \{\theta_{s1}, \theta_{s2}, \dots\}$ is countable, the sigma-algebra $\mathcal{F}_s = 2^{\Theta}_s$, and λ_s is the counting measure. As $\mathcal{L}(D)$ is now a countable union of countable sets, it is also countable. Eq. (3.5) is therefore written as a sum over $E \in \mathcal{E}$ for some $\mathcal{E} \subset \mathcal{L}$. «

3.3.1 Bayesian Synthesis via Markov Chain Monte Carlo

Algorithm 3.1 shows the template of an MCMC sampling algorithm for generating approximate solutions to Problem 3.13. In lines 2–4, the algorithm repeatedly calls GENERATE-EXPRESSION-FROM-PRIOR until it obtains an initial expression $E_0 \in \mathcal{L}$ such that Likelihood $[\![E]\!] (O) > 0$. In lines 5–6, the algorithm iteratively invokes GENERATE-NEW-EXPRESSION to produce E_i from E_{i-1} using a *transition operator* $\mathcal{T}$. The transition operator takes as input expression E and observation O and stochastically samples an expression E' with probability denoted $\mathcal{T}(O, E \to E')$, where $\sum_{E' \in \mathcal{L}} \mathcal{T}(O, E \to E') = 1$ for all $E \in \mathcal{L}$. The next three conditions on the transition operator $\mathcal{T}$ are sufficient to prove that Algorithm 3.1 returns expressions $(E_1, \dots, E_n)$ from an arbitrarily close approximation to Post $[\![E]\!] (O)$.

Condition 3.15 (Posterior invariance)**.** If an expression $E \in \mathcal{L}$ is sampled from the posterior distribution and a new expression $E' \in \mathcal{L}$ is sampled with probability $\mathcal{T}(O, E \to E')$, then E' is also a sample from the posterior distribution:

$$\sum_{E \in \mathcal{L}} \text{Post}\,[\![E]\!]\,(O) \cdot \mathcal{T}(O, E \to E') = \text{Post}\,[\![E']\!]\,(O). \tag{3.11}$$

«

Condition 3.16 (Posterior irreducibility)**.** Every expression $E' \in \mathcal{L}$ with nonzero likelihood is reachable from every $E \in \mathcal{L}$ in a finite number of steps. More specifically, for all pairs of expressions (E, E') for which Likelihood $[\![E']\!]\,(O) > 0$ there exists an integer $n \geq 1$ and a sequence of expressions $E_1, E_2, \dots, E_n$ where $E_1 = E$ and $E_n = E'$ such that $\mathcal{T}(O, E_{i-1} \to E_i) > 0$ for all $i \in \{2, \dots, n\}$. «

Condition 3.17 (Aperiodicity)**.** There exists some expression $E \in \mathcal{L}$ such that the transition operator has a nonzero probability of returning to the same expression, i.e., $\mathcal{T}(O, E \to E) > 0$. «

It is next established that Algorithm 3.1 returns asymptotically sound samples whenever these three conditions hold. First, it is necessary to prove that an initial expression $E_0 \in \mathcal{L}$ for which Likelihood $[\![E_0]\!]\,(O) > 0$ can be obtained using a finite number of invocations of Generate-Expression-From-Prior.

Proposition 3.18. *The do-while loop of Algorithm 3.1 will terminate with probability 1, and the expected number of iterations is at most $c_{tx}^{\max}/c_{tx}$.* «

Proof. The number of iterations of the loop is geometrically distributed with mean

$$\begin{aligned} p &= \sum_{E \in \mathcal{L}} \text{Prior}\,[\![E]\!] \cdot \mathbb{I}[\text{Likelihood}\,[\![E]\!]\,(O) > 0] && (3.12)\\ &= \frac{1}{c_{tx}^{\max}} \sum_{E \in \mathcal{L}} \text{Prior}\,[\![E]\!] \cdot c_{tx}^{\max} \cdot \mathbb{I}[\text{Likelihood}\,[\![E]\!]\,(O) > 0] && (3.13)\\ &\geq \frac{1}{c_{tx}^{\max}} \sum_{E \in \mathcal{L}} \text{Prior}\,[\![E]\!] \cdot \text{Likelihood}\,[\![E]\!]\,(O) = \frac{c_{tx}}{c_{tx}^{\max}}. && (3.14) \end{aligned}$$

Therefore, the expected number of iterations of the do-while loop is at most $1/p = c_{tx}^{\max}/c_{tx} < \infty$. ■

The notation $\mathcal{T}^n(O, E_0 \to (E_1, \dots, E_n))$ denotes the probability that Algorithm 3.1 returns expressions $(E_1, \dots, E_n)$ given $n \geq 1$ inputs, observation $O \in \mathcal{X}$, and initial expression E_0. For each $E \in \mathcal{L}$, the marginal probability that the return value of Algorithm 3.1 satisfies $E_n = E$ is denoted $\text{ApproxPost}_{E_0}^{(n)}\,[\![E]\!]\,(O)$ and can be defined inductively

$$\begin{aligned} \text{ApproxPost}_{E_0}^{(1)}\,[\![E]\!]\,(O) &::= \mathcal{T}(O, E_0 \to E) && (3.15)\\ \text{ApproxPost}_{E_0}^{(n)}\,[\![E]\!]\,(O) &::= \sum_{E' \in \mathcal{L}} \mathcal{T}(O, E' \to E) \cdot \text{ApproxPost}_{E_0}^{(n-1)}\,[\![E']\!]\,(O) \qquad (n > 1). && (3.16) \end{aligned}$$

The next two convergence theorems are due to Tierney [1994].

Theorem 3.19 (Convergence of MCMC [Tierney, 1994, Theorem 1])**.** *If Conditions 3.15–3.17 all hold for language $\mathcal{L}$, transition operator $\mathcal{T}$, and observation $O \in \mathcal{X}$, then for all $E_0 \in \mathcal{L}$ such that* Likelihood $[\![E_0]\!]\,(O) > 0$,

$$\sup_{\mathcal{E} \subset \mathcal{L}} \left| \sum_{E \in \mathcal{E}} \left[\text{ApproxPost}_{E_0}^{(n)}\,[\![E]\!]\,(O) - \text{Post}\,[\![E]\!]\,(O) \right] \right| \xrightarrow[n \to \infty]{} 0. \tag{3.17}$$

«

Eq. (3.17) is a very strong form of convergence known as convergence in total variation. It states that the maximum difference that the probability measures $\mathrm{ApproxPost}_{E_0}^{(n)} [\![\cdot]\!] (O)$ and $\mathrm{Post} [\![\cdot]\!] (O)$ assign to any set $\mathcal{E} \subset \mathcal{L}$ of expressions in the language can be made arbitrarily close to 0, for sufficiently large n.

Theorem 3.20 (Strong Law of Large Numbers for MCMC [Tierney, 1994, Theorem 3])**.** *If Conditions 3.15–3.17 hold for language $\mathcal{L}$, transition operator $\mathcal{T}$, and data $O \in \mathcal{X}$, then for all $E_0 \in \mathcal{L}$ with* $\mathrm{Likelihood} [\![E_0]\!] (O) > 0$ *and any real function $\phi : \mathcal{L} \to \mathbb{R}$ that satisfies $\sum_{E \in \mathcal{L}} |\phi(E)| \cdot \mathrm{Post} [\![E]\!] (O) < \infty$,*

$$\frac{1}{n}\sum_{i=1}^{n} \phi(E_i) \xrightarrow[n\to\infty]{} \sum_{E\in\mathcal{L}} \phi(E) \cdot \mathrm{Post} [\![E]\!] (O) =: \mathbb{E}_{\mathrm{Post}} [\phi(E)] \qquad \textit{almost surely}, \tag{3.18}$$

where $(E_1, \dots, E_n) \sim \mathcal{T}^n(E_0 \to \cdot)$. «

Eq. (3.18) states that the time average of $\phi(E_i)$ evaluated along a random sample path of the Markov chain converges with probability one to the expected value $\mathbb{E}_{\mathrm{Post}} [\phi(E)]$ under the posterior.

Remark 3.21. It is common to run Algorithm 3.1 several times to obtain $M \geq 1$ independent MCMC chains $\{(E_1^j, \dots, E_n^j)\}_{j=1}^{M}$. To estimate $\mathbb{E} [\phi(E)]$, the average-of-averages is computed

$$\frac{1}{m}\sum_{j=1}^{m} \left[\frac{1}{n} \sum_{i=B}^{n} \phi(E_i^j) \right], \tag{3.19}$$

where $B \in \{1, \dots, n\}$ denotes a "burn-in" period. The heuristics of multiple chains and burn-in may improve the variance of the estimator and they do not impact the correctness of Theorem 3.20. «

3.3.2 Bayesian Synthesis via Resample-Move Sequential Monte Carlo

In many data modeling problems, the data has a natural interpretation as a stream of observations

$$(t_1, x_1), (t_2, x_2), (t_3, x_3), \dots \tag{3.20}$$

where, using the notation from Definition 3.1, each $t_i \in T$ and $x_i \in X_{t_i}$ for $i \geq 1$. For example, in the Gaussian process DSL from Chapter 2, t_i is the list of all time points up to and including step i and x_i is the list of corresponding time series values. When the data has this kind of sequential structure, the same three primitives (P1)–(P3) can be used to design SMC synthesis algorithms that are more scalable than the MCMC approach in Algorithm 3.1.

Overview of SMC There are many algorithms that fall under the broad umbrella of "sequential Monte Carlo". The specific variant considered here is called "resample-move" SMC [Gilks and Berzuini, 2001]. Let $O_j ::= (t_j, x_j)$ be the j^{th} element in the sequence (3.20). The sequence of target distributions

$$\{\mathrm{Post} [\![\cdot]\!] (O_j)\}_{j=0}^{J} \tag{3.21}$$

are all defined over the same space $\mathcal{L}$ of DSL expressions[1]. For $j = 0$, the target distribution $\mathrm{Post} [\![\cdot]\!] (O_0)$ is simply $\mathrm{Prior} [\![\cdot]\!]$. At each step $j = 1, \dots, J$, the SMC algorithm maintains a set $\{(E_j^\ell, w_j^\ell)\}_{\ell=1}^{M}$ of $M \geq 1$ weighted expressions, where (informally) the weight w_j^ℓ represents how well the expression E_j^ℓ

[1] The setting where all the target distributions are defined on the same space is sometimes referred to as "static sequential Monte Carlo", to distinguish from the more common setting of state-space models where the dimensionality of the latent space increases at each step.

(S0) At step $j = 0$, initialize for each $\ell = 1, \dots, M$

$$E_0^\ell \sim \textsc{Generate-Expression-From-Prior}(), \tag{3.23}$$

For iterations $j = 1, \dots, J$, run steps (S1)–(S3):

(S1) **Reweight**: For $\ell = 1, \dots, M$, incorporate observation O_j by recomputing the weights:

$$w_j^\ell \leftarrow \frac{\textsc{Evaluate-Likelihood}(O_j, E_{j-1}^\ell)}{\textsc{Evaluate-Likelihood}(O_{j-1}, E_{j-1}^\ell)}, \tag{3.24}$$

(S2) **Resample**: For $\ell = 1, \dots, M$; if $j < J$ then resample the parents and reset the weights:

$$u \sim \text{Categorical}(w_j^1, \dots, w_j^M), \quad E_j^\ell \leftarrow E_{j-1}^u \tag{3.25}$$

(S3) **Rejuvenate**: For $\ell = 1, \dots, M$, rejuvenate E_j^ℓ times by running $n \geq 0$ iterations of MCMC:

$$E_j^\ell \sim \textsc{Generate-New-Expression}(O_j, E_j^\ell). \tag{3.26}$$

Listing 3.1: Resample-move sequential Monte Carlo for Bayesian synthesis.

explains O_j, relative to other particles. This set of weighted expressions forms a discrete "particle-based" approximation of the target distribution at step j,

$$\sum_{\ell=1}^{M} \frac{w_j^\ell}{\sum_{k=1}^{M} w_j^k} \delta_{E_j^\ell}(\cdot) \approx \text{Post}\, [\![\cdot]\!]\, (O_j), \tag{3.22}$$

where δ_E is an atomic probability measure at the expression $E \in \mathcal{L}$. Listing 3.1 explains how to sequentially construct such a particle approximation. In (S0), a set of M particles are initialized i.i.d. from the prior, as in line 3 of Algorithm 3.1. For $j = 1, \dots, J$, the observations are incorporated, which involves: (S1) reweighting each particle by the ratio of likelihood of the full new data $X_{1:j}$ to the likelihood of the previous data $X_{1:j-1}$; (S2) resampling the particles based on their weights; (S3) rejuvenating the particles by running $n \geq 0$ iterations of $\mathcal{T}$, as in line 6 of Algorithm 3.1.

Remark 3.22. In the reweighting step (S1) of Listing 3.1, the ratio (3.24) is well defined across all executions of the SMC algorithm whenever, in addition to Conditions 3.7 and 3.9, the extra condition Likelihood $[\![E]\!]\,(O) > 0$ for all E and O holds. Otherwise the particle weights may collapse to zero and the algorithm as presented would fail. Del Moral et al. [2015] discuss alternative techniques for handling this type of particle collapse. «

Remark 3.23. In Listing 3.1, elements of the observation sequence $(O_1, \dots, O_j)$ need not be incorporated one by one. Any batch of $B \in \{1, \dots, J\}$ observations can be used. «

Remark 3.24. In the resampling step (S2) of Listing 3.1, a common heuristic is to use an adaptive resampling strategy rather than resampling at each iteration. Resampling is triggered at step $j \in \{1, \dots, J\}$ if the effective sample size (a measure of the particle diversity) satisfies

$$\text{ESS}(w_j^{1:\ell}) ::= \frac{1}{\sum_{\ell=1}^{M} \left[(w_j^\ell / (\sum_{k=1}^{M} w_j^k))^2 \right]} < \text{ESS}_{\min} \in \{1, \dots, M\}. \tag{3.27}$$

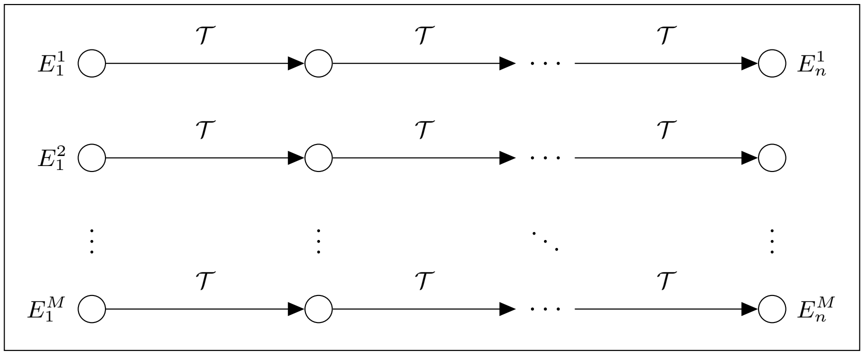


Figure 3.2: Markov chain Monte Carlo with M parallel chains executed for n iterations.

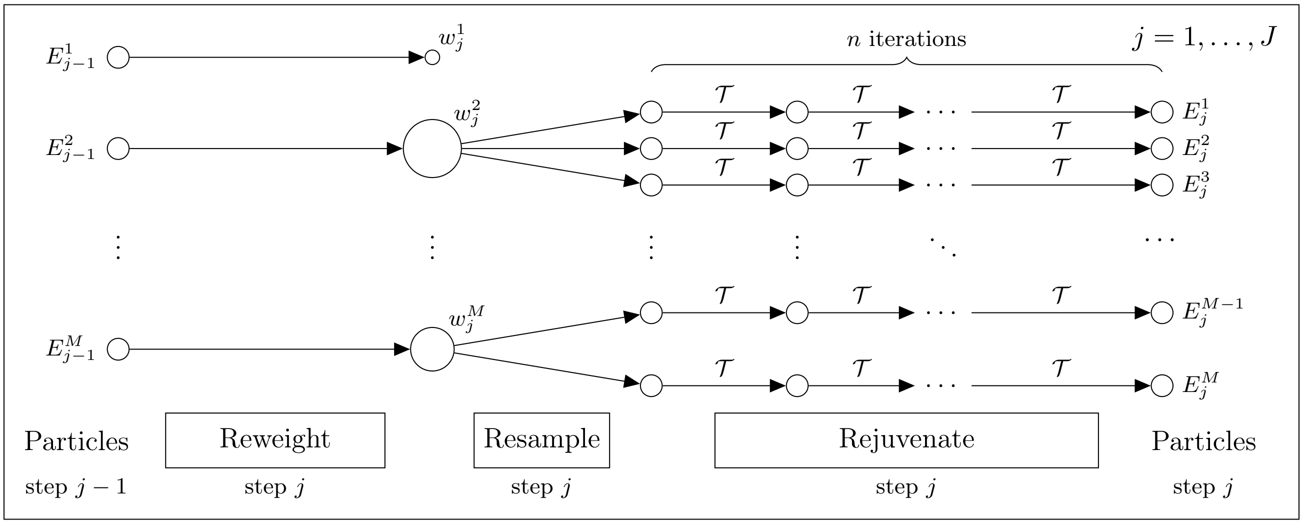


Figure 3.3: Resample-move sequential Monte Carlo with M particles and n rejuvenation iterations.

As with the MCMC heuristic from Remark 3.21, adaptive resampling does not impact correctness and may reduce variance at later steps. Resampling does not reduce variance at the present step, which is why resampling at the final step $j = J$ is avoided in (S2). Algorithm 3.2 shows an implementation of the resample-move SMC template in Listing 3.1 with adaptive resampling. «

Figures 3.2 and 3.3 show diagrams for MCMC and resample-move SMC, respectively. In situations where the observed data can be treated sequentially, SMC improves MCMC upon in the following ways:

- In MCMC with $M \geq 1$ parallel chains (Remark 3.21), each chain operates independently of the rest, whereas SMC with $M \geq 1$ particles redirects computational effort to more promising parts of the DSL $\mathcal{L}$ via resampling.
- MCMC does not easily apply to sequences of observations for online or streaming settings, as the target (posterior) distribution is fixed, whereas SMC handles streaming data through the sequence of posteriors (3.21).
- SMC delivers unbiased estimates of the marginal likelihood (3.29) of the observed data, which have many uses; for example, performing model comparison among a collection set of DSLs $\{\mathcal{L}_1, \ldots, \mathcal{L}_T\}$ that operate over the same data space $\mathcal{X}$.

Algorithm 3.2 Resample-move sequential Monte Carlo algorithm for Bayesian synthesis.

```
Require: observations $(O_1, \dots, O_J)$ where $O_i \in \mathcal{X}$; number of move iterations $n \geq 0$; number of
    particles $M \geq 1$; $\mathrm{ESS}_{\min} \in \{1, \dots, M\}$
procedure Bayesian-Synthesis-SMC($X, n, M$)
    for $\ell = 1 \dots M$ do                                              ▷ initialize $M$ particles
        $E_0^\ell \sim$ Generate-Expression-From-Prior()
        $w_0^\ell \leftarrow 1$;   $\hat{w}_{j-1}^\ell \leftarrow 1$
    for $j = 1 \dots J$ do                                   ▷ for each observation in the sequence
        // REWEIGHT
        for $\ell = 1 \dots M$ do
            $w_j^\ell \leftarrow \hat{w}_{j-1}^\ell \cdot \dfrac{\textsc{Evaluate-Likelihood}(O_j, E_{j-1}^\ell)}{\textsc{Evaluate-Likelihood}(O_{j-1}, E_{j-1}^\ell)}$
        // RESAMPLE (ADAPTIVE)
        if $\mathrm{ESS}(w_{j-1}^{1:M}) < \mathrm{ESS}_{\min}$ and $j < J$ then          ▷ resampling triggered
            for $\ell = 1 \dots M$ do                                          ▷ resample particle
                $A_j^\ell \leftarrow \mathrm{Categorical}(M; w_j^{1:M})$                        ▷ reset weight
                $\hat{w}_j^\ell \leftarrow 1$
        else
            for $\ell = 1 \dots M$ do                                   ▷ no resampling triggered
                $A_i^\ell \leftarrow \ell$                                                ▷ copy particle
                $\hat{w}_j^\ell \leftarrow w_{j-1}^\ell$                                          ▷ reset weight
        // REJUVENATE
        for $\ell = 1 \dots M$ do
            $E_j^\ell \leftarrow E_{j-1}^{A_j^\ell}$                                ▷ set parent to new ancestor
            for $i = 1 \dots n$ do
                $E_j^\ell \sim$ Generate-New-Expression($O_j, E_j^\ell$)         ▷ run transition operator
    return $E_{0:J}^{1:M}, w_{0:J}^{1:M}, A_{1:J}^{1:M}$
```

Convergence Properties If the transition operator $\mathcal{T}(O, E \to E')$, which is invoked by calling Generate-New-Expression in (S3), satisfies Conditions 3.15–3.17 for each target distribution in the sequence (3.21), then estimates of expectations $\mathbb{E}_{\mathrm{Post}}[\phi(E)]$ using the particle-based approximation (3.22) have a similar consistency property to Theorem 3.20, albeit for a more restricted class of functions ϕ. That is for each $j = 1, \dots, J$,

$$\frac{1}{M} \sum_{\ell=1}^{M} W_j^\ell \phi(E_j^\ell) \xrightarrow[M \to \infty]{} \sum_{E \in \mathcal{L}} \phi(E) \cdot \mathrm{Post} \, [\![E]\!] \, (O_j) \qquad \text{almost surely.} \tag{3.28}$$

Precise statements about a central limit theorem and technical conditions on ϕ can be found in Gilks and Berzuini [2001, Theorem 1], Chopin [2004, Theorem 1], and Del Moral et al. [2006, Proposition 2]. Unlike MCMC, however, SMC has the remarkable property that it also delivers unbiased estimates of the marginal likelihood of each O_j $(j = 1, \dots, J)$. Using the notation from Algorithm 3.2,

$$\mathbb{E}\left[\prod_{j=1}^{J}\left[\frac{1}{\sum_{\ell=1}^{M} \hat{w}_{j-1}^\ell} \sum_{\ell=1}^{M} w_j^\ell\right]\right] = \sum_{E \in \mathcal{L}} \mathrm{Likelihood} \, [\![E]\!] \, (O_j) \cdot \mathrm{Prior} \, [\![E]\!] \, . \tag{3.29}$$

3.4 Bayesian Synthesis for Context-Free Probabilistic DSLs

The preceding sections have introduced probabilistic DSLs (Section 3.1), formalized the Bayesian synthesis problem (Section 3.2), and presented MCMC and SMC algorithm templates (Section 3.3) that deliver sound approximate solutions to Bayesian synthesis under well characterized technical conditions. This section introduces a new class of context-free grammars that are used to define "context-free" probabilistic DSLs and outlines a provably sound implementation of the end-to-end Bayesian synthesis framework that applies to any context-free probabilistic DSL.

- Sections 3.4.1 and 3.4.2 presents a class of context-free grammars for specifying probabilistic DSLs.
- Section 3.4.3 shows how to implement (P1) Generate-Expression-From-Prior for context-free probabilistic DSLs by defining a Prior semantics that is guaranteed to be normalized as in Condition 3.6.
- Section 3.4.4 shows how to implement (P3) Generate-New-Expression for any context-free probabilistic DSL and proves that the transition operator satisfies Conditions 3.15–3.17 which ensure convergence of the MCMC and SMC algorithms in Section 3.3.

Remark 3.25. The procedure (P2) Evaluate-Likelihood depends on the modeling application (e.g., Eq. (3.63))—any semantics that satisfies Conditions 3.7 and 3.9 is sufficient. «

3.4.1 Context-Free Grammars for Specifying Probabilistic DSLs

Formal grammars describe a recipe for how to form strings of a language starting from an alphabet of terminal symbols [Jelinek et al., 1992]. This section describes a new class of context-free grammars for defining probabilistic domain-specific data modeling languages whose expressions resemble Lisp-style symbolic expressions, or "s-expressions". These context-free grammars have two main properties:

(i) The grammar produces s-expressions that contain a unique phrase tag for each production rule, which uniquely identifies the production rule that produced a given subexpression;

(ii) Each production rule is associated with a probability distribution over terminal symbols that are jointly sampled to fill a finite number of "holes" in the production rule.

The first property guarantees that the grammar is unambiguous, i.e., each expression has a unique parse, which makes it straightforward to compute the probability of generating a given expression without enumerating over all possible parses. The second property is used to model random parameters.

Definition 3.26. A *context-free grammar for a probabilistic DSL* consists of six entities.

- $N ::= \{N_1, \dots, N_m\}$ is a finite set of non-terminal symbols.
- $R ::= \{r_{ik} \mid i = 1, \dots, m; k = 1, \dots, r_i\}$ is a set of production rules, where r_{ik} is the k^{th} production rule of non-terminal N_i. Each production rule r_{ik} is a tuple of the form

$$r_{ik} ::= (N_i, t_{ik}, h_{ik}, \widetilde{N}^1_{ik}, \dots, \widetilde{N}^{n_{ik}}_{ik}), \tag{3.30}$$

  where h_{ik} is the number of holes in the production; n_{ik} the number of non-terminals; and $\widetilde{N}^j_{ik} \in N$ for $j = 1, \dots, n_{ik}$. If $n_{ik} = 0$ then r_{ik} is a *non-recursive* production rule, otherwise it is *recursive.*
- $T ::= \{t_{ik} \mid i = 1, \dots, m; k = 1, \dots, r_i\}$ is a set of phrase tag symbols, disjoint from N, where t_{ik} is a unique symbol that identifies the production rule r_{ik}.

- $P ::= \{p_{ik} \mid i = 1, \dots, m; k = 1, \dots, r_i\}$ is a set of probabilities, where each phrase tag t_{ik} is assigned a probability $p_{ik} \in (0, 1]$ that non-terminal N_i selects production rule r_{ik}.
- $W ::= \{(\Theta_{ik}, \Sigma_{ik}, \mu_{ik}) \mid i = 1, \dots, m; k = 1, \dots, r_i; h_{ik} > 0\}$ is a family of probability spaces, which specify a joint distribution over the allowable valuables that the h_{ik} holes in the production rule r_{ik} may take.
- $N^{\text{start}} \in N$ is a designated start symbol.

«

Example 3.27. Consider the Gaussian process DSL in Listing 2.1 from Chapter 2. Listing 3.2 shows a context-free grammar that more formally defines this DSL. The start symbol $N^{\text{start}} = N_1$ and the phrase tags $T = \{t_{11}, t_{21}, t_{31}, \dots, t_{37}\}$ are shown in teletype font. The notation $B(U)$ denotes the Borel sigma-algebra of a topological space U, which is generated by the open sets. «

3.4.2 Defining a Probabilistic DSL from a Context-Free Grammar

This section shows how a context-free grammar G (Definition 3.26) defines the structure space S and measure spaces $(\Theta_s, \mathcal{F}_s, \lambda_s)$ (for each $s \in S$) of a probabilistic DSL D (Definition 3.4).

Eq. (3.30) describes how the production rules of G convert non-terminal symbols into tagged s-expressions with holes. The evaluation $N_i \Downarrow_G^p E$ means that, starting from non-terminal N_i, the s-expression E is yielded with probability p. Recalling that both h_{ik} and n_{ik} in Eq. (3.30) can be zero, the following rule applies for all $i = 1, \dots, n$ and $k = 1, \dots, r_i$:

$$\frac{(N_i, t_{ik}, h_{ik}, \widetilde{N}_{ik}^1, \dots, \widetilde{N}_{ik}^{n_{ik}}) \in R \quad \widetilde{N}_{ik}^1 \Downarrow_G^{p_1} E_1 \quad \dots \quad \widetilde{N}_{ik}^{n_{ik}} \Downarrow_G^{p_{n_{ik}}} E_{n_{ik}}}{N_i \Downarrow_G^{p_{ik} \prod_{j=1}^{n_{ik}} p_j} (t_{ik} \;\square_1 \;\dots\; \square_{h_{ik}} \; E_1 \;\dots\; E_{n_{ik}})} \tag{3.31}$$

Remark 3.28. The rule (3.31) always yields an s-expression where the first element is a phrase tag t_{ik} that unambiguously identifies the production rule r_{ik} that yielded the expression. Thus, every s-expression has a unique parse tree [Turbak and Gifford, 2008, Section 2.3.3]. «

The language $\mathcal{L}_\square(G, N_i)$ denotes the set of all s-expressions with holes that can be yielded starting from non-terminal N_i $(i = 1, \dots, m)$ with positive probability according to the relation defined by Eq. (3.31). Further, the language $\mathcal{L}_\square(G) ::= \mathcal{L}_\square(G, N^{\text{start}})$ is the set of all s-expressions derivable from the start symbol. With these notations, the structure space S and probabilities p_s $(s \in S)$ are

$$S ::= \mathcal{L}_\square(G) \tag{3.32}$$

$$p_s ::= w \in (0, 1] \text{ such that } N^{\text{start}} \Downarrow_G^w s. \tag{3.33}$$

Each probability p_s is unique by Remark 3.28. Next, the probability spaces $\{(\Theta_s, \mathcal{F}_s, \mu_s), s \in S\}$ of D are obtained from a transition relation $s \Downarrow_G (\Theta_s, \Sigma_s, \mu_s)$, defined as follows:

$$\frac{}{(t_{ik} \;\square_1 \;\dots\; \square_{h_{ik}}) \Downarrow_G (\Theta_{ik}, \Sigma_{ik}, \mu_{ik})} \tag{\textsc{Primitive-Production}}$$

$$\frac{s_1 \Downarrow_G^\Theta (\Theta_1, \mathcal{F}_1, \mu_1) \quad \dots \quad s_{n_{ik}} \Downarrow_G^\Theta (\Theta_{n_{ik}}, \mathcal{F}_{n_{ik}}, \mu_{n_{ik}})}{(t_{ik} \;\square_1 \;\dots\; \square_{h_{ik}} \; s_1 \dots\; s_{n_{ik}}) \Downarrow_G (\prod_{j=1}^{n_{ik}} \Theta_j, \otimes_{j=1}^{n_{ik}} \mathcal{F}_j, \times_{j=1}^{n_{ik}} \mu_j)} \tag{\textsc{Recursive-Production}}$$

where

$$\prod_{j=1}^{n_{ik}} \Theta_j ::= \{(\theta_1, \dots, \theta_{n_{ik}}) \mid \theta_j \in \Theta_j, j = 1, \dots n_{ik}\} \qquad \text{(Cartesian product)} \tag{3.34}$$

$$\otimes_{j=1}^{n_{ik}} \mathcal{F}_j ::= \sigma\left(\left\{\prod_{j=1}^{n_{ik}} A_j \mid A_j \in \mathcal{F}_j, j = 1, \dots, n_{ik}\right\}\right) \qquad \text{(product sigma-algebra)} \tag{3.35}$$

$$(\times_{j=1}^{n_{ik}} \mu_j)\left(\prod_{j=1}^{n_{ik}} A_j\right) ::= \prod_{j=1}^{n_{ik}} \mu_j(A_j),\ A_j \in \mathcal{F}_j, j = 1, \dots, n_{ik} \qquad \text{(product measure).} \tag{3.36}$$

Remark 3.29. By Billingsley [1995, Theorem 18.2], the product measure (3.36) defined on the generating sets extends to a unique probability measure on the entire sigma-algebra (3.35). «

Remark 3.30. The context-free property of G means that the joint probability distributions for holes that belong to different subexpressions are obtained in Eq. (Recursive-Production) by taking product measures. While the h_{ik} holes in a s-expression with phrase tag t_{ik} have an arbitrary joint distribution μ_{ik}, the holes in different subexpressions are mutually independent. «

Eqs. (3.32)–(3.36) how a context-free grammar G uniquely defines the set of expressions (3.1) of a probabilistic DSL D.

Definition 3.31. Let G be a context-free grammar and D a probabilistic DSL whose expressions match those generated by G. For each $(s, \theta) \in \mathcal{L}(D)$, the notation $s \oplus \theta$ denotes the unique s-expression obtained by syntactically replacing each hole in s with the corresponding parameter in θ. The languages

$$\mathcal{L}(G, N_i) ::= \{s \oplus \theta \mid s \in \mathcal{L}_\square(G, N_i), \theta \in \Theta_s\} \qquad (i = 1, \dots, m) \tag{3.37}$$

are defined to be the set of all s-expressions that can be yielded starting from the non-terminal N_i and where the holes are replaced with valid parameter values. Moreover, the language $\mathcal{L}(G) ::= \mathcal{L}(G, N^{\text{start}})$. «

Example 3.32. Let G be the grammar in Listing 3.2 and D a probabilistic DSL whose expressions match those generated by G. Consider the following structure $s \in \mathcal{L}_\square(G)$ and parameter $\theta \in \Theta_s$:

$$s ::= \texttt{(GaussianProcess (Noise }\square\texttt{) (* (Constant }\square\texttt{) (Periodic }\square\ \square\texttt{)))} \tag{3.38}$$

$$\theta ::= ((), (0.7, (1.8, (2.1, 3.1)))). \tag{3.39}$$

As $(s, \theta) \in \mathcal{L}(D)$, applying Definition 3.31 gives

$$s \oplus \theta = \texttt{(GaussianProcess (Noise 0.7) (* (Constant 1.8) (Periodic 2.1 3.1)))}.$$

The formal definition of $s \oplus \theta$ is straightforward. The uniqueness of $s \oplus \theta$ follows from Remark 3.28. «

Remark 3.33. As the languages $\mathcal{L}(G)$ and $\mathcal{L}(D)$ are in 1-1 correspondence, the expressions (s, θ) and $s \oplus \theta$ will be used interchangeably, whichever is clearer or more notationally compact. Any semantic function defined on one of these domains is therefore also defined on the other. «

3.4.3 A Sound Prior Semantics

Consider a context-free grammar G and the corresponding language $\mathcal{L}_{\Box}(G)$ of all derivable s-expressions with holes. To meet Condition 3.6, setting Prior $[\![(s,\theta)]\!]\, \lambda_s(\mathrm{d}\theta) = p_s \mu_s(\mathrm{d}\theta)$ is sufficient since

$$\sum_{s \in S} \left[\int_{\theta \in \Theta_s} \text{Prior}\, [\![(s,\theta)]\!]\, \mu_s(\mathrm{d}\theta) \right] = \sum_{s \in S} p_s \mu(\Theta_s) = 1. \tag{3.40}$$

Eq. (3.40) is well formed if and only if the structure probabilities $\{p_s \mid s \in \mathcal{L}_{\Box}(G)\}$ defined in Eq. (3.33) sum to one. A classic result from Booth and Thompson [1973] can be used to verify this property.

Definition 3.34 (Consistency [Booth and Thompson, 1973])**.** A context-free grammar G is consistent if the probabilities assigned to all strings derivable from G sum to one. «

Theorem 3.35 (Sufficient Condition for Consistency [Booth and Thompson, 1973])**.** *A proper probabilistic context-free grammar G is consistent if the largest eigenvalue (in modulus) of the expectation matrix of G is less than one.* «

Remark 3.36. Gecse and Kovács [2010] show how to construct the expectation matrix of a context-free grammar G from the production rule probabilities $\{p_{ik}\}$. If the production rules enforce finite recursion depth, then G is consistent since $\mathcal{L}_{\Box}(G)$ is finite. If the production rules allow arbitrary recursion depth, then consistency of G is equivalent to having a finite expected number of steps in the rewriting rule Eq. (3.31). Every context-free grammar is henceforth assumed to be consistent. «

In most modeling applications, the distributions μ_{ik} for the holes in the context-free grammar G have a density γ_{ik} with respect to a sigma-finite dominating measure λ_{ik} (e.g., some combination of Lebesgue and counting measures) over Σ_{ik}. For $s \in \mathcal{L}_{\Box}(G)$, the measure λ_s is defined to be the product the measure over $(\Omega_s, \mathcal{F}_s)$ analogously to Eq. (3.36) and $\gamma_s : \Theta_s \to \mathbb{R}_{\geq 0}$ denotes the product densities of μ_s. A default denotational semantics Prior : $\mathcal{L}(G) \to \mathbb{R}_{>0}$ can now be described for this class of applications. To aid with the construction, the semantic function Expand : $\mathcal{L}(G) \to N \to \mathbb{R}_{\geq 0}$ is used to map an expression and a non-terminal symbol to the probability density that the non-terminal is expanded to the given expression:

$$\text{Expand}\, [\![\texttt{(}t_{ik}\ \theta_1\ \ldots\ \theta_{h_{ik}}\ E_1\ \ldots\ E_{n_{ik}}\ \texttt{)}]\!]\, (N_i) ::= p_{ik} \prod_{j=1}^{h_{ik}} \gamma_{ik}(\theta_1, \ldots, \theta_{h_{ik}}) \prod_{j=1}^{n_{ik}} \text{Expand}\, [\![E_j]\!]\, (\widetilde{N}_{ik}^{j}) \tag{3.41}$$

for $i = 1, \ldots, n$ and $k = 1, \ldots, r_i$. The prior probability of an expression E is then

$$\text{Prior}\, [\![E]\!] ::= \text{Expand}\, [\![E]\!]\, (N^{\text{start}}). \tag{3.42}$$

Proposition 3.37. *The prior density of $E = (s,\theta)$ in Eq.* (3.42) *factorizes as*

$$\text{Prior}\, [\![(s,\theta)]\!] = p_s \gamma_s(\theta). \tag{3.43}$$

«

Proof. Combine Eq. (3.31) and Eq. (3.41) and perform structural induction on $(s,\theta) \in \mathcal{L}$. ■

Remark 3.38. The context-free structure of G reflects two types of probabilistic independencies. First, the joint probability of $(s,\theta) \in \mathcal{L}(D)$ factorizes according to $p_s\gamma_s(\theta)$, which means that the entire structure is sampled independently of the numeric parameters. Second, the parameters $(\theta_1, \ldots, \theta_{h_{ik}})$ in any subexpression $\texttt{(}t_{ik}\ \theta_1\ \ldots\ \theta_{h_{ik}}\ E_1\ \ldots\ E_{n_{ik}}\ \texttt{)}$ are independent of the parameters that appear in $E_1, \ldots, E_{n_{ik}}$, which are themselves mutually independent of one another. «

Algorithm 3.3 Transition operator $\mathcal{T}$ for a context-free language.

```
procedure Generate-New-Expression($O, E$)        ▷ observation $O \in \mathcal{X}$ and input expression $E \in \mathcal{L}$
    $a \sim \text{Uniform}(A_E)$        ▷ randomly select a node in parse tree
    $(N_i, E_{\text{sev}}) \leftarrow \text{Sever}_a \llbracket E \rrbracket$        ▷ sever parse tree and return non-terminal symbol at sever point
    $E_{\text{sub}} \sim \text{Expand} \llbracket \cdot \rrbracket (N_i)$        ▷ generate random $E_{\text{sub}}$ with probability $\text{Expand} \llbracket E_{\text{sub}} \rrbracket (N_i)$
    $E' \leftarrow E_{\text{sev}}[E_{\text{sub}}]$        ▷ fill hole in $E_{\text{sev}}$ with expression $E_{\text{sub}}$
    $L \leftarrow \text{Likelihood} \llbracket E \rrbracket (O)$        ▷ evaluate likelihood for expression $E$ and observation $O$
    $L' \leftarrow \text{Likelihood} \llbracket E' \rrbracket (O)$        ▷ evaluate likelihood for expression $E'$ and observation $O$
    $p_{\text{accept}} \leftarrow \min\{1, (|A_E|/|A_{E'}|) \cdot (L'/L)\}$        ▷ compute the probability of accepting the mutation
    $r \sim \text{Uniform}([0, 1])$        ▷ draw a random number from the unit interval
    if $r < p_{\text{accept}}$ then        ▷ if-branch has probability $p_{\text{accept}}$
        return $E'$        ▷ accept and return the mutated expression
    else        ▷ else-branch has probability $1 - p_{\text{accept}}$
        return $E$        ▷ reject the mutated expression and return the input expression
```

The notation $E \sim \text{Expand} \llbracket \cdot \rrbracket (N_i)$ is used to mean that E is randomly sampled according to the density defined in Eq. (3.41). That is, to expand N_i, a production rule r_{ik} is chosen with probability p_{ik}, the parameters $(\theta_1, \dots, \theta_{h_{ik}})$ are sampled jointly from μ_s to fill in the holes, and the subexpressions (if any) for $\widetilde{N}_{ik}^{j}$ $(j = 1, \dots, n_{ik})$ are sampled recursively. (P1) Generate-Expression-From-Prior() returns $E \sim \text{Expand} \llbracket \cdot \rrbracket (N^{\text{start}})$, which is guarnateed to halt whenever G is consistent.

3.4.4 A Sound Markov Chain Transition Operator

Remark 3.39. This section makes the countability assumption from Remark 3.14. Following Remark 3.33, it is also assumed that each expression $(s, \theta) \in \mathcal{L}(D)$ is written in its unique s-expression form $(s \oplus \theta) \in \mathcal{L}(G)$, where the holes in s are syntactically replaced with parameters θ. «

This section defines a generic implementation of (P3) Generate-New-Expression for any language $\mathcal{L} ::= \mathcal{L}(G)$ whose structure and parameter spaces are defined by a context-free grammar G and proves that the implementation satisfies Conditions 3.15–3.17. Algorithm 3.3 shows the transition operator. Given an initial expression $E \in \mathcal{L}$, a randomly selected subexpression in E is replaced to obtain a new expression E', which is accepted according to the usual Metropolis-Hastings rule. The notation $\mathcal{T}(O, E \to E')$ denotes the probability that Generate-New-Expression(O, E) returns E'. Before proving correctness, a scheme for uniquely identifying syntactic locations in the parse tree of expressions is described.

Definition 3.40. Let $A ::= \{(a_1, a_2, \dots, a_l) \mid a_i \in \{1, 2, \dots, h_{\max}\}, l \in \{0, 1, 2, \dots\}\}$ be a countably infinite set whose elements index nodes in parse trees of s-expressions. The integer $h_{\max}$ denotes the maximum number of symbols that appear on the right of any production rule (3.30). Each $a \in A$ is a sequence of subexpression positions on the path from the root node of a parse tree to another node. Further, the set $A_E \subset A$ denotes the finite subset of nodes that exist in the parse tree of $E \in \mathcal{L}$. «

Example 3.41. Suppose that $E = (t_0\ E_1\ E_2)$ where $E_1 = (t_1\ E_3\ E_4)$ and $E_2 = (t_2\ E_5\ E_6)$. The root node of the parse tree has index $a_{\text{root}} ::= ()$; the node for E_1 has index (1); the node for E_2 has index (2); the nodes for E_3 and E_4 have indices $(1, 1)$ and $(1, 2)$, respectively; and the nodes for E_5 and E_6 have indices $(2, 1)$ and $(2, 2)$, respectively. Thus, $A_E = \{(), (1), (2), (1, 1), (1, 2), (2, 1), (2, 2)\}$. «

Definition 3.42. The syntactic operation Sever, parametrized by $a \in A$, takes as input an expression E. If E contains a node a, then Sever returns a tuple (N_i, E_{sev}) where E_{sev} is the expression with the subexpression of E located at a replaced with a symbol $\triangle$ and where N_i is the non-terminal symbol from which the removed subexpression is produced. Otherwise, Sever fails. Define the relation $\xrightarrow[\text{sever}]{}$

$$\frac{a = ()}{(a, (T_{ik}\ \theta_1\ \ldots\ \theta_{h_{ik}}\ E_1\ \ldots\ E_{n_{ik}})) \xrightarrow[\text{sever}]{} (N_i, \triangle)} \tag{3.44}$$

$$\frac{((a_2, a_3, \ldots), E_j) \xrightarrow[\text{sever}]{} (N_i, E_{\text{sev}}) \text{ and } a_1 = j}{\begin{aligned}(a, (T_{ik}\ \theta_1\ \ldots\ \theta_{h_{ik}}\ E_1\ \ldots\ E_{n_{ik}})) \xrightarrow[\text{sever}]{} (N_i, (T_{ik}\ \theta_1\ \ldots\ \theta_{h_{ik}}& \\ E_1\ \ldots\ E_{j-1}\ E_{\text{sev}}\ E_{j+1}\ \ldots\ E_{n_{ik}}))&\end{aligned}} \tag{3.45}$$

for $i = 1, \ldots, m$, $k = 1, \ldots, r_i$ and put

$$\text{Sever}_a\, [\![(T_{ik}\ \theta_1\ \ldots\ \theta_{h_{ik}}\ E_1\ \ldots\ E_{n_{ik}})]\!] ::= \begin{cases} (N_i, E_{\text{sev}}) & \text{if } (a, (T_{ik}\ \theta_1\ \ldots\ \theta_{h_{ik}}\ E_1\ \ldots\ E_{n_{ik}})) \\ & \qquad \xrightarrow[\text{sever}]{} (N_i, E_{\text{sev}}) \\ \textbf{undefined} & \text{otherwise.} \end{cases} \tag{3.46}$$

«

Remark 3.43. For any expression $E \in \mathcal{L}$, setting $a = a_{\text{root}} \equiv ()$ gives $((), E) \xrightarrow[\text{sever}]{} (N^{\text{start}}, \triangle)$. Further, every $E_{\text{sev}} \notin \mathcal{L}$ because E_{sev} contains one or more instances of the special symbol $\triangle$. «

Remark 3.44. To handle s-expressions that contain a $\triangle$ subexpression, Eq. (3.41) is extended with the rule $\text{Expand}\,[\![\triangle]\!]\,(N_i) ::= 1$ for $i = 1, \ldots, m$. «

Definition 3.45. Let E_{sev} be an expression that contains a single $\triangle$ for which $(a, E) \xrightarrow[\text{sever}]{} (N_i, E_{\text{sev}})$ for some $E \in \mathcal{L}$, $a \in A$, and $i \in [m]$. The replacement $E_{\text{sev}}[E_{\text{sub}}] \in \mathcal{L}$ denotes the expression formed by replacing $\triangle$ with E_{sub}, where $E_{\text{sub}} \in \mathcal{L}(G, N_i)$ «

Definition 3.46. The operation Subexpr, parametrized by a, takes an expression E and extracts the subexpression corresponding to node a in the parse tree. That is, for $i = 1, \ldots, m$ and $k = 1, \ldots, r_i$,

$$\text{Subexpr}_a\, [\![(T_{ik}\ \theta_1\ \ldots\ \theta_{h_{ik}}\ E_1\ E_2\ \ldots\ E_{n_{ik}})]\!] ::= \begin{cases} \varnothing & \text{if } a = () \text{ or } a_1 > l \\ E_j & \text{if } a = (j) \text{ for some } 1 \le j \le l \\ \text{Subexpr}_{(a_2, a_3, \ldots)}\, [\![E_j]\!] & \text{if } a \ne (j) \text{ and} \\ & \quad a_1 = j \text{ for some } 1 \le j \le l. \end{cases}$$

«

Consider the probability that $\mathcal{T}$ takes an expression E to another expression E', which by total probability is an average over the uniformly chosen node index a:

$$\mathcal{T}(O, E \to E') = \frac{1}{|A_E|} \sum_{a \in A_E} \mathcal{T}(O, E \to E'; a) = \frac{1}{|A_E|} \sum_{a \in A_E \cap A_{E'}} \mathcal{T}(O, E \to E'; a), \tag{3.47}$$

where

$$\mathcal{T}(O, E \to E'; a) ::= \begin{cases} \text{Expand}\,[\![\text{Subexpr}_a\,[\![E']\!]]\!]\,(N_i) \cdot \alpha(E, E') & \text{if } \text{Sever}_a\,[\![E]\!] = \text{Sever}_a\,[\![E']\!] \\ & \qquad = (N_i, E_{\text{sev}}) \text{ for some } i \text{ and } E_{\text{sev}}, \\ 0 \quad \text{otherwise} & \end{cases} \tag{3.48}$$

$$\alpha(E, E') ::= \min\left\{1, \frac{|A_E| \cdot \text{Likelihood}\,[\![E']\!]\,(O)}{|A_{E'}| \cdot \text{Likelihood}\,[\![E]\!]\,(O)}\right\}.$$

The second equality in Eq. (3.47) discards terms $a \in A_E \setminus A_{E'}$ from the sum, as these terms have $\text{Sever}_a \llbracket E' \rrbracket = \varnothing$ and $\mathcal{T}(O, E \to E'; a) = 0$.

Proposition 3.47. *For $E \in \mathcal{L}$, if $\text{Sever}_a \llbracket E \rrbracket = (N_i, E_{\text{sev}})$ then $\text{Expand} \llbracket \text{Subexpr}_a \llbracket E \rrbracket \rrbracket (N_i) > 0$.* «

Proof. If $\text{Sever}_a \llbracket E \rrbracket = (N_i, E_{\text{sev}})$ then $\text{Subexpr}_a \llbracket E \rrbracket$ is an expression with tag t_{ik} for some k. Since $E \in \mathcal{L}$, it must be that $\text{Expand} \llbracket \text{Subexpr}_a \llbracket E \rrbracket \rrbracket (N_i) > 0$, as otherwise $\text{Prior} \llbracket E \rrbracket = 0$ (a contradiction). ■

Proposition 3.48. *For $E \in \mathcal{L}$, if $\text{Sever}_a \llbracket E \rrbracket = (N_i, E_{\text{sev}})$ then:*

$$\text{Expand} \llbracket E \rrbracket (S) = \text{Expand} \llbracket \text{Subexpr}_a \llbracket E \rrbracket \rrbracket (N_i) \cdot \text{Expand} \llbracket E_{\text{sev}} \rrbracket (S). \tag{3.49}$$

«

Proof. Each factor in $\text{Expand} \llbracket E \rrbracket (S)$ corresponds to a particular node a' in the parse tree. If a' is a descendant of a, then each factor corresponding to a' appears in $\text{Expand} \llbracket \text{Subexpr}_a \llbracket E \rrbracket \rrbracket (N_i)$. Otherwise, it appears in $\text{Expand} \llbracket E_{\text{sev}} \rrbracket (S)$. ■

Proposition 3.49. *For any $E, E' \in \mathcal{L}$ where $\text{Sever}_a \llbracket E \rrbracket = \text{Sever}_a \llbracket E' \rrbracket = (N_i, E_{\text{sev}})$, it holds that*

$$\text{Prior} \llbracket E' \rrbracket = \text{Prior} \llbracket E \rrbracket \cdot \frac{\text{Expand} \llbracket \text{Subexpr}_a \llbracket E' \rrbracket \rrbracket (N_i)}{\text{Expand} \llbracket \text{Subexpr}_a \llbracket E \rrbracket \rrbracket (N_i)}. \tag{3.50}$$

«

Proof. Use $\text{Prior} \llbracket E \rrbracket = \text{Expand} \llbracket E \rrbracket (S)$ and $\text{Prior} \llbracket E' \rrbracket = \text{Expand} \llbracket E' \rrbracket (S)$. Invoke Proposition 3.48. ■

Lemma 3.50. *Algorithm 3.3 satisfies Condition 3.15 (posterior invariance).* «

Proof. It suffices to establish detailed balance for $\mathcal{T}$ with respect to the posterior:

$$\text{Post} \llbracket E \rrbracket (O) \cdot \mathcal{T}(O, E \to E') = \text{Post} \llbracket E' \rrbracket (O) \cdot \mathcal{T}(O, E' \to E) \qquad (E, E' \in \mathcal{L}). \tag{3.51}$$

First, if $\text{Post} \llbracket E \rrbracket (O) \cdot \mathcal{T}(O, E \to E') = 0$ then $\text{Post} \llbracket E' \rrbracket (O) \cdot \mathcal{T}(O, E' \to E) = 0$ as follows. Either $\text{Post} \llbracket E \rrbracket (O) = 0$ or $\mathcal{T}(O, E \to E') = 0$. There are two cases.

Case 1. If $\text{Post} \llbracket E \rrbracket (O) = 0$ then $\text{Likelihood} \llbracket E \rrbracket (O) = 0$.
Thus $\alpha(E', E) = 0$.
Then $\mathcal{T}(O, E' \to E) = 0$.

Case 2. If $\mathcal{T}(O, E \to E') = 0$ then $\mathcal{T}(O, E \to E'; a_{\text{root}}) = 0$.
Thus, either $\alpha(E, E') = 0$ or $\text{Expand} \llbracket \text{Subexpr}_{a_{\text{root}}} \llbracket E' \rrbracket \rrbracket (S) = 0$.
If $\alpha(E, E') = 0$ then $\text{Likelihood} \llbracket E' \rrbracket (O) = 0$ and $\text{Post} \llbracket E' \rrbracket (O) = 0$.
But $\text{Expand} \llbracket \text{Subexpr}_{E'} \llbracket a_{\text{root}} \rrbracket \rrbracket (S) = 0$ is a contradiction since

$$\text{Expand} \llbracket \text{Subexpr}_{E'} \llbracket a_{\text{root}} \rrbracket \rrbracket (S) = \text{Expand} \llbracket E' \rrbracket (S) = \text{Prior} \llbracket E' \rrbracket > 0.$$

Next, consider $\text{Post} \llbracket E \rrbracket (O) \cdot \mathcal{T}(O, E \to E') > 0$ and $\text{Post} \llbracket E' \rrbracket (O) \cdot \mathcal{T}(O, E' \to E) > 0$. It suffices to show that $\text{Post} \llbracket E \rrbracket (O) \cdot |A_{E'}| \cdot \mathcal{T}(O, E \to E'; a) = \text{Post} \llbracket E' \rrbracket (O) \cdot |A_E| \cdot \mathcal{T}(O, E' \to E; a)$ for all $a \in A_E \cap A_{E'}$. If $E = E'$, then this statement is vacuously true. If $E \neq E'$, there are two cases:

Case 1. If $\text{Sever}_a \llbracket E \rrbracket = \text{Sever}_a \llbracket E' \rrbracket = (N_i, E_{\text{sev}})$ for some i and E_{sev}, then it suffices to show that

$$\begin{aligned} \text{Post} \llbracket E \rrbracket (O) \cdot |A_{E'}| \cdot \text{Expand} \llbracket \text{Subexpr}_a \llbracket E' \rrbracket \rrbracket (N_i) \cdot \alpha(E, E') \qquad (3.52) \\ = \text{Post} \llbracket E' \rrbracket (O) \cdot |A_E| \cdot \text{Expand} \llbracket \text{Subexpr}_a \llbracket E \rrbracket \rrbracket (N_i) \cdot \alpha(E', E) \qquad (3.53) \end{aligned}$$

Both sides are nonzero because

$$\text{Post}\,[\![E]\!]\,(O) > 0 \implies \text{Likelihood}\,[\![E]\!]\,(O) > 0 \implies \alpha(E', E) > 0 \tag{3.54}$$

by Proposition 3.47, and similarly for Post $[\![E']\!]\,(O) > 0$. It suffices to show that

$$\begin{aligned}\frac{\alpha(E, E')}{\alpha(E', E)} &= \frac{\text{Post}\,[\![E']\!]\,(O) \cdot |A_E| \cdot \text{Expand}\,[\![\text{Subexpr}_a\,[\![E]\!]]\!]\,(N_i)}{\text{Post}\,[\![E]\!]\,(O) \cdot |A'_E| \cdot \text{Expand}\,[\![\text{Subexpr}_a\,[\![E']\!]]\!]\,(N_i)} &(3.55)\\ &= \frac{\text{Prior}\,[\![E']\!] \cdot \text{Likelihood}\,[\![E']\!]\,(O) \cdot |A_E| \cdot \text{Expand}\,[\![\text{Subexpr}_a\,[\![E]\!]]\!]\,(N_i)}{\text{Prior}\,[\![E]\!] \cdot \text{Likelihood}\,[\![E]\!]\,(O) \cdot |A'_E| \cdot \text{Expand}\,[\![\text{Subexpr}_a\,[\![E']\!]]\!]\,(N_i)} &(3.56)\\ &= \frac{\text{Likelihood}\,[\![E']\!]\,(O) \cdot |A_E|}{\text{Likelihood}\,[\![E]\!]\,(O) \cdot |A'_E|}, &(3.57)\end{aligned}$$

where the last uses Proposition 3.49 to expand Prior $[\![E']\!]$, followed by cancellation of factors. Note that if $\alpha(E, E') < 1$ then $\alpha(E', E) = 1$ and

$$\frac{\alpha(E, E')}{\alpha(E', E)} = \frac{\alpha(E, E')}{1} = \frac{\text{Likelihood}\,[\![E']\!]\,(O) \cdot |A_E|}{\text{Likelihood}\,[\![E]\!]\,(O) \cdot |A'_E|}. \tag{3.58}$$

If $\alpha(E', E) < 1$ then $\alpha(E, E') = 1$ and

$$\frac{\alpha(E, E')}{\alpha(E', E)} = \frac{1}{\alpha(E', E)} = \frac{\text{Likelihood}\,[\![E']\!]\,(O) \cdot |A_E|}{\text{Likelihood}\,[\![E]\!]\,(O) \cdot |A'_E|}. \tag{3.59}$$

If $\alpha(E, E') = 1 = \alpha(E', E)$ then $\alpha(E, E')/\alpha(E', E) = 1 = \alpha(E, E')$.

Case 2. If $\text{Sever}_a\,[\![E]\!] \neq \text{Sever}_a\,[\![E']\!]$, then $\mathcal{T}(O, E \to E'; a) = \mathcal{T}(O, E' \to E; a) = 0$.

If $E \neq E'$, then $\mathcal{T}(O, E \to E'; a) = \text{Prior}\,[\![\text{Subexpr}_a\,[\![E']\!]]\!] \cdot \alpha(E, E')$. Finally, posterior invariance follows from detailed balance:

$$\textstyle\sum_{E\in\mathcal{L}} \text{Post}\,[\![E]\!]\,(O)\mathcal{T}(O, E \to E') = \sum_{E\in\mathcal{L}} \text{Post}\,[\![E']\!]\,(O)\mathcal{T}(O, E' \to E) = \text{Post}\,[\![E']\!]\,(O). \tag{3.60}$$

■

Lemma 3.51. *Algorithm 3.3 satisfies Condition 3.16 (irreducibility).* «

Proof. It suffices to show that for all $E \in \mathcal{L}$ and for all $E' \in \mathcal{L}$ such that Post $[\![E']\!]\,(O) > 0$, E' is reachable from E in one step, i.e., $\mathcal{T}(O, E \to E') > 0$. By the definition of a_{root}, it holds that for all $E, E' \in \mathcal{L}$, $a_{\text{root}} \in A_E \cap A_{E'}$ and $\text{Sever}_{a_{\text{root}}}\,[\![E]\!] = \text{Sever}_{a_{\text{root}}}\,[\![E']\!] = (S, \triangle)$. Further, since Post $[\![E']\!]\,(O) > 0$, it must hold that Prior $[\![E']\!] = \text{Expand}\,[\![E']\!]\,(S) > 0$ and Likelihood $[\![E']\!]\,(O) > 0$, which together imply $\alpha(E, E') > 0$. Finally,

$$\mathcal{T}(O, E \to E') \geq \frac{1}{|A_E|} \cdot \mathcal{T}(O, E \to E'; a) \geq \text{Expand}\,\left[\!\left[E'\right]\!\right]\,(S) \cdot \alpha(E, E') > 0. \tag{3.61}$$

■

Lemma 3.52. *Algorithm 3.3 satisfies Condition 3.17 (aperiodicity).* «

Proof. For all $E \in \mathcal{L}$,

$$\mathcal{T}(O, E \to E) \geq (1/|A_E|) \cdot \text{Expand}\,[\![E]\!]\,(S) = (1/|A_E|) \cdot \text{Prior}\,[\![E]\!] > 0 \tag{3.62}$$

The inequality derives from choosing only $a = a_{\text{root}}$ in the sum over $a \in A_E$ in Eq. (3.47). ■

(r_{11})	N_1 ::= `(NoisyGP` N_2 N_3`)`	
(r_{21})	N_2 ::= `(Noise □)`	$(\Theta_{21}, \Sigma_{21}, \mu_{21}) = (\mathbb{R}, B(\mathbb{R}), \text{Gamma}(1,1))$
(r_{31})	N_3 ::= `(Constant □)`	$(\Theta_{31}, \Sigma_{31}, \mu_{31}) = (\mathbb{R}, B(\mathbb{R}), \text{Uniform}(0,10))$
(r_{32})	| `(Linear □)`	$(\Theta_{32}, \Sigma_{32}, \mu_{32}) = (\mathbb{R}, B(\mathbb{R}), \text{Uniform}(-10,10))$
(r_{33})	| `(Smooth □)`	$(\Theta_{33}, \Sigma_{33}, \mu_{33}) = (\mathbb{R}, B(\mathbb{R}), \text{Uniform}(0,10))$
(r_{34})	| `(Periodic □ □)`	$(\Theta_{34}, \Sigma_{34}, \mu_{34}) = (\mathbb{R}^2, B(\mathbb{R}^2), \text{Uniform}(0,10) \cdot \text{Uniform}(0,10))$
(r_{35})	| `(*` N_3 N_3`)`	
(r_{36})	| `(+` N_3 N_3`)`	
(r_{37})	| `(Changepoint □ □` N_3 N_3`)`	$(\Theta_{37}, \Sigma_{37}, \mu_{37}) = (\mathbb{R}^2, B(\mathbb{R}^2), \text{Uniform}(-10,10) \cdot \text{Uniform}(0,10))$

Listing 3.2: Context-free grammar expressing the Gaussian process DSL for univariate time series.

Lemmas 3.50–3.52 together establish that the transition operator in Algorithm 3.3 satisfies Conditions 3.15–3.17, which makes it a sound transition operator for synthesis using MCMC (Section 3.3.1) and resample-move SMC (Section 3.3.2) in any probabilistic DSL defined by a context-free grammar.

3.5 Formalizing the Gaussian Process DSL

Chapter 2 demonstrated a probabilistic DSL for Gaussian process models of univariate time series data, whose formal definition using a context-free grammar is shown in Listing 3.2. The Prior semantics are as described in Section 3.4.3, which applies directly to any probabilistic DSL defined by a context-free grammar. Recalling Definition 3.1, the input space $T ::= \biguplus_{n=1}^{\infty} \mathbb{R}^n$ consists of finite sequences of real numbers. For any input $t ::= (t_1, \dots, t_n) \in T$ the output space $\mathcal{X}_t ::= \mathbb{R}^n$ contains the corresponding time series values $x ::= (x_1, \dots, x_n)$, the sigma-algebra $\mathcal{F}_t ::= B(\mathbb{R}^n)$ and λ_t is the n-dimensional Lebesgue measure. The Likelihood semantics are

$$\text{Likelihood}\,[\![\texttt{(NoisyGP (Noise } \epsilon\texttt{)}\ K\texttt{)}]\!]\,(t,x) ::= \exp\left(-\frac{1}{2}\Big[x^\top W x - \log\left(\det(W)\right) - n\log\left(2\pi\right)\Big]\right), \tag{3.63}$$

where $K \in \mathcal{L}(G, N_3)$ is a covariance expression and W is the $n \times n$ covariance matrix encoded by (ϵ, K) whose entries $W_{ij} ::= [\![K]\!]\,(t_i, t_j) + \mathbf{1}[t = t'](\epsilon + \epsilon_{\min})$ $(1 \le i, j \le n)$ are obtained from the covariance semantics in Listing 2.2. The parameter $\epsilon_{\min} > 0$ is a "noise floor" that ensure the likelihood is bounded. The next two lemmas establish that the Gaussian process DSL satisfies the conditions needed for Bayesian synthesis described in Problem 3.13 to be well defined.

Proposition 3.53. *The* Prior *and* Likelihood *semantics of the Gaussian process DSL satisfy Conditions 3.6–3.9.* «

Proof for Condition 3.6. Since the non-terminals N_1 and N_2 in Listing 3.2 are non-recursive, it suffices to show that the language generated by the production rule N_3 satisfies the condition for consistency in Theorem 3.35. A Chomsky normal form of this production rule is

$$\begin{aligned}
N_3 &\to \underset{7/40}{H_1H_0} \quad | \underset{7/40}{H_2H_0} \quad | \underset{7/40}{H_3H_0} \quad | \underset{7/40}{H_4H_0} \quad | \underset{4/40}{H_5N_3} \quad | \underset{4/40}{H_6N_3} \quad | \underset{4/40}{H_7N_3}\\
H_0 &\to \square\\
H_1 &\to \texttt{Constant}\\
H_2 &\to \texttt{Linear}\\
H_3 &\to \texttt{Smooth}\\
H_4 &\to H_8H_0\\
H_5 &\to H_9N_3\\
H_6 &\to H_{10}N_3\\
H_7 &\to H_{11}N_3\\
H_8 &\to \texttt{Periodic}\\
H_9 &\to \texttt{*}\\
H_{10} &\to \texttt{+}\\
H_{11} &\to H_{12}H_0\\
H_{12} &\to H_{13}H_0\\
H_{13} &\to \texttt{ChangePoint}.
\end{aligned}$$

Following the notation from Gecse and Kovács [2010, Eq. (2)], define:

$$m_{ij} = \textstyle\sum_{k=1}^{r_i} p_{ik}n_{ikj}, \text{ where} \qquad \begin{aligned} r_i &::= \text{ number of productions with } N_i \in N \text{ premise,}\\ p_{ik} &::= \text{ probability assigned to production } k \text{ of } N_i,\\ n_{ikj} &::= \text{ number of occurrence of } N_j \in N \text{ in production } k \text{ of } N_i. \end{aligned}$$

The expectation matrix $M ::= [m_{ij}]$ is therefore

$$\begin{array}{c|ccccccccccccccc|}
 & N_3 & H_0 & H_1 & H_2 & H_3 & H_4 & H_5 & H_6 & H_7 & H_8 & H_9 & H_{10} & H_{11} & H_{12} & H_{13}\\
N & 12/40 & 28/40 & 7/40 & 7/40 & 7/40 & 7/40 & 4/40 & 4/40 & 4/40 & 0 & 0 & 0 & 0 & 0 & 0\\
H_0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
H_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
H_2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
H_3 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
H_4 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0\\
H_5 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0\\
H_6 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\\
H_7 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0\\
H_8 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
H_9 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
H_{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\\
H_{11} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0\\
H_{12} & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1\\
H_{13} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}.$$

The nonzero eigenvalues of M are $0.71789089\ldots$ and $-0.41789089\ldots$ which lie in the interval $[-1, 1]$. The conclusion follows from Theorem 3.35. ■

Proof for Conditions 3.7 and 3.8. Since $\epsilon_{\min} > 0$, the Likelihood semantic function is the density of a non-degenerate multivariate normal, which is normalized and has full support over its domain. ■

Proof for Condition 3.9. Let $K \in \mathcal{L}(G, N_3)$ be a covariance expression, $C ::= [\llbracket K \rrbracket (t_i, t_j)]_{i,j=1}^{n}$ be the $n \times n$ covariance matrix induced by K, and I the $n \times n$ identity matrix. Eq. (3.63) is then

$$
\begin{aligned}
&\text{Likelihood} \llbracket \texttt{(NoisyGP (Noise } \epsilon\texttt{)}\ K\texttt{)} \rrbracket (t, x) \\
&::= \exp\left(-\frac{1}{2}\Big[x^{\top}[C + (\epsilon + \epsilon_{\min})I]^{-1}x - \log\left(\det(C + (\epsilon + \epsilon_{\min})I)\right) - n\log(2\pi) \Big] \right) && (3.64)\\
&= ((2\pi)^n \det(C + (\epsilon + \epsilon_{\min})I))^{-1/2} \exp\left(x^{\top}[C + (\epsilon + \epsilon_{\min})I]^{-1}x \right) && (3.65)\\
&\le ((2\pi)^n \det(C + (\epsilon + \epsilon_{\min})I))^{-1/2} && (3.66)\\
&\le (\det(C + (\epsilon + \epsilon_{\min})I))^{-1/2} && (3.67)\\
&\le (\det(C) + \det((\epsilon + \epsilon_{\min})I))^{-1/2} && (3.68)\\
&\le (\det((\epsilon + \epsilon_{\min})I))^{-1/2} && (3.69)\\
&= \sqrt{\epsilon + \epsilon_{\min}} && (3.70)\\
&\le \sqrt{\epsilon_{\min}}, && (3.71)
\end{aligned}
$$

where Eq. (3.66) follows from the positive semi-definiteness of $C + (\epsilon + \epsilon_{\min})I$ and the identity $e^{-z} < 1$ for $z > 0$; Eq. (3.67) from $(2\pi)^{-n/2} < 1$; Eq. (3.68) from Minkowski's determinant inequality [Marcus and Minc, 1992, Theorem 4.1.8]; Eq. (3.69) from positive semi-definiteness of C; Eq. (3.71) from $\epsilon > 0$. As (K, ϵ) are arbitrary and the upper bound $\sqrt{\epsilon_{\min}}$ is independent of these terms, the result follows. ■

Since the Gaussian process DSL is generated by a context-free grammar, the transition operator from Algorithm 3.3 is sound by Lemmas 3.50–3.52 and can be used within the MCMC (Algorithm 3.1) and the SMC (Algorithm 3.2) synthesis procedures described in Section 3.3.

3.5.1 Time Complexity Analysis

Each iteration of the transition operator in Algorithm 3.3 takes as an input existing DSL expression E ::= `(NoisyGP (Noise` ϵ`)` K`)` and proposes a new subexpression E' ::= `(NoisyGP (Noise` ϵ'`)` K'`)` by applying the following operations:

1. Severing the parse tree at a randomly selected node using Sever (Algorithm 3.3, lines 2 and 3). The cost is linear in the number of subexpressions in E, which is denoted $|E|$.
2. Sampling the PCFG using Expand (3.41) (Algorithm 3.3, line 4). The expected cost is linear in the average length ℓ_G of strings generated by the PCFG, which can be determined from the transition probabilities [Gecse and Kovács, 2010, Eq. (3)]).
3. Assessing the likelihood under the existing and proposed expressions using Likelihood $\llbracket E \rrbracket (O)$ (Algorithm 3.3, line 6). For a Gaussian process with n observed time points, the cost of constructing the covariance matrix $[\llbracket K \rrbracket (t_i, t_j)]_{i,j=1}^{n}$ is $O(|K|n^2)$ and computing the inverse and determinant is $O(n^3)$.

The overall time complexity of an iteration of MCMC or rejuvenation step in SMC is thus $O(\ell_G + \max(|K|, |K'|)n^2 + n^3)$. This term is typically dominated by the n^3 cost of assessing Likelihood. There is a broad literature in sparse approximations to Gaussian processes that reduce this complexity to $O(m^2 n)$ where $m \ll n$ is a parameter representing the size of a subsample of the full data to be used [Rasmussen and Williams, 2006, Chapter 8]. These approximations trade off predictive accuracy with substantial improvements in scalability. Quiñonero-Candela and Rasmussen [2005] show that many sparse Gaussian process approximations correspond to well-defined probabilistic models that have different likelihood functions than the standard Gaussian process, and so the Bayesian synthesis framework can naturally incorporate sparse Gaussian processes by adapting the Likelihood semantics. As for the quadratic factor $O(\max(|K|, |K'|)n^2)$, this scaling depends largely on the observed data: simpler patterns can be described fewer components in the covariance expression whereas complex patterns require larger covariance expressions.

3.6 Related Work

Synthesizing programs from data is a longstanding activity in computer science. This chapter has established the formal foundations for Bayesian synthesis of programs in both general and context-specific probabilistic DSLs. Chapter 2 demonstrated Bayesian synthesis for univariate time series data in a probabilistic context-free DSL and Chapters 4–6 present Bayesian synthesis techniques in context-sensitive DSLs for cross-sectional data, multivariate time series, and relational data. It should be noted that the terminology of "Bayesian synthesis" in this thesis is to be understood in the sense used in programming languages, and should not be confused with unrelated uses of the term such as in Givens et al. [1994], Green et al. [2000], McAlinn and West [2019], and others.

This section gives a broad literature overview of related work in five related areas: (i) Bayesian synthesis of probabilistic programs where the formalism and proofs in this chapter, a subset of which appear in Saad et al. [2019a], are the first; (ii) non-Bayesian synthesis of probabilistic programs; (iii) probabilistic synthesis of non-probabilistic programs; (iv) non-probabilistic synthesis of non-probabilistic programs; (v) probabilistic structure discovery in machine learning.

Bayesian Synthesis of Probabilistic Programs

Within the literature of Bayesian synthesis, the main contributions of this chapter include

(1) Formalizing probabilistic domain-specific data modeling languages, where each expression specifies the structure and parameters of a given model within a family of probabilistic models for a given domain.

(2) Identifying sufficient conditions on the prior and likelihood semantics of probabilistic expressions needed to obtain a well-defined posterior distribution over DSL programs.

(3) Identifying sufficient conditions to obtain a sound Bayesian synthesis algorithm.

(4) Defining a general family of domain-specific languages generated by context-free grammars with default semantics that ensure the posterior distribution is well defined.

(5) Presenting sound synthesis algorithms using Markov chain Monte Carlo and sequential Monte Carlo that apply to any language generated by a context-free grammar.

(6) Presenting a specific domain-specific language—the Gaussian process language for modeling time series data reviewed in Chapter 2—that is verified to satisfy the required soundness conditions.

Nori et al. [2015] introduce PSKETCH, a system designed to complete partial sketches of probabilistic programs for modeling tabular data. The technique is based on applying sequences of program mutations to a base program written in a probabilistic sketching language. Specifically, Nori et al. [2015] propose to use Metropolis-Hastings sampling obtain maximum a posteriori (MAP) solutions to the sketching problem, where the data likelihood is approximated using a mixture of Gaussians. However, the paper does not establish that the prior or posterior distributions on programs are well defined.

In particular, PSKETCH is formalized to use a uniform prior distribution over an unbounded space of programs, even though there is no valid uniform probability measure over this space. Because the posterior is defined using the prior and the marginal likelihood of all datasets is not shown to be finite, the posterior distribution over sketches is also not well defined. The paper also asserts that the synthesis algorithm converges because the MH algorithm always converges, but it does not establish that the PSKETCH framework satisfies the properties required for MH to converge. And because the posterior is not well defined, there is no probability distribution to which the MH algorithm can converge. Moreover, while the paper claims to use MH to determine whether a proposed program mutation is accepted or rejected, there is no tractable algorithm that can be used to compute the reverse probability of going back from a proposed program to an existing program, which means that MH cannot be used with the program mutation proposals described in the paper.

Moreover, the PSKETCH system in Nori et al. [2015] is based on a probabilistic sketching language with constructs drawn from general-purpose programming languages, including such as if-then-else, for loops, variable assignment, and arithmetic operations. Working with these general constructs produces a search space that is too unconstrained to yield practical solutions to real-world data modeling problems, especially in the absence of other sources of information that can more effectively narrow the search. As with traditional synthesis, domain-specific languages such as the ones presented in this thesis make it possible to effectively solve automatic data modeling problems. For example, while the PSKETCH language in Nori et al. [2015] contains general-purpose programming constructs, it does not have domain-specific constructs that compactly encapsulate Gaussian processes or valid covariance functions.

Hwang et al. [2011] use beam search over programs in a subset of the Church probabilistic programming language [Goodman et al., 2008a] for synthesizing tree-like s-expressions. The resulting search space is so unconstrained that, as the authors note in the conclusion, the synthesis technique does not easily scale real-world problems. This drawback highlights the benefit of controlling the search space via an appropriate domain-specific data modeling language. Although Hwang et al. [2011] also use the vocabulary of Bayesian synthesis, the proposed program-length prior over an unbounded program space is not shown to always be probabilistically well formed and may not lead to a valid Bayesian posterior distribution over programs. In addition, the stochastic approximation of the program likelihood does not result in a sound synthesis algorithm.

Several papers in the cognitive science literature have used Bayesian inference to synthesize expressions in probabilistic domain-specific languages of intuitive theories [Ullman and Tenenbaum, 2020]. For example, Goodman et al. [2008b] synthesize logical rules in a "concept language" generated by a context-free grammar; Ullman et al. [2012] synthesize Horn clauses generated by a probabilistic Horn clause grammar; and Ullman et al. [2018] synthesize Church expressions that define rudimentary rules of intuitive physics. Cranefield and Dhiman [2021] apply the techniques in this chapter to synthesize programs for identifying social norms that govern agent interactions. Inference in these works is performed using Markov Chain Monte Carlo sampling. Many of these works can be seen as specific instantiations of the general framework presented in Section 3.4 for Bayesian synthesis over probabilistic DSLs generated by context-free languages.

Non-Bayesian Synthesis of Probabilistic Programs

Ellis et al. [2015] introduce a method for synthesizing probabilistic programs by using SMT solvers to optimize the likelihood of the observed data with a regularization term that penalizes the program length. The research works with a domain-specific language for morphological rule learning. Perov and Wood [2014] propose to synthesize code for simple random number generators using approximate Bayesian computation. These two techniques are fundamentally different from the Bayesian synthesis problem presented in this chapter. Neither technique is based on Bayesian inference and neither presents soundness proofs or states a soundness property. Moreover, both approaches attempt to find a single highest-scoring program as opposed to synthesizing ensembles of programs that approximate a posterior distribution over probabilistic programs. As this previous research demonstrates, obtaining soundness proofs or characterizing the uncertainty of the synthesized programs against the data generating process is particularly challenging for non-Bayesian approaches.

Lake et al. [2015] learn probabilistic programs that solve challenging one-shot learning of visual concepts for recognizing handwritten characters. Due to the sophisticated nature of the prior distribution over probabilistic programs that produce handwritten characters, inference is performed using a combination of discrete approximations, continuous optimization, and MCMC sampling [Lake et al., 2015, supplementary material, Section S3], which together trade off soundness for runtime performance. Guimerá et al. [2020] use a Bayesian framework for synthesizing mathematical expressions of noisy regression functions and apply the technique to discover models from finance, systems biology, and physics data. The prior over mathematical expressions is estimated from a corpus of 4080 expressions mined from Wikipedia. The likelihood that an expression assigns to observed data is approximated using the Bayesian information criterion (BIC) heuristic. It is fruitful to explore resample-move sequential Monte Carlo learning algorithms from Section 3.3.2 for more principled, scalable, and fully-Bayesian probabilistic program synthesis in these important problems.

Tong and Choi [2016] describe a method which uses the off-the-shelf model discovery system from Lloyd [2014] to learn Gaussian process models and the render the models in the Stan [Carpenter et al., 2017] probabilistic programming language. However, the approach in Tong and Choi [2016] does not formalize the program synthesis problem; nor does it present any formal semantics; nor is it able to extract posterior probabilities that qualitative properties hold in the data; nor does it apply to multiple model families in a single problem domain, let alone multiple problem domains. Chasins and Phothilimthana [2017] present a technique for synthesizing probabilistic programs of tabular datasets in a subset of BLOG [Milch et al., 2005]. Unlike the probabilistic DSL for cross-sectional tabular data from Chapter 4, this approach requires the user to manually specify a causal ordering among the variables. This knowledge is rarely available for real-world data, and it is not obvious how to infer causal orderings from observational data. Second, the technique of Chasins and Phothilimthana [2017] is based on using linear correlation, which fail adequately capture complex relationships. Finally, the synthesis technique uses simulated annealing not Bayesian learning, has no probabilistic interpretation, approximates the program likelihood using mixtures of Gaussians, and does not provide a notion of soundness.

Probabilistic Synthesis of Non-probabilistic Programs

Schkufza et al. [2013] describe a technique for synthesizing high-performance X86 binaries by using MCMC to stochastically search through a space of programs. The output is a single X86 program which satisfies the hard constraint of correctness and the soft constraint of performance improvement. While both the Bayesian synthesis framework in this chapter and the superoptimization technique from Schkufza et al. [2013] can leverage MCMC algorithms to implement the synthesis, they use MCMC in a fundamentally different way. In Schkufza et al. [2013], MCMC is used to search through a space of deterministic programs that seek to minimize a cost function that has no probabilistic semantics. In contrast, Bayesian synthesis uses MCMC (or SMC) to approximately sample from a space of probabilistic programs whose semantics specify a well-defined posterior distribution over programs.

Bayesian approaches for sampling from a posterior distribution over deterministic programs have also been investigated. Liang et al. [2010] use adapter grammars [Johnson et al., 2006] to build a hierarchical nonparametric Bayesian prior over programs specified in combinatory logic [Schönfinkel, 1924]. Ellis et al. [2016] describe a method to sample from a bounded program space with a uniform prior where the posterior probability of a program is equal to zero if it does not satisfy an observed input-output constraint, or geometrically decreasing in its length otherwise. This method is used to synthesize arithmetic operations, text editing routines, and list manipulation programs. Both Liang et al. [2010] and Ellis et al. [2016] specify prior distributions over programs similar to the prior in this chapter. However, these two techniques assume that the synthesized programs have deterministic input-output behavior and cannot be easily extended to synthesize programs that have probabilistic input-output behavior.

Lee et al. [2018] present a technique to speed up program synthesis of non-probabilistic programs, where A* search is used to enumerate programs that satisfy a set of input-output constraints in order of decreasing prior probability. The prior distribution over programs is itself learned using a probabilistic higher-order grammar. The technique is used to synthesize programs in domain-specific languages for bit-vector, circuit, and string manipulation tasks. Similar to the Bayesian synthesis framework, Lee et al. [2018] use PCFG priors to specify domain-specific languages. However the fundamental differences are that the synthesized programs in Lee et al. [2018] are non-probabilistic and the objective is to enumerate valid programs sorted by their prior probability, while in Bayesian synthesis the programs are probabilistic and the objective is to sample programs according to their posterior probabilities.

Non-probabilistic synthesis of non-probabilistic programs

Over the last decade program synthesis has become a highly active area of research in programming languages, and a full literature review is too vast to summarize here. Key techniques include deductive logic with program transformations [Burstall and Darlington, 1977, Manna and Waldinger, 1979, 1980], genetic programming [Koza, 1992, Koza et al., 1997], solver and constraint-based methods [Solar-Lezama et al., 2006, Gulwani et al., 2011, Gulwani, 2011, Alur et al., 2013, Feser et al., 2015, Kim et al., 2021], and neural networks [Graves et al., 2014, Reed and de Freitas, 2016, Balog et al., 2017, Chaudhuri et al., 2021]. These approaches have generally focused on areas where uncertainty is not essential or relevant to the problem being solved. Synthesis tasks in these works apply to programs that exhibit deterministic input-output behavior, typically for discrete problem domains. The usual formulation is to define an “optimal” solution and the goal of program synthesis to find this solution or some approximation of it. In contrast, in Bayesian probabilistic program synthesis, the problem domain is fundamentally about automatically learning models of non-deterministic data generating processes given a set of noisy observations. Given this characteristic of the problem domain, the solutions from Bayesian synthesis capture uncertainty at two levels. First, each synthesized probabilistic program exhibits noisy, non-deterministic input-output behavior. Second, the ensemble of synthesized programs characterizes uncertainty over the structure and parameters of expressions in the DSL, which is inherent in real-world empirical phenomenon.

Probabilistic Structure Discovery

Researchers have developed several probabilistic techniques for discovering statistical model structures from observed data [Tenenbaum et al., 2011]. Examples include Bayesian network structures [Friedman and Koller, 2003, Mansinghka et al., 2006]; sparse deep graphical models [Adams et al., 2010]; matrix-composition models [Grosse et al., 2012]; multivariate data tables [Mansinghka et al., 2016], further discussed in Chapter 4; univariate time series [Wilson and Adams, 2013, Duvenaud et al., 2013]; multivariate time series [Saad and Mansinghka, 2018], further discussed in Chapter 5; relational data [Kemp et al., 2006, Saad and Mansinghka, 2021], further discussed in Chapter 6. Several of these methods also approach the structure learning problem in terms of a hierarchical Bayesian inference problem. The formalism presented in this chapter helps explain, formalize, and unify these methods in terms of synthesizing programs in probabilistic domain-specific data modeling languages. The formalisms establish general conditions for Bayesian synthesis over structured expressions to be well defined and introduces sound synthesis algorithms that apply to a broad class of DSLs produced by context-free grammars. The learned model ensemble can then be used to compute the approximate posterior probability that various structures are present or absent in the data and a full distribution over predicted values. This capability rests on a distinctive aspect of the presented formalism, namely that soundness is defined in terms of sampling programs from a distribution that converges to the Bayesian posterior, as opposed to finding a single “highest scoring” program. These capabilities improve upon the Gaussian process learning technique in Duvenaud et al. [2013], for example, which leverages greedy search for inferring covariance structure, does not support online learning with streaming data, and does not rest on any notion of soundness. The workflow in Figure 3.1 shows one way that probabilistic programming languages makes structure discovery techniques more usable. Specifically, the proposed approach is to translate programs from domain-specific languages to probabilistic programming languages (such as SPPL, described in Chapter 7) that have built-in constructs for solving probabilistic inference queries, instead of developing new inference engines for each model class.

Chapter 4

Synthesizing Models for Cross-Sectional Data

Dilbert	“We need 3 more programmers.”
Boss	“Use agile programming methods.”
Dilbert	“Agile programming doesn’t just mean doing more work with fewer people.”
Boss	“Find me some words that *DO* mean that and ask again.”

Scott Adams

This chapter describes a probabilistic domain-specific modeling language for cross-sectional data tables called MultiMixture. Programs in this DSL specify generative models of data tables, where rows represent different “individuals” and columns represent “variables” or “attributes”. A central advantage of learning generative models for data tables is that they can be reused multiple times to automate many data analysis tasks that repeatedly arise in applications, for example: (i) computing the probability of a predicate on the variables; (ii) imputing likely values of missing cells; (iii) quantifying mutual information between groups of variables, possibly conditioned on others; (iv) finding individuals that are similar to a given record of interest; (v) identifying anomalous entries; and (vi) generating realistic synthetic data. A second advantage is that all queries are formally represented as functionals (Eqs. (3.9) and (3.10)) of a coherent joint probability distribution over model structure, latent variables, and data. This approach avoids the need for expensive hypothesis tests of significance or multiple testing corrections across many queries as in frequentist analyses.

A key challenge in modeling cross-sectional data tables is designing a family of models that is flexible enough to emulate a broad range of data generating process while also being tractable enough for Bayesian synthesis and fast querying. Traditional approaches based on learning the structure of Bayesian networks [Friedman and Koller, 2003, Daly et al., 2011] specify a vastly unconstrained space of directed graphical models that is computationally infeasible to search effectively. In addition, the resulting Bayesian network models cannot always be queried efficiently. The approach taken here is to instead synthesize generative models in a domain-specific language for sum-product networks [Poon and Domingos, 2011], which belong to the class of probabilistic circuits [Darwiche, 2021] that specify “deep” probabilistic mixture models. The MultiMixture DSL goes beyond previous work in structure learning for sum-product networks [Gens and Domingos, 2013, Vergari et al., 2019, Trapp et al., 2019] by introducing a Bayesian nonparametric prior over the structure. The variables are first nonparametrically divided into independent groups (Product node) using a Chinese restaurant process and the rows are then nonparametrically divided (Sum nodes) using Dirichlet process mixture models.

Syntax Rules

$$
\begin{aligned}
n &::= \text{number of rows}\\
m &::= \text{number of columns}\\
&a \in \{1,\dots,m\} \quad s \in \{1,\dots,n\} \quad \mu \in \mathbb{R} \quad \sigma \in \mathbb{R}_{>0} \quad \gamma \in \mathbb{R}_{>0} \quad w \in [0,1]\\
P &::= \texttt{(factorize } M_1\texttt{)} \mid \texttt{(factorize } M_1\ M_2\texttt{)} \mid \cdots \mid \texttt{(factorize } M_1\ \dots\ M_m\texttt{)}\\
B &::= \texttt{(sum } C_1\texttt{)} \mid \texttt{(sum } C_1\ C_2\texttt{)} \mid \cdots \mid \texttt{(sum } C_1\ \dots\ C_n\texttt{)}\\
C &::= \texttt{(product } s\ V_1\texttt{)} \mid \texttt{(product } s\ V_1\ V_2\texttt{)} \mid \cdots \mid \texttt{(product } s\ V_1\ \dots\ V_m\texttt{)}\\
V &::= \texttt{(leaf } a\ D\texttt{)}\\
D &::= \texttt{(gaussian } \mu\ \sigma\texttt{)} \mid \texttt{(poisson } \gamma\texttt{)} \mid \texttt{(bernoulli } w\texttt{)} \mid \dots
\end{aligned}
$$

Prior Denotation

$$
\begin{aligned}
&\text{Prior}\,\llbracket\texttt{(factorize } M_1\ \dots\ M_k\texttt{)}\rrbracket && ::= 1/m!\prod_{i=1}^{k}\text{Prior}\,\llbracket M_k\rrbracket && (k=1,\dots,m)\\
&\text{Prior}\,\llbracket\texttt{(sum } C_1\ \dots\ C_k\texttt{)}\rrbracket && ::= (k-1)!/n!\prod_{i=1}^{k}\text{Prior}\,\llbracket C_k\rrbracket && (k=1,\dots,n)\\
&\text{Prior}\,\llbracket\texttt{(product } s\ V_1\ \dots\ V_k\texttt{)}\rrbracket && ::= (s-1)!\prod_{i=1}^{k}\text{Prior}\,\llbracket V_i\rrbracket && (k=1,\dots,m)\\
&\text{Prior}\,\llbracket\texttt{(leaf } a\ D\texttt{)}\rrbracket && ::= \text{Prior}\,\llbracket D\rrbracket\\
&\text{Prior}\,\llbracket\texttt{(gaussian } \mu\ \sigma\texttt{)}\rrbracket && ::= \sqrt{\frac{\lambda}{\sigma^2 2\pi}}\,\frac{\beta^\alpha}{\Gamma(\alpha)}\left(\frac{1}{\sigma^2}\right)^{\alpha+1}\exp\left(\frac{-(2\beta+\lambda(\mu-\eta)^2)}{2\sigma^2}\right)\\
&\text{Prior}\,\llbracket\texttt{(poisson } \gamma\texttt{)}\rrbracket && ::= \frac{\xi^\nu\gamma^{\nu-1}e^{-\xi\gamma}}{\Gamma(\nu)}\\
&\text{Prior}\,\llbracket\texttt{(bernoulli } w\texttt{)}\rrbracket && ::= w^{\tau-1}(1-w)^{\kappa-1}/B(\tau,\kappa)
\end{aligned}
$$

where $\lambda, \alpha, \beta, \nu, \xi, \tau, \kappa$ are statistical constants

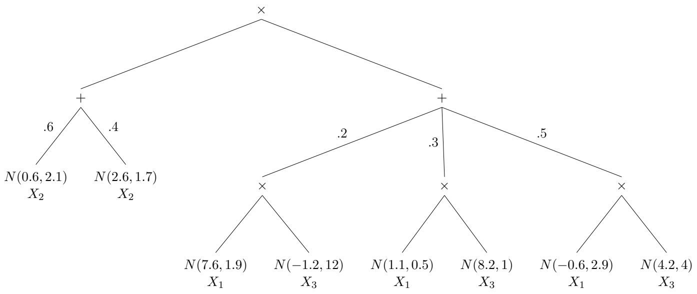
Likelihood Denotation

$$
\begin{aligned}
&\text{Likelihood}\,\llbracket\texttt{(factorize } M_1\ \dots\ M_k\texttt{)}\rrbracket\,((n,m),x) && ::= \prod_{i=1}^{k}\text{Likelihood}\,\llbracket M_k\rrbracket\,((n,m),x)\\
& && (k=1,\dots,m)\\
&\text{Likelihood}\,\llbracket\texttt{(sum } C_1\ \dots\ C_k\texttt{)}\rrbracket\,((n,m),x) && ::= \prod_{i=1}^{n}\sum_{j=1}^{k}\text{Likelihood}\,\llbracket C_j\rrbracket\,(x[i,:])\\
& && (k=1,\dots,n)\\
&\text{Likelihood}\,\llbracket\texttt{(product } s\ V_1\ \dots\ V_k\texttt{)}\rrbracket\,((n,m),x) && ::= \frac{s}{n}\prod_{i=1}^{k}\text{Likelihood}\,\llbracket V_i\rrbracket\,((n,m),x)\\
& && (k=1,\dots,m)\\
&\text{Likelihood}\,\llbracket\texttt{(leaf } a\ D\texttt{)}\rrbracket\,((n,m),x) && ::= \text{Likelihood}\,\llbracket D\rrbracket\,(x[1,a])\\
&\text{Likelihood}\,\llbracket\texttt{(gaussian } \mu\ \sigma\texttt{)}\rrbracket\,((n,m),x) && ::= \frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{x[1,1]-\mu}{\sqrt{2}\sigma}\right)^2\\
&\text{Likelihood}\,\llbracket\texttt{(poisson } \gamma\texttt{)}\rrbracket\,((n,m),x) && ::= \begin{cases}\gamma^{x[1,1]}e^{-\gamma}/(x[1,1]!) & \text{if } x[1,1]\in\mathbb{Z}_{\geq 0}\\ 0 & \text{otherwise}\end{cases}\\
&\text{Likelihood}\,\llbracket\texttt{(bernoulli } w\texttt{)}\rrbracket\,((n,m),x) && ::= \begin{cases}w^{x[1,1]}(1-w)^{(1-x[1,1])} & \text{if } x[1,1]\in\{0,1\}\\ 0 & \text{otherwise}\end{cases}
\end{aligned}
$$

Listing 4.1: MultiMixture DSL for cross-sectional data tables. This DSL is context-sensitive as the constituent expressions are subject to several syntactic constraints described in the main text.

```
(factorize
 (sum
  (product 6 (leaf 2 (gaussian 0.6 2.1)))
  (product 4 (leaf 2 (gaussian 2.3 1.7))))
 (sum
  (product 2 (leaf 1 (gaussian 7.6 1.9)
             (leaf 3 (gaussian -1.2 12))))
  (product 3 (leaf 1 (gaussian 1.1 0.5)
             (leaf 3 (gaussian 8.2 1))))
  (product 5 (leaf 1 (gaussian -0.6 2.9)
             (leaf 3 (gaussian 4.2 4))))))
```

(a) DSL Program



(b) Sum-Product Network Representation

Figure 4.1: Representing a program in the MultiMixture DSL as a sum-product network.

4.1 MultiMixture DSL for Modeling Data Tables

Following the terminology of probabilistic DSLs from Definition 3.1, the input space T is all pairs $t ::= (n, m)$ of positive integers and the corresponding data space X_t is a matrix x with n columns and m rows. Each column c represents a distinct random variable and each row $[x[r, 1], \dots, x[r, m]]$ (for $r = 1, \dots, n$) represents a joint instantiation of all the m random variables, possibly with missing values. The DSL describes the data generating process for cells in this table using a variant of the "Cross-Categorization" model [Shafto et al., 2011, Mansinghka et al., 2016] which was originally introduced in the cognitive science and machine learning literature and later integrated into multiple Bayesian database systems [Mansinghka et al., 2015, Saad and Mansinghka, 2016a,b, 2017, Saad et al., 2017, Schaechtle et al., 2022].

Listing 4.1 shows the syntax , Prior denotation, and Likelihood denotation for the MultiMixture DSL. The syntax is parameterized by (n, m) to ensure the normalization property in Condition 3.6. Each program in this DSL represents a sum-product network [Poon and Domingos, 2011] with three levels, as shown in the example from Figure 4.1. Sum-product networks are a type of probabilistic graphical model that specify deep probabilistic mixtures—this class of models will be explained in Chapter 7. Briefly, each leaf node in a sum-product network represent a primitive random variable such as a normal, Poisson, or geometric distribution. Each sum node represents a probabilistic mixture model, whose weights are written along directed edges to the children. Each product node represents probabilistic factorization (independence). For a 3×3 data table, there are possibly structures

In terms of the sum-product network representation in Figure 4.1b, each DSL program specifies a different structure and set of parameters for modeling the data table, which together determine:

1. The number of branches under the product node at the root (between 1 and m).
2. An assignment of each of the m variables to exactly one subtree of the product at the root.
3. The number of branches under each sum node in the second level (between 1 and n).
4. The weights of the children of all the sum nodes.
5. The numeric parameters at the leaf nodes.

Figure 4.2 shows an enumeration of all 57 structures in the MultiMixture DSL for modeling a data table with $m = 3$ columns and $n = 3$ rows. More generally, the number of possible structures for an $n \times m$ data table is

$$\sum_{b \in B(m)} n^{|b|} = \sum_{k=1}^{m} \left\{ {m \atop k} \right\} n^k, \tag{4.1}$$

where $B(m)$ is the set of all partitions of $\{1, \dots, m\}$ and $\left\{ {m \atop k} \right\}$ is a Stirling number of the second kind.

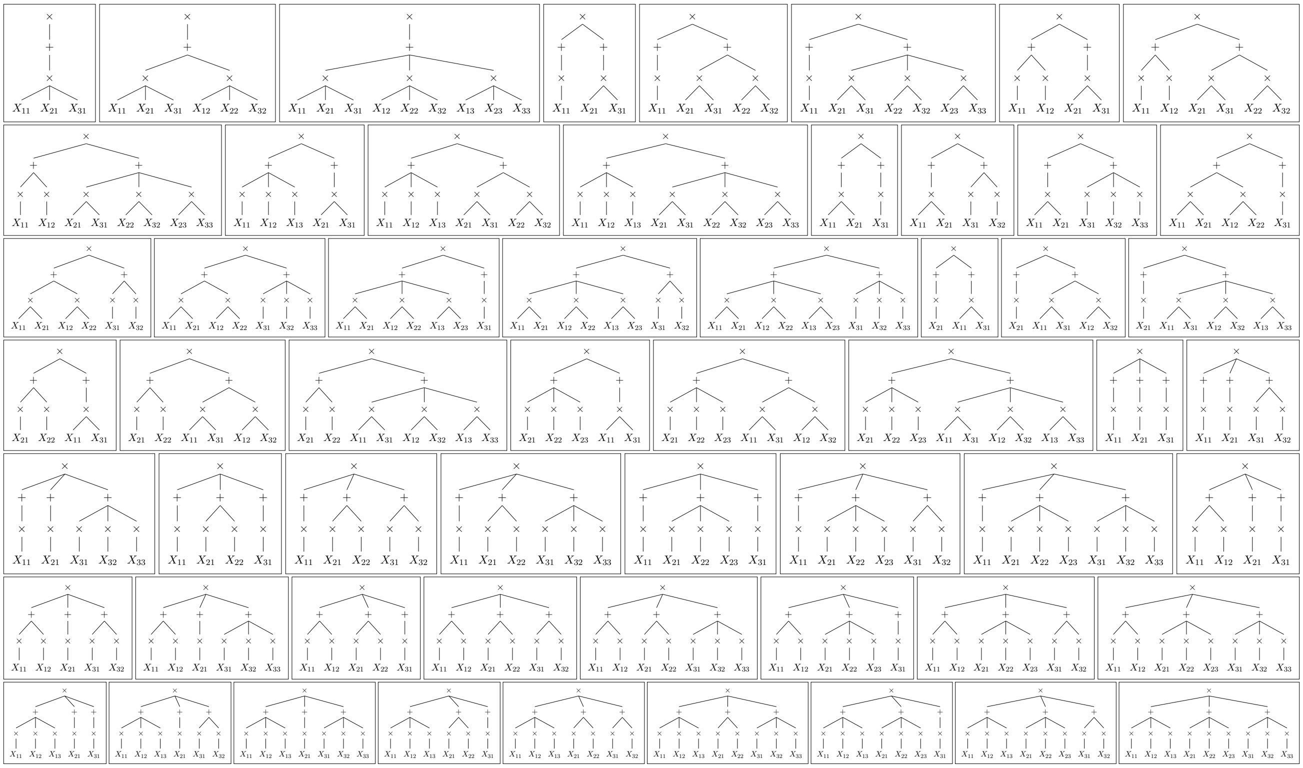


Figure 4.2: The space of all 57 structures in the MultiMixture DSL for a 3×3 cross-sectional data table.

Listing 4.1 shows only three primitive distributions, whereas the reference implementation (Section 1.4) contains eight primitives—Bernoulli, beta, categorical, exponential, Gaussian, geometric, Poisson, and von Mises—for modeling a range statistical data types. The Prior semantics recursively describes the joint probability of all terms in a MultiMixture expression. The Likelihood semantics decompose the full probability of an $n \times m$ data table x into the cell wise probabilities, computed by summing out each mixture component. MultiMixture improves on Cross-Categorization by using a sum-product network representation that removes exchangeable coupling between the rows, which makes the model easier to interpret and easier to translate into the SPPL system presented in Chapter 7.

4.2 Algorithms for Posterior Inference

Recall from Problem 3.13 that the goal of Bayesian synthesis is to generate expressions $E \in \mathcal{L}$ given an observation $O ::= (t, x)$. In MultiMixture, $t ::= (n, m)$ specifies the dimensions of the data table x. As discussed in Section 3.3, both MCMC and resample-move SMC algorithms for Bayesian synthesis require an implementation of the procedure $\textsc{Generate-New-Expression}(O, E)$, which returns a new expression E' from the current expression E and observation O. Figure 4.3 shows five examples of program mutation operators that are used to generate new expressions in the MultiMixture DSL. These mutations are interleaved over the course of inference. For the MultiMixture DSL, the likelihood ratio (3.24) can be efficiently computed without revisiting the entire data, by using standard properties of sampling-based inference for Dirichlet process mixture models [Escobar and West, 1995, Neal, 2000]. A formal description of the transition operators, which are based on the Gibbs kernels described in Mansinghka et al. [2016, Section 2.4], as well as correctness proofs are beyond the scope of this thesis.

```
(factorize
 (sum
  (product 6 (leaf 1 (gaussian 0.6 2.1)))
  (product 4 (leaf 1 (gaussian 0.3 1.7))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))
              (leaf 3 (gaussian -2.6 7.7)))))
```

→

```
(factorize
 (sum
  (product 6 (leaf 1 (gaussian 0.6 2.1))
             (leaf 3 (gaussian -0.3 0.9)))
  (product 4 (leaf 1 (gaussian 0.3 1.7)
             (leaf 3 (gaussian -6.1 4.8)))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9)))))
```

(a) Move a variable into an existing sum node and sample new distribution parameters.

```
(factorize
 (sum
  (product 6 (leaf 1 (gaussian 0.6 2.1)))
  (product 4 (leaf 1 (gaussian 0.3 1.7))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))
              (leaf 3 (gaussian -2.6 7.7)))))
```

→

```
(factorize
 (sum
  (product 6 (leaf 1 (gaussian 0.6 2.1)))
  (product 4 (leaf 1 (gaussian 0.3 1.7))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))))
 (sum
  (product 7 (leaf 3 (gaussian 8.4 0.9)))
  (product 3 (leaf 3 (gaussian 0.1 2.9)))))
```

(b) Move a variable into a new sum node of its own and sample new distribution parameters.

```
(factorize
 (sum
  (product 6 (leaf 1 (gaussian 0.6 2.1)))
  (product 4 (leaf 1 (gaussian 0.3 1.7))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))
              (leaf 3 (gaussian -2.6 7.7)))))
```

→

```
(factorize
 (sum
  (product 6 (leaf 1 (gaussian 0.6 2.1)))
  (product 4 (leaf 1 (gaussian 8.3 0.6))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))
              (leaf 3 (gaussian -2.6 7.7)))))
```

(c) Change the parameters of a variable's distribution at a leaf node.

```
(factorize
 (sum
  (product 6 (leaf 1 (gaussian 0.6 2.1)))
  (product 4 (leaf 1 (gaussian 0.3 1.7))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))
              (leaf 3 (gaussian -2.6 7.7)))))
```

→

```
(factorize
 (sum
  (product 7 (leaf 1 (gaussian 0.6 2.1)))
  (product 3 (leaf 1 (gaussian 0.3 1.7))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))
              (leaf 3 (gaussian -2.6 7.7)))))
```

(d) Within one sum node, increase the weight of a branch by 1 and decrease the weight of another by 1.

```
(factorize
 (sum
  (product 6 (leaf 1 (gaussian 0.6 2.1)))
  (product 4 (leaf 1 (gaussian 0.3 1.7))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))
              (leaf 3 (gaussian -2.6 7.7)))))
```

→

```
(factorize
 (sum
  (product 5 (leaf 1 (gaussian 0.6 2.1)))
  (product 4 (leaf 1 (gaussian 0.3 1.7)))
  (product 1 (leaf 1 (gaussian -5.0 2.3))))
 (sum
  (product 10 (leaf 2 (gaussian 7.6 1.9))
              (leaf 3 (gaussian -2.6 7.7)))))
```

(e) Within one sum node, create a new branch with weight 1 and decrease the weight of an existing branch by 1.

Figure 4.3: Examples of transition operators applied to a program in the MultiMixture DSL during Bayesian synthesis via MCMC or resample-move SMC.

4.3 Evaluation

MultiMixture DSL programs produced by Bayesian synthesis are evaluated in three ways. Section 4.3.1 checks how well the learned DSL programs are able to detect complex nonlinear dependence structures between multiple pairs of variables obtained from datasets in the UCI Machine Learning repository. Section 4.3.2 computes the predictive accuracy on held-out observations using MultiMixture DSL programs learned from data produced from a benchmark of 14 ground-truth probabilistic programs. Section 4.3.3 compares the quality of synthetic data generated from synthesized MultiMixture programs to those from statistical and deep learning baselines on a dataset of real-world earth satellites. All three evaluations include distributions with discrete and continuous variables. The results confirm that the synthesized MultiMixture programs deliver accurate solutions for all these data analysis tasks.

4.3.1 Detecting Dependence Relationships

A central goal of data analysis for cross-sectional data is identifying predictive relationships between variables [Ezekiel, 1930]. Recall from Section 4.1 that each MultiMixture program specifies a partition of the m variables into independent groups that factorize under the product node at the root. In Figure 4.1, for example, X_2 is independent of X_1 and X_3. Figure 4.4 shows scatter plots for sixteen pairs of variables selected from six datasets in the UCI machine learning repository [Dheeru and Graff, 2017]: automobile data (26 variables), wine data (14 variables), hepatitis data (20 variables), heart disease data (15 variables), and census data (15 variables). The variable pairs exhibit a broad class of relationships, including linear, nonlinear, heteroskedastic, and multi-modal patterns. Four of the benchmarks, shown in the final row, have no predictive relationship. Table 4.1 compares the ability of the MultiMixture programs produced by Bayesian synthesis to detect dependencies between variables to that of Pearson correlation. Using Eq. (3.9) from Section 3.2.2, a dependence between a pair of variables is reported whenever a 80% of the programs in the ensemble assign these variable to the same subexpression under the root. For Pearson correlation, dependence is reported if the value exceeds 0.20 and is statistically significant at the 5% level, with Bonferroni correction for multiple testing. The last two columns in Table 4.1 show these numbers in the final columns. MultiMixture programs deliver 10/12 true positives and 4/4 true negatives, whereas Pearson correlation delivers 4/12 true positives and 4/4 true negatives. The 8/12 false negatives from Pearson correlation are for variable pairs that exhibit nonlinear, bimodal, or heteroskedastic relationships. The evaluation in Section 4.3.3 shows that MultiMixture programs can not only detect the presence of complex predictive relationships between variables but also generate synthetic data that matches patterns in the observed data.

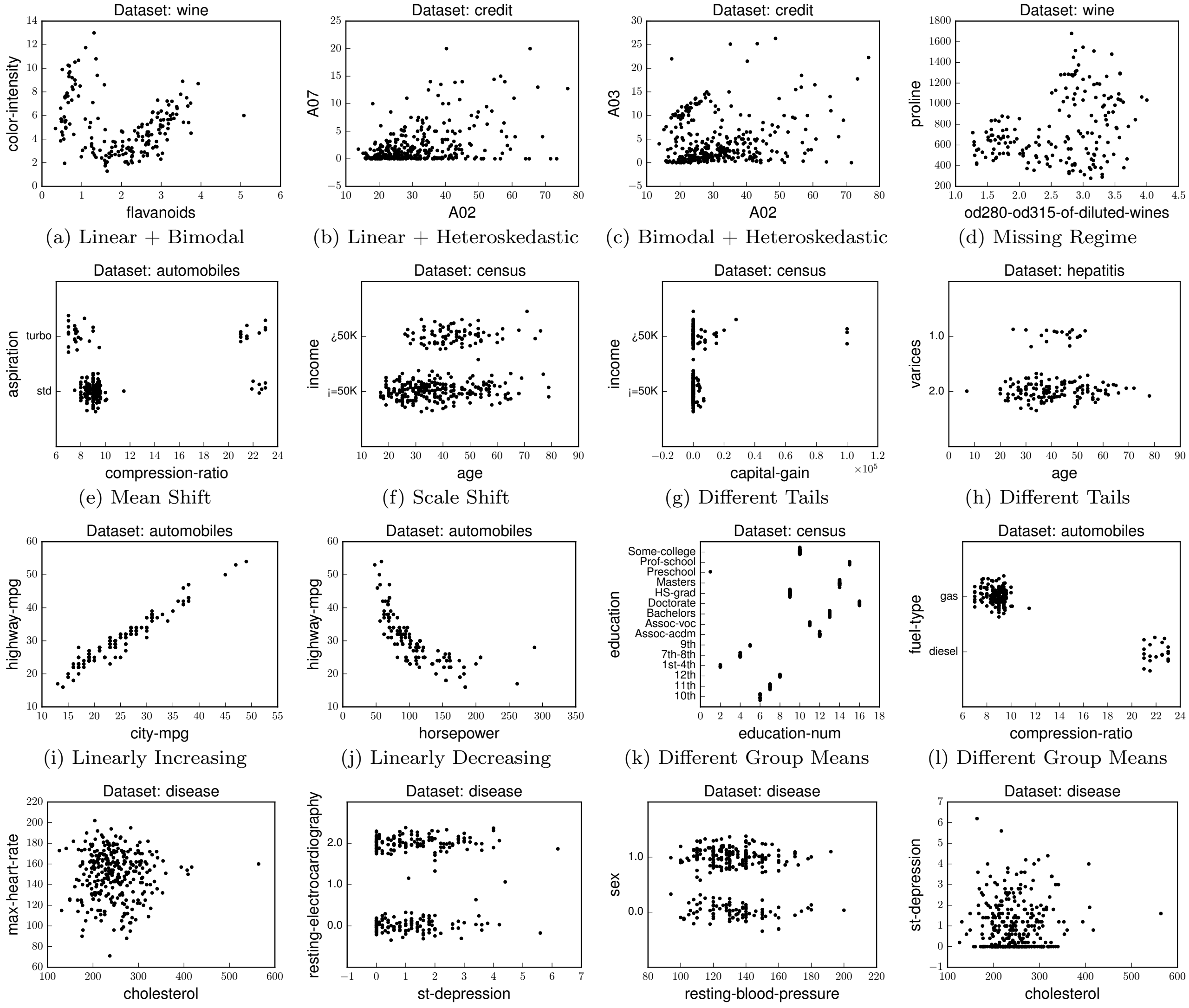
4.3.2 Estimating Probabilities

Another central goal of data analysis for cross-sectional data tables is learning probability distributions that can be used to accurately compute the probability (density) of new data, which is a task known as "density estimation" [Silverman, 1986]. Accurate density estimation enables several tasks, as discussed in the beginning of this chapter. After synthesizing programs in the MultiMixture DSL, the probability of new data records is computed by translating the programs into SPPL (right column of 3.1) and using the exact inference engine to automatically solve density evaluation queries. Density estimates from MultiMixture programs are compared to the widely used multivariate kernel density estimation (KDE) method from Racine and Li [2004] which supports both continuous and discrete data.

The quality of the density estimates are assessed on a benchmark of 14 programs adapted from Chasins and Phothilimthana [2017], listed in the first column in Table 4.2. For each benchmark problem, a training set of 10,000 data records was generated from a ground-truth probabilistic program expressed in the BLOG language [Milch et al., 2005]. These data records were used to learn MultiMixture programs and to fit KDE models. To measure how well each method captures the true distribution, 10,000 new data

Table 4.1: Comparison of detected dependencies for the 16 variable pairs in Figure 4.4.

				Detected Dependence Structure	
	Variable 1	Variable 2	True Dependence Structure	Pearson Correlation	Bayesian Synthesis
(a)	flavanoids	color-intensity	✓ linear + bimodal	× (0.03)	✓ (0.97)
(b)	A02	A07	✓ linear + heteroskedastic	× (0.16)	✓ (0.89)
(c)	A02	A03	✓ linear + bimodal + heteroskedastic	× (0.03)	× (0.66)
(d)	proline	od280-of-wines	✓ nonlinear + missing regime	× (0.09)	✓ (0.97)
(e)	compression-ratio	aspiration	✓ mean shift	× (0.07)	✓ (0.98)
(f)	age	income	✓ different group tails	× (0.06)	✓ (0.90)
(g)	age	varices	✓ scale shift	× (0.00)	✓ (0.90)
(h)	capital-gain	income	✓ different group tails	× (0.05)	× (0.77)
(i)	city-mpg	highway-mpg	✓ linearly increasing	✓ (0.95)	✓ (1.00)
(j)	horsepower	highway-mpg	✓ linearly decreasing	✓ (0.65)	✓ (1.00)
(k)	education-years	education-level	✓ different group means	✓ (1.00)	✓ (1.00)
(l)	compression-ratio	fuel-type	✓ different group means	✓ (0.97)	✓ (0.98)
(m)	cholesterol	max-heart-rate	× none (+ outliers)	× (0.00)	× (0.08)
(n)	cholesterol	st-depression	× none (+ outliers)	× (0.00)	× (0.00)
(o)	blood-pressure	sex	× none	× (0.01)	× (0.26)
(p)	st-depression	electrocardiography	× none	× (0.04)	× (0.00)



(a) Linear + Bimodal

(b) Linear + Heteroskedastic

(c) Bimodal + Heteroskedastic

(d) Missing Regime

(e) Mean Shift

(f) Scale Shift

(g) Different Tails

(h) Different Tails

(i) Linearly Increasing

(j) Linearly Decreasing

(k) Different Group Means

(l) Different Group Means

(m) No Dependence + Outliers

(n) No Dependence + Outliers

(o) No Dependence

(p) No Dependence

Figure 4.4: Sixteen variable pairs in the UCI repository that exhibit various dependence structures.

Table 4.2: Median absolute error of estimates of the log probability of held-out data using kernel density estimation and Bayesian synthesis for 14 density estimation benchmarks.

		Median Absolute Error	
	Distributions	Kernel Density Estimation	Bayesian Synthesis
`biasedtugwar`	Discrete–Continuous	1.58×10^{-1}	3.13×10^{-2}
`burglary`	Discrete	4.25×10^{-1}	1.45×10^{-3}
`csi`	Discrete	1.00×10^{-1}	4.35×10^{-4}
`easytugwar`	Discrete–Continuous	2.96×10^{-1}	9.96×10^{-2}
`eyecolor`	Discrete	4.69×10^{-2}	5.31×10^{-3}
`grass`	Discrete–Continuous	3.91×10^{-1}	4.49×10^{-2}
`healthiness`	Discrete	1.35×10^{-1}	3.00×10^{-3}
`hurricane`	Discrete	1.30×10^{-1}	2.11×10^{-4}
`icecream`	Discrete–Continuous	1.51×10^{-1}	7.07×10^{-2}
`mixedCondition`	Continuous	1.06×10^{-1}	1.43×10^{-2}
`multipleBranches`	Continuous	4.12×10^{-2}	1.22×10^{-2}
`students`	Discrete–Continuous	1.74×10^{-1}	5.47×10^{-2}
`tugwarAddition`	Discrete–Continuous	2.60×10^{-1}	1.38×10^{-1}
`uniform`	Continuous	2.72×10^{-1}	1.26×10^{-1}

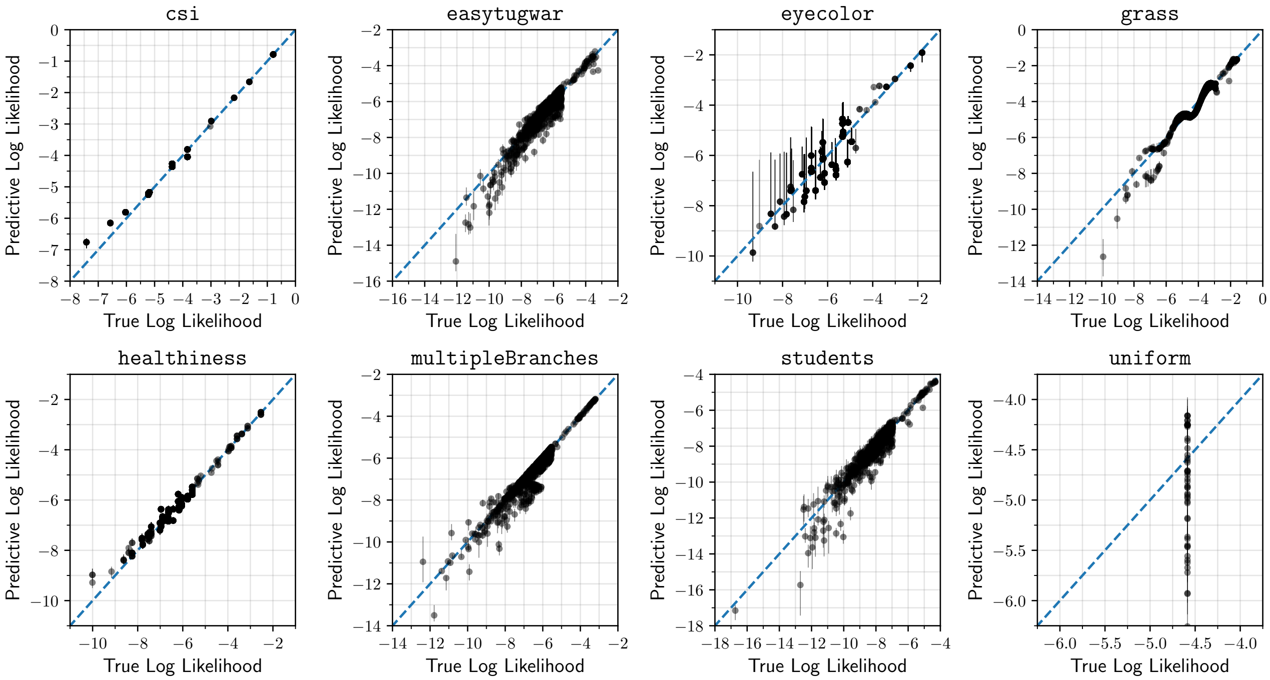


Figure 4.5: Comparing ground-truth probabilities of held-out data to estimated probabilities according to synthesized programs, for eight of the 14 density estimation benchmarks in Table 4.2. Each dot represents one held-out observation. The error in each estimated probability is the horizontal distance between the diagonal blue line and the dot. Error bars are computed across the ensemble of synthesized programs. In most cases, the estimates are most accurate and certain in the bulk of the distribution and are less accurate and certain in the tails.

records were used to assess the held-out predictive probabilities according to KDE and MultiMixture. To compute the estimation errors, the predictive probabilities were compared to the actual probabilities from the ground-truth BLOG programs used to generate the training data. The results in Table 4.2

	Name	Country of Operator	Operator Owner	Users	Purpose	Class of Orbit	Type of Orbit
1	Prometheus 1A	USA	Los Alamos Nati	Military	Technology Develo	LEO	Sun-Synchronous
2	Eutelsat 28A	Multinational	European Teleco	Commercial	Communications	GEO	NaN
3	SMDC-ONE 1.2	USA	U.S. Army Space	Military	Technology Develo	LEO	NaN
4	Lacrosse/Onyx	USA	National Reconn	Military	Surveillance	LEO	Intermediate
5	SMOS (Soil Mo	ESA	Centre National	Government	Earth Observation	LEO	Sun-Synchronous
6	Compass G-11	China (PR)	Chinese Defense	Military	Navigation/Global	GEO	NaN
7	Echostar 6	USA	Echostar Techno	Commercial	Communications	GEO	NaN
8	INMARSAT 4 F2	United Kingdom	INMARSAT, Ltd.	Commercial	Communications	GEO	NaN
9	Eutelsat 25C	Multinational	European Teleco	Commercial	Communications	GEO	NaN
10	Vinasat 2	Vietnam	Vietnamese Post	Government	Communications	GEO	NaN

	Perigee km	Apogee km	Eccentricity	Period minutes	Launch Mass kg	Dry Mass kg	Power watts
1	500	506	0.00044	94.68	NaN	NaN	NaN
2	35788	35794	0.00007	1436.10	2950	1375	5900
3	483	789	0.02184	97.40	3	NaN	NaN
4	574	676	0.00729	97.21	14500	NaN	NaN
5	759	760	0.00007	100.00	658	630	1065
6	35776	35799	0.00027	1436.15	2300	NaN	NaN
7	35775	35798	0.00027	1436.12	3700	1493	11000
8	35773	35800	0.00032	1436.11	5458	NaN	13000
9	35780	35790	0.00012	1436.04	3170	1900	5900
10	35742	35776	0.00040	1434.69	2970	NaN	NaN

	Date of Launch	Anticipated Lifetime	Contractor	Launch Site	Launch Vehicle	Longitude Radians	Inclination radians
1	41597	NaN	Los Alamos Nation	Wallops Island Fl	Minotaur 1	NaN	0.707033
2	36958	12	Alcatel Space Ind	Guiana Space Cent	Ariane 5	0.498466	0.001222
3	41165	NaN	Miltec	Vandenberg AFB	Atlas 5	NaN	1.127483
4	36755	9	Lockheed Martin A	Vandenberg AFB	Titan IV	NaN	1.186824
5	40119	3	Thales Alenia Spa	Plesetsk Cosmodro	Breeze KM	NaN	1.717404
6	40963	8	Space Technology	Xichang Satellite	Long March 3A	1.029744	0.032638
7	36721	12	Lockheed Martin M	Cape Canaveral	Atlas 2 AS	-1.269029	0.001222
8	38664	15	EADS Astrium	Sea Launch (Odyss	Zenit 3SL	-0.920836	0.040666
9	37580	12	Alcatel Space Ind	Cape Canaveral	Atlas 2 AS	0.445059	0.000349
10	41044	15	Lockheed Martin C	Guiana Space Cent	Ariane 5 ECA	2.300344	0.001396

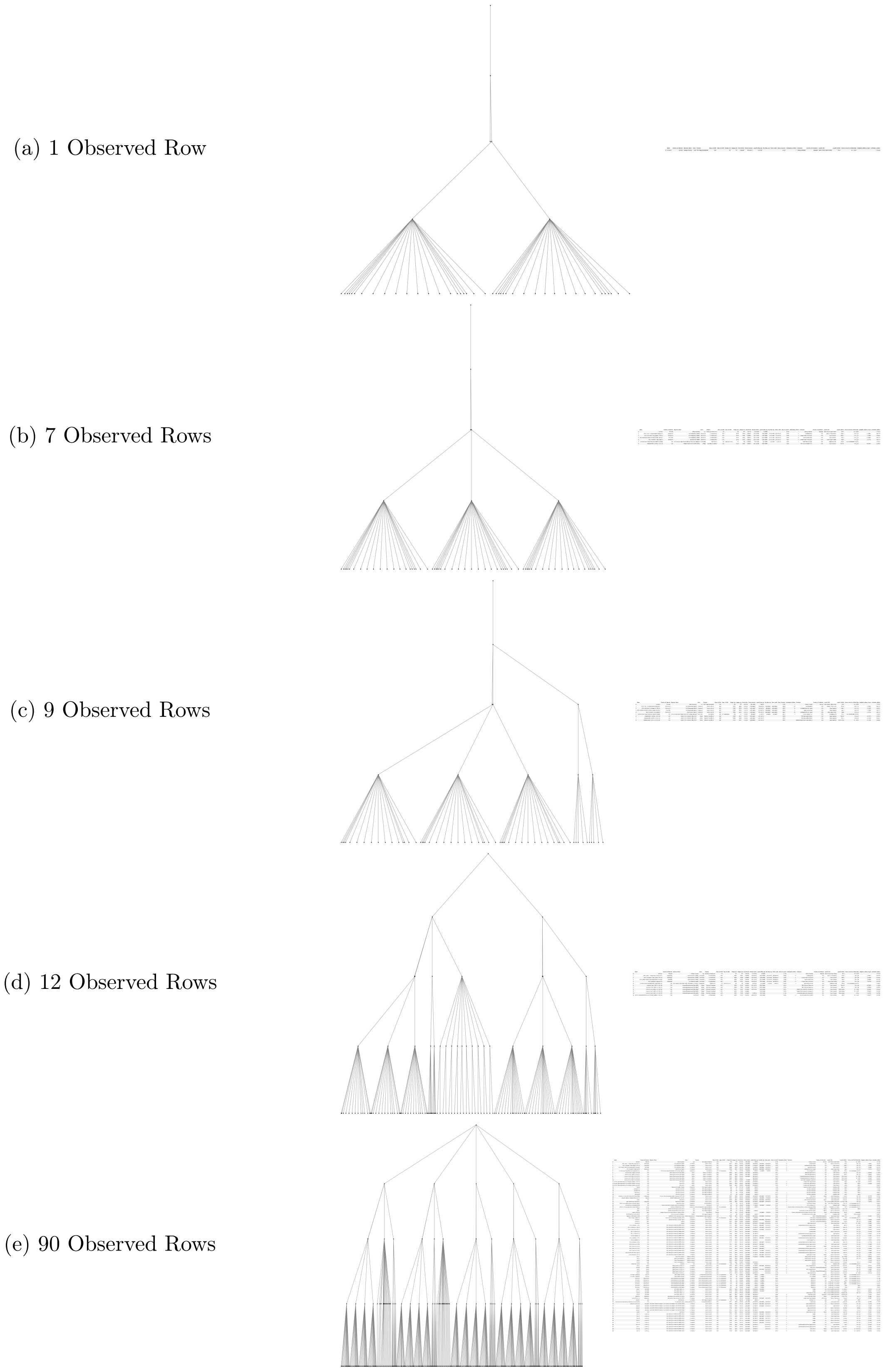
Table 4.3: Ten records from the satellites dataset showing 21 numeric and nominal variables.

show that the synthesized programs deliver more accurate density estimates, with improvement ranging between 10x–1000x. Moreover, an advantage of Bayesian synthesis over frequentist methods like KDE is that the ensemble of synthesized programs makes it possible to approximate the full posterior distribution over model structure and parameters, which can be used to compute error bars for predicted probability values. In contrast, KDE only gives point estimates of predictive probabilities that are not associated with any measure of uncertainty. Figure 4.5 shows that the reported uncertainties around the predictions are generally well calibrated, in the sense that the error bars are wider for held-out data records where the estimates are less accurate. The `uniform` benchmark is the most challenging. Because the MultiMixture DSL does not contain a primitive uniform distribution over a continuous domain, it must form an approximation using a mixture of one of the eight existing nonuniform primitives mentioned in Section 4.1, which requires a very large number of components approximate accurately.

4.3.3 Generating Synthetic Data

Synthetic data generation is an emerging application area of machine learning where the goal is to simulated "virtual" datasets that accurately emulate the statistical characteristics of an observed dataset. The ability to learn accurate synthetic data generators has broad applications in privacy [Near and Darais, 2021], software testing and reliability analysis [Soltana et al., 2017], and generating new training data to improve the performance of machine algorithms [Jahanian et al., 2021].

This evaluation assess the quality of synthetic data produced by MultiMixture programs that are learned from an open-science dataset of 1167 earth satellites [Union of Concerned Scientists, 2016]. Table 4.3 shows ten example satellite data records and all 21 variables. There are numeric variables, such as the orbital period and launch mass; nominal variables, such as the class and type of orbit; and arbitrary patterns of missing data (NaN). Figure 4.6 shows a snapshot of online Bayesian synthesis of MultiMixture programs at various iterations of resample-move SMC inference (Algorithm 3.2), where



(a) 1 Observed Row

(b) 7 Observed Rows

(c) 9 Observed Rows

(d) 12 Observed Rows

(e) 90 Observed Rows

Figure 4.6: Online Bayesian synthesis in the MultiMixture DSL using resample-move sequential Monte Carlo for the satellites data in Table 4.3. Each panel shows a graphical representation of the ensemble of synthesized DSL programs and the subset of the data table observed so far (1167 total rows).

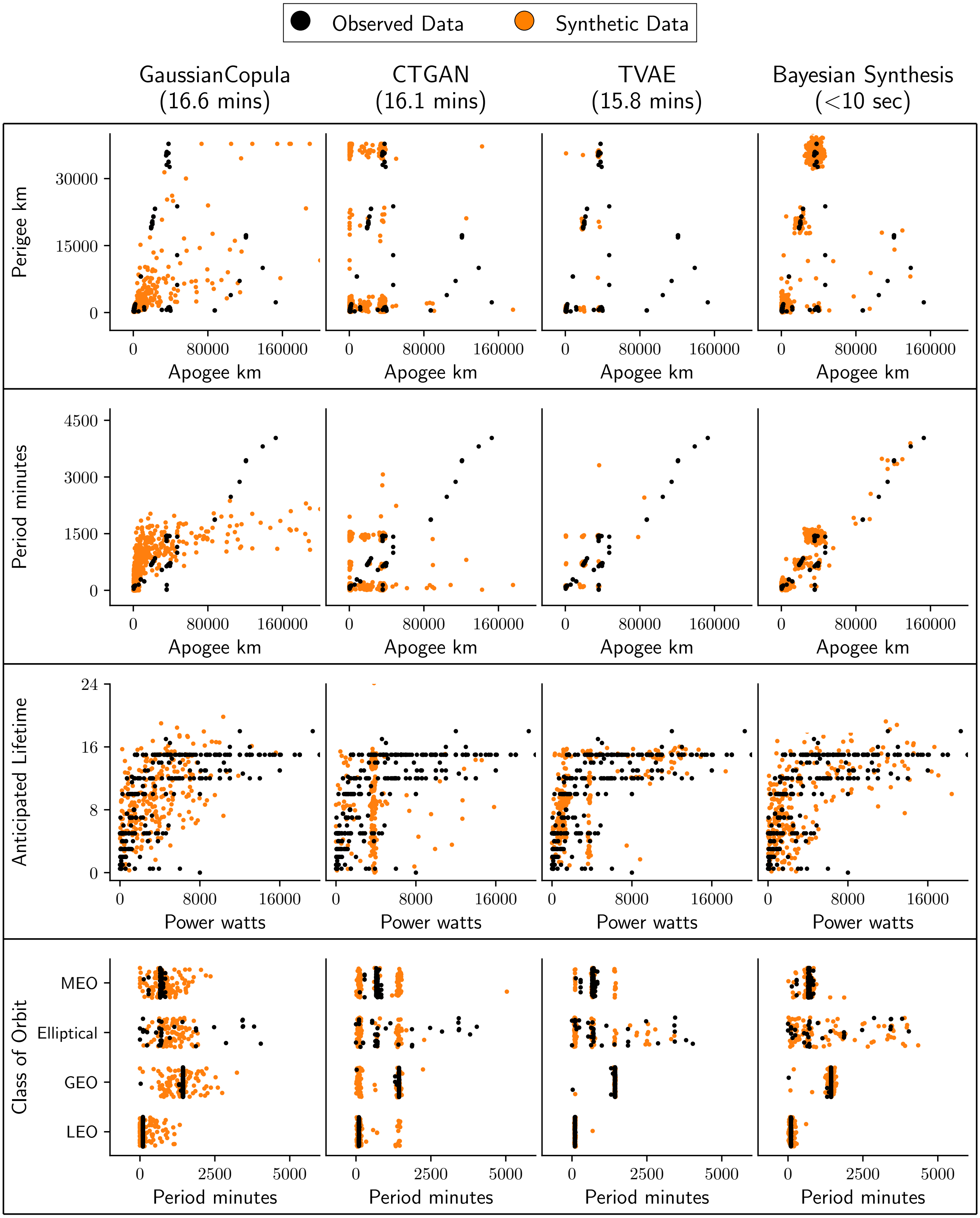


Figure 4.7: Comparison of synthetic data quality. Each row shows observed data (black) for a pair of satellite variables and synthetic data (orange) produced by various methods. Bayesian synthesis of MultiMixture programs is faster and produces more accurate data as compared to the baselines.

the rejuvenation operators are shown in Figure 4.3. The structure of the synthesized programs in the ensemble adapts to handle the increasing statistical complexity as more satellite records are observed, which is enabled by the nonparametric Prior denotation in Listing 4.1. In particular, the number of possible structures (4.1) grows with the number n of observed rows. After observing all 1167 rows, synthetic data is generated by sampling from the Likelihood defined in Listing 4.1.

Baselines Figure 4.7 compares synthetic data for four pairs of satellite variables generated using four techniques: multivariate Gaussian copulas [Patki et al., 2016], conditional tabular generative adversarial networks [CTGAN; Xu et al., 2019], tabular variational autoencoder [TVAE; Xu et al., 2019], and synthesized MultiMixture programs. Implementations of the three baselines were obtained from an open-source online repository [Synthetic Data Vault, 2022].

Runtime Comparison The top row of Figure 4.7 shows the runtime required to learn models of satellites data table, which takes around 16 minutes for the three baselines. In contrast, the resample-move SMC strategy from Figure 4.6 takes less than 10 seconds to complete a full streaming pass over the data. In all cases, runtime is dominated by learning models rather than producing synthetic data.

Synthetic Data Comparison The plots in Figure 4.7 show qualitatively that synthetic data produced by MultiMixture programs more accurately emulate the observed data than those from the three baselines. For Gaussian copula, the simulations follow a unimodal distribution with large variance that is spread over more multiple modes in the data. CTGAN produces an artificial grid of simulations (rows 1, 2, and 4) obtained from the Cartesian product of the modes in the observed data or an artificial vertical line (row 3), which means that the synthetic data appears in areas of the two-dimensional plane where there is no observed data. The TVAE suffers from a similar issue to CTGAN. In row 2, TVAE produces a subgrid of nine modes, four of which are artificial, and it fails to capture the linear tail for apogee values greater than 80,000. In row 4, TVAE produces an artificial mode for the MEO orbital class. The synthesized MultiMixture programs consistently capture modes in the data and do not produce artificial modes or other artifacts such as vertical or horizontal lines. The simulations from MultiMixture in row 3, however, appear too noisy around the anticipated lifetime values of 16, which can be improved by running additional steps of parameter inference.

Chapter 5

Synthesizing Models for Multivariate Time Series

> Although the future is not predictable in any detail, it is manageable as an aggregate phenomenon.
>
> Herbert A. Simon

Multivariate time series data is ubiquitous, arising in domains such as macroeconomics, neuroscience, and public health. Unfortunately, data analysis problems such forecasting, imputation, and clustering are exceptionally difficult to solve when there are dozens or hundreds of time series. One challenge in these settings is that the data may reflect underlying processes with widely varying and nonstationary dynamics [Fulcher and Jones, 2014]. Another challenge is that standard parametric approaches such as state-space models or vector autoregression often become numerically unstable in sparse high-dimensional data [Koop, 2013]. These approaches also require users to perform significant custom modeling for each dataset or search over a large number of possible parameter settings. As discussed in Gruber and West [2017], there is an increasing need for multivariate methods that exploit sparsity, are computationally efficient, and can simultaneously model many hundreds time series jointly.

This chapter presents the temporally-reweighted Chinese restaurant mixture (TR-CRPM), a nonparametric Bayesian model family multivariate time series that is designed to address some of the above challenges. The model is based on two extensions to traditional Dirichlet process mixture models [Lo, 1984]. The first extension is a recurrent version of the Chinese restaurant process mixture model for capturing temporal dependencies in the data. The second extension is a structure learning prior for discovering groups of time series whose underlying dynamics are modeled independently from one another. The TR-CRPM is designed to interpolate in regimes where it has seen similar history before and reverts to a broad ignorance prior in novel regimes. This inductive bias does not sacrifice predictive accuracy when there is sufficient signal in the previous data to make forecasts or impute missing data.

The TR-CRPM is applied to solve challenging flu forecasting problems using data from the US Center for Disease Control and Prevention (CDC). The TR-CRPM outperforms several baselines, including Facebook Prophet, multi-output Gaussian processes, seasonal ARIMA, and the HDP-HMM, by integrating multiple sources of information that include historical flu rates, weather measurements, and Twitter data. Imputation accuracy for missing data is also highly competitive with the state-of-the-art Amelia method [Honaker et al., 2011]. Finally, the TR-CRPM is shown to detect interpretable clusters that correspond to commonsense categories given hundreds of macroeconomic time series from the Gapminder Foundation [2022] database.

5.1 Temporally-Reweighted Chinese Restaurant Mixture Model

Notation Let $\mathbf{x}^n ::= \{x^n_t \mid t = 1, \dots, T\}$, $n = 1, \dots, N$, be a set of N time series, each observed during T discrete time points. Subsequences of variables are indexed using slice notation, so that $\mathbf{x}^n_{t_1:t_2} ::= (x^n_{t_1}, x^n_{t_1+1}, \dots, x^n_{t_2-1}, x^n_{t_2})$ whenever $t_1 \le t_2$ and is empty if $t_2 < t_1$. The superscript n is omitted when discussing a single time series. This section develops a nonparametric Bayesian model that describes the joint distribution of $\{\mathbf{x}^n \mid n = 1, \dots, N\}$, which is presented in stages.

5.1.1 Background: Chinese Restaurant Process Mixtures Models

The Dirichlet process mixture (DPM) is a Bayesian nonparametric model of an exchangeable sequence $(x_1, \dots, x_m)$ of m data points. Given a a measurable space Θ, a probability distribution π_Θ over Θ, and a real-valued concentration parameter $\alpha > 0$, the generative process of the DPM is

$$P \sim \text{DirichletProcess}(\alpha, \pi_\Theta), \quad \theta^*_j \mid P \sim P, \quad x_j \mid \theta^*_j \sim F(\cdot \mid \theta^*_j), \tag{5.1}$$

where P is an almost-surely discrete random probability measure whose atoms are drawn from π_Θ and F is a probability distribution over the space in which the data x_j take their values that is parameterized by $\theta \in \Theta$. The random measure P is a draw from a Dirichlet process in the sense that, for any finite partition $A_1, \dots, A_n$ of Θ into measurable n subsets, the random vector $[P(A_1), \dots, P(A_n)] \sim \text{Dirichlet}(\alpha\pi_\Theta(A_1), \dots, \alpha\pi_\Theta(A_n))$. A constructive representation of the generative process (5.1) can be obtained in terms of a distribution over partitions called the Chinese restaurant process [Aldous, 1985]. As P is almost-surely discrete, the draws $\{\theta^*_j\} \sim P$ may contain repeated values with positive probability, which induces a clustering among data x_j ($j = 1, \dots, m$). In particular, let $\{\theta_k\}$ be unique values among the $\{\theta^*_j\}$ and z_j denote the cluster assignment of x_j which satisfies $\theta^*_j = \theta_{z_j}$. Define n_{jk} to be the number of observations x_i with $z_i = k$ for $i < j$. Using the conditional distribution of z_j given previous cluster assignments $\mathbf{z}_{1:j-1}$, an equivalent generative process to Eq. (5.1) is

$$\{\theta_k\} \overset{\text{iid}}{\sim} \pi_\Theta(\cdot \mid \lambda_F) \tag{5.2}$$

$$z_j \mid \mathbf{z}_{1:j-1} \sim \sum_{k=1}^{M_j} \frac{n_{jk}}{\alpha + j - 1}\delta_k + \frac{\alpha}{\alpha + j - 1}\delta_{M_j+1} \qquad (j = 1, 2, \dots) \tag{5.3}$$

$$x_j \mid z_j, \{\theta_k\} \sim F(\cdot \mid \theta_{z_j}) \qquad (j = 1, 2, \dots), \tag{5.4}$$

where δ_k denotes an atomic measure at k; $M_j ::= \max(\mathbf{z}_{1:j-1})$ is the total number of clusters among the first $j - 1$ data items; and λ_F are hyperparameters of the base measure π_Θ.

5.1.2 TR-CRP Mixtures for Modeling a Single Time Series

Overview The traditional CRP mixture model has no notion of temporal dependence between the observations x_j and is therefore inappropriate for modeling a nonexchangeable discrete-time series $(x_1, x_2, \dots)$, where the ordering among the variables induces temporal dependencies. Instead of assuming that (x_t, z_t) are conditionally independent of previous data $\mathbf{x}_{1:t-1}$ given $\mathbf{z}_{1:t-1}$ and $\{\theta_k\}$ as in Eqs. (5.3) and (5.4), temporal dependencies can be introduced by using the previous observations $\mathbf{x}_{1:t-1}$ when simulating z_t. The main idea is to modify the CRP prior by letting the cluster probability that $\{z_t = k\}$ at step t additionally account for (i) the p most recent observations $\mathbf{x}_{t-p:t-1}$; and (ii) the collection of all lagged values $D_{tk} := \{\mathbf{x}_{t'-p:t'-1} \mid z_{t'} = k, 1 \le t' < t\}$ of earlier data points $x_{t'}$ that are

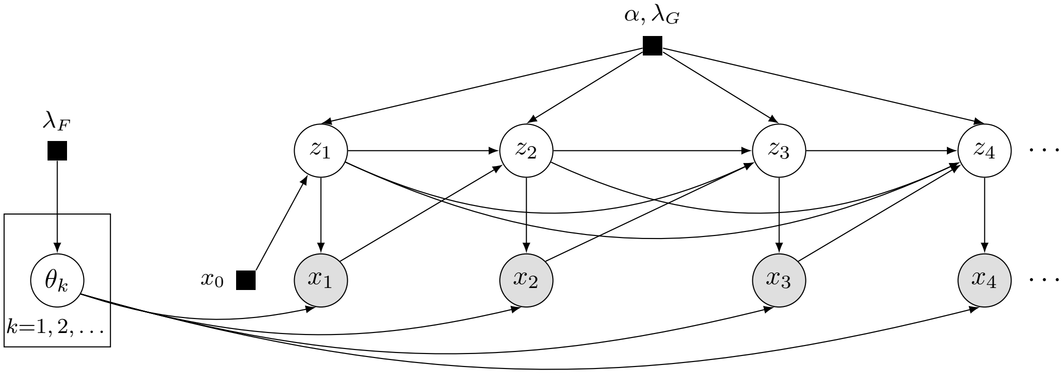


Figure 5.1: Graphical representation of the TR-CRP mixture model for a single time series $\mathbf{x} = (x_1, x_2, \dots)$. The window size $p = 1$.

assigned to cluster k. The distribution of $(x_1, x_2, \dots)$ in the TR-CRP mixture is thus

$$\{\theta_k\} \overset{\text{iid}}{\sim} \pi_\Theta(\cdot \mid \lambda_F) \tag{5.5}$$

$$z_t \mid \mathbf{z}_{1:t-1}, \mathbf{x}_{1:t-1} \sim \sum_{k=1}^{M_t} \frac{n_{tk}\gamma_{tk}}{Z}\delta_k + \frac{\alpha\gamma_{M_{t+1}}}{Z}\delta_{M_t+1} \qquad (t = 1, 2, \dots) \tag{5.6}$$

$$\text{where} \begin{cases} \gamma_{tk} ::= G(\mathbf{x}_{t-p:t-1}; D_{tk}, \lambda_G) \\ Z ::= \sum_{k=1}^{M_t} n_{tk}\gamma_{tk} + \alpha\lambda'_t \end{cases}$$

$$x_t \mid z_t, \{\theta_k\} \sim F(\cdot \mid \theta_{z_t}) \qquad (t = 1, 2, \dots) \tag{5.7}$$

where $M_t ::= \max(\mathbf{z}_{1:t-1})$ is the total number of clusters among the first $t-1$ elements in the time series. The main difference between the standard CRP mixture in Eqs. (5.2)–(5.4) and TR-CRP mixture in Eqs. (5.5)–(5.7) is the reweighting terms $\gamma_{tk} ::= G(\mathbf{x}_{t-p:t-1}; \lambda_G, D_{tk})$, where $G : \mathbb{R}^p \to \mathbb{R}^+$ is a "cohesion" function parametrized by the previous data D_{tk} at cluster k and parameters λ_G. The function G measures how well the current lagged values $\mathbf{x}_{t-p:t-1}$ match the collection of lagged values of earlier data D_{tk} in each cluster k, thereby introducing temporal dependence to the model. The smoothness of the process depends on the choice of the window size p: if t_1 and t_2 are close in time (relative to p) then they have overlapping lagged values $\mathbf{x}_{t_1-p:t_1-1}$ and $\mathbf{x}_{t_2-p:t_2-1}$, so G increases the probability that $\{z_{t_1} = z_{t_2}\}$. More generally, any pair of time points t_1 and t_2 that share similar lagged values are a-priori more likely to have similar distributions for generating x_{t_1} and x_{t_2}, because G increases the probability that $\{z_{t_1} = z_{t_2} = k\}$, so that x_{t_1} and x_{t_2} are both drawn from $F(\cdot \mid \theta_k)$.

Graphical Model Figure 5.1 shows a graphical model for the TR-CRP mixture with window size $p = 1$. The initial p observations $(x_{-p+1}, \dots, x_0)$ are assumed to be given. At step t, the cluster assignment z_t is sampled, whose probability of joining cluster k is a product of (i) the CRP probability for $\{z_t = k\}$ given all previous cluster assignments $\mathbf{z}_{1:t-1}$, and (ii) the "cohesion" term $G(\mathbf{x}_{t-p:t-1}; \lambda_G, D_{tk})$. In Figure 5.1, edges between the z_t's denote the CRP probabilities, while edges from x_{t-1} up to z_t represent reweighting the CRP by G. Cluster assignment z_t identifies a hidden temporal state that dictates the distribution of $x_t \sim F(\cdot \mid \theta_{z_t})$. If $p = 0$ or $G \propto 1$, then the temporal model reduces to a standard CRP mixture model for exchangeable data, since (z_t, x_t) are conditionally independent of the entire time series history $\mathbf{x}_{1:t-1}$ given $\mathbf{z}_{1:t-1}$. The model does not have Markov structure due to the infinite coupling among the hidden states z_t, which is different than the recurrent switching linear dynamical system described in Barber [2006]. Figure 5.2 shows an example of how observation of the

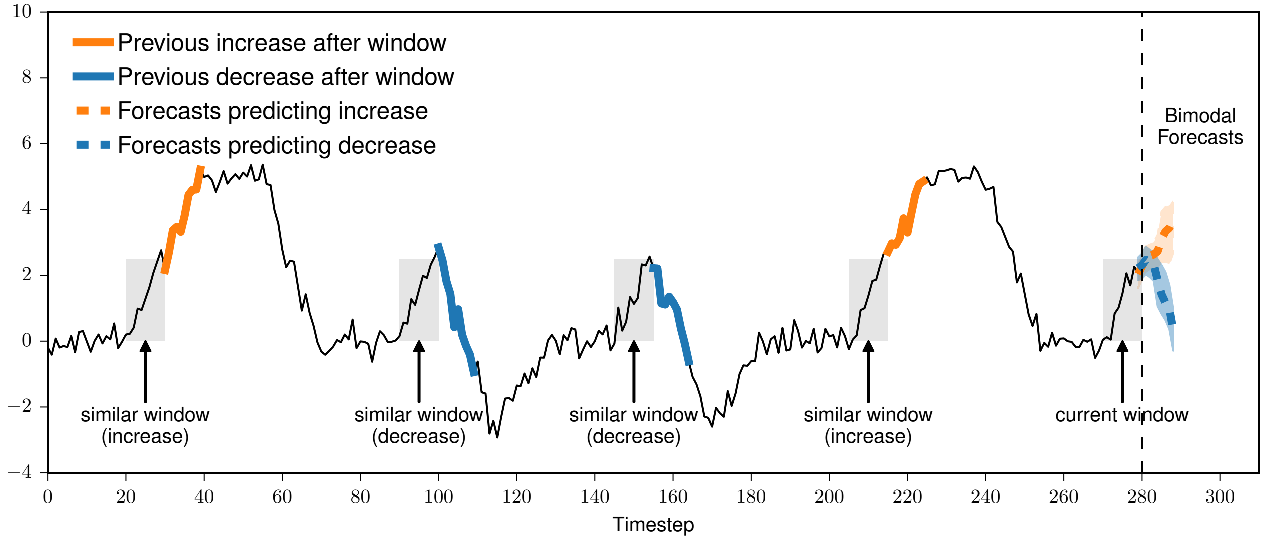


Figure 5.2: Bimodal forecasts from the TR-CRP mixture model with a window size $p = 10$. The observed time series data is shown as a solid black line. At time $t = 280$, the current size-p window of lagged values contains the observations $(x_{271}, \dots, x_{280})$, which are highlighted in gray. The model detects four previous time windows of length 10 that are most similar to the current window, also highlighted in gray, which are $[20, 30)$, $[95, 105)$, $[150, 159)$, and $[210, 219)$. The time series values increased in two of these windows (solid orange line) and decreased in two of these windows (solid blue line). The model hypothesizes that the future data after the current window is equally likely to be either of these two possibilities, which produces bimodal forecasts for the future data $(x_{281}, \dots, x_{290})$ in the next window (dashed orange and blue lines).

dynamics after previous size-p windows that are "similar" to the present window (as measured by G) are used to generate the hidden state and time series data for the next window.

Data Distribution The distribution F in Eq. (5.7) that governs the time series values can be arbitrary, depending on the data type of the observations. Examples include Bernoulli, Poisson, categorical, normal, gamma, and beta distributions. For the real-valued time series analyzed in this chapter, the distribution F is a Normal distribution with Normal-InverseGamma prior π_Θ:

$$\pi_\Theta(\mu_k, \sigma_k^2 \mid m, V, a, b) = \mathrm{N}(\mu_k \mid m, \sigma_k^2 V)\mathrm{InverseGamma}(\sigma_k^2 \mid a, b) \tag{5.8}$$

$$F(x_t \mid \mu_k, \sigma_k) = \mathrm{N}(x_t \mid \mu_k, \sigma_k^2), \tag{5.9}$$

where $\theta_k = (\mu_k, \sigma_k^2)$ are the parameters of F within cluster k and $\lambda_F = (m, V, a, b)$ are hyperparameters of π_Θ. As F and π_Θ are conjugate [Bernardo and Smith, 1994], the parameters θ_k can be marginalized out of the generative model as discussed in Section 5.2.

Cohesion Function The cohesion function G must be nonnegative but otherwise arbitrary. However, to ensure good performance, G must be able to assign a high value to lagged data vectors $\mathbf{x}_{t-p:t-1}$ and $\mathbf{x}'_{t-p:t-1}$ of length p that are similar to one another, so that the cluster probabilities are reweighted appropriately. Previous approaches in Bayesian nonparametric regression leveraged kernel-based reweighting schemes [Dunson et al., 2007]. The TR-CRP mixture takes a different approach, where G is a product of p Student-T distributions whose location, scale, and degrees of freedom depend on the previous data

D_{tk} assigned to cluster k:

$$G(\mathbf{x}_{t-p:t-1}; D_{tk}, \lambda_G) ::= \prod_{i=1}^{p} G_i(x_{t-i}; D_{tki}, \lambda_{Gi}) \tag{5.10}$$

$$G_i(x_{t-i}; D_{tki}, \lambda_{Gi}) ::= \mathrm{T}_{2a_{tki}}\left(x_{t-i}; m_{tki}, b_{tki}\frac{1+V_{tki}}{a_{tki}}\right) \tag{5.11}$$

$$D_{tki} ::= \{x_{t'-i} : z_{t'} = k, 1 \leq t' < t\} \tag{5.12}$$

$$n_{tki} ::= |D_{tki}| \tag{5.13}$$

$$\bar{x}_{tki} ::= \textstyle\sum_{t' \in D_{tki}} x_{t'-i}/n_{tki} \tag{5.14}$$

$$V_{tki} ::= 1/(V_{i0}^{-1} + n_{tki}) \tag{5.15}$$

$$m_{tki} ::= V_{tki}(V_{i0}^{-1} m_{i0} + n_{tki}\bar{x}_{tki}) \tag{5.16}$$

$$a_{tki} ::= a_{i0} + n_{tki}/2 \tag{5.17}$$

$$b_{tki} ::= b_{k0} + \left(m_{i0}^2 V_{i0}^{-1} + \textstyle\sum_{t'} x_{t'-i}^2 - m_{itk}^2 V_{tki}^{-1}\right)/2. \tag{5.18}$$

The function G is used for reweighting only: it does not define a probability distribution over the lagged data. Mathematically, G attracts x_t towards a cluster k that assigns $\mathbf{x}_{t-p:t-1}$ a high density value under the posterior predictive of an axis-aligned multivariate Gaussian conditioned on D_{tk} [Murphy, 2007].

5.1.3 TR-CRP Mixtures for Modeling Multiple Dependent Time Series

The univariate TR-CRP mixture can be extended to model a collection $\mathbf{x}^1, \dots, \mathbf{x}^N$ of $N > 0$ time series, which for now are assumed to be all dependent with one another. At time t, there is a single hidden state assignment z_t shared among all the time series. The lagged values of all N time series are used to reweight the CRP probabilities by the cohesion term G. Listing 5.1 shows the generative process for the multivariate TR-CRP mixture and Figure 5.3 applies the model to flu data. While there is a single hidden state z_t at time t, each time series has its own cluster parameters $\{\theta_k^n\}$ since the dynamics within each state might be different. Moreover, the model makes a "naive Bayes" assumption that data $\{x_t^n\}_{n=1}^N$ at time t are conditionally independent given z_t and that the reweighting term G in step 5.1 of Listing 5.1 factorizes as a product. This independence assumption improves numerical stability in sparse high-dimensional settings as compared to assuming arbitrary covariance structure, and maintains the ability to produce complex temporal patterns through the infinite mixture model.

5.1.4 Discovering Independence Structure Between Multiple Time Series

The multivariate TR-CRP mixture described in Listing 5.1 makes the restrictive assumption that all N time series $\{\mathbf{x}^1, \dots, \mathbf{x}^N\}$ are probabilistically dependent with one another. However, when modeling dozens or hundreds of time series with widely varying dynamics, forcing a shared hidden state z_t at each step t for all time series will cause the CRP to over-segment the time course and produce a large number of duplicate parameters to explain the data. The assumption that there is a single hidden state is relaxed by introducing a structure learning prior to determine which subsets of the N time series are well modeled by a joint TR-CRP mixture. This prior induces sparsity in the dependencies between the N time series as it first nonparametrically partitions them using an "outer" CRP. Each group of dependent time series is then modeled using the multivariate TR-CRP mixture described in Listing 5.1:

$$(c^1, c^2, \dots, c^N) \sim \mathrm{CRP}\,(\cdot \mid \alpha_0) \tag{5.19}$$

$$\{\mathbf{x}^n \mid c^n = k\} \sim \text{TR-CRP Mixture} \qquad \left(k = 1, \dots, \max \mathbf{c}^{1:N}\right), \tag{5.20}$$

where c^n is the cluster assignment of $\mathbf{x}^n$ and $\alpha_0 > 0$ the concentration parameter of the "outer" CRP.

```
1. Sample concentration parameter of CRP.
     $\alpha \sim \text{Gamma}(1,1)$
2. Sample model hyperparameters.
     $\lambda_G^n \sim H_G^n$                                   $(n = 1, \dots, N)$
     $\lambda_F^n \sim H_F^n$                                   $(n = 1, \dots, N)$
3. Sample distribution parameters of $F$
     $\theta_1^n, \theta_2^n, \dots \overset{\text{iid}}{\sim} \pi_\Theta(\cdot|\lambda_F^n)$          $(n = 1, \dots, N)$
4. Assume first $p$ values are known
     $\mathbf{x}_{-p+1:0}^n ::= (x_{-p+1}^n, \dots, x_0^n)$            $(n = 1, \dots, N)$
5. Sample time series for $t = 1, 2, \dots$
     5.1 Sample temporal cluster assignment $z_t$
         $P(z_t = k \mid \dots) \propto \text{CRP}(k|\alpha, \mathbf{z}_{1:t-1})$
                   $\prod_{n=1}^{N} G(\mathbf{x}_{t-p:t-1}^n; D_{tk}^n, \lambda_G^n)$
         where $D_{tk}^n ::= \{\mathbf{x}_{t'-p:t'-1}^n \mid z_{t'} = k, 1 \le t' < t\}$
         and $k = 1, \dots, \max(\mathbf{z}_{1:t-1}) + 1$
     5.2 Sample data $x_t^n$
         $x_t^n \mid z_t, \{\theta_k^n\} \sim F(\cdot|\theta_{z_t}^n)$          $(n = 1, \dots, N)$
```

Listing 5.1: Multivariate TR-CRP mixture model.

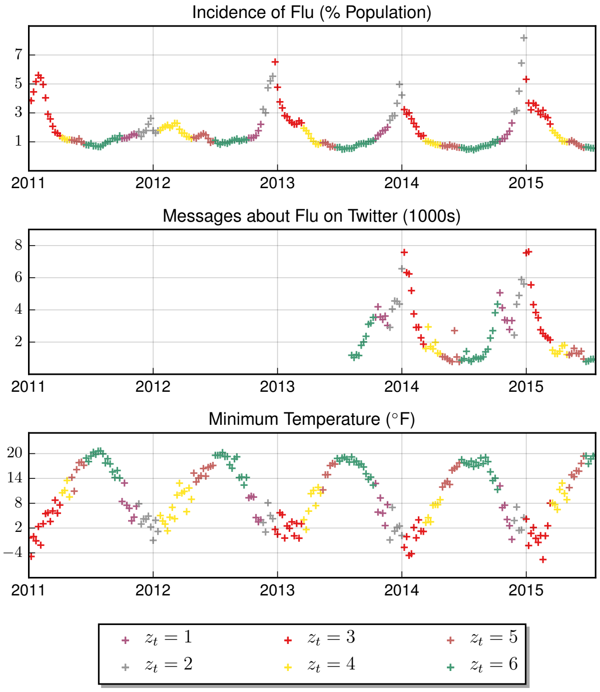
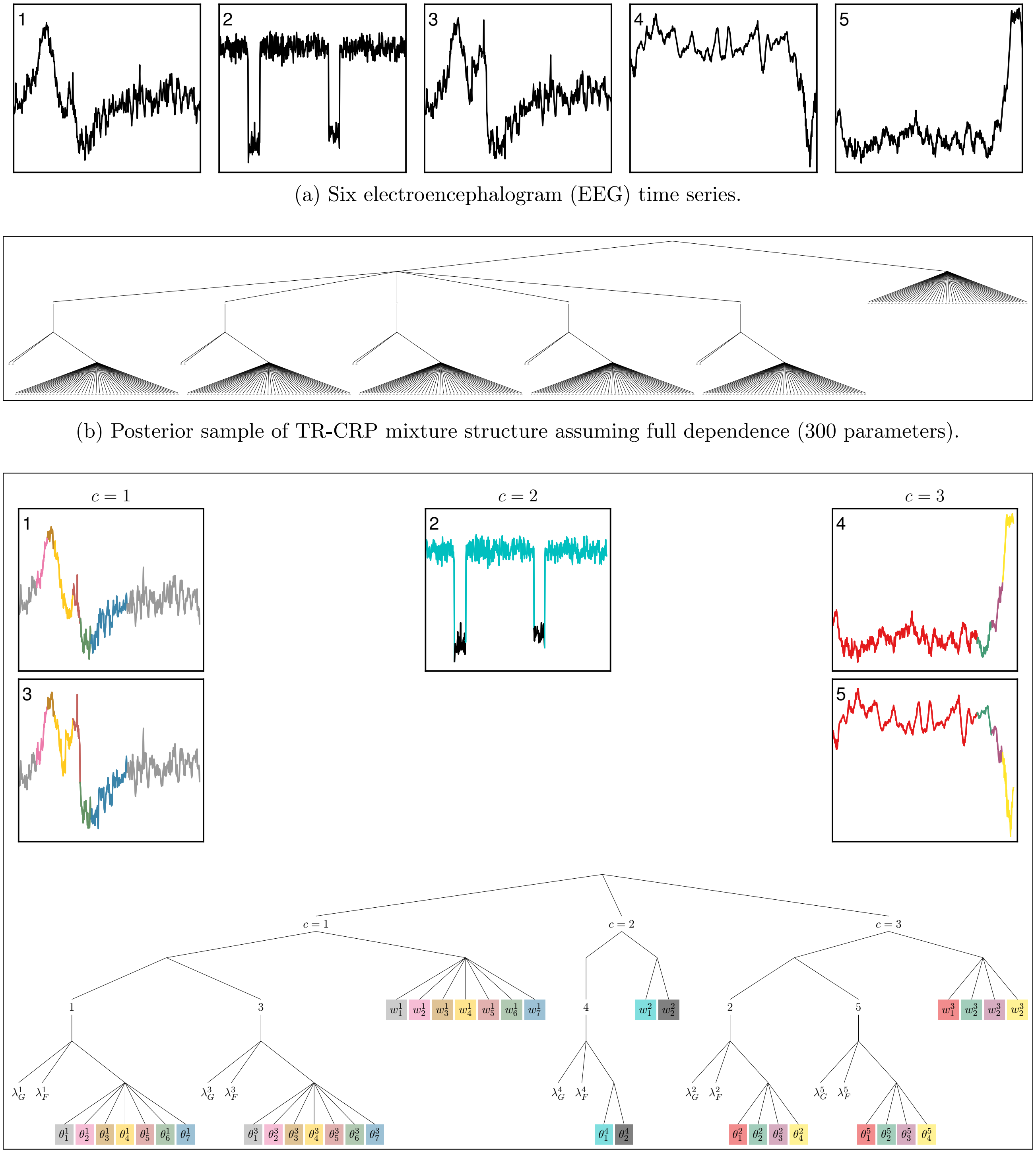


Figure 5.3: Applying the TR-CRP mixture with $p = 10$ weeks to model $\mathbf{x}^{\text{flu}}$, $\mathbf{x}^{\text{tweet}}$, and $\mathbf{x}^{\text{temp}}$ in US Region 4. Six regimes describing the seasonal behavior are detected in this posterior sample. Purple, gray, and red are the pre-peak rise, peak, and post-peak decline during the flu season; and yellow, brown, and green represent the rebound in between successive seasons. In 2012, the model reports no post-peak regime (red), reflecting the relatively mild flu season that year.

Figure 5.4 compares the learned model structures given EEG time series using the TR-CRP mixture with full dependence (Section 5.1.3) and the TR-CRP mixture with independence discovery (Section 5.1.4). As both model variants use nonparametric CRP priors over the hidden states, the number of parameters $\{\theta_k^n\}$ for $n = 1, \dots, N$ are inferred from data. Figure 5.4a shows the five observed EEG time series. Figure 5.4b shows the learned model structure assuming full dependence, which explains the data using 300 parameters. In Figure 5.4c, using the independence discovery prior to model the EEG data delivers a more concise model structure with only 47 parameters. The independence prior learns a separate segmentation of the time course into hidden states within each group of dependent time series. In contrast, the full dependence model produces the cross-product of the hidden states, which causes many excessive duplicated parameters. Synthesizing model structures that have independent groups of time series leads to fewer parameters, improving both interpretability and data efficiency. While there is a risk of learning spurious independencies that lead to under-fitting and weaker predictive performance, synthesizing an ensemble of model structures rather than a single structure effectively quantifies posterior uncertainty over the factorization structure. The applications in Section 5.4 further show that the independence discovery prior learns interpretable groups of independent time series given real-world GDP and cell phone subscriber data.



(a) Six electroencephalogram (EEG) time series.

(b) Posterior sample of TR-CRP mixture structure assuming full dependence (300 parameters).

(c) Posterior sample of TR-CRP mixture model structure with dependence discovery (47 parameters).

Figure 5.4: Learning independence relationships delivers more concise probabilistic model structures.

5.2 Algorithms for Posterior Inference

The goal of posterior inference in the TR-CRP mixture is to sample from the conditional distribution over all latent variables, given observed time series values $x_1^n, \dots x_T^n$ for $n = 1, \dots, N$. Because the factorization prior (5.19) produces $M = \max(\mathbf{c}^{1:N})$ separate TR-CRP mixtures, all latent variables in Listing 5.1 will include a subscript $m = 1, \dots, M$. In particular, α^m is the CRP concentration parameter and $\mathbf{z}_{1:T}^m$ the hidden state vector for all time series $\mathbf{x}^n$ such that $c_n = m$. Further, let $K_m = \max(\mathbf{z}_{1:T}^m)$ denote the number of unique states in $\mathbf{z}_{1:T}^m$. Given window size p and the initial observations $\mathbf{x}^n_{-p+1:0}$ for each $n = 1, \dots, N$, the full data likelihood is

$$\begin{aligned}
&P\left(\alpha_0, \mathbf{c}^{1:N},\ \alpha^{1:M}, \lambda_G^{1:N}, \lambda_F^{1:N}, \{\theta_j^n \mid 1{\le}j{\le}K_{c^n}\}_{n=1}^N, \mathbf{z}_{1:T}^{1:M}, \mathbf{x}_{1:T}^{1:N}; \mathbf{x}_{-p+1:0}^{1:N}, p\right) \qquad (5.21)\\
&= \Gamma(\alpha_0; 1, 1)\mathrm{CRP}(\mathbf{c}^{1:N} \mid \alpha_0) \left(\prod_{n=1}^N H_G^n(\lambda_G^n)\right) \left(\prod_{n=1}^N H_F^n(\lambda_F^n)\right) \left(\prod_{n=1}^N \prod_{j=1}^{K_{c^n}} \pi_\Theta^n(\theta_j^n)\right)\\
&\quad \prod_{m=1}^M \left(\Gamma(\alpha^m; 1, 1) \prod_{t=1}^T \left[b_t^m \mathrm{CRP}(z_t^m \mid \mathbf{z}_{1:t-1}^m, \alpha^m) \prod_{n \mid c_n = m} G(\mathbf{x}_{t-p:t-1}^n; D_{tz_t^m}^n, \lambda_G^n) F(x_t^n \mid \theta_{z_t^m}^n) \right]\right).
\end{aligned}$$

The b_t^m terms normalizes the term between the square brackets and are obtained by summing over $k_t^m = 1, \dots, \max(\mathbf{z}_{1:t-1}^m) + 1$. Eq. (5.21) defines the unnormalized posterior distribution of all latent variables given the data. Inference is based on both Markov chain Monte Carlo and sequential Monte Carlo methods, discussed in Section 3.3 of Chapter 3. At a high level, the hidden state assignments $(z_t^m \mid \mathbf{z}_{1:T\setminus t}^m \dots)$ are sampled using a variant of Neal [2000, Algorithm 3], taking care to handle the temporal coupling term b_t^m which is not found in traditional DPM samplers. An alternative SMC scheme to sample $(\mathbf{z}_{1:T}^m \mid \dots)$ jointly as a block is also described. Time series cluster assignments $(c^n \mid \mathbf{c}^{1:N \setminus n}, \dots)$ are transitioned by proposing to migrate $\mathbf{x}^n$ into either an existing or a new cluster, and computing the MH acceptance ratio for each case. Model hyperparameters are sampled using an empirical Bayes approach [Robbins, 1964] and a discrete approximation that resembles the “griddy Gibbs” method of Ritter and Tanner [1992].

Collapsed Model Representation When F and π_Θ in Eq. (5.9) are conjugate, the parameters θ_k^n can be analytically marginalized. The TR-CRP mixture model from Listing 5.1 becomes

$$\begin{aligned}
\alpha &\sim \text{Gamma(1,1)}\\
\lambda_G^n &\sim H_G^n && n = 1, 2, \dots, N\\
\lambda_F^n &\sim H_F^n && n = 1, 2, \dots, N\\
\mathbf{x}_{-p+1:0}^n &::= (x_{-p+1}^n, \dots, x_0^n) && n = 1, 2, \dots, N
\end{aligned}$$

and for each $t = 1, 2, \dots$

$$\begin{aligned}
P\left(z_t = k \mid \mathbf{z}_{1:t-1}, \mathbf{x}_{t-p:t-1}^{1:N}, \alpha, \lambda_G^{1:N}\right) &\propto \mathrm{CRP}(k \mid \alpha, \mathbf{z}_{1:t-1}) \textstyle\prod_{n=1}^N G(\mathbf{x}_{t-p:t-1}^n; D_{tk}^n, \lambda_G^n) && n = 1, \dots, N\\
&\text{where } D_{tk}^n ::= \{\mathbf{x}_{t'-p:t'-1}^n \mid z_{t'} = k, 1 \le t' < t\}\\
&\text{and } k = 1, \dots, \max(\mathbf{z}_{1:t-1}) + 1\\
x_t^n \big| \{z_t = k, \mathbf{x}_{1:t-1}^{1:N}\} &\sim \int_\theta F(\cdot|\theta) \pi_\Theta(\theta \mid D_{tk}'^n, \lambda_F^n) \mathrm{d}\theta && n = 1, \dots, N\\
&\text{where } D_{tk}'^n ::= \{\mathbf{x}_{t'}^n \mid z_{t'} = k, 1 \le t' < t\}.
\end{aligned}$$

Integrating the likelihood F against the prior π_Θ yields a Student-T distribution as in Eq. (5.10), whose hyperparameter updates given $D_{tz_t}'^n$ and λ_F^n are identical to those in Eqs. (5.14)–(5.18) with $i = 0$.

Inferring Hidden State Assignments $(z_t \mid \mathbf{z}_{1:T\setminus t}, \dots)$ Since the factorization prior (5.19) for learning independence gives $M = \max(\mathbf{c}^{1:N})$ separate TR-CRP mixtures, it suffices to describe how to infer $\mathbf{z}_{1:T}$ in one component assuming all N time series are dependent. The joint probability is then

$$\begin{aligned}
&P\left(\alpha, \lambda_G^{1:N}, \lambda_F^{1:N}, \mathbf{z}_{1:T}, \mathbf{x}_{1:T}^{1:N} \; ; \; \mathbf{x}_{-p+1:0}^{1:N}, p\right) \\
&= \Gamma(\alpha; 1, 1)\left(\prod_{n=1}^{N} H_G^n(\lambda_G^n)\right)\left(\prod_{n=1}^{N} H_F^n(\lambda_F^n)\right)\prod_{t=1}^{T}\Bigg[b_t \mathrm{CRP}(z_t \mid \mathbf{z}_{1:t-1}, \alpha) \\
&\qquad\qquad\qquad\qquad\qquad \prod_{n=1}^{N} G(\mathbf{x}_{t-p:t-1}^n; D_{tz_t}^n, \lambda_G^n) F(x_t^n \mid D_{tz_t}'^n, \lambda_F^n)\Bigg].
\end{aligned} \tag{5.22}$$

The normalizing terms for times $t = 1, \dots, T$ are

$$b_t(\mathbf{x}_{1:t-1}^{1:N}, \mathbf{z}_{1:t-1}) ::= \left(\sum_{k=1}^{K_t} \mathrm{CRP}(k \mid \alpha, \mathbf{z}_{1:t-1}) \prod_{n=1}^{N} G(\mathbf{x}_{t-p:t-1}^n; D_{tk}^n, \lambda_G^n)\right)^{-1}, \tag{5.23}$$

where $K_t ::= \max(\mathbf{z}_{1:t-1}) + 1$. These normalizing terms ensure that distribution over hidden states sums to one. It will also be convenient to define the predictive density q_t at time t of data $\mathbf{x}_t^{1:N}$, which is obtained by summing out all possible values of z_t:

$$q_t(\mathbf{x}_{1:t}^{1:N}, \mathbf{z}_{1:t-1}) ::= b_t(\mathbf{x}_{1:t-1}^{1:N}, \mathbf{z}_{1:t-1}) \sum_{k=1}^{K_t}\left[\mathrm{CRP}(k \mid \alpha, \mathbf{z}_{1:t-1}) \prod_{n=1}^{N} G(\mathbf{x}_{t-p:t-1}^n; D_{tk}^n, \lambda_G^n) F(x_t^n \mid D_{tk}'^n, \lambda_F^n)\right]. \tag{5.24}$$

Let the current state of the Markov chain be $(\alpha, \lambda_G^{1:N}, \lambda_F^{1:N}, \mathbf{z}_{1:T})$. Algorithm 5.1 shows a single-site Metropolis-Hastings procedure that targets $(z_t \mid \mathbf{z}_{1:T\setminus t}, \dots)$. Algorithm 5.2 show an SMC scheme to block sample $(\mathbf{z}_{1:T} \mid \dots)$ using particle learning [Carvalho et al., 2010]. In both cases, arbitrary observations may be missing as they are sampled over the course of inference.

There are several computational trade-offs between MH (Algorithm 5.1) and SMC (Algorithm 5.2) for inferring the hidden states. In step 1 of Algorithm 5.1, the terms in Eq. (5.25) must be computed $K = O(\max(\mathbf{z}_{1:T}))$ times. Each assessment requires $O(Np)$ computations, where the factor of N is the product over the time series, and the factor of p is the cost of assessing G in Eq. (5.10). In step 3, computing the terms $b_{t'}$ in Eq. (5.26) requires revisiting $O(T)$ data points. Therefore a single iteration requires $O(TKNp)$ computations, so that the cost of a full sweep over all T time points is $O(T^2KNp)$. It is not necessary to sum over K_t in Eq. (5.23) when computing the $b_{t'}$ terms in Eq. (5.26), since the data in at most two clusters will change when proposing z_t to $z_{t'}$. The sufficient statistics can be updated in constant time using a simple dynamic programming approach.

Computational approximations can help improve the scalability of single-site MH in Algorithm 5.1. In Eq. (5.25), the term D_{Tk}^n in the expression $G(\mathbf{x}_{t-p:t-1}^n; D_{Tk}^n \setminus \{x_t^n\}, \lambda_G^n)$ for the cohesion function includes both observed and imputed data. An alternative approach is to define a "data-dependent" prior following the strategy described by Dunson et al. [2007] in the context of Bayesian density regression. Namely, letting o_t^n be the indicator for having observed x_t^n, the reweighting function G considers only those data points that have actually been observed. With this approximation, Eq. (5.10) becomes

$$G(\mathbf{x}_{t-p:t-1}; D_{tk}, \lambda_G) = \prod_{i=1}^{p} (G_i(x_{t-i}; D_{tki}, \lambda_{Gi}))^{o_{t-i}^n}. \tag{5.29}$$

which reduces the cost of computing Eqs. (5.25) and (5.26) in the presence of missing data. Second,

Algorithm 5.1 Metropolis-Hastings sampler for hidden state assignments in the TR-CRP mixture.

This algorithm resamples $(z_t|\mathbf{z}_{1:T\backslash t}, \dots)$. Let o_t^n be the "observation indicator" so that $o_t^n = 1$ if x_t^n is observed and 0 if it is missing $(n = 1, 2, \dots, N; t = 1, 2, \dots, T)$.

1. Propose z'_t from the categorical distribution:

$$\begin{aligned} P\left(z'_t = k \mid \mathbf{z}_{1:T\backslash t}, \mathbf{x}^{1:N}, \alpha\right) &\propto \mathrm{CRP}(k \mid \alpha, \mathbf{z}_{1:T\backslash t}) \\ &\prod_{n=1}^{N} G(\mathbf{x}_{t-p:t-1}^n; D_{Tk}^n \backslash \{x_t^n\}, \lambda_G^n) \left(F(x_t^n \mid D_{Tk}'^n \backslash \{x_t^n\}, \lambda_F^n)\right)^{o_t^n}, \end{aligned} \tag{5.25}$$

   for $k \in \mathrm{unique}(\mathbf{z}_{1:T\backslash t}) \cup \{\max(\mathbf{z}_{1:T\backslash t}) + 1\}$.

2. For each $n = 1, \dots, N$, if $o_t^n = 0$, then propose $x'^n_t \sim F(\cdot \mid \theta_{z_t}^n)$ otherwise set $x_t'^n \leftarrow x_t^n$.

3. Compute the MH acceptance ratio r, using b_t defined in Eq. (5.23):

$$r = \frac{\prod_{t'>t} b_{t'}(\mathbf{z}_{1:t'-1\backslash t} \cup z'_t, \mathbf{x}_{1:t'-1\backslash t}^{1:N} \cup \mathbf{x}_{t:t}'^{1:N})}{\prod_{t'>t} b_{t'}(\mathbf{z}_{1:t'}, \mathbf{x}_{1:t'-1}^{1:N})}. \tag{5.26}$$

4. Set $(z_t, \mathbf{x}_{t:t}^{1:N}) \leftarrow (z'_t, \mathbf{x}_{t:t}'^{1:N})$ with probability $\min(1, r)$, otherwise leave unchanged. unchanged.

the MH proposal (5.25) resembles the Gibbs proposal in Neal [2000, Algorithm 3], except that it additionally accounts for the temporal coupling which is needed to ensure that the transition leaves Eq. (5.22) invariant. Empirical evidence suggests that, when using the proposal (5.25), the acceptance ratio centers around one. This observation suggests a good initialization or "warm-up" strategy for the Markov chain, prior to running the full MH algorithm, is to run several rounds of step 1 and always accepting the proposal without computing Eq. (5.26), which removes the additional $O(T)$ factor.

Unlike single-site MH in Algorithm 5.1, SMC in Algorithm 5.2 requires $O(KNp)$ to assess Eq. (5.27) in step 2.1.1. The total cost of a complete pass through all T data points (step 2) and all J particles (step 2.1) is therefore $O(JTKNp)$. In SMC, the normalizers b_t need not to be retroactively computed, which is the main overhead of single-site MH in Algorithm 5.1. In addition to its linear scaling in T, SMC more tractably handles missing data. Particle rejuvenation can also be added to the SMC algorithm by applying the MH strategy from Algorithm 5.1 in between observing batches of data (e.g., every 10 time steps), which delivers more accurate inference at the expense of more computation.

Inferring Time Series Cluster Assignments $(c^n \mid \mathbf{c}^{1:N\backslash n}, \dots)$ MCMC is used to infer the cluster assignments in Eq. (5.19) that factorize the time series into independent groups. Let $B \subseteq [N]$ and put

$$L^m(\mathbf{z}_{1:T}, \mathbf{x}_{1:T}^B) ::= \prod_{t=1}^{T} \left[b_t \mathrm{CRP}(z_t \mid \mathbf{z}_{1:t-1}, \alpha^m) \prod_{n=1}^{N} G(\mathbf{x}_{t-p:t-1}^n; D_{tz_t}^n, \lambda^n) F(x_t^n \mid D_{tz_t}'^n, \lambda_F^n) \right]. \tag{5.31}$$

The term L^m is a shorthand for the product from $t = 1$ to T in the full model likelihood (5.22) for a single TR-CRP mixture with latent sequence $\mathbf{z}_{1:T}$, data $\mathbf{x}_{1:T}^B$, and CRP concentration α^m. Further, let $A^m ::= \{n \mid n = 1, \dots, N, c^n = m\}$ be the indices of the time series currently assigned to cluster m.

Algorithm 5.3 shows an MH transition operator for resampling c^n given all other variables. Proposing the hidden state vector z^{M+1} for a singleton from the conditional prior in step 1 avoids the need for transdimensional adjustments such as those in reversible jump MCMC [Green, 1995]. When computing

Algorithm 5.2 Sequential Monte Carlo sampler for hidden state assignments in the TR-CRP mixture.

This algorithm block samples $(\mathbf{z}_{1:T} \mid \dots)$. Let o^n_t be the "observation indicator" so that $o^n_t = 1$ if x^n_t is observed and 0 if it is missing $(n = 1, 2, \dots, N; t = 1, 2, \dots, T)$. Let $J > 0$ be the number of particles. As missing values are simulated over the course of inference, all data has a superscript j to identify any imputed values by particle j.

```
1. Set $w^j \leftarrow 1$ for $j = 1, 2, \dots, J$
2. Repeat for $t = 1, 2, \dots, T$
    2.1. Repeat for $j = 1, 2, \dots, J$
        2.1.1. Sample $z^j_t$ from the multinomial distribution:
               $P\left(z^j_t = k \mid \mathbf{z}^j_{1:t-1}, \mathbf{x}^{j,1:N}, \alpha\right) \propto \mathrm{CRP}(k \mid \alpha, \mathbf{z}^j_{1:t-1}) \prod_{n=1}^{N} G(\mathbf{x}^{n,j}_{t-p:t-1}; D^{n,j}_{tk}, \lambda^n_G) \prod_{n=1}^{N} \left(F(x^n_t \mid D'^{n,j}_{tk}, \lambda^n_F)\right)^{o^n_t},$   (5.27)
               for $k = 1, 2, \dots, \max(\mathbf{z}^j_{1:t-1}) + 1$.
        2.1.2. Update particle weight using predictive density $q_t$ defined in Eq. (5.24):
               $w^j \leftarrow w^j q_t \left(\mathbf{x}^{j,1:N}_{1:t-1} \cup \{x^n_t \mid o^n_t = 1\}, \mathbf{z}^j_{1:t-1}\right).$   (5.28)
        2.1.3. For each $n$ such that $o^n_t = 0$, simulate a value $x^{n,j}_t \sim F(\cdot \mid D'^{n}_{tz^j_t}, \lambda^n_F)$.
    2.2. If resampling criterion met, then:
        2.2.1. Resample $(\mathbf{z}^j_{1:t}, \mathbf{x}^{j,1:N}_{1:t})$ proportionally to $w^j$, $j = 1, 2, \dots, J$.
        2.2.2. Renormalize weights $w^j \leftarrow w^j / \sum_{j'} w^{j'}$, $j = 1, 2, \dots, J$.
3. Resample $j \sim \mathrm{Categorical}(w^1, \dots, w^J)$ and return $(\mathbf{z}^j_{1:T}, \mathbf{x}^{j,1:N}_{1:T})$.
```

the MH acceptance ratio (5.30) in step 5, it is not necessary to recompute all the L^m terms at each iteration. Writing out the full products in Eq. (5.31) reveals several duplicate terms in the numerator and denominator of Eq. (5.30) that cancel one another. The b^m_t terms that do not cancel contain several duplicated components, which can be cached and reused from one transition to the other. Empirical evidence suggests that a similar initialization heuristic to the one described for Algorithm 5.1 provides good transitions with a lower computational overhead, given the similarities between Algorithm 5.3 and the Gibbs proposal in Neal [2000, Algorithm 8].

Algorithm 5.3 Metropolis-Hastings sampler for partition assignments in the TR-CRP mixture.

Let the current state of the Markov chain be $(\alpha_0, \mathbf{c}^{1:N}, \alpha^{1:M}, \lambda_G^{1:N}, \lambda_F^{1:N}, \mathbf{z}_{1:T}^{1:M})$ with observations $\mathbf{x}_{1:T}^{1:N}$. This algorithm resamples the assignment $(c^n \mid \mathbf{c}^{1:N \setminus n}, \dots)$ for time series $\mathbf{x}^n$.

1. If c^n is not a singleton cluster ($|A^{c^n}| > 1$), then generate a proposal sequence by forward sampling $\mathbf{z}_{1:T}^{M+1}$ from the conditional prior where the data $\mathbf{x}_{1:T}^n$ fixed at the observed values. Otherwise, if c^n is a singleton ($|A^{c^n}| = 1$) then reuse the current hidden states by setting $\mathbf{z}_{1:T}^{M+1} ::= \mathbf{z}_{1:T}^{c^n}$.

2. For $m \in \text{unique}(\mathbf{c}^{1:N \setminus n})$, compute

$$p^m = \begin{cases} |A^m| L^m\left(\mathbf{z}_{1:T}^m, \mathbf{x}_{1:T}^n\right) & \text{if } c^n \neq m, \\ (|A^m| - 1) L^m\left(\mathbf{z}_{1:T}^m, \mathbf{x}_{1:T}^n\right) & \text{if } c^n = m. \end{cases}$$

3. Compute the singleton proposal probability:

$$p^{M+1} = \alpha_0 L_{m+1}\left(\mathbf{z}_{1:T}^{M+1}, \mathbf{x}_{1:T}^n\right)$$

4. Sample $c' \sim \text{Categorical}(p^1, \dots, p^{M+1})$.

5. Compute the MH acceptance ratio

$$r = \left(\frac{L^{c'}(\mathbf{z}_{1:T}^{c'}, \mathbf{x}_{1:T}^{A^{c'}} \cup \mathbf{x}_{1:T}^n) L^{c^n}(\mathbf{z}_{1:T}^{c^n}, \mathbf{x}_{1:T}^{A^{c^n}} \setminus \mathbf{x}_{1:T}^n)}{L^{c'}(\mathbf{z}_{1:T}^{c'}, \mathbf{x}_{1:T}^{A^{c'}}) L^{c^n}(\mathbf{z}_{1:T}^{c^n}, \mathbf{x}_{1:T}^{A^{c^n}})} \right) \left(\frac{L^{c^n}(\mathbf{z}_{1:T}^{c^n}, \mathbf{x}_{1:T}^n)}{L^{c'}(\mathbf{z}_{1:T}^{c'}, \mathbf{x}_{1:T}^n)} \right). \tag{5.30}$$

6. Set $c^n \leftarrow c'$ with probability $\min(1, r)$, else leave c^n unchanged.

Inferring Hyperparameters $(\alpha_0, \{\alpha^m\}, \{\lambda_G^n\}, \{\lambda_F^n\} \mid \dots)$ For each hyperparameter, a grid of 30 data-dependent logarithmically-spaced bins is constructed as follows:

$$\begin{aligned}
\text{grid}(\alpha_0) &= \text{logspace}(1/N, N) && \text{(Outer CRP Concentration)} && (5.32) \\
\text{grid}(\alpha^m) &= \text{logspace}(1/T, T) && \text{(TR-CRP Concentration)} && (5.33) \\
\text{grid}(m_0^n) &= \text{logspace}(\min(\mathbf{x}_{1:T}^n) - 5, \max(\mathbf{x}_{1:T}^n) + 5) && \text{(Normal-InverseGamma Location)} && (5.34) \\
\text{grid}(V_0^n) &= \text{logspace}(1/T, T) && \text{(Normal-InverseGamma Variance)} && (5.35) \\
\text{grid}(a_0^n) &= \text{logspace}(\text{ssqdev}(\mathbf{x}_{1:T}^n)/100, \text{ssqdev}(\mathbf{x}_{1:T}^n)) && \text{(Normal-InverseGamma Shape)} && (5.36) \\
\text{grid}(b_0^n) &= \text{logspace}(1, T), && \text{(Normal-InverseGamma Scale)} && (5.37)
\end{aligned}$$

where ssqdev denotes the sum of squared deviations from the mean. The grids for the Normal-InverseGamma hyperparameters are used for both λ_F^n and λ_G^n ($n = 1, 2, \dots, N$). The sampler cycles through the grid points of each hyperparameter and assess the full likelihood at each bin using Eq. (5.21). This method is simple, computationally feasible, and finds reasonable hyperparameter settings. Alternative approaches based on expectation-maximization or slice sampling are also possible, although they require more complex implementations such as gradient computation, inverse densities, or numerical methods to compute slices. It is also possible to normalize the time series values to the unit interval before performing inference, which would make it easier to specify a broad prior over valid hyperparameter values for data within that range.

5.3 Forecasting, Clustering, and Imputation with TR-CRP Mixtures

A collection of S approximate posterior samples $\{\hat{\xi}^1, \dots, \hat{\xi}^S\}$ of all latent variables can be used to solve a variety of predictive inference queries about the time series $\mathbf{x}^1, \dots, \mathbf{x}^N$.

Forecasting For future time points $T{+}1, \dots, T{+}h$ over an h step horizon, the forecasts $x^n_{T+1}, \dots, x^n_{T+h}$ ($n = 1, \dots, N$) are generated by ancestral sampling. That is, first draw $\tilde{s} \sim \mathrm{Uniform}[1 \dots S]$, then simulate step 5 of Listing 5.1 using the latent variables in chain $\xi^{\tilde{s}}$.

Clustering For a pair of time series $(\mathbf{x}^i, \mathbf{x}^j)$, the posterior probability that they are dependent is the fraction of samples in which they are in the same cluster:

$$P\left(c^i = c^j \middle| \mathbf{x}^{1:N}\right) \approx \frac{1}{S} \sum_{s=1}^{S} \mathbb{I}\left[\hat{c}^{i,s} = \hat{c}^{j,s}\right]. \tag{5.38}$$

Imputation Posterior inference yields samples of each hidden state $\hat{z}^{\cdot,s}_t$ for all in-sample time points $1 \le t \le T$. The posterior distribution of a missing value is thus:

$$P\left(x^n_t \in B \middle| \mathbf{x}^{1:N} \setminus \{x^n_t\}\right) \approx \frac{1}{S} \sum_{s=1}^{S} F(B \mid \hat{\theta}^{n,s}_{\hat{z}^{\hat{c}^{n,s}}_t}). \tag{5.39}$$

5.4 Applications to Macroeconomic and Flu Data

The TR-CRP mixture model can be used to effectively solve challenging data analysis tasks that involve multivariate time series. Section 5.4.1 shows how the TR-CRP mixture is able to automatically detect clusters of interpretable temporal patterns in macroeconomic time series from the Gapminder Foundation [2022] database. Sections 5.4.2 and 5.4.3 apply the TR-CRP mixture to solve imputation and forecasting tasks on seasonal flu data from the US CDC, with quantitative results that outperform several widely used baselines on these challenging problems.

5.4.1 Clustering Macroeconomic Data

The Gapminder Foundation [2022] database contains dozens of macroeconomic time series for 170 countries spanning 50 years. Because fluctuations due to events such as natural disasters, financial crises, or healthcare epidemics are poorly described by parametric or hand-designed causal models that ignore these unpredictable events, a key objective is to automatically discover the number and kinds of patterns in the data. The top panel of Figure 5.5 shows all 170 GDP time series in the Gapminder dataset and the remaining nine panels show inferred clusters from the TR-CRP mixture ($p = 5$ years). These clusters reflect commonsense and visually distinct patterns: within each group of dependent time series there is a similar 50-year trajectory of GDP, which include temporal patterns such as linear growth, exponential growth, boom-and-bust cycles, and outliers.

Figure 5.6 shows discovered structure in historical cell phone subscription data. The left-most plot in Figure 5.6a shows the observed 170 time series and the remaining plots show three representative clusters in one posterior sample of the TR-CRP mixture. Each cluster corresponds to countries whose changepoint in cell phone subscribers from zero to nonzero lies in a distinct window: 1985–1995 in cluster 1, 1995–2000 in cluster 2, and 2000–2005 in cluster 3. Figures 5.6b and 5.6c compare the pairwise dependence probability matrix with the pairwise cross-correlation matrix for the 170 time series, which show that the TR-CRP mixture captures more refined independence structure in the data.

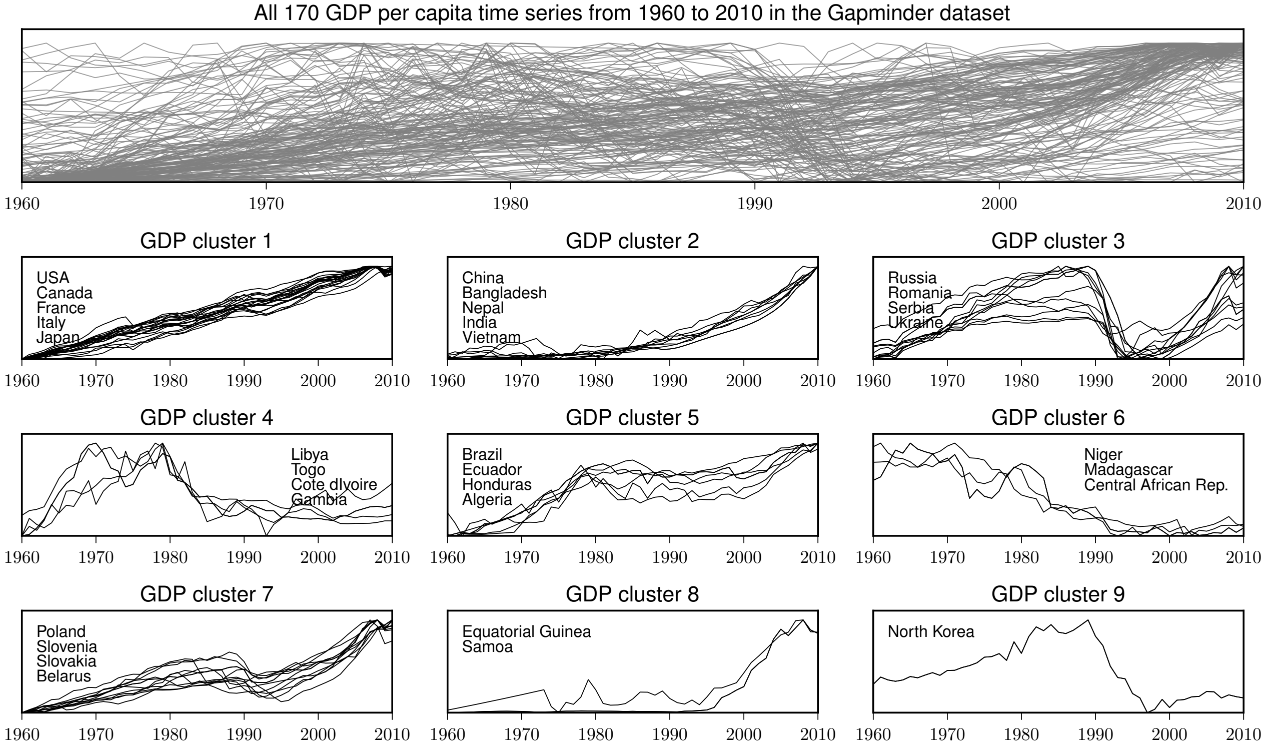
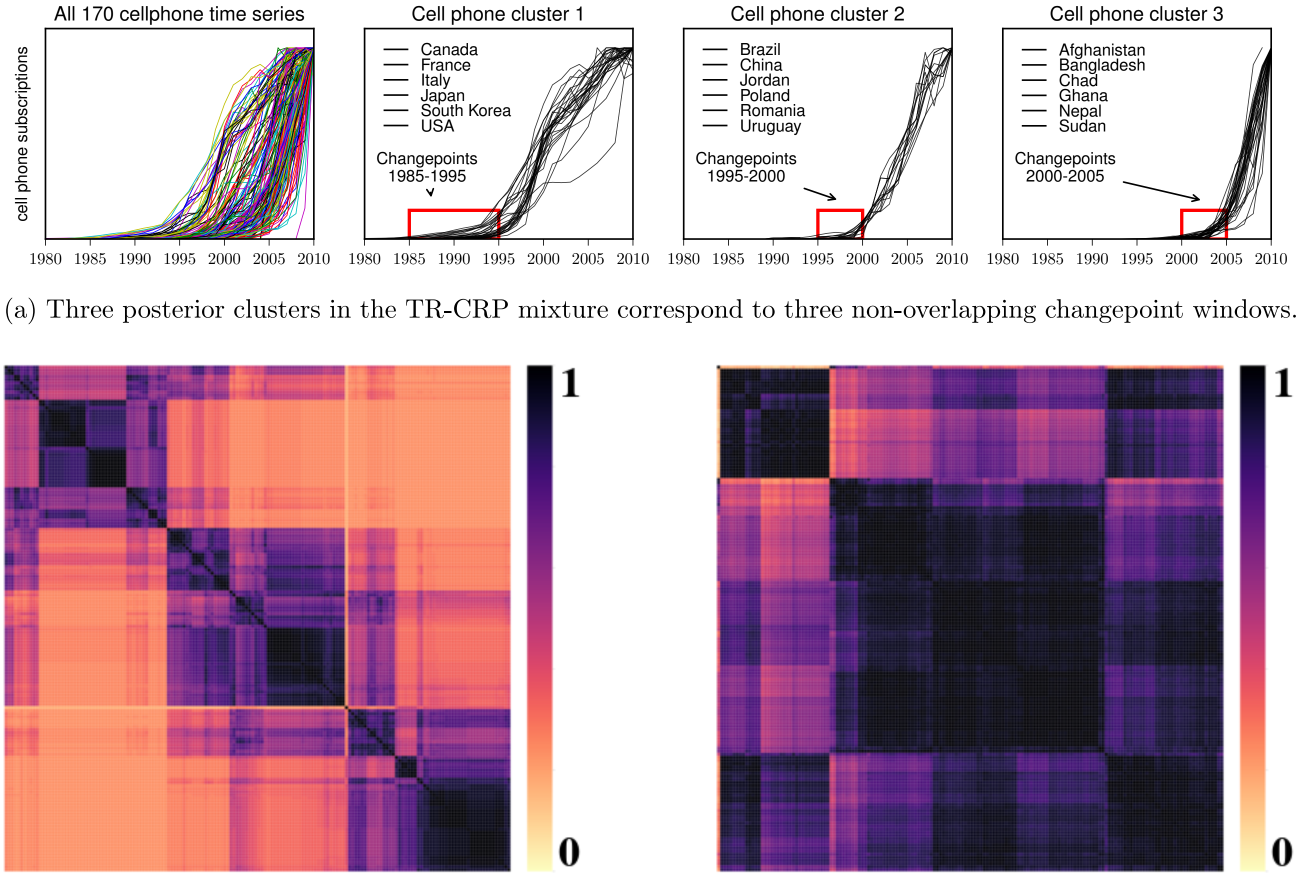


Figure 5.5: Given GDP per capita data for 170 countries from 1960–2010, the TR-CRP mixture detects groups of time series with qualitatively distinct temporal patterns. The top panel shows an overlay of all the GDP time series and nine representative clusters in the remaining panels. Countries within each cluster, of which a subset are labeled, share similar political, economic, and/or geographic characteristics. For instance, cluster 1 contains Western democracies with steady economic growth. Cluster 2 includes China and India, whose GDP growth rates have outpaced those of industrialized nations. Cluster 3 contains former communist nations, whose economies crashed after fall of the Soviet Union. Outliers such as Samoa, Equatorial Guinea, and North Korea can be seen in clusters 8 and 9.

5.4.2 Imputing Multivariate Flu Rates

Predicting flu rates is a fundamental task in public health. The US CDC maintains an extensive dataset of flu rates and related time series such as temperature measurements and vaccination rates.

Figure 5.3 shows the influenza-like-illness rate (ILI, or flu), tweets, and minimum temperature time series in United States Region 4, as well as six temporal regimes detected by one posterior sample of the TR-CRP mixture model ($p = 10$ weeks). Measurements are taken weekly from January 1998 to June 2015. To investigate the performance of the TR-CRP mixture on multivariate imputation, windows of length 10 were dropped at a rate of 5% from flu time series in US Regions 1–10. Figure 5.7 shows flu time series for US Regions 2, 4, 7, and 9, as well joint imputations obtained from the TR-CRP mixture using Eq. (5.39) and example imputations for the 2013 flu season in Region 9. Table 5.1 shows a quantitative comparison of imputation accuracy to several baselines, where statistically significant lowest errors are shown in bold. The results show that the TR-CRP mixture achieves comparable accuracy to the state-of-the art Amelia II [Honaker et al., 2011] baseline, although neither method is consistently more accurate across all regions.



(a) Three posterior clusters in the TR-CRP mixture correspond to three non-overlapping changepoint windows.

(b) Pairwise dependence probability heatmap

(c) Pairwise cross-correlation heatmap

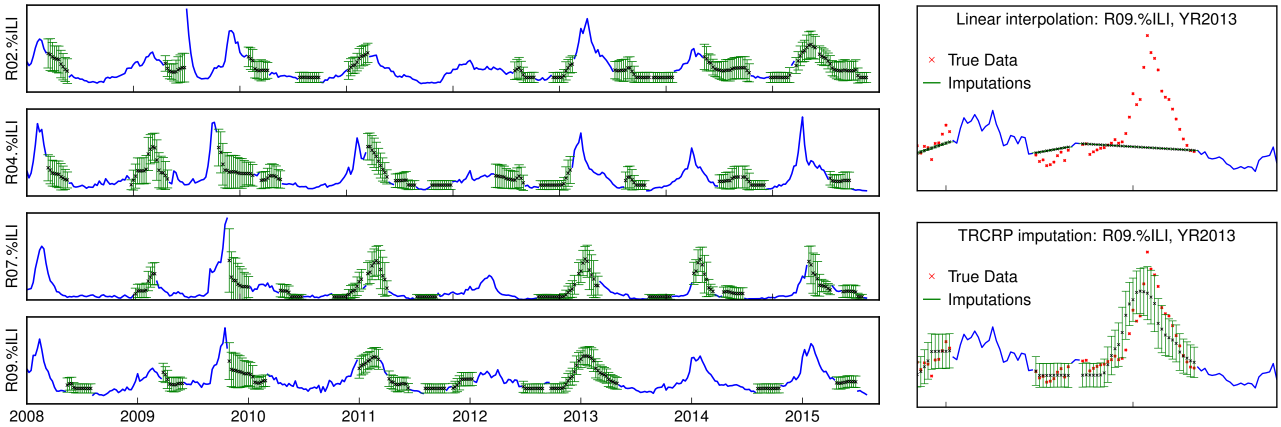
Figure 5.6: Discovering changepoint locations in cell phone subscriptions for 170 countries in the Gapminder dataset. (a) The three clusters, extracted from one posterior sample, correspond to three regimes each with non-overlapping changepoint windows, annotated by red boxes. The representative countries in each cluster have similar adoption times of cell phone technology, a feature which differs across the clusters. (b)–(c) Comparison of pairwise dependence probabilities, averaged over 60 posterior samples using Eq. (5.38), and the matrix of pairwise cross-correlations (bottom) between all pairs 170 time series. Each row and column is a time series, and the color of a cell is a value between 0 and 1 that indicates the posterior dependence probability, resp. the cross-correlation coefficient (if significant at the 0.05 level with Bonferroni correction). The TR-CRP mixture detects more refined dependence structures than those captured by linear correlation.

5.4.3 Forecasting Multivariate Flu Rates

To quantitatively assess the forecasting quality of the TR-CRPM mixture, the entire 2015 season for all 10 US regions was held out. Forecasts for the unobserved weeks were generated on a rolling basis. In particular, for each week $t = 2014.40, \dots, 2015.20$ the task is to generate a forecast for $\mathbf{x}^{n,\text{flu}}_{t:t+h}$, conditioned on $\mathbf{x}^{n,\text{flu}}_{1:t-2}$ and all available covariate data up to time t, where the horizon $h = 1, \dots, 10$. A key challenge is that when forecasting $\mathbf{x}^{n,\text{flu}}_{t:t+h}$, the most recent flu measurement $x^{n,\text{flu}}_{t-2}$ is two weeks old. Moreover, covariate time series are themselves sparsely observed in the training data; for example, all Twitter data is missing before June 2013, as shown in the top panel of Figure 5.3. Table 5.2 shows the forecasting accuracy from several widely used and domain-general baselines that do not require custom modeling for obtaining forecasts. Not all baselines can include the weather and tweets covariate information, as

Table 5.1: Mean absolute imputation errors in 10 United States flu regions using various baselines.

	United States CDC Flu Region									
	R01	R02	R03	R04	R05	R06	R07	R08	R09	R10
Mean Imputation	0.65	0.85	0.91	1.07	0.66	1.20	1.17	0.75	0.80	1.10
Linear Interpolation	0.43	0.63	0.57	**0.42**	0.44	0.71	0.71	0.35	0.43	0.72
Cubic Splines	1.01	0.72	0.61	0.89	0.69	1.68	1.42	0.63	0.99	1.47
Multi-output GP	0.36	0.57	0.32	0.58	0.30	0.57	0.62	**0.34**	0.43	0.56
Amelia II	0.29	0.52	0.25	0.45	**0.29**	**0.53**	**0.53**	0.37	0.39	**0.51**
TR-CRP Mixture	**0.23**	**0.47**	**0.23**	0.49	0.31	0.55	0.75	**0.34**	**0.37**	0.67



(a) Example joint imputations in four US regions.

(b) Example imputations in R09.

Figure 5.7: Jointly imputing missing flu data in 10 US Regions over eight seasons.

indicated in the legend of Table 5.2. For example, the seasonal ARIMA [Hyndman and Khandakar, 2008] baseline is not able to handle missing covariates in either the training set or over the course of the forecast horizon. The multivariate TR-CRP mixture, which uses the flu, weather, and tweet data, consistently produces the most accurate forecasts for all horizons, with statistically significant lowest errors in bold. Figure 5.8 compares the forecasts using various baselines at two timepoints in the 2014–2015 flu season. Both Facebook Prophet [Taylor and Letham, 2018] and ARIMA incorrectly forecast the peak behavior and produced biased forecasts in the post-peak regime. The HDP-HSMM [Johnson and Willsky, 2013] is able to include weather data because it is is fully observed but fails to accurately detect flu peaks. The univariate TR-CRP mixture, which only models the flu, produces similar errors to the Gaussian process with periodic kernel, although the latter gives wider posterior error bars even in the relatively noiseless post-peak regime. The multi-output GP [Alvarez and Lawrence, 2008] uses both weather and tweet covariates, but does not deliver an improvement in predictive accuracy over univariate methods. The multivariate TR-CRP mixture produces the most accurate and calibrated forecasts on this benchmark set.

5.5 Related Work

The TR-CRP mixture is a time series extension of product partition models for cross-sectional data [Ishwaran and James, 2003, Shahbaba and Neal, 2009, Park and Dunson, 2010, Mueller and Quintana, 2010, Mueller et al., 2011]. Product partition models apply to an exchangeable sequence $(x_1, \dots, x_m)$ of random variables with exogenous covariates $(y_1, \dots, y_m)$ by reweighting the prior CRP cluster prob-

Table 5.2: Mean absolute error of flu predictions for 10 forecast horizons averaged over US flu regions.

		Length of forecast horizon (weeks)									
	Covariates	1	2	3	4	5	6	7	8	9	10
Linear Extrapolation	–	0.65	0.79	0.93	1.08	1.24	1.39	1.55	1.70	1.86	2.01
GP(SE+PER+WN)	–	0.53	0.60	0.66	0.71	0.75	0.79	0.82	0.85	0.87	0.89
GP(SE×PER+WN)	–	0.50	0.57	0.62	0.67	0.71	0.74	0.78	0.81	0.84	0.86
Facebook Prophet	–	0.83	0.84	0.85	0.85	0.85	0.86	0.86	0.87	0.87	0.87
Seasonal ARIMA	–	0.64	0.76	0.84	0.92	0.98	1.04	1.08	1.13	1.16	1.19
TR-CRP Mixture	–	0.54	0.58	0.62	0.67	0.71	0.76	0.80	0.83	0.86	0.89
HDP-HSMM	Weather	0.69	0.72	0.76	0.79	0.82	0.84	0.86	0.88	0.89	0.90
Multi-output GP	Weather + Tweets	0.70	0.77	0.84	0.88	0.91	0.93	0.95	0.97	0.99	1.01
TR-CRP Mixture	Weather + Tweets	**0.46**	**0.49**	**0.51**	**0.53**	**0.56**	**0.58**	**0.59**	**0.61**	**0.62**	**0.64**

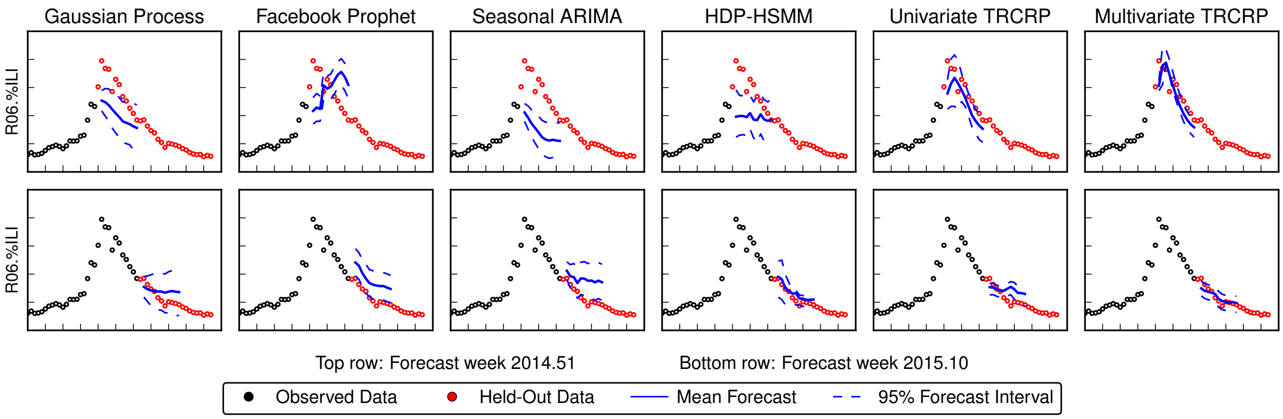


Figure 5.8: Comparison of forecasts in US Region 6 for week 2014.51 (pre-peak) and week 2015.10 (post-peak) using the TR-CRP mixture and several baselines. The multivariate TR-CRP mixture accurately forecasts seasonal dynamics in both cases, whereas baseline methods produce inaccurate forecasts or miscalibrated uncertainties.

ability $P(z_i = k)$ for each observation x_i using the covariate y_i. The TR-CRP mixture extends this idea to time series, where the prior CRP cluster probability $P(z_t = k)$ for x_t is reweighted based on the p most recent values $\mathbf{x}_{t-1:t-p}$. Moreover, when using the independence discovery prior in Section 5.1.4 and removing all temporal dependencies by setting $p = 0$ or $G \propto 1$, the TR-CRP mixture coincides with the CrossCat model [Mansinghka et al., 2016] and generalizes the MultiMixture DSL in Chapter 4.

Zhu et al. [2005] and Ahmed and Xing [2008] develop temporal extensions to Chinese restaurant process mixtures for dynamic clustering. The latter work derives a recurrent CRP as the limit of a finite dynamic mixture model. Unlike the TR-CRP mixture, these models are used to cluster batched data and dynamic topic modeling [Blei and Lafferty, 2006] rather than data analysis tasks such as forecasting or imputing multivariate time series.

Nonparametric Bayesian techniques for modeling multivariate time series include dependent Dirichlet processes [Rodriguez and ter Horst, 2008]; hierarchical Dirichlet process hidden Markov models [Fox et al., 2008, Johnson and Willsky, 2013]; Dirichlet Process and Pitman-Yor mixtures of nonlinear state-space models [Nieto-Barajas and Contreras-Cristan, 2014, Lin et al., 2019]; and linear dynamic models with Dirichlet Process mixture and Polya tree priors over the innovations [Caron et al., 2008, Nieto-Barajas and Quintana, 2016]. As extensions of state-space models, these techniques support explicit temporal structures, such as periodicity, trends, and autoregression, for modeling a specific dataset pattern. These features are useful if the practitioner has strong prior knowledge about the underlying structure, but they require more expensive inference algorithms such as running forward-filtering backward-smoothing in the inner loop of MCMC. In contrast to these methods, the TR-CRP mixture is a purely empirical model that detects and emulates patterns in the data by effectively maintaining an infinite memory, as illustrated in Figure 5.2. The resulting model is designed to provide a good baseline explanation of many datasets with widely varying structures, such as the GDP trajectories in Figure 5.5, where expert modeling knowledge is either not available or too expensive to obtain. Another distinction is that the TR-CRP mixture leverages a Bayesian nonparametric structure learning prior (Section 5.1.4) to infer groups of independent time series that can be modeled separately, which many of these previous techniques do not discover.

Recent methods in the literature have also extended the TR-CRP mixture. Tong and Torenvliet [2020], for example, describe an alternative learning algorithm for the TR-CRP mixture based on online variational inference using a stick-breaking representation and apply the method to solve anomaly detection tasks in streaming data from industrial processes.

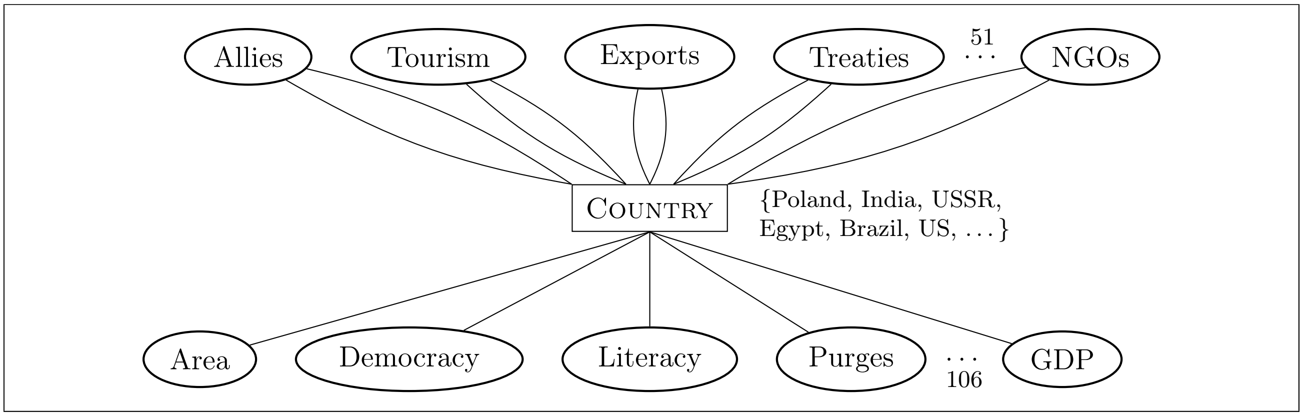
Chapter 6

Synthesizing Models for Relational Data

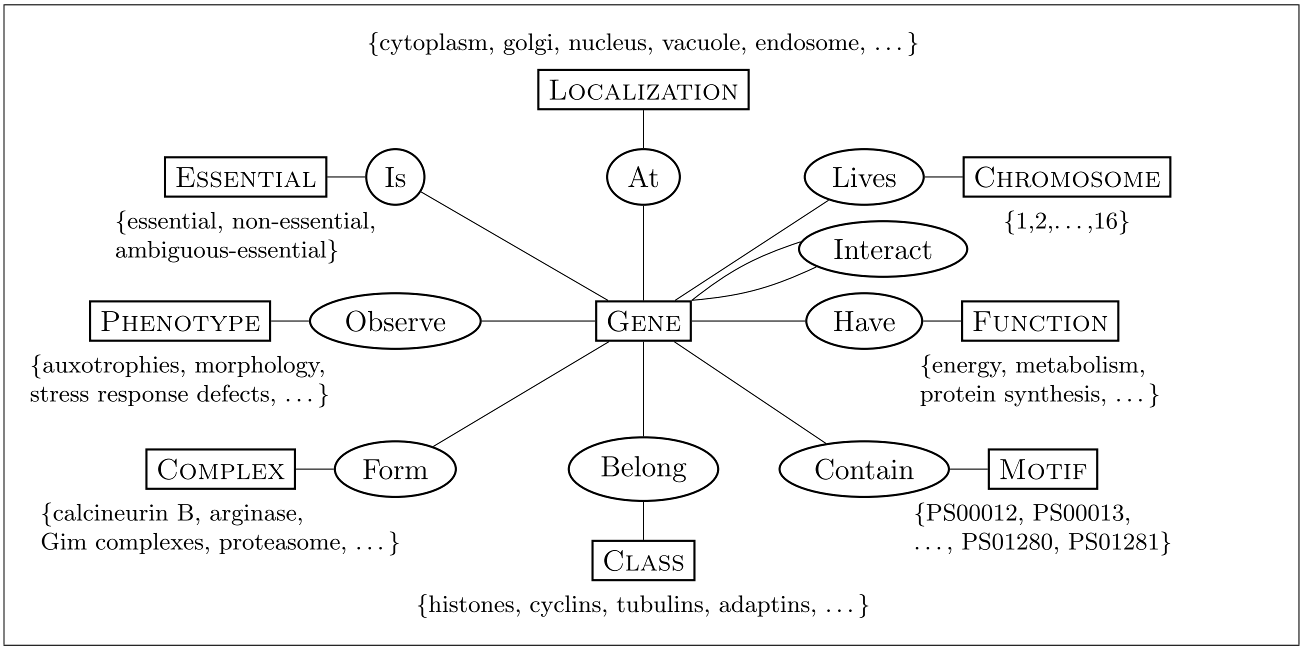
Learning models for relational data is a widely studied problem that arises in a number of settings such as business intelligence [Chaudhuri et al., 2011], social networks [Carrington et al., 2005], bioinformatics [Rual et al., 2005], and recommendation systems [Su and Khoshgoftaar, 2009], amongst many others [Džeroski and Lavrač, 2001]. In this setting, the data consists of (i) a set of entities; (ii) a set of attributes of these entities; and (iii) a set of interactions that occur between one more entities. The goal is to learn models that are useful for explaining or making predictions about the data. Figure 6.1 shows examples of relational systems from politics and genomics. In the political data, one problem could be to discover what attributes of a particular country and interactions with other countries are likely to make it an attractive tourist hub. In the genomics data, the objective could be to predict which complexes a particular gene is likely to form, given information about its motifs, functions, and interactions with other genes. This chapter addresses the problem of automatically learning probabilistic models for a variety of relational systems, given a dataset of noisy sparse observations.

Learning probabilistic models for relational data, which generalize cross-sectional data (Chapter 4), is an exceptionally difficult task. One approach to simplify the relational learning problem is to posit a collection of hidden variables that decouple the relationships between observed variables. Using Bayesian nonparametrics, the values and dimensionality of these hidden variables can be inferred from data. Several papers have explored Bayesian nonparametric models for relational data [Kemp et al., 2006, Xu et al., 2006, Roy and Teh, 2008, Sutskever et al., 2009, Kim et al., 2013, Nakano et al., 2014, Xuan et al., 2017, Fan et al., 2018]; refer to Fan et al. [2021] for a survey. The technique presented in this chapter builds on the infinite relational model [IRM; Kemp et al., 2006, Xu et al., 2006], a widely used, flexible, and lightweight Bayesian nonparametric method for modeling a variety relational systems. The IRM is a cluster-based model: informally, to decide whether a binary relation R holds between a pair of entities i and j, the IRM flips a coin whose weight depends on the (latent) cluster assignments of i and j. A strength of the IRM, reviewed in Section 6.1, is its ability to extract meaningful partitions from observational data. However, as shown in Section 6.2, two limitations inherent to the IRM make the model susceptible to combinatorial over-clustering and fail to discover certain predictive structure between dependent but non-identically distributed relations.

To address these limitations, Section 6.3 introduces the hierarchical infinite relational model (HIRM), which combines the flexibility of the IRM with a structure learning prior that infers subsets of relations that are probably independent of one another. By allowing different relations to be explained by different models, the HIRM specifies a large hypothesis space that includes the standard IRM as well as more compact models that can only be approximated by an IRM using a combinatorially large number of clusters. Section 6.4 presents algorithms for posterior inference. The evaluations in 6.5 show that the HIRM makes more accurate predictions and discovers more interpretable clustering structure as compared to the IRM, while retaining a highly flexible framework for automatic Bayesian structure discovery in a variety of relational systems.



(a) Political Data [Rummel, 1999]


{cytoplasm, golgi, nucleus, vacuole, endosome, . . . }
LOCALIZATION
ESSENTIAL
Is
At
Lives
CHROMOSOME
{essential, non-essential,
ambiguous-essential}
{1,2,. . . ,16}
Interact
PHENOTYPE
Observe
GENE
Have
FUNCTION
{auxotrophies, morphology,
stress response defects, . . . }
{energy, metabolism,
protein synthesis, . . . }
COMPLEX
Form
Belong
Contain
MOTIF
{calcineurin B, arginase,
Gim complexes, proteasome, . . . }
{PS00012, PS00013,
. . . , PS01280, PS01281}
CLASS
{histones, cyclins, tubulins, adaptins, . . . }

(b) Yeast Genomics Data [Cheng et al., 2002]

Figure 6.1: Two relational systems analyzed using the hierarchical infinite relational model introduced in this chapter. Domains appear in rectangular boxes, relations appear in round ellipses, and the entities within a domain are written between curly braces. (a) One domain, five unary relations, and five binary relations. (b) Nine domains and nine binary relations. Unary relations represent "attributes" of entities in a domain. Binary and higher-order relations represent "interactions" between domain entities.

6.1 Background: Infinite Relational Model

This section reviews the IRM using a generalization of the relational systems originally described in Kemp et al. [2006] and Xu et al. [2006].

Definition 6.1. A *relational system* S consists of n domains $D_1, \dots, D_n$ and m relations $R_1, \dots, R_m$. Each *domain* $D_i ::= \{e^i_1, e^i_2, \dots\}$ $(1 \le i \le n)$ is a countably infinite set of distinct *entities*. Each *relation* R_k is a map from the Cartesian product of t_k domains to an arbitrary codomain C_k $(1 \le k \le m)$. The symbol d_{ki} denotes the index of the domain for the i^{th} argument of R_k $(1 \le k \le m,\ 1 \le i \le t_k)$. «

Example 6.2. Suppose system S has $n = 4$ domains D_1, D_2, D_3, D_4; and $m = 3$ relations R_1, R_2, R_3:

$$R_1 : D_1 \times D_1 \to \{0, 1\}, \tag{6.1}$$
$$R_2 : D_1 \times D_3 \times D_4 \to \{1, 2, \dots\}, \tag{6.2}$$
$$R_3 : D_2 \to (-\infty, \infty). \tag{6.3}$$

That is, R_1 is a binary relation taking binary values, R_2 is a ternary relation taking positive integer values, and R_3 is a unary relation taking real values. The following quantities can be directly read off from Eqs. (6.1)–(6.3):

$$\begin{aligned}
&t_1 = 2; \quad d_{11} = 1, d_{12} = 1; && C_1 = \{0, 1\};\\
&t_2 = 3; \quad d_{21} = 1, d_{22} = 3, d_{23} = 4; && C_2 = \{1, 2, \dots\};\\
&t_3 = 1; \quad d_{31} = 2; && C_3 = (-\infty, \infty).
\end{aligned}$$

«

Remark 6.3. To simplify notation, for a given relation $R : D_1 \times \cdots \times D_n \to C$ and entity indexes $i_1, \dots, i_n \in \mathbb{N}$, the notation $R(i_1, \dots, i_n)$ means $R(e^1_{i_1}, \dots, e^n_{i_n})$. «

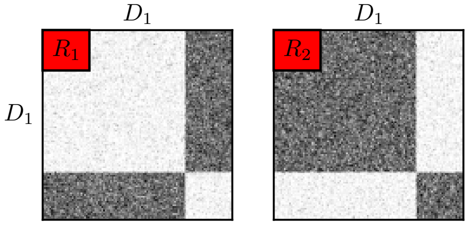
Consider a system S with n domains and m relations. For each $i = 1, \dots, n$, the IRM assumes that the entities $\{e^i_1, e^i_2, \dots\}$ in domain D_i have integer cluster assignments $\{z^i_1, z^i_2, \dots\} =: z^i$. The IRM defines a joint probability distribution over cluster assignments and relation values with the following factorization structure:

$$P(z^1, \dots, z^n, R_1, \dots, R_m) = \prod_{i=1}^{n} P(z^i) \prod_{k=1}^{m} P(R_k \mid z^1, \dots, z^n). \tag{6.4}$$

To allow the IRM to discover an arbitrary number of clusters, the cluster assignments z^i for the entities in D_i are given a nonparametric prior that assigns a positive probability to all possible partitions using the Chinese restaurant process [CRP; Aldous, 1985]. For each $i = 1, \dots, n$, the cluster assignment probabilities $P(z^i) = P(z^i_1, z^i_2, \dots)$ in Eq. (6.4) are defined inductively with $z^i_1 ::= 1$, and

$$P(z^i_\ell = j \mid z^i_1, \dots, z^i_{\ell-1}) ::= \frac{1}{(\ell - 1) + \alpha} \begin{cases} n_j & \text{if } 1 \le j \le M \\ \alpha & \text{if } j = M + 1, \end{cases} \qquad (\ell = 2, 3, \dots) \tag{6.5}$$

where $n_j ::= \sum_{c=1}^{\ell-1} \mathbf{1}\left[z^i_c = j\right]$ is the number of previous entities at cluster j; $M ::= \max\{z^i_1, \dots, z^i_{\ell-1}\}$ is the number of clusters among the first $\ell - 1$ entities; and $\alpha > 0$ is the concentration parameter. The cluster assignments $z^1, \dots, z^n$ across the n domains are mutually independent, each drawn from a separate CRP (6.5). Next, for each relation R_k $(1 \le k \le m)$, a set of parameters $\theta_k(j_1, \dots, j_{t_k})$ is used to dictate the distribution of $R_k(i_1, \dots, i_{t_k})$, where $j_1, \dots, j_{t_k}, i_1, \dots, i_{t_k} \in \mathbb{N}$. The relation values depend only the cluster assignments, i.e., $R_k(i_1, \dots, i_{t_k})$ and $R_k(i'_1, \dots, i'_{t_k})$ share the same parameter



Relational System	**IRM Encoding**
$R_1 : D_1 \times D_1 \to \{0,1\}$	$R' : D' \times D_1 \times D_1 \to \{0,1\}$
$R_2 : D_1 \times D_1 \to \{0,1\}$	where $D' ::= \{R_1, R_2\}$

Figure 6.2: When used to cluster relations, the standard IRM uses a higher-ordering encoding that requires relations to be defined on the same domain and assumes that all relations within a cluster are identically distributed. While the relations R_1 and R_2 shown above are based on identical partitions of D_1 they are not identically distributed and thus assigned to different clusters by the IRM. In contrast, the HIRM can learn clusters of relations defined on different domains and can assign non-identically distributed relations to the same cluster. Refer to Figure 6.6a for a real-world example.

whenever $z^{d_{k\ell}}_{i_\ell} = z^{d_{k\ell}}_{i'_\ell}$ for each $\ell = 1, \dots, t_k$. Therefore, the generative model of the IRM is

$$\{z^i_1, z^i_2, \dots\} \sim \text{CRP}(\alpha_i) \qquad (i = 1, \dots, n) \tag{6.6}$$

$$\theta_k(j_1, \dots, j_{t_k}) \sim \pi_k(\lambda_k) \qquad (k = 1, \dots, m;\ j_1, \dots, j_{t_k} \in \mathbb{N}) \tag{6.7}$$

$$R_k(i_1, \dots, i_{t_k}) \sim L_k(\theta_k(z^{d_{k1}}_{i_1}, \dots, z^{d_{kt_k}}_{i_{t_k}})) \qquad (k = 1, \dots, m;\ i_1, \dots, i_{t_k} \in \mathbb{N}), \tag{6.8}$$

where $\{\alpha_i\}_{i=1}^n$ and $\{\lambda_k\}_{k=1}^m$ are hyperparameters. Eq. (6.8) ensures that all entities within the same cluster are identically distributed. The prior π_k and likelihood L_k distributions in Eqs. (6.7) and (6.8) can be set depending on the codomain C_k of R_k (e.g., beta-Bernoulli for binary data, gamma-Poisson for counts, chisquare-normal for real values, etc.). Kemp et al. [2006] used the IRM to discover structure in a variety of real-world relational systems, including:

(a) Random graphs, with one domain D representing the vertices and one relation $R : D \times D \to \{0,1\}$ representing the edges, so that $R(i,j) = 1$ if there is an edge from e_i to e'_j.

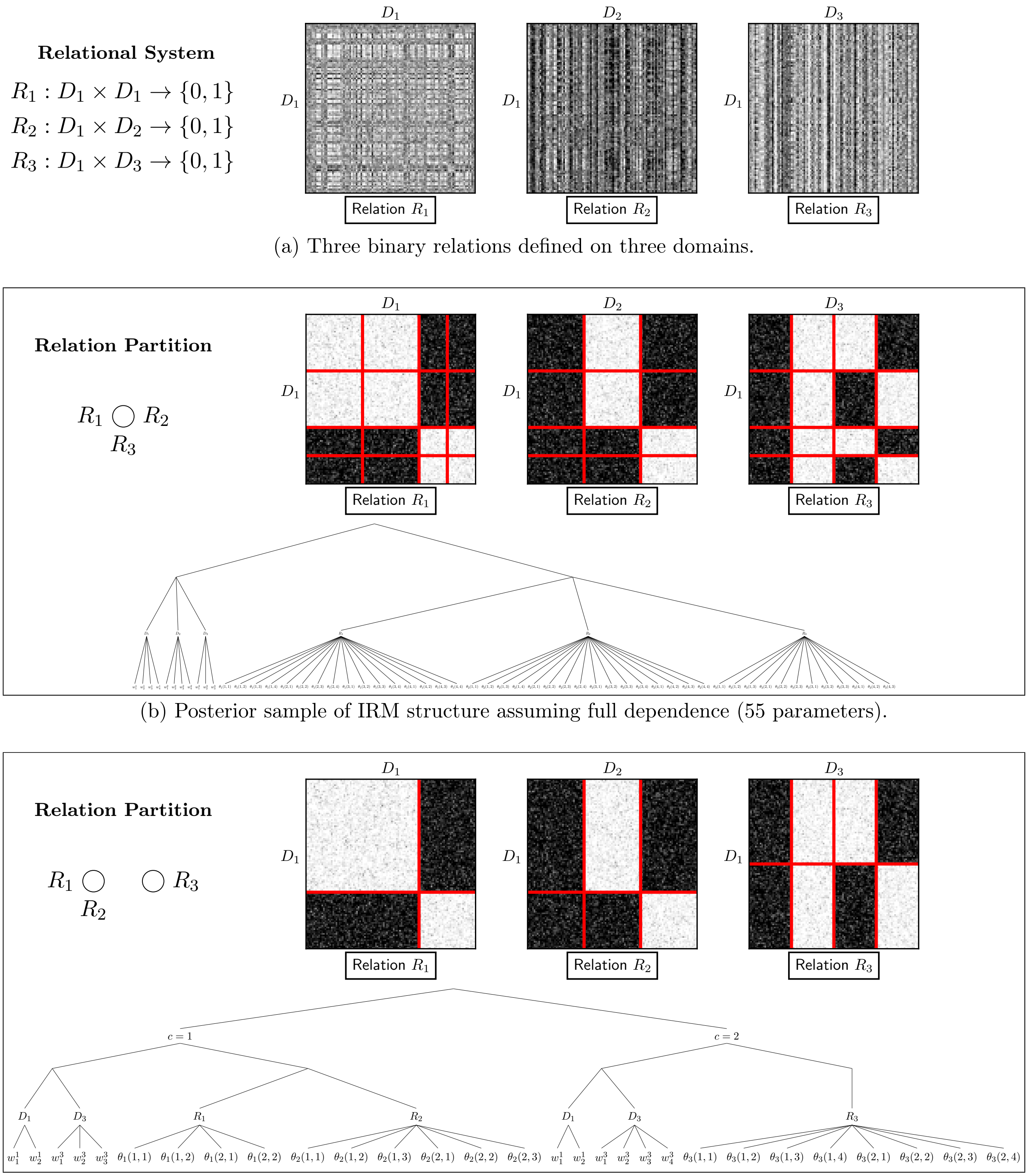
(b) Object-attribute data (Chapter 4), where D_1 are the objects (individuals), D_2 are the attributes and the relation $R : D_1 \times D_2 \to \{0,1\}$ is such that $R(i,j) = 1$ if entity e^1_i has attribute e^2_j.

(c) Political data with multiple attributes and interactions (Figure 6.1a), where D_1 are countries, D_2 are attributes, and D_3 are interactions. Attributes are represented by the relation $R_1 : D_1 \times D_2 \to \{0,1\}$ and interactions are represented by the relation $R_2 : D_1 \times D_1 \times D_3 \to \{0,1\}$ where $R_2(i,j,k) = 1$ if countries e^1_i and e^1_j perform interaction e^3_k.

6.2 Limitations of the Infinite Relational Model

There are two fundamental modeling limitations of the standard IRM that motivate the hierarchical IRM with independence discovery presented in Section 6.3.

Enforcing Shared Domain Clustering Leads to Overfitting The IRM assumes that each domain D_i has a single clustering $z^i = \{z^i_1, z^i_2, \dots\}$ that globally dictates the partition of its entities $\{e^i_1, e^i_2, \dots\}$. That is, the same cluster assignments z^i are used for all of the relations in which D_i participates. Figure 6.3 shows that this inductive bias can lead to substantial over-clustering of the domain entities and a failure to accurately model data when there exists structural independencies between relations.


Relational System
$R_1 : D_1 \times D_1 \to \{0,1\}$
$R_2 : D_1 \times D_2 \to \{0,1\}$
$R_3 : D_1 \times D_3 \to \{0,1\}$
D_1
D_2
D_3
D_1
Relation R_1
Relation R_2
Relation R_3
Relation Partition
R_1 ○ R_2
R_3
Relation Partition
R_1 ○ ○ R_3
R_2
$c = 1$
$c = 2$
D_1
D_3
R_1
R_2
R_3
w^1_1 w^1_2 w^3_1 w^3_2 w^3_3 $\theta_1(1,1)$ $\theta_1(1,2)$ $\theta_1(2,1)$ $\theta_1(2,2)$ $\theta_2(1,1)$ $\theta_2(1,2)$ $\theta_2(1,3)$ $\theta_2(2,1)$ $\theta_2(2,2)$ $\theta_2(2,3)$
w^1_1 w^1_2 w^3_1 w^3_2 w^3_3 w^3_4 $\theta_3(1,1)$ $\theta_3(1,2)$ $\theta_3(1,3)$ $\theta_3(1,4)$ $\theta_3(2,1)$ $\theta_3(2,2)$ $\theta_3(2,3)$ $\theta_3(2,4)$

(a) Three binary relations defined on three domains.

(b) Posterior sample of IRM structure assuming full dependence (55 parameters).

(c) Posterior sample of HIRM structure with dependence discovery ($(R_1, R_2) \perp R_3$) (39 parameters).

Figure 6.3: Learning independence relationships delivers more concise probabilistic model structures. (b) The IRM forces all relations to be dependent and learns, for each domain, a single clustering (red lines) shared by all the relations. This inductive bias leads to many spurious clusters. (c) The HIRM infers that R_3 is independent of (R_1, R_2) and learns two separate clusterings of D_1. While both (b) and (c) are in the HIRM hypothesis space, the structure in (c) is $\approx 7 \times 10^{13}$ times more likely under the posterior than (c), since the prior over structure penalizes excessively complex hypotheses.

Restrictions When Clustering Multiple Relations Kemp et al. [2006] applied the IRM to clustering multiple *relations* (i.e., interactions) by treating the relations themselves as entities in a domain. More specifically, consider a system with relations $R_1, \dots, R_m$ all defined on same domain and codomain, say D and C. This system can be encoded using a single higher-order relation $R' : D' \times D \to C$, where the entities of D' are relations over D, i.e., $R'(j, i) ::= R_j(i)$ (for $1 \le j \le m$, $i \in D$). While an IRM for R' will learn a clustering of both D' (the relations) and D (the original domain), there are at least two restrictions with this modeling approach: (i) it only applies to relations defined on the same domain and codomain; and (ii) it clusters relations R_i and R_j together only if they are dependent and identically distributed, which follows from Eq. (6.8). Figure 6.2 shows an illustrative example of this limitation.

6.3 Hierarchical Infinite Relational Model

The hierarchical infinite relational model (HIRM) is designed to address the aforesaid limitations of the IRM by using a structure learning prior to infer probable independencies among relations. This factorization cannot be represented by the model structure of a standard IRM. Given a system S with domains $D_1, \dots, D_n$ and relations $R_1, \dots, R_m$, the HIRM first nonparametrically partitions the m *relations* using a CRP (6.5), where the cluster assignments of the relations are denoted $c ::= (c_1, \dots, c_m)$. This partition induces a random number $K ::= \max(c_1, \dots, c_m)$ of subsystems $S_1, \dots, S_K$ of S. For each $\ell = 1, \dots, K$, the relations $\{R_i \mid 1 \le i \le m, c_i = \ell\}$ assigned to subsystem S_ℓ are modeled jointly by an IRM (6.6)–(6.8), independently of all relations assigned to another subsystem $S_{\ell'}$ ($\ell' \ne \ell$). The HIRM thus defines a distribution over relation clusters, entity clusters, and relation values as follows:

$$P(c_1, \dots, c_m, \{z^{\ell 1}, \dots, z^{\ell n}\}_{\ell=1}^{K}, R_1, \dots, R_m) = P(c) \prod_{\ell=1}^{K} \prod_{i=1}^{n} P(z^{\ell i}) \prod_{k \mid c_k = \ell} P(R_k \mid z^{\ell 1}, \dots, z^{\ell n}). \tag{6.9}$$

The generative specification of the HIRM is

$$\begin{aligned}
\{c_1, \dots, c_m\} &\sim \mathrm{CRP}(\alpha_0) && && (6.10)\\
\{z_1^{\ell i}, z_2^{\ell i}, \dots\} &\sim \mathrm{CRP}(\alpha_{\ell i}) && (\ell = 1, \dots, \max(c_1, \dots, c_m); i = 1, \dots, n) && (6.11)\\
\theta_k(j_1, \dots, j_{t_k}) &\sim \pi_k(\lambda_k) && (k = 1, \dots, m;\ j_1, \dots, j_{t_k} \in \mathbb{N}) && (6.12)\\
R_k(i_1, \dots, i_{t_k}) &\sim L_k(\theta_k(z_{i_1}^{c_k, d_{k1}}, \dots, z_{i_{t_k}}^{c_k, d_{kt_k}})) && (k = 1, \dots, m;\ i_1, \dots, i_{t_k} \in \mathbb{N}), && (6.13)
\end{aligned}$$

where α_0, $\{\{\alpha_{\ell i}\}_{i=1}^{n}\}_{\ell=1}^{K}$, and $\{\lambda_k\}_{k=1}^{m}$ are hyperparameters.

The HIRM (6.10)–(6.13) generalizes and extends the IRM (6.6)–(6.8). When $\alpha_0 = 0$, the HIRM recovers the IRM. For $\alpha_0 > 0$, Eq. (6.10) specifies a CRP partition prior to factorize the relations, where relations in the same block are modeled jointly using a standard IRM. In Eq. (6.11), each domain D_i is associated with a different partition $z^{\ell i}$ for each subsystem S_ℓ in which it participates. This inductive bias allows the HIRM to express structural independencies between relations and avoid modeling a Cartesian product of domain partitions when the data for (a subset of) relations in the system are not well aligned (Figure 6.3). Eq. (6.10) allows the HIRM to directly cluster dependent relations together, without using higher-order encodings discussed that are limited to relations defined on the same domain as in the IRM (Figure 6.2). Eqs. (6.12) and (6.13) also imply that any relations R_k and $R_{k'}$ clustered together in a subsystem S_ℓ need not be identically distributed (resp. Figure 6.2), as they each have their own parameters θ_k and $\theta_{k'}$, respectively. The dependence is instead modeled by the shared domain partitions $\{z^{\ell 1}, \dots, z^{\ell n}\}$ within subsystem S_ℓ. In total, the structure learning prior (6.10) retains the benefits of the standard IRM while addressing the limitations discussed in Section 6.2, all within a Bayesian nonparametric model discovery framework.

Algorithm 6.1 Scan of Gibbs sampling for HIRM (sketch).

```
Require: Markov chain state $\mathcal{S}$ containing relation cluster assignments $(c_1, \dots, c_m)$, entity cluster
    assignments $\{z^{\ell i}_1, \dots, z^{\ell i}_{N_i}\}$, and parameters $\{\theta_k(j_1, \dots, j_{t_k})\}$, for $1 \le i \le m$, $1 \le \ell \le K$, and
    $1 \le k \le m$; dataset $r$.
 1: for $k = 1, \dots, m$ do                                              ▷ For each relation $R_k$;
 2:     resample $c_k$ given $(r, \mathcal{S} \setminus \{c_k\})$                       ▷ resample partition assignment.
 3: for $\ell = 1, \dots, \max(c_1, \dots, c_m)$ do                                ▷ For each subsystem $S_\ell$;
 4:     $I_\ell \leftarrow \{d_{kj} \mid 1 \le k \le m, 1 \le j \le t_k, c_k = \ell\}$       ▷ $I_\ell$ is the set of domains in subsystem $S_\ell$;
 5:     for $i \in I_\ell$ do                                             ▷ for each domain $D_i$ in $S_\ell$;
 6:         for $j = 1, \dots, N_i$ do                                     ▷ for each entity $e^i_j$ in domain $D_i$;
 7:             resample $z^{\ell i}_j$ given $(r, \mathcal{S} \setminus \{z^{\ell,i}_j\})$               ▷ resample cluster assignment.
 8:     $T_\ell \leftarrow \{k \mid 1 \le k \le m, c_k = \ell, (\pi_k, L_k)$ nonconjugate$\}$   ▷ $T_\ell$ is the set of relations in subsystem $S_\ell$
 9:     for $k \in T_\ell$ do                                             ▷ For each relation $R_k$ in $S_\ell$;
10:         for $j_1 = 1, \dots, \max(z^{\ell d_{k1}}_1, \dots, z^{\ell d_{k1}}_{N_{d_{k1}}})$ do     ▷ for each cluster in the first domain of $R_k$;
11:             ...                                                    ▷ ...
12:             for $j_{t_k} = 1, \dots, \max(z^{\ell d_{kt_k}}_1, \dots, z^{\ell d_{kt_k}}_{N_{d_{kt_k}}})$ do   ▷ for each cluster in the last domain of $R_k$;
13:                 resample $\theta_k(j_1, \dots, j_{t_k})$ given $(r, \mathcal{S})$          ▷ resample cluster parameter.
```

6.4 Algorithms for Posterior Inference

An observed dataset for a relational system (Definition 6.1) consists of a finite number of realizations of relation values, i.e., observations of random variables of the form $\{R_k(i_1, \dots, i_{t_k}) = r_k(i_1, \dots, i_{t_k})\}$, written $\{r_1, \dots, r_m\}$ for short. Without loss of generality, it is assumed in this section that the relation values are fully observed for $N_i \ge 1$ entities $\{e^i_1, \dots, e^i_{N_i}\}$ of each domain D_i $(i = 1, \dots, n)$ across all relations that it participates in. Thus, the number of observations of $R_k : D_{d_{k1}} \times \cdots \times D_{d_{kt_k}} \to C_k$ is precisely $N_{d_{k1}} \times \cdots \times N_{d_{kt_k}}$, for $k = 1, \dots m$. The reference implementation of the HIRM (Section 1.4) relaxes this requirement and can handle arbitrary index combinations with missing data.

Posterior inference in the HIRM is performed by simulating an ergodic Markov chain that converges to the conditional distribution of Eq. (6.9) given the observed dataset. The state $\mathcal{S}$ of the chain contains:

(S1) cluster assignments $\{c_1, \dots, c_m\}$ of the relations, which define a partition of $\{R_1, \dots, R_m\}$ into $K ::= \max\{c_1, \dots, c_m\}$ subsystems $(S_1, \dots, S_M)$;

(S2) cluster assignments $\{z^{\ell i}_1, \dots, z^{\ell i}_{N_i}\}$ of the items in domain D_i $(1 \le i \le n)$ within subsystem S_ℓ $(1 \le \ell \le M)$, which define a partition of $\{e^i_1, \dots e^i_{N_i}\}$ into $W_{\ell i} ::= \max\{z^{\ell i}_1, \dots, z^{\ell i}_{N_i}\}$ clusters;

(S3) cluster parameters $\theta_k(j_1, \dots, j_{t_k})$ for relation indexes $k = 1, \dots, m$ and cluster indexes $j_s = 1, \dots, \max(z^{\ell d_{ks}}_1, \dots, z^{\ell d_{ks}}_{N_{d_{ks}}})$ $(1 \le s \le t_k)$, whenever (π_k, L_k) do not form a conjugate pair.

An initial state $\mathcal{S}$ can be obtained by sampling from the prior (6.10)–(6.12) and then iterating using Gibbs sampling. Algorithm 6.1 shows one full Gibbs scan through all the variables in a given state $\mathcal{S}$. The transition operators for the updates in lines 2, 7 and 13 of Algorithm 6.1 are described next. It is straightforward to embed these Gibbs kernels within a resample-move sequential Monte Carlo algorithm that incorporates batches of observations one at a time rather than all the data at once.

Resampling Relation Cluster Assignments c_k This kernel uses the auxiliary Gibbs sampler [Neal, 2000, Algorithm 8]. Let $C_\ell ::= |\{k \mid 1 \le k \le K, c_k = \ell\}|$ be the number of relations in S_ℓ and $W_{\ell i} ::= \max\{z^{\ell i}_1, \dots, z^{\ell i}_{N_i}\}$ be the number of clusters for domain D_i within S_ℓ $(1 \le \ell \le K)$.

Case 1. If c_k is a singleton ($C_k = 1$), then it is resampled from the categorical distribution

$$P\left(c_k^{\text{new}} = \ell \mid \mathcal{S} \setminus \{c_k\}\right) = t_{k\ell} \prod_{j_1=1}^{W_{\ell d_{k1}}} \cdots \prod_{j_{t_k}=1}^{W_{\ell d_{kt_k}}} w_{k\ell}(\mathbf{j}, \theta_k) \qquad (\ell = 1, \ldots, K), \tag{6.14}$$

where $\mathbf{j} ::= (j_1, \ldots, j_{t_k})$ and

$$t_{k\ell} ::= \begin{cases} \alpha_0/(m-1+\alpha_0) & \text{if } \ell = c_k \\ C_\ell/(m-1+\alpha_0) & \text{otherwise,} \end{cases} \tag{6.15}$$

$$w_{k\ell}(\mathbf{j}, \theta_k) ::= \prod_{\mathbf{i} \in A_{k\ell}(\mathbf{j})} L_k(r_k(\mathbf{i}); \theta_k(\mathbf{j})). \tag{6.16}$$

Eq. (6.15) is the conditional probability from the CRP prior (6.5), and in Eq. (6.16)

$$A_{k\ell}(\mathbf{j}) ::= \{\mathbf{i} \mid z_{i_1}^{\ell d_{k1}} = j_1, \ldots, z_{i_{t_k}}^{\ell d_{kt_k}} = j_{t_k}\} \tag{6.17}$$

denotes the set of entity indexes $\mathbf{i} ::= (i_1, \ldots, i_{t_k})$ for domains $(d_{k1}, \ldots, d_{kt_k})$ that are assigned to cluster $\mathbf{j}$ of subsystem S_ℓ (where $1 \le k \le m$; $1 \le \ell \le M$; $1 \le j_1 \le W_{\ell d_{k1}}$; $\ldots$; $1 \le j_{t_k} \le W_{\ell d_{kt_k}}$). If (π_k, L_k) is a conjugate pair, the parameters θ_k can be analytically integrated out, and Eq. (6.16) becomes

$$w_{k\ell}(\mathbf{j}) ::= \int_\theta \Big[\prod_{\mathbf{i} \in A_{k\ell}(\mathbf{j})} L_k(r_k(\mathbf{i}); \theta) \Big] \pi_k(\theta; \lambda_k) d\theta. \tag{6.18}$$

Case 2. If c_k is not a singleton ($C_k > 1$), then

1. For domain indexes $i = 1, \ldots, n$, draw cluster assignments for a fresh entity partition:

$$\{z_1^{K+1,i}, \ldots, z_{N_i}^{K+1,i}\} \sim \text{CRP}(\alpha), \tag{6.19}$$

$$W_{K+1,i} ::= \max\{z_1^{K+1,i}, \ldots z_{N_i}^{K+1,s}\}. \tag{6.20}$$

2. Draw parameters $\theta_k(j_1, \ldots, j_{t_k})$ for relation indexes $k = 1, \ldots, m$ and cluster indexes $j_1 = 1, \ldots, W_{K+1,d_{k1}}$; $\ldots$; $j_{t_k} = 1, \ldots, W_{K+1,d_{kt_k}}$.

3. Resample c_k to take a new value $\ell \in \{1, \ldots, K+1\}$ using the same terms in Eqs. (6.14)–(6.16) from the previous case, except the CRP weight $t_{k\ell}$ in Eq. (6.15) is

$$t_{k\ell} ::= \begin{cases} (C_\ell - 1)/(m-1+\alpha_0) & \text{if } \ell = c_k \\ C_\ell/(m-1+\alpha_0) & \text{if } \ell \neq c_k, \ell \le K \\ \alpha_0/(m-1+\alpha_0) & \text{if } \ell = K+1. \end{cases} \tag{6.21}$$

Resampling Entity Cluster Assignments $z_j^{\ell i}$ Within each subsystem S_ℓ, the entity cluster assignments are transitioned using the collapsed Gibbs sampler [Neal, 2000, Alg. 3]. Alternatively, the split-merge algorithm [Jain and Neal, 2004] can be used. Xu et al. [2007] discuss additional sampling-based and variational approaches for inferring these variables.

Resampling Cluster Parameters $\theta_k(j_1, \ldots, j_{t_k})$ Sample $\theta'_k(\mathbf{j}) \sim q_k(\theta_k(\mathbf{j}))$ from a proposal distribution, such as the prior $\pi_k(\lambda_k)$ or Gaussian drift $\mathcal{N}(\theta_k(\mathbf{j}), \sigma_k)$, and accept the move according to the

Table 6.1: Comparison of average log-likelihood of held-out test data on a benchmark of 20 binary object-attribute datasets using the HIRM and two Bayesian nonparametric baselines.

	Dataset Statistics			Average Test Log-Likelihood		
	N_{cols}	$N^{\text{train}}_{\text{rows}}$	$N^{\text{test}}_{\text{rows}}$	HIRM	IRM	DPMM
NLTCS	16	18338	3236	-06.00	-06.01 ●	-06.01 ●
MSNBC	17	330212	58265	-06.19	-06.27 ●	-06.22 ●
KDDCup 2000	64	199999	34955	-02.13	-02.13 ●	-02.13 ●
Plants	69	19733	3482	-13.75	-14.23 ●	-13.81
Audio	100	17000	3000	-39.99	-40.34	-40.02
Jester	100	10000	4116	-52.91	-52.96	-52.92
Netflix	100	3500	3000	-56.96	-57.48 ●	-56.96
Accidents	111	14458	2551	-33.85	-39.43 ●	-38.93 ●
Retail	135	24979	4408	-10.90	-10.99	-10.92
Pumsb-star	163	13897	2452	-32.77	-38.95 ●	-38.02 ●
DNA	180	2000	1186	-87.65	-97.44 ●	-97.62 ●
Kosarek	190	37825	6675	-10.91	-10.99	-10.95
MSWeb	294	62191	5000	-10.23	-11.20 ●	-10.26
Book	500	9859	1739	-34.43	-34.52	-34.76
EachMovie	500	5526	591	-52.23	-52.09	-54.86
WebKB	839	3361	838	-156.67	-157.27	-158.26
Reuters-52	889	7560	1540	-90.22	-90.06	-89.34
20 Newsgroup	910	15057	3764	-153.52	-156.46 ●	-153.95
BBC	1058	1895	330	-253.36	-253.86	-254.59
Ad	1556	2788	491	-45.19	-46.17	-52.40 ●

● indicates significantly worse than HIRM ($p = 0.05$ Mann-Whitney U test).

Table 6.2: Summary of the number wins, ties, and losses of HIRM on benchmarks from Table 6.1, compared to two Bayesian nonparametric baselines and two probabilistic deep learning baselines.

	IRM	DPMM	LearnSPN	RAT-SPN
HIRM # better	11	7	6	4
HIRM # tie	9	13	8	13
HIRM # worse	0	0	6	3

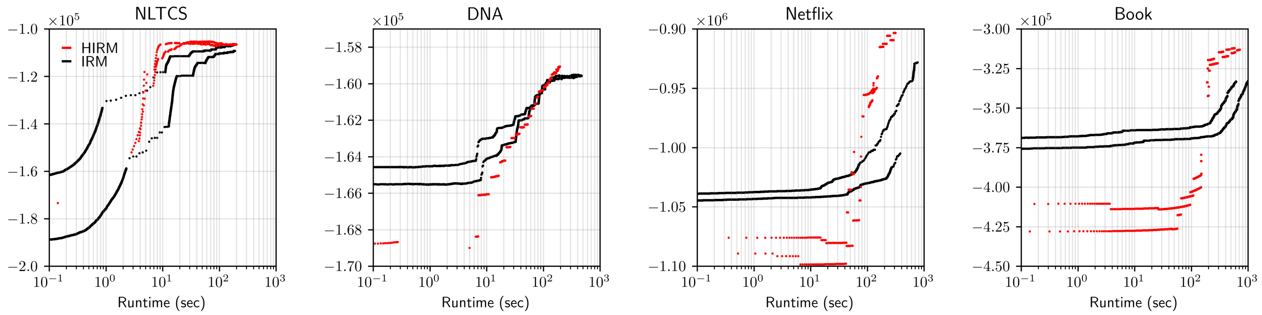


Figure 6.4: Runtime versus log joint probability of latent variables and observed data using the HIRM (red) and IRM (black) in four representative object-attribute benchmarks from Table 6.1. Each subplot shows measurements for two independent runs of posterior inference in each method.

MH probability

$$\min\left(1, \frac{\pi_k(\theta'_k(\mathbf{j}); \lambda_k) w_{k\ell}(\mathbf{j}, \theta'_k)}{\pi_k(\theta_k(\mathbf{j}); \lambda_k) w_{k\ell}(\mathbf{j}, \theta_k)} \cdot \frac{q_k(\theta_k(\mathbf{j}); \theta'_k(\mathbf{j}))}{q_k(\theta'_k(\mathbf{j}); \theta_k(\mathbf{j}))}\right), \tag{6.22}$$

where is $w_{k\ell}$ (Eq. (6.16)) is the data likelihood for cluster $\mathbf{j} = (j_1, \dots, j_{t_k})$.

Resampling Hyperparameters $(\alpha_0, \{\alpha_{\ell i}\}_{i=1}^{n}, \{\lambda_k\}_{k=1}^{m})$ Broad exponential hyperpriors are used for all the hyperparameters that appear in Eqs. (6.10)–(6.12) and are resampled using an empirical Bayes approach similar to Eqs. (5.32)–(5.37) from Chapter 5.

6.5 Applications to Object-Attribute, Political, and Genomics Data

The HIRM is evaluated on three problems. Section 6.5.1 benchmarks the predictive accuracy on object-attribute (cross-sectional) data tables against several baselines. Sections 6.5.2 and 6.5.3 applies the HIRM to make discoveries in the relational systems for politics and genomics shown in Figure 6.1, which discovers interpretable relationships between the entities in these challenging domains.

6.5.1 Predictive Accuracy on Object-Attribute Benchmarks

This evaluation assesses the predictive performance of the HIRM on a benchmark of 20 object-attribute datasets from Gens and Domingos [2013] and compares the results to various baselines. In Table 6.1, the first four columns summarize the dataset statistics, which range between 16–1556 columns and 2000–330212 rows. The last three columns show the test log-likelihood from the HIRM, IRM [Kemp et al., 2006, Xu et al., 2006], and Dirichlet process mixture model [DPMM; Lo, 1984]. As in Kemp et al. [2006], the IRM encodes object-attribute data using one binary relation $R : \text{Attr} \times \text{Obj} \to \{0, 1\}$. The HIRM encodes each dataset using N_{cols} unary relations $\{R_i : \text{Obj} \to \{0, 1\} \mid i \in \text{Attr}\}$ with structure learning over the dependence between these attributes. The DPMM uses the same encoding as the HIRM but without structure learning, so all attributes are modeled as dependent. In Table 6.1, the black dots indicate significantly worse accuracy than the HIRM. The HIRM consistently outperforms these baselines—it is significantly better in 17 cases and worse in zero cases. Figure 6.4 shows a plot of inference runtime versus held-in data log score using the HIRM and IRM on four of the benchmarks from Table 6.1. Despite the structure learning prior, the runtime of the HIRM matches or outperforms the IRM. In fact, the HIRM often infers simpler partitions within the independent subsystems, which can improve not only runtime scaling but also model accuracy.

To further assess the density estimation results, Table 6.2 compares the HIRM test log-likelihood to those obtained from two probabilistic deep learning baselines for object-attribute data: LearnSPN [Gens and Domingos, 2013] and RAT-SPN [Peharz et al., 2019]. A win, loss, and tie means statistically significant better, worse, or equal test log likelihoods as compared to the HIRM. The results show that the HIRM, which is a relatively shallow Bayesian model, is competitive on object-attribute data with higher capacity probabilistic deep learning baselines that fit the data using greedy optimization. The HIRM is distinguished by being additionally applicable to more general relational systems, presented in the next two subsections, which LearnSPN and RAT-SPN cannot handle.

6.5.2 Political Interactions

Figure 6.1a shows the schema for political data from the "Dimensionality of Nations" project [Rummel, 1999] and Figure 6.5 shows the observations for a subset of 56 interactions and 15 countries, using the version of the dataset from Kemp et al. [2006]. In each graph, a directed edge from one country to another indicates an observed interaction; the absence of an edge means either there is no interaction

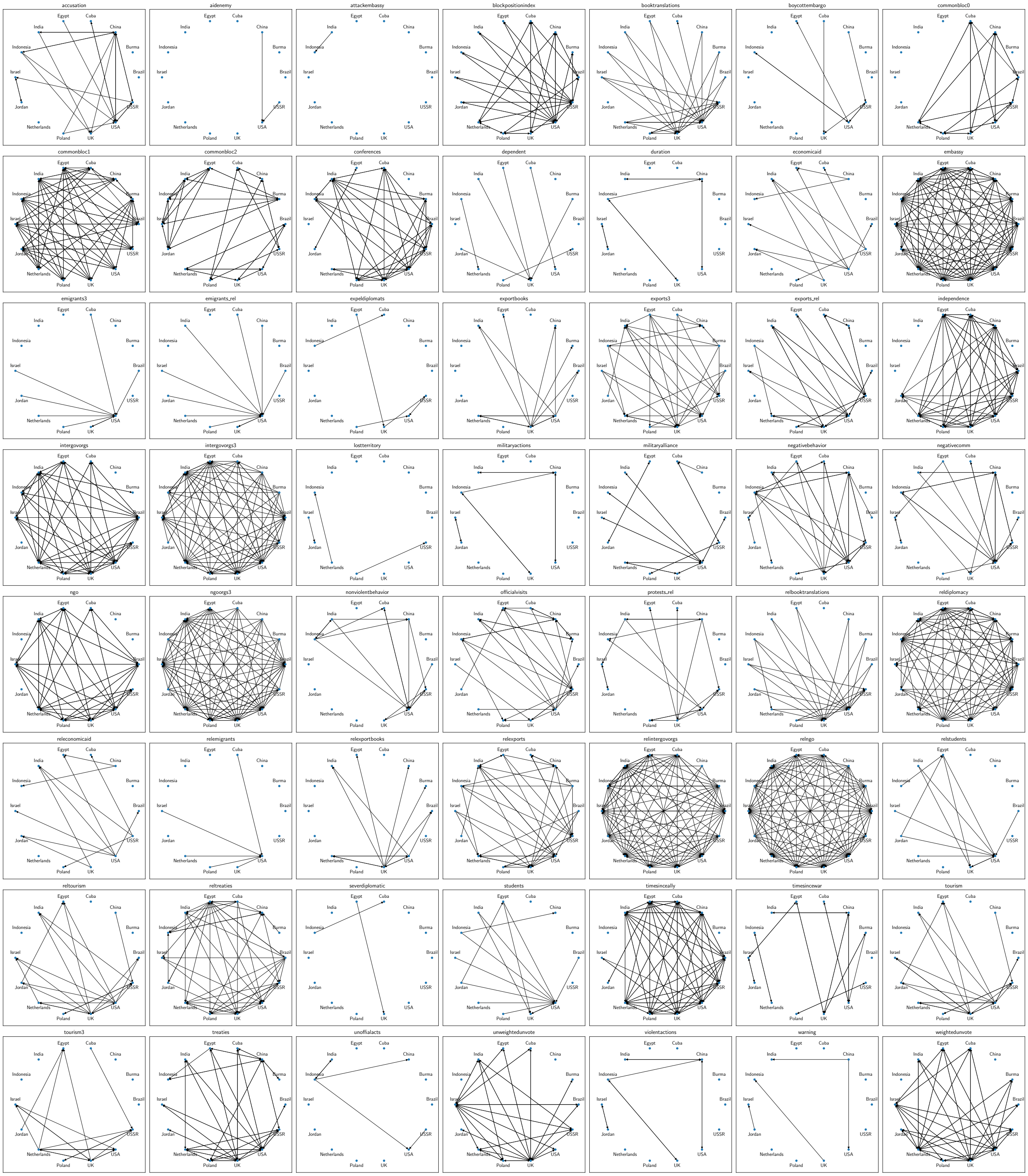
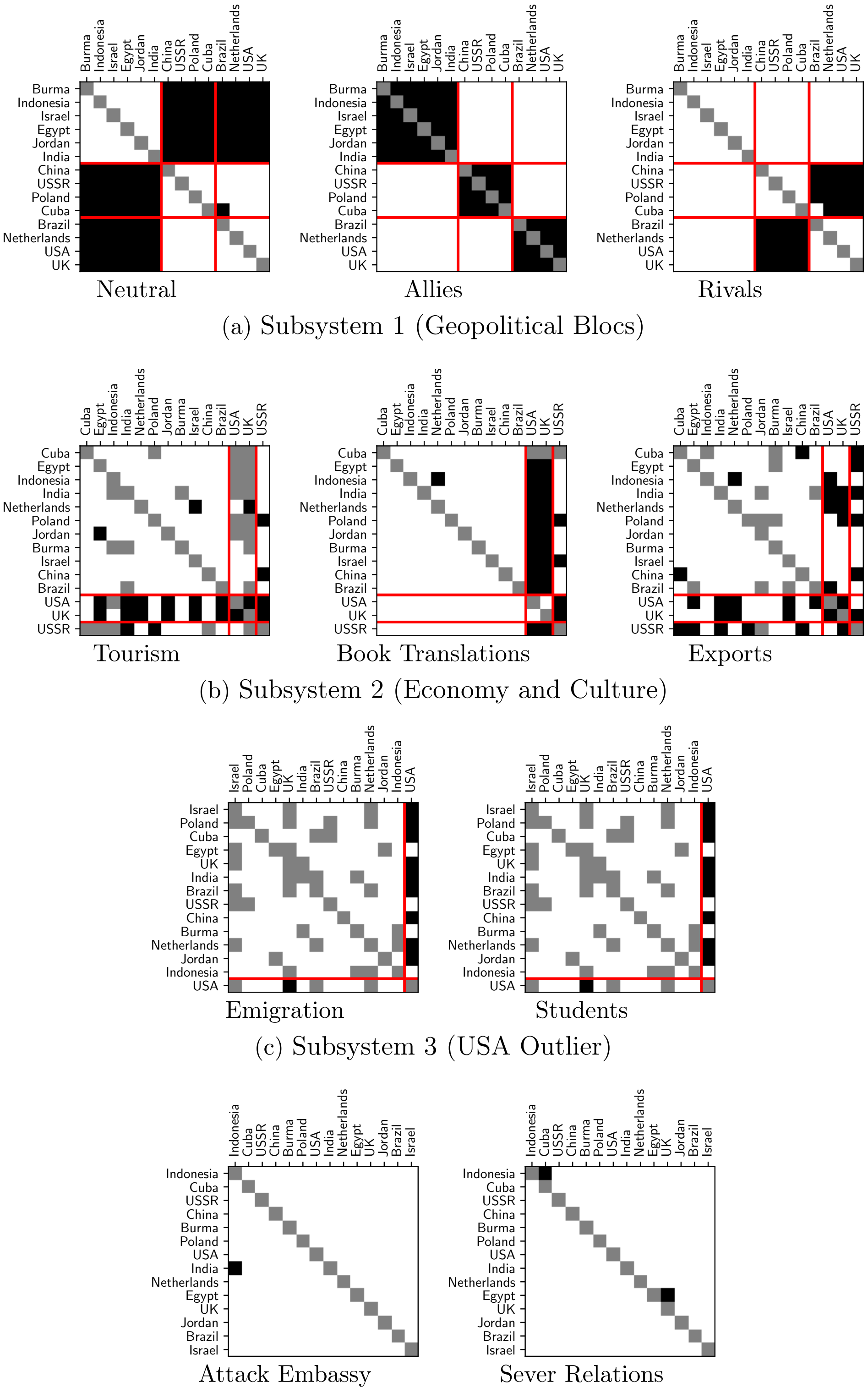


Figure 6.5: Graphs for 56 interactions between countries in the “Dimensionality of Nations” data.



Neutral Allies Rivals

(a) Subsystem 1 (Geopolitical Blocs)

Tourism Book Translations Exports

(b) Subsystem 2 (Economy and Culture)

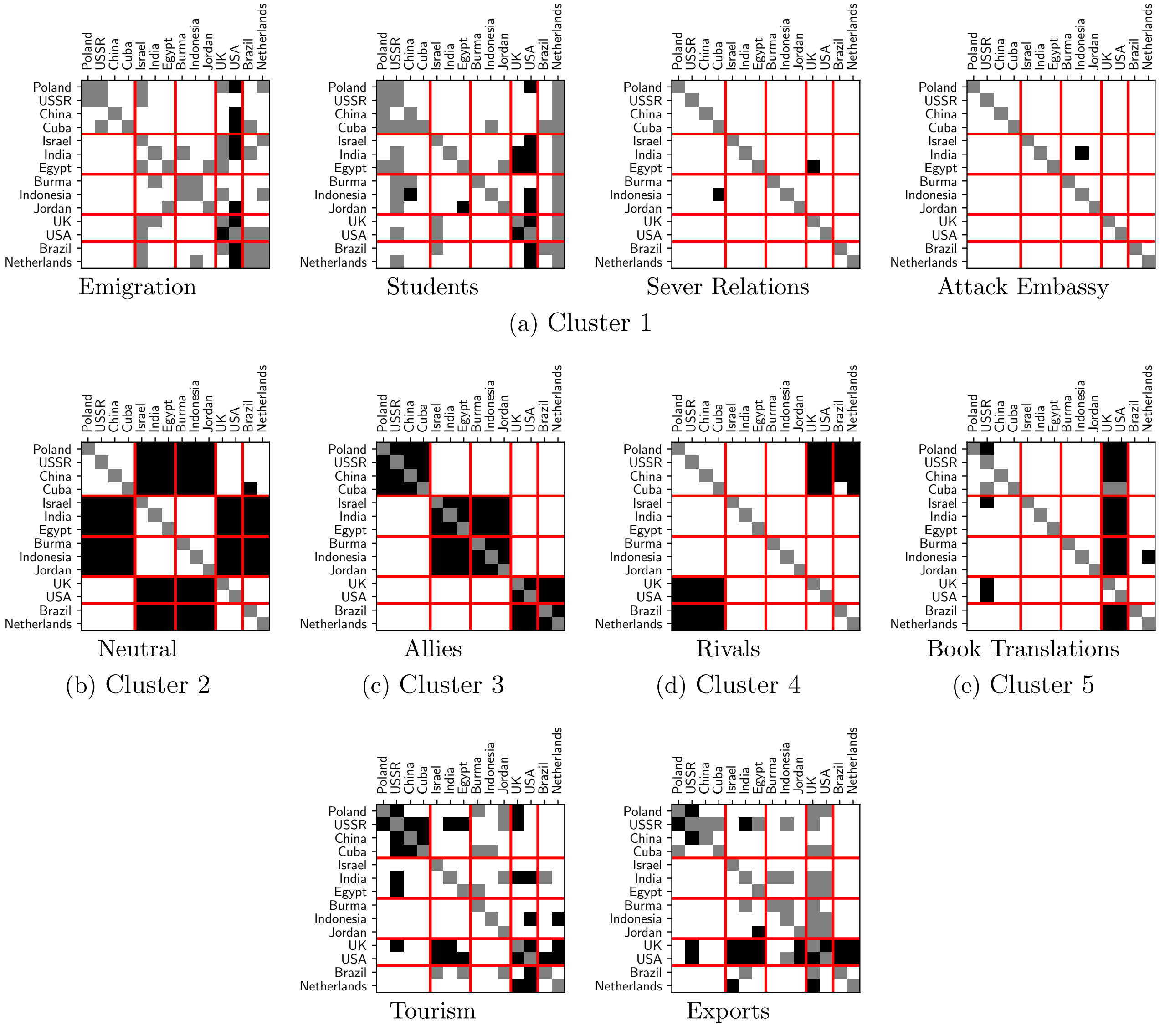
Emigration Students

(c) Subsystem 3 (USA Outlier)

Attack Embassy Sever Relations

(d) Subsystem 4 (Sparse Interactions)

Figure 6.6: Subsystems inferred by the HIRM on the "Dimensionality of Nations" data. Each panel shows an independent system of interactions that has its own partition of the nations. Interactions are represented as binary adjacency matrices of the graphs in Figure 6.5.



(a) Cluster 1

(b) Cluster 2

(c) Cluster 3

(d) Cluster 4

(e) Cluster 5

(f) Cluster 6

Figure 6.7: Clusters of interactions inferred by the IRM on the “Dimensionality of Nations” data. Since the IRM uses a shared partition of the nations across all relations (red lines), the concepts discovered are more complicated and less interpretable than those of the HIRM in Figure 6.6.

or the data is missing. There are also 111 country attributes (not shown) encoded as unary relations on the Country domain.

Figure 6.6 shows four independent subsystems of relations discovered by the HIRM. Each relation is shown as an adjacency matrix (white means 0, black means 1, and gray means missing data) of the corresponding graph in Figure 6.5. Red lines show the country partition inferred for each subsystem, which capture varying properties of the data. For example, in Figure 6.6a, the HIRM infers a subsystem representing three geopolitical interactions for whether countries are allies, rivals, or have neutral relations. The attributes assigned to this subsystem include “electoral system”, “political leadership”, and “constitutional”.[1] In Figure 6.6b, which represents economic and cultural ties, the learned attributes include “absolute income”, “agricultural population”, and “arts and culture NGOs”. This subsystem reflects the fact that tourists from the UK and USA travel to countries from all clusters; all countries translate books from the USA and UK; and UK and USA translate books from the USSR. Figure 6.6c represents a subsystem of relations in which the USA is unique, due to its exceptionally high number of immigrants and foreign students. The HIRM has inferred that immigration and foreign students are independent of the geopolitical interactions in Figure 6.6a. This discovery corresponds to commonsense: the geopolitical rivalry between the USA and China/USSR does not impact the flow of immigrants and students between the latter to the former. Finally, Figure 6.6d contains sparse relations such as “Attack Embassy” and “Sever Relations”, which form a subsystem with a single country cluster with a small probability for the presence of a hostile event.

In contrast to the HIRM, the IRM cannot detect subsystem structure because it uses a single country partition for all interactions, as shown in Figure 6.7. These clusters provide a less interpretable explanation of the data. For example, Figure 6.7a shows that the IRM explains the four interactions of emigration, foreign students, sever relations, and embassy attacks using a single latent process, which is less plausible than the HIRM explanation in Figures 6.6c and 6.6d. Moreover, the IRM cannot detect dependence between the geopolitical interactions in Figure 6.6a and instead assigns them to separate clusters (Figures 6.7b–6.7d) due to the limitation described in Figure 6.2.

6.5.3 Genomic Properties

The final experiment applies the HIRM to structure discovery in a dataset of yeast genomes [Cheng et al., 2002]. Figure 6.1b shows the relational schema. The Gene domain has 1,243 unique identifiers and there is a binary relation Interact : $\text{Gene} \times \text{Gene} \to \{0, 1\}$ between gene pairs. Moreover, there is a binary relation between Gene and each of the eight other domains, for example At : $\text{Gene} \times \text{Localization} \to \{0, 1\}$ specifies whether a given gene is at a specific location (cytoplasm, golgi, etc.). Each gene is associated with exactly one entity in the Essential and Chromosome domains, and possibly multiple entities in the Complex, Phenotype, Class, Motif, and Function. Tables 6.3 and 6.4 show example records for genes G235131 and G234936. Gene G235131 has a missing Class; it is associated with two entities in the Complex domain; two entities in the Function domain; five entities in the Phenotype domain; and interacts with 11 other genes (three of which are listed).

Figure 6.8a shows two heatmaps that summarize the clusters of genes learned by the HIRM under two different contexts. Each row and column in these heatmaps represents a unique Gene and the color of a cell is the posterior probability (between 0 and 1) that the two genes are assigned to the same latent cluster, estimated by an ensemble of 100 posterior HIRM samples. The top (resp. bottom) heatmap in Figure 6.8a shows posterior co-clustering probabilities within the subsystem that contains the “Gene At Localization” (resp. “Gene Belong Class”) relation. These heatmaps reflect the fact that the HIRM discovers context-specific clusters that are different across the learned subsystems.

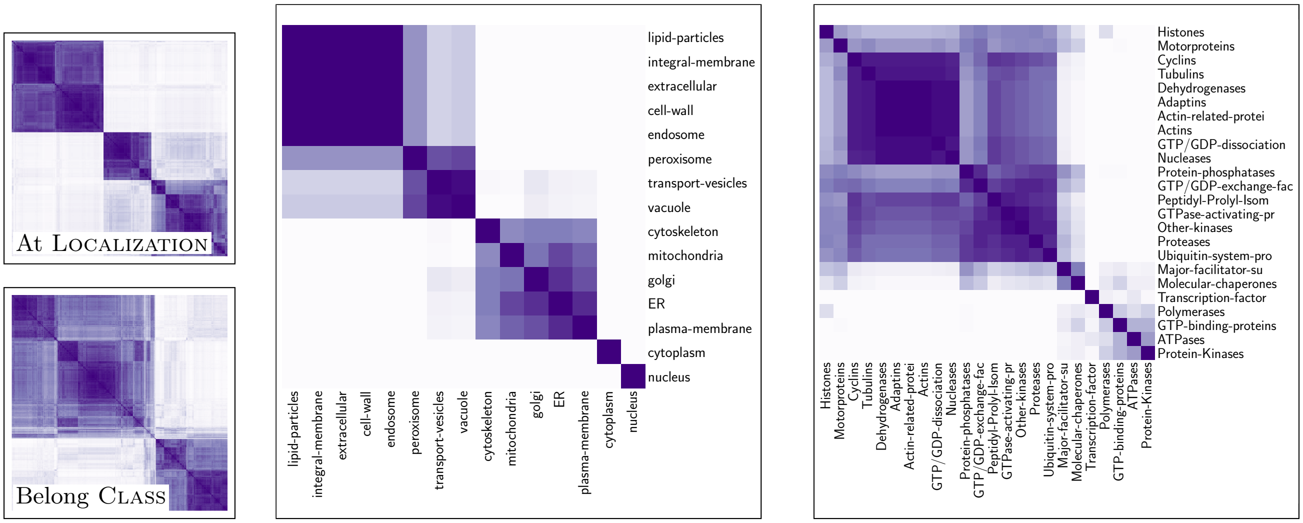
[1] In Figure 6.6a, the Cuba–Brazil relationship is neutral despite the countries belonging to rival geopolitical blocs. This outlier is explained by the so-called the American–Brazilian–Cuban “triangular diplomacy” during the 1962 missile crisis [Hershberg, 2004].

Table 6.3: Data for gene G235131

Field	**Value**
Gene	G235131
Essential	Non-Essential
Class	?
Complex	Histone Acetyltransferase
—	Transcriptosome
Phenotype	Auxotrophies
—	Carbohydrate & Lipid Biosynth.
—	Conditional Phenotypes
—	Mating & Sporulation Defects
—	Nucleic Acid Metab. Defects
Motif	PS00633
Chromosome	2
Function	Transcription
—	Cellular Organization
Localization	Nucleus
Interactions	G234980, G235780, G235278, . . .

Table 6.4: Data for gene G234936

Field	**Value**
Gene	G234936
Essential	Ambiguous
Class	Protein-Kinases
Complex	Casein Kinase
Phenotype	Cell Morphology Organelle Mutants
—	Conditional Phenotypes
—	Nucleic Acid Metab. Defects
Motif	PS00108
—	PS00107
Chromosome	15
Function	Transcription
—	Cellular Organization
—	Cellular Growth
Localization	Nucleus
Interactions	G238510, G235309, G234122, . . .



(a) Gene Clusters

(b) Localization Clusters

(c) Class Clusters

Figure 6.8: Posterior co-clustering probabilities for various relational domains in yeast genome data.

Table 6.5: Posterior co-clustering probabilities for gene G235131 (Table 6.3) and five other genes within two of the subsystems (Figure 6.8a) inferred by the HIRM.

		Co-clustering probability within subsystem for		
Gene 1	Gene 2	Localization	Class	Pattern
G235131	G235278	0.98	0.87	Low–Low
G235131	G239017	0.52	0.47	Med–Med
G235131	G236063	0.03	0.13	Low–Low
G235131	G235388	0.83	0.27	High–Low
G235131	G240065	0.03	0.68	Low–High

Table 6.5 lists various co-clustering probabilities between G235131 and five other genes, which show that a pair of genes that are similar in the LOCALIZATION context need not be similar in the CLASS context. Further, even though G235131 belongs to an unknown CLASS, the HIRM can compute its co-clustering probabilities within this context by using observations of other properties that are inferred to be predictive of the missing value.

Figure 6.8b shows posterior co-clustering probabilities for entities in domains that describe gene properties. The HIRM infers a likely cluster of LOCALIZATION entities that includes cell wall, extracellular, integral membrane, and lipid particles, whereas cytoplasm and nucleus are inferred as singletons. Figure 6.8c shows co-clustering probabilities for CLASS, which reflect a probable cluster (cyclins, tublins, adaptins, . . .) embedded within a larger more noisy cluster, as well as singletons such as transcription factor and polymerases. These heatmaps show quantitative estimates of posterior uncertainty in the partition structures detected by the HIRM, which cannot be captured using inference approaches such as maximum a posteriori estimation or variational inference, and highlight an advantage of using fully Bayesian sampling for probabilistic structure learning in complex domains.

6.6 Related Work

Several variations of the standard IRM have been developed in the literature on nonparametric relational Bayesian models [Ishiguro et al., 2012, Ohama et al., 2013, Jonas and Kording, 2015, Briercliffe, 2016]. The hierarchical IRM is distinguished by being the first extension that uses a nonparametric structure learning prior over the relations themselves to improve modeling capacity and address shortcomings of the IRM identified in Section 6.2, which include combinatorial over-clustering and failing to detect relationships between dependent but non-identically distributed relations. These limitations have not been addressed by previous extensions of the IRM and, as shown in Sections 6.5.2 and 6.5.3, enable new types of discoveries to be made from data. An advantage of the hierarchical IRM is that it can be easily composed with many variants of the IRM that are designed to address other shortcomings, such as (i) the subset IRM [Ishiguro et al., 2012], which detects and filters out irrelevant observations in the presence of extreme sparsity; and (ii) the logistic regression IRM [Jonas and Kording, 2015], which improves predictive accuracy for semi-supervised tasks that specify one or more target variables as well as exogenous (non-probabilistic) covariates.

Other approaches to relational modeling include relational extensions of Bayesian networks [Heckerman et al., 2004, Koller and Pfeffer, 1997, Friedman et al., 1999, Getoor et al., 2007] and Markov random fields [Muggleton and de Raedt, 1994, Taskar et al., 2002, Richardson and Domingos, 2006]. While these approaches may be more expressive than purely nonparametric models, they inherit traditional challenges of structure learning and model selection. In directed models [Daly et al., 2011], for example, there is a super-exponential number of graphs to consider [Robinson, 1977]. In undirected models, structure learning requires expensive tuning of evaluation measures, clause construction operators, or search strategies [Kok and Domingos, 2005]. The HIRM instead builds on Bayesian nonparametric relational models that use latent variables to simplify the learning problem as compared to searching over arbitrary graphical structures. This approach makes it possible to use principled Bayesian structure learning algorithms that deliver uncertainty over model structure, latent variables, and predictions.

Deep generative models have also been developed for relational data [Kipf and Welling, 2016, Mehta et al., 2019, Fan et al., 2019, Qu et al., 2019]. These methods generally assume that there is one binary adjacency matrix being modeled (i.e., a random graph relation) or operate in a semi-supervised setting of predicting labels. In contrast, the HIRM discovers generative models for datasets with more complex relational schemas than a single binary matrix (e.g., Figure 6.1) and operates in a fully unsupervised setting. This approach allows the HIRM to model sparse and noisy systems with multiple entities, attributes, and interactions in Sections 6.5.2 and 6.5.3. Table 6.2 shows that, in the special case of object-attribute (cross-sectional) data, density estimates of the HIRM are competitive with modeling techniques from probabilistic deep learning that cannot handle more expressive relational structure.

Using the Chinese restaurant process as a structure learning prior (Eq. (6.10)), which is a recurring theme in this thesis that also appears in Chapters 4 and 5, has been considered by Salakhutdinov et al. [2013], Blei et al. [2010], Mansinghka et al. [2016] in other contexts. The same insight of using an outer CRP to partition relations in the IRM can also be applied to relational models such as the Mondrian process [Roy and Teh, 2008]. More broadly, it would be fruitful to investigate a representation theorem for the ergodic distributions of a relational system modeled by an HIRM within the framework of exchangeable random structures from Orbanz and Roy [2015].

In addition to generalizing the IRM, the HIRM generalizes several other Bayesian nonparametric models which include the MultiMixture DSL from Chapter 4, the infinite hidden relational model [Xu et al., 2006], the infinite mixture model [Rasmussen, 1999], the Dirichlet process mixture model [Lo, 1984], and the Cross-Categorization model [Mansinghka et al., 2016]. By generalizing the likelihood term (6.13) to include regression on relation values that are endogenous to the system, the HIRM could be further extended to express a relational variant of Dirichlet process mixtures of generalized linear models [Hannah et al., 2011].

Part II

Exact Bayesian Inference via Symbolic Program Analysis

Chapter 7

Sum-Product Probabilistic Language

> The most important thing in a programming language is the name. A language will not succeed without a good name. I have recently invented a very good name, and now I am looking for a suitable language.
>
> Donald Knuth

This chapter presents the Sum-Product Probabilistic Language (SPPL), a probabilistic programming language that automatically delivers exact solutions to a broad range of probabilistic inference queries. SPPL translates probabilistic programs into *sum-product expressions*, a new symbolic representation and associated semantic domain that extends standard sum-product networks to support mixed-type distributions, numeric transformations, logical formulas, and pointwise and set-valued constraints. SPPL is formalized via a novel translation strategy from probabilistic programs to sum-product expressions and give sound exact algorithms for conditioning on and computing probabilities of events. The language imposes certain syntactic restrictions on probabilistic programs to ensure they can be translated into sum-product expressions, which can be used to efficiently solve queries. The four probabilistic DSLs from Part I of this thesis can be translated into SPPL source syntax, which enables substantial reuse of the underlying inference machinery and eliminates the need to develop custom prediction engines for each DSL. Evaluations on benchmarks the system targets show that it obtains orders of magnitude speedup over state-of-the-art exact solvers across several challenging tasks.

Several probabilistic programming languages leverage approximate inference techniques [Gilks et al., 1994, Goodman et al., 2008a, Wingate and Weber, 2013] which have been used effectively in diverse settings [Sankaranarayanan et al., 2013, Carpenter et al., 2017, Cusumano-Towner et al., 2019]. Drawbacks of approximate inference, however, include a lack of accuracy and/or soundness guarantees [Dagum and Luby, 1993, Lew et al., 2020]; difficulties with programs that combine continuous, discrete, or mixed-type distributions [Carpenter et al., 2017, Wu et al., 2018]; challenges assessing the quality of iterative solvers [Brooks and Gelman, 1998]; and the substantial expertise needed to write custom inference programs that deliver acceptable performance [Mansinghka et al., 2018, Cusumano-Towner et al., 2019]. To address the shortcomings of approximate inference, other probabilistic programming languages leverage exact symbolic techniques [Bhat et al., 2013, Narayanan et al., 2016, Gehr et al., 2016, Carette and Shan, 2016, Zhang and Xue, 2019]. These languages use some form of computer algebra to solve inference queries. However, the generality of the symbolic representation causes them to fail frequently, even on problems with tractable solutions. SPPL occupies a new point in the expressiveness vs. performance trade-off space for exact symbolic inference. A key idea in SPPL is to incorporate certain modeling restrictions that avoid the need for general computer algebra, instead using a new, specialized

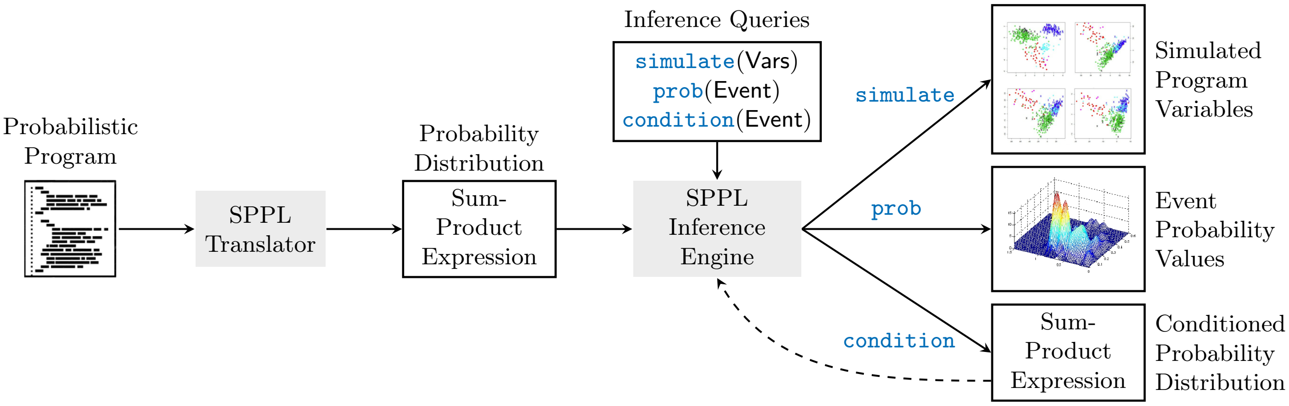


Figure 7.1: SPPL system architecture. Probabilistic programs, either machine-synthesized (Part I) or handwritten, are translated into symbolic sum-product expressions that can be used to deliver exact solutions to many probabilistic inference queries.

class of "sum-product" symbolic expressions to exactly represent probability distributions specified by SPPL programs. These new symbolic expressions extend and generalize sum-product networks [Poon and Domingos, 2011], which are computational graphs that have received widespread attention for their clear probabilistic semantics and tractable properties for exact inference [Darwiche, 2021]. The sum-product expressions used in SPPL are used to automatically obtain exact solutions to probabilistic inference queries about probabilistic programs, which are fast and scalable in tractable regimes.

System Overview Figure 7.1 shows an overview of SPPL. Given a probabilistic program expressed in the source syntax from Listing 7.1, a program translator produces a sum-product expression that represents the prior distribution over all program variables. Given this expression and a query specified by the user, the SPPL inference engine returns an exact answer, where:

`simulate`(Vars)	returns random samples of program variables from their joint distribution;
`prob`(Event)	returns the probability of an event, which is a predicate on program variables;
`condition`(Event)	returns a new sum-product expression that represents the posterior distribution over all program variables given that the specified event is true.

The system architecture is designed to be modular, in the sense that modeling, conditioning, and querying are factored into distinct stages that reflect the essential components of a Bayesian workflow. Moreover, the dashed back-edge in Figure 7.1 indicates that the new sum-product expression returned by `condition` can be reused to invoke additional queries on the posterior distribution. This "closure-under-conditioning" property enables substantial runtime gains across multiple datasets and queries.

Trade-offs SPPL imposes restrictions on probabilistic programs that specifically rule out the following constructs: (i) unbounded loops; (ii) multivariate numeric transformations; and (iii) arbitrary prior distributions on continuous parameters. As a result, SPPL is not designed to express model classes such as regression with a prior on real coefficients, neural networks, support-vector machines, spatial Poisson processes, urn processes, or hidden Markov models with unknown transition matrices. These model classes cannot be represented as sum-product expressions and most of them do not have tractable algorithms for exact predictive inference.

The restrictions in SPPL ensure that the probabilistic programs can always be translated into finite sum-product expressions, thereby avoiding the need for more general computer algebra to represent probability distributions. The sum-product expressions produced by SPPL have a number of characteristics that make them a particularly useful translation target for probabilistic programs:

- **Completeness and Decomposibility**: By satisfying completeness (C3) and decomposability (C4) conditions from the literature [Poon and Domingos, 2011, Definitions 3 and 6], sum-product expressions are guaranteed to represent normalized probability distributions.

- **Efficient Factorization**: By specifying multivariate probability distributions compositionally in terms of sums and products of simpler distributions, sum-product expressions can be made more compact by applying algebraic "factorization" operations (Figures 7.3d and 7.6a).

- **Efficient Deduplication**: When an SPPL program specifies a generative model with conditional independence structure, the translated sum-product expression contains duplicate subexpressions that can be "deduplicated" to again produce a more compact symbolic expression (Figures 7.3d and 7.6b).

- **Efficient Caching**: Inference algorithms for sum-product expressions proceed from root to leaves to root, allowing intermediate results to be cached and quickly retrieved at internal subexpressions in a depth-first traversal of the computation graph.

- **Closure Under Conditioning**: Sum-product expressions are closed under probabilistic conditioning (Theorem 7.5), which allows them to be reused across multiple datasets and inference queries about the same probabilistic program.

- **Linear-Time Exact Inference**: For a well-defined class of queries, inference scales linearly in the expression size (Theorem 7.7). As a result, when SPPL constructs delivers a "small" sum-product expression, inference is also guaranteed to be fast.

It is well known that a very large class of tractable models can be cast as sum-product networks [Poon and Domingos, 2011, Theorem 2]. SPPL automatically constructs these representations from generative probabilistic programs that include standard constructs such as arrays, if/else branches, for-loops, and numeric and logical operators. To enable this translation, SPPL introduces a new class of sum-product expressions and inference algorithms that extend standard sum-product networks by supporting (many-to-one) univariate transformations, mixed-type base measures, and pointwise and set-valued constraints. These constructs make SPPL expressive enough to solve prominent inference tasks in the PPL literature [Albarghouthi et al., 2017, Nori et al., 2014, Wu et al., 2018, Laurel and Misailovic, 2020] for which standard sum-product networks have not been previously used. Example model classes that can be expressed include most finite discrete models, latent variable models with discrete hidden states and arbitrary observed states, and decision trees over discrete and continuous variables. They also include expressions in the probabilistic DSLs from Part I. Taken together, these characteristics make SPPL particularly effective for automatic and fast inference on tractable problems. The evaluations in Section 7.5 indicate that SPPL delivers these benefits on the problems it is designed to solve, whereas more general and expressive techniques in previous solvers [Gehr et al., 2016, Albarghouthi et al., 2017, Bastani et al., 2019] can exhibit orders of magnitude worse performance on these problems, runtime has higher variance, or return an unusable result.

Key Contributions This chapter makes the following contributions

- **New semantic domain for sum-product expressions** (Section 7.2) that extends the modeling expressiveness of sum-product networks by including mixed-type distributions, numeric transforms, logical formulas, and events with pointwise and set-valued constraints.

- **Provably sound exact symbolic inference algorithms** (Section 7.3) based on a proof that sum-product expressions are closed under conditioning on any event that can be specified in the domain. These algorithms enable an efficient and multi-stage inference architecture that separates model translation, conditioning, and querying into distinct stages, enabling interactive workflows and substantial computation reuse.
- **The Sum-Product Probabilistic Language** (Section 7.4), a probabilistic programming language built on a novel translation semantics from generative code to sum-product expressions. Several optimization techniques are introduced to improve the scalability of translation and inference by exploiting conditional independence structure.
- **Empirical measurements of efficacy** (Section 7.5) on inference tasks from the literature that SPPL targets, which show that it delivers substantial improvements over existing baselines. Examples include 20x–$550,000$x speedups over state-of-the-art fairness verification methods [Albarghouthi et al., 2017, Bastani et al., 2019] and computer algebra solvers [Gehr et al., 2016]; and many orders of magnitude speedup over sampling-based inference in [Milch et al., 2005].

7.1 Tutorial Examples

7.1.1 Indian GPA Problem

The Indian GPA problem, originally posed by Michael I. Jordan in a personal communication to Stuart J. Russell, has been widely considered in the probabilistic programming literature [Nitti et al., 2016, Srivastava et al., 2017, Wu et al., 2018, Riguzzi, 2018, Narayanan and Shan, 2020] for containing a “mixed-type” random variable that takes both continuous and discrete values. Most probabilistic programming languages fail to deliver sound inference in this setting.

Specifying the Prior Figure 7.2a shows the generative process for three variables (`Nationality`, `Perfect` and `GPA`) of a student. The student’s nationality is either India or USA with equal probability (line 1). Students from India (line 2) have a 10% probability of a perfect 10 GPA (lines 3-4), otherwise the GPA is uniform over $[0, 10]$ (line 5). Students from USA (line 6) have a 15% probability of a perfect 4 GPA (lines 6-7), otherwise the GPA is uniform over $[0, 4]$ (line 8). Recall that typical schools in India use a GPA scale between 0 and 10, whereas typical schools in the USA use a scale between 0 and 4.

Prior Sum-Product Expression The graph in Figure 7.2d shows the translated sum-product expression for this program, which represents a sampler for the distribution over program variables as follows: (i) if a node is a sum $(+)$, visit a random child with probability equal to the weight of the directed edge to the child; (ii) if a node is a product $(\times)$, visit each child exactly once, in no specific order; (iii) if a node is a leaf, sample a value from the distribution at the leaf and assign it to corresponding variable. Similarly, the graph encodes the joint distribution of the variables by treating (i) each sum node as a probabilistic mixture; (ii) each product node as a probabilistic factorization; and (iii) each leaf node as a primitive random variable. Thus, the prior distribution specified by the program is:

$$\begin{aligned}\Pr[\texttt{Nationality} = n, \texttt{Perfect} = p, \texttt{GPA} \leq g] = 0.5\big[&\delta_{\text{India}}(n) \cdot (0.1[(\delta_{\text{True}}(p) \cdot \mathbf{1}[10 \leq g])] \\ &+ 0.9[(\delta_{\text{False}}(p) \cdot (g/10 \cdot \mathbf{1}[0 \leq g < 10] + \mathbf{1}[10 \leq g]))])\big] \\ + 0.5\big[&\delta_{\text{USA}}(n) \cdot (0.15[(\delta_{\text{True}}(p) \cdot \mathbf{1}[4 \leq g])] \\ &+ 0.85[(\delta_{\text{False}}(p) \cdot (g/4 \cdot \mathbf{1}[0 \leq g < 4] + \mathbf{1}[4 \leq g]))])\big].\end{aligned} \tag{7.1}$$

```
Nationality ~ choice({'India': 0.5, 'USA': 0.5})
if (Nationality == 'India'):
    Perfect ~ bernoulli(p=0.10)
    if Perfect:         GPA ~ atom(10)
    else:               GPA ~ uniform(0, 10)
else: # Nationality is 'USA'
    Perfect ~ bernoulli(p=0.15)
    if Perfect:         GPA ~ atom(4)
    else:               GPA ~ uniform(0, 4)
```

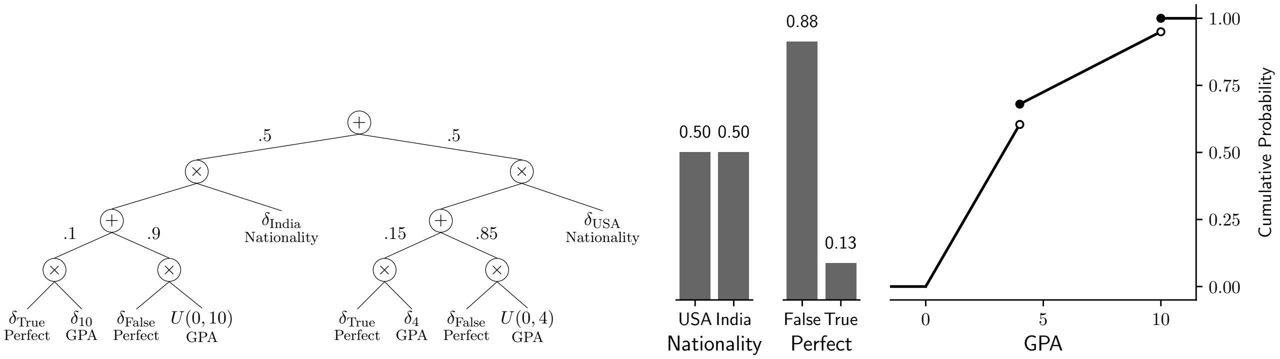
(a) Probabilistic Program

```
prob (Nationality == 'USA');
prob (Perfect == 1);
prob (GPA <= x/10) # for x = 0, ..., 120
```

(b) Example Queries on Marginal Probabilities

```
prob ((Perfect == 1)
  or (Nationality == 'India') and (GPA > 3))
```

(c) Example Query on Joint Probabilities

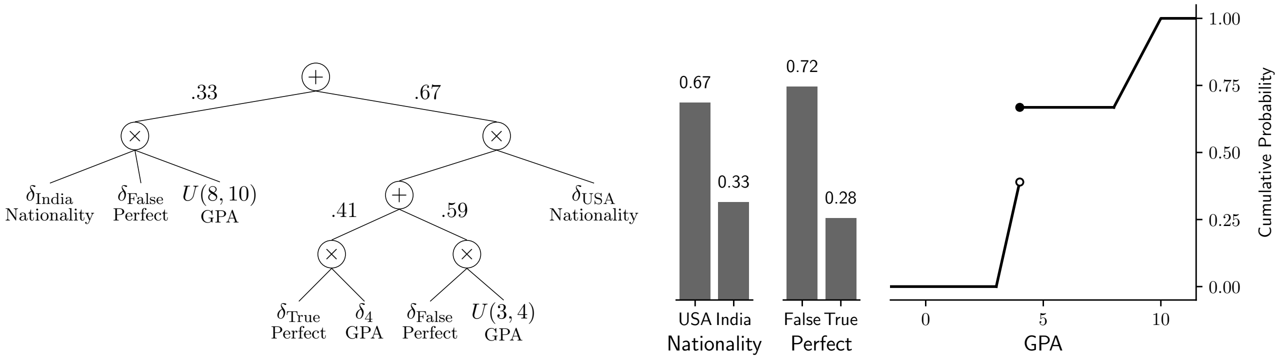


(d) Prior Sum-Product Expression

(e) Prior Marginal Distributions

```
condition ((Nationality == 'USA') and (GPA > 3)) or (8 < GPA < 10)
```

(f) Conditioning the Program



(g) Posterior Sum-Product Expression

(h) Posterior Marginal Distributions

Figure 7.2: Analyzing the Indian GPA problem in SPPL.

Figure 7.2b shows SPPL queries for the prior marginal distributions of the three variables, plotted in Figure 7.2e. The jumps in the cumulative distribution function (CDF) of `GPA` at 4 and 10 correspond to the atoms that occur when `Perfect` is true and the `Nationality` is USA and India, respectively.

Conditioning the Program Figure 7.2f shows an example of the `condition` query, which specifies an event e on which to constrain executions of the program. An event is a predicate on (possibly transformed) program variables that can be used for both `condition` (Figure 7.2f) and `prob` (Figure 7.2c) queries. SPPL is the first system with inference algorithms for sum-product expressions that handle predicates of this form. Given e, the object of inference is to compute full posterior distribution:

$$\Pr[\texttt{Nationality} = n, \texttt{Perfect} = p, \texttt{GPA} \le g \mid e] ::= \frac{\Pr[\texttt{Nationality}\,{=}\,n, \texttt{Perfect}\,{=}\,p, \texttt{GPA} \le g, e]}{\Pr[e]}. \tag{7.2}$$

Posterior Sum-Product Expression Given the prior expression (Figure 7.2d) and conditioning event e (Figure 7.2f), SPPL produces a new expression (Figure 7.2g) that specifies a distribution which is precisely equal to Eq. (7.2), From Theorem 7.5, conditioning an SPPL program on any event that can be specified in the language results in a posterior distribution that can be represented as a sum-product expression. Conditioning on e performs several transformations on the prior expression, including:

1. Eliminating the subtree rooted at the parent of leaf δ_{10}, which is inconsistent with the condition.

2. Rescaling the distribution $U(0, 10)$ at the leaf node in the India subtree to $U(8, 10)$.

3. Rescaling the distribution $U(0, 4)$ at the leaf node in the USA subtree to $U(3, 4)$.

4. Reweighting the branch probabilities of the sum node in the USA subtree from $[.15, .85]$ to $[.41, .59]$, where $.41 = .15/(.15 + .2125)$ is the posterior probability of $\{\texttt{Perfect} = 1, \texttt{GPA} = 4\}$ given the condition e:

$$\begin{aligned}\Pr[\texttt{Perfect} = 1, \texttt{GPA} = 4 \mid \texttt{Nationality} = \texttt{'USA'}, \texttt{GPA} > 3] &= (0.15 \times 1)/c = 0.15/c\\ \Pr[\texttt{Perfect} = 0, 3 < \texttt{GPA} < 4 \mid \texttt{Nationality} = \texttt{'USA'}, \texttt{GPA} > 3] &= (0.85 \times 0.25)/c = 0.2125/c.\end{aligned}$$

5. Reweighting the branch probabilities at the root from $[.5, .5]$ to $[.33, .67]$, using the same rules as in the previous step.

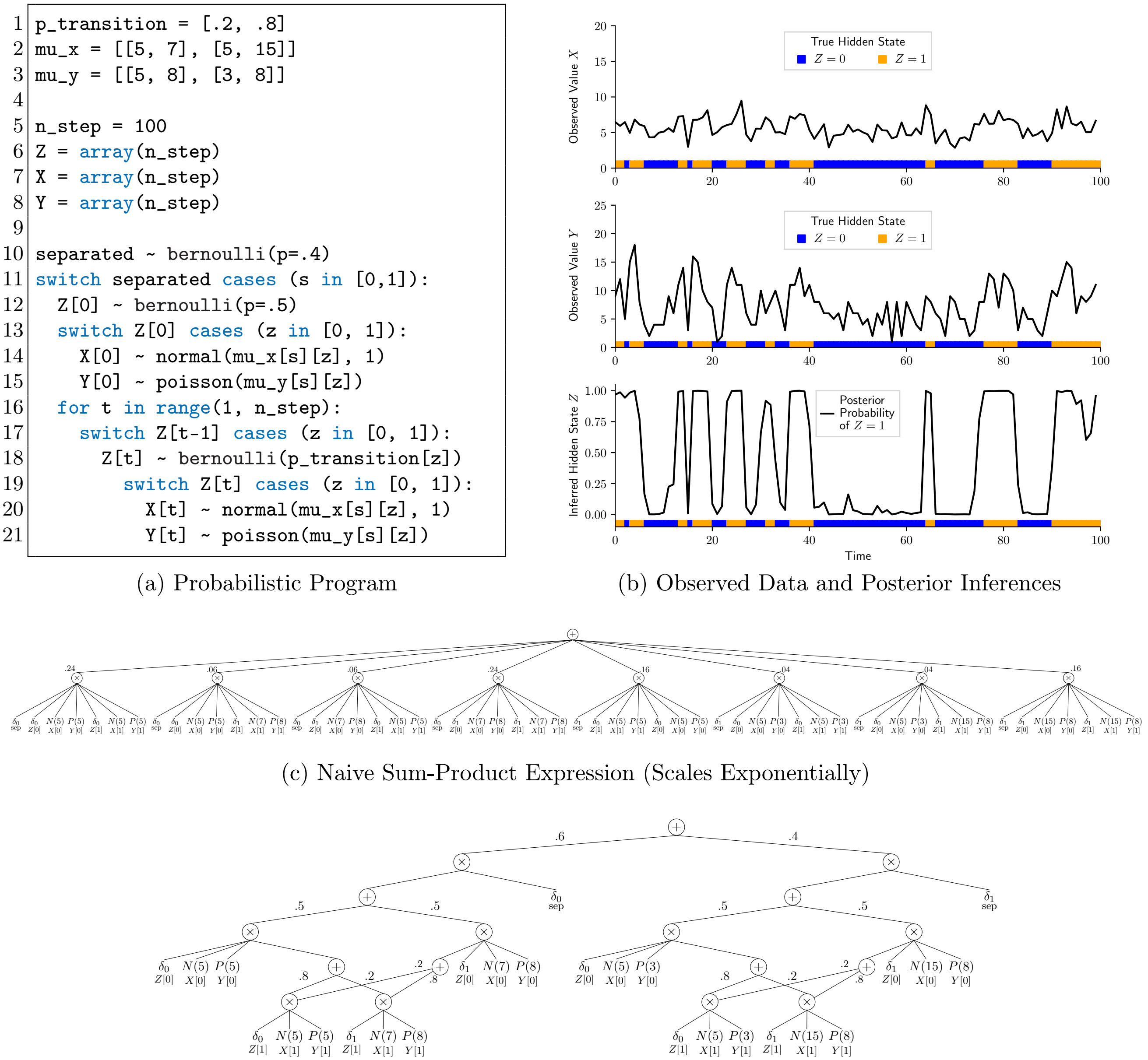
Figure 7.2g shows the posterior expression obtained by applying these symbolic transformations. Using the posterior expression, the right-hand side of Eq. (7.2) is

$$\begin{aligned}&\Pr[\texttt{Nationality} = n, \texttt{Perfect} = p, \texttt{GPA} \le g \mid e] \qquad (7.3)\\ &= 0.33 \left[\delta_{\text{India}}(n) \cdot \delta_{\text{False}}(p) \cdot \left(\frac{g-8}{2} \cdot \mathbf{1}[8 \le g < 10] + \mathbf{1}[10 \le g]\right)\right]\\ &+ 0.67\Bigg[\delta_{\text{USA}}(n) \cdot \Big(0.41 \left[\delta_{\text{True}}(p) \cdot \mathbf{1}[4 \le g]\right]\\ &\qquad\qquad + 0.59 \left[\delta_{\text{False}}(p) \cdot \left(\frac{g}{4} \cdot \mathbf{1}[0 \le g < 4] + \mathbf{1}[4 \le g]\right)\right]\Big)\Bigg].\end{aligned}$$

(Floats are written to two decimal places.) The `prob` queries in Figure 7.2b can be executed again on the conditioned program to plot the posterior marginal distributions, which are shown in Figure 7.2h. The example in Figure 7.2 illustrates a typical modular workflow in SPPL (Figure 7.1), where modeling (Figure 7.2a), conditioning (Figure 7.2f) and querying (Figures 7.2b and 7.2c) are separated into distinct and reusable stages that together express the main components of Bayesian modeling and inference.

7.1.2 Scalable Inference in a Hierarchical HMM

The next example shows how to perform efficient inference in a hierarchical hidden Markov model [HMM; Murphy and Paskin, 2001] and illustrates optimization techniques used by the SPPL translator (Section 7.4.1), which exploit conditional independence to ensure that the size of the sum-product expression grows linearly in the number of time steps. The code box in Figure 7.3a shows a hierarchical



```
p_transition = [.2, .8]
mu_x = [[5, 7], [5, 15]]
mu_y = [[5, 8], [3, 8]]

n_step = 100
Z = array(n_step)
X = array(n_step)
Y = array(n_step)

separated ~ bernoulli(p=.4)
switch separated cases (s in [0,1]):
  Z[0] ~ bernoulli(p=.5)
  switch Z[0] cases (z in [0, 1]):
    X[0] ~ normal(mu_x[s][z], 1)
    Y[0] ~ poisson(mu_y[s][z])
  for t in range(1, n_step):
    switch Z[t-1] cases (z in [0, 1]):
      Z[t] ~ bernoulli(p_transition[z])
        switch Z[t] cases (z in [0, 1]):
          X[t] ~ normal(mu_x[s][z], 1)
          Y[t] ~ poisson(mu_y[s][z])
```

(a) Probabilistic Program

(b) Observed Data and Posterior Inferences

(c) Naive Sum-Product Expression (Scales Exponentially)

(d) Optimized Sum-Product Expression (Scales Linearly)

Figure 7.3: Fast smoothing in a hierarchical hidden Markov model using SPPL. The systems constructs an efficient sum-product expression that exploits conditional independencies in the generative process.

HMM with Bernoulli hidden states Z_t and Normal–Poisson observations (X_t, Y_t). The `separated` variable indicates whether the mean values of X_t and Y_t at $Z_t = 0$ and $Z_t = 1$ are well separated. For example, `mu_x` specifies that if `separated` $= 0$, then the mean of X_t is 5 when $Z_t = 0$ and 7 when $Z_t = 1$; otherwise if `separated` $= 1$, then the mean of X_t is 4 when $Z_t = 0$ and 15 when $Z = 1$. The `p_transition` vector specifies that the current state Z_t switches from the previous state Z_{t-1} with 20% probability. This example shows the SPPL `array`, `for`, and `switch`-`cases` statements, where the latter is a macro that expands to `if`-`else` statements (as in, e.g., the C language):

$$\texttt{switch}\ x\ \texttt{cases}\ \texttt{(}x'\ \texttt{in}\ \mathit{values}\texttt{)}\ \texttt{\{}C\texttt{\}} \overset{\text{desugar}}{\rightsquigarrow} \begin{array}{l} \texttt{if (}x\ \texttt{in}\ \mathit{values}[0]\texttt{)}\ \texttt{\{}C[x'/\mathit{values}[0]]\texttt{\}} \\ \texttt{elif}\ \ldots \\ \texttt{elif (}x\ \texttt{in}\ \mathit{values}[n{-}1]\texttt{)}\ \texttt{\{}C[x'/\mathit{values}[n-1]]\texttt{\}}, \end{array} \tag{7.4}$$

where n is the length of *values* and $C[x/E]$ indicates syntactic replacement of the variable x with the expression E in command C.

The top and middle plots in Figure 7.3b show a realization of X and Y that result from simulating the process for 100 time steps. The blue and orange regions along the horizontal axes indicate whether the true hidden state Z is 0 or 1, respectively. The goal of inference is to compute posterior probabilities over Z given X and Y; the ground-truth values of Z are shown for illustration only. The bottom plot in Figure 7.3b shows the exact posterior marginal probabilities $\Pr[Z_t = 1 \mid x_{0:99}, y_{0:99}]$ for each $t = 0, \dots, 99$ as inferred by SPPL, which is an inference known as "smoothing". These probabilities track the true hidden state, i.e., the probability $Z_t = 1$ is low in the blue regions and high in the orange regions.

Figure 7.3c shows a "naive" sum-product expression for the distribution of all program variables up to the first two time steps. This expression is a sum-of-products. The products in the second level are an enumeration of all possible realizations of program variables, which causes the number of terms to scale exponentially in the number of time steps. Figure 7.3d shows the optimized expression constructed by SPPL, which is based on factoring and sharing common terms in the two level sum-of-products in Figure 7.3c. These factorization and deduplication optimizations exploit conditional independencies and repeated structure in the program, which for the hierarchical HMM delivers a expression whose size scales linearly in the number of time points. SPPL can also efficiently solve the following queries using the same underlying inference machinery: (i) filtering: $\Pr[Z_t \mid X_{1:t}, Y_{1:t}]$, for all t; (ii) smoothing: $\Pr[Z_t \mid X_{1:n}, Y_{1:n}]$, for all $t < n$; (iii) full joint: $\Pr[Z_{1:n} \mid X_{1:n}, Y_{1:n}]$; (iv) forecasting: $\Pr[X_{n+1:n+k}, Y_{n+1:n+k}, \mid X_{1:n}, Y_{1:n}]$, for all k; and (v) marginal likelihood: $\Pr[X_{1:n}, Y_{1:n}]$, for all n.

7.2 Core Calculus for Sum-Product Expressions

SPPL is formalized in terms of a semantic domain of sum-product expressions which generalizes sum-product networks and enables precise probabilistic reasoning. This domain will be used to (i) establish the closure of sum-product expressions under conditioning on events expressible in the calculus (Theorem 7.5); (ii) describe sound algorithms for exact Bayesian inference (Section 7.3 and Appendix 7.D); and (iii) describe a procedure for translating generative probabilistic programs into sum-product expressions in the core calculus (Section 7.4). Listing 7.1 shows the domains in the core calculus, which includes basic sets (Listing 7.1a) outcomes (Listing 7.1b); real transforms (Listing 7.1c); predicates with pointwise and set-valued constraints (Listing 7.1d); primitive distributions (Listing 7.1e); and multivariate distributions specified compositionally as sums and products of primitive distributions (Listing 7.1f). Listing 7.2 shows the semantics of the core calculus, which are described next in detail.

Basic Outcomes Random variables in the calculus take values in the $\mathsf{Outcome} ::= \mathsf{Real} + \mathsf{String}$ domain. The symbol $+$ here indicates a sum (disjoint-union) data type, whose elements are formed by the injection operation, e.g., $\downarrow^{\mathsf{Real}}_{\mathsf{Outcome}} r$ for $r \in \mathsf{Real}$ $\downarrow^{\mathsf{String}}_{\mathsf{Outcome}} s$ for $s \in \mathsf{String}$ This domain is used to model mixed-type random variables, such as `X` in the SPPL program from Listing 7.3. The $\mathsf{Outcomes}$ domain denotes a subset of $\mathsf{Outcome}$ as defined by the valuation function $\mathbb{V}$ shown in Listing 7.2a. For example, `((`$b_1\, r_1$`)` `(`$r_2\, b_2$`))` specifies a (open, closed, or clopen) real interval and `{`$s_1 \dots s_m$`}`^{b} is a set of strings, where $b =$ `#t` indicates the complement. In this notation, meta-variables such as m indicate an arbitrary but finite number of repetitions of a domain variable or subexpression. The operations *union*,

$x \in \mathsf{Var}$
$n \in \mathsf{Natural}$
$b \in \mathsf{Boolean} ::= \{\texttt{\#t}, \texttt{\#f}\}$
$u \in \mathsf{Unit} ::= \{\texttt{\#u}\}$
$w \in [0,1]$
$r \in \mathsf{Real} \cup \{-\infty, \infty\}$
$s \in \mathsf{String} ::= \mathsf{Char}^*$

(a) Basic Sets

$rs \in \mathsf{Outcome} ::= \mathsf{Real} + \mathsf{String}$
$v \in \mathsf{Outcomes}$

$::= \varnothing$	[Empty]
$\mid \texttt{\{}s_1 \dots s_m\texttt{\}}^b$	[FiniteStr]
$\mid \texttt{\{}r_1 \dots r_m\texttt{\}}$	[FiniteReal]
$\mid \texttt{((}b_1\, r_1\texttt{)}\ \texttt{(}r_2\, b_2\texttt{))}$	[Interval]
$\mid v_1 \amalg \cdots \amalg v_m$	[Union]

(b) Outcomes

$t \in \mathsf{Transform}$

$::= \texttt{Id(}x\texttt{)}$	[Identity]
$\mid \texttt{Reciprocal(}t\texttt{)}$	[Reciprocal]
$\mid \texttt{Abs(}t\texttt{)}$	[AbsValue]
$\mid \texttt{Root(}t\, n\texttt{)}$	[Radical]
$\mid \texttt{Exp(}t\, r\texttt{)}$	[Exponent]
$\mid \texttt{Log(}t\, r\texttt{)}$	[Logarithm]
$\mid \texttt{Poly(}t\, r_0 \dots r_m\texttt{)}$	[Polynomial]
$\mid \texttt{Piecewise((}t_1\, e_1\texttt{)}$ $\dots$ $\texttt{(}t_m\, e_m\texttt{))}$	[Piecewise]

(c) Transformations

$e \in \mathsf{Event}$

$::= \texttt{(}t\ \texttt{in}\, v\texttt{)}$	[Containment]
$\mid e_1 \sqcap \cdots \sqcap e_m$	[Conjunction]
$\mid e_1 \sqcup \cdots \sqcup e_m$	[Disjunction]

(d) Events

$F \in \mathsf{CDF} \subset \mathsf{Real} \to [0,1]$
$::= \texttt{Norm(}r_1, r_2\texttt{)} \mid \texttt{Poisson(}r\texttt{)} \dots$
where F is càdlàg;
$\lim_{r\to\infty} F(r) = 1$; $\lim_{r\to-\infty} F(r) = 0$;
and $F^{-1}(u) ::= \inf\{r \mid u \leq F(r)\}$.

$d \in \mathsf{Distribution}$

$::= \texttt{DistR(}F\, r_1\, r_2\texttt{)}$	[DistReal]
$\mid \texttt{DistI(}F\, r_1\, r_2\texttt{)}$	[DistInt]
$\mid \texttt{DistS((}s_1\, w_1\texttt{)} \dots \texttt{(}s_m\, w_m\texttt{))}$	[DistStr]

(e) Primitive Distributions

$\sigma \in \mathsf{Environment} ::= \mathsf{Var} \to \mathsf{Transform}$
$S \in \mathsf{SPE}$

$::= \texttt{Leaf(}x\, d\, \sigma\texttt{)}$	[Leaf]
$\mid \texttt{(}S_1\, w_1\texttt{)} \oplus \cdots \oplus \texttt{(}S_m\, w_m\texttt{)}$	[Sum]
$\mid S_1 \otimes \cdots \otimes S_m$	[Product]

(f) Sum-Product

Listing 7.1: Syntax of core calculus for sum-product expressions and related domains.

intersection, and *complement* operate on Outcomes in the usual set-theoretic way, while preserving the semantic invariants described in Appendix 7.B.

A Sigma Algebra of Outcomes To speak precisely about random variables and measures on Outcome, a sigma-algebra $\mathcal{B}(\mathsf{Outcome}) \subset \mathcal{P}(\mathsf{Outcome})$ is constructed as follows:

1. Let τ_{Real} be the usual topology on Real.

2. Let τ_{String} be the discrete topology on String.

3. Let $\tau_{\mathsf{Outcome}} ::= \tau_{\mathsf{Real}} \uplus \tau_{\mathsf{String}}$ be the disjoint-union topology on Outcome, where a set $U \subset \mathsf{Outcome}$ is open iff $\{r \mid (\downarrow^{\mathsf{Real}}_{\mathsf{Outcome}} r) \in U\}$ is open in Real and $\{s \mid (\downarrow^{\mathsf{String}}_{\mathsf{Outcome}} s) \in U\}$ is open in String.

4. Let $\mathcal{B}(\mathsf{Outcome})$ be the Borel sigma-algebra of τ_{Outcome}.

Remark 7.1. As measures on Real are defined by their values on open intervals and measures on String on singletons, mappings from Outcomes to $[0,1]$ induce probability measures on $\mathcal{B}(\mathsf{Outcome})$. «

Real Transformations Listing 7.2b describes real transformations that can be applied to variables in the core calculus. The Identity domain, written $\texttt{Id(}x\texttt{)}$, is a terminal subexpression of any Transform t and contains a single variable name that specifies the "dimension" over which t is defined. The main operation involving transforms is computing the preimage of Outcomes v under t using the *preimg* : Transform → Outcomes → Outcomes operation, which satisfies the following properties:

$$(\downarrow^{\mathsf{Real}}_{\mathsf{Outcome}} r) \in \mathbb{V}[\![\mathit{preimg}\ t\ v]\!] \iff \mathbb{T}[\![t]\!](r) \in \mathbb{V}[\![v]\!] \tag{7.5}$$

$$(\downarrow^{\mathsf{String}}_{\mathsf{Outcome}} s) \in \mathbb{V}[\![\mathit{preimg}\ t\ v]\!] \iff (t \in \mathsf{Identity}) \wedge (s \in \mathbb{V}[\![v]\!]). \tag{7.6}$$

$$
\begin{array}{lr}
\mathbb{V} : \textsf{Outcomes} \to \mathcal{P}(\textsf{Outcome}) & \\
\mathbb{V}\llbracket \varnothing \rrbracket ::= \varnothing & \textsf{[Empty]} \\
\mathbb{V}\left\llbracket \texttt{\{}s_1 \dots s_m\texttt{\}}^{b} \right\rrbracket ::= \textbf{if } b \textbf{ then } \cup_{i=1}^{m} \{(\downarrow {}^{\textsf{String}}_{\textsf{Outcome}}\, s_i)\} & \textsf{[FiniteStr]} \\
\qquad\qquad\qquad\quad \textbf{else } \{(\downarrow {}^{\textsf{String}}_{\textsf{Outcome}}\, s) \mid \forall i. s \neq s_i\} & \\
\mathbb{V}\llbracket \texttt{\{}r_1 \dots r_m\texttt{\}} \rrbracket ::= \cup_{i=1}^{m}\{(\downarrow {}^{\textsf{Real}}_{\textsf{Outcome}}\, r_i)\} & \textsf{[FiniteReal]} \\
\mathbb{V}\llbracket \texttt{((}b_1\, r_1\texttt{)}\ \texttt{(}r_2\, b_2\texttt{))} \rrbracket ::= \{(\downarrow {}^{\textsf{Real}}_{\textsf{Outcome}}\, r) \mid r_1 <_{b_1} r <_{b_2} r_2\} & \textsf{[Interval]} \\
\qquad\qquad\qquad\qquad \text{where } <_{\texttt{\#t}} ::= <; <_{\texttt{\#f}} ::= \leq; r_1 < r_2 & \\
\mathbb{V}\llbracket v_1 \amalg \cdots \amalg v_m \rrbracket ::= \cup_{i=1}^{m} \mathbb{V}\llbracket v_i \rrbracket & \textsf{[Union]}
\end{array}
$$

(a) Outcomes

$$
\begin{array}{l}
\mathbb{T} : \textsf{Transform} \to \textsf{Real} \to \textsf{Real} \\
\mathbb{T}\llbracket \texttt{Id(}x\texttt{)} \rrbracket ::= \lambda r'. r';\ \mathbb{T}\llbracket \texttt{Reciprocal(}t\texttt{)} \rrbracket ::= \lambda r'. 1/\left(\mathbb{T}\llbracket t \rrbracket (r')\right); \\
\mathbb{T}\llbracket \texttt{Abs(}t\texttt{)} \rrbracket ::= \lambda r'. |\mathbb{T}\llbracket t \rrbracket (r')|; \mathbb{T}\llbracket \texttt{Root(}t\, n\texttt{)} \rrbracket ::= \lambda r'. \sqrt[n]{\mathbb{T}\llbracket t \rrbracket (r')}; \\
\mathbb{T}\llbracket \texttt{Poly(}t\ r_0\ \dots\ r_m\texttt{)} \rrbracket ::= \lambda r'. \sum_{i=0}^{m} r_i \left(\mathbb{T}\llbracket t \rrbracket (r')\right)^{i}; \dots
\end{array}
$$

(b) Transformations

$$
\begin{array}{lr}
\mathbb{E} : \textsf{Event} \to \textsf{Var} \to \textsf{Outcomes} & \\
\mathbb{E}\llbracket \texttt{(}t\, \texttt{in}\, v\texttt{)} \rrbracket x ::= \textbf{if } (\mathit{vars}\, t) = \{x\} \textbf{ then } (\mathit{preimg}\, t\, v) \textbf{ else } \varnothing & \textsf{[Containment]} \\
\mathbb{E}\llbracket e_1 \sqcap \cdots \sqcap e_m \rrbracket x ::= (\mathit{intersection}\, \mathbb{E}\llbracket e_1 \rrbracket x\ \dots\ \mathbb{E}\llbracket e_m \rrbracket x) & \textsf{[Conjunction]} \\
\mathbb{E}\llbracket e_1 \sqcup \cdots \sqcup e_m \rrbracket x ::= (\mathit{union}\, \mathbb{E}\llbracket e_1 \rrbracket x\ \dots\ \mathbb{E}\llbracket e_m \rrbracket x) & \textsf{[Disjunction]}
\end{array}
$$

(c) Events

$$
\begin{array}{lr}
\mathbb{D} : \textsf{Distribution} \to \textsf{Outcomes} \to [0,1] & \\
\mathbb{D}\llbracket \texttt{DistS((}s_i\, w_i\texttt{)}_{i=1}^{m}\texttt{)} \rrbracket v ::= \textbf{match } (\mathit{intersection}\, v\, \texttt{\{}s_1 \dots s_m\texttt{\}}^{\texttt{\#f}}) & \textsf{[DistStr]} \\
\quad \rhd \varnothing \mid \texttt{\{}r'_1 \dots r'_m\texttt{\}} \mid \texttt{((}b_1\, r_1\texttt{)}\ \texttt{(}r_2\, b_2\texttt{))} \Rightarrow 0 & \\
\quad \rhd v_1 \amalg \cdots \amalg v_m \Rightarrow \sum_{i=1}^{m} \mathbb{D}\llbracket \texttt{DistS((}s_i\, w_i\texttt{)}_{i=1}^{m}\texttt{)} \rrbracket v_i & \\
\quad \rhd \texttt{\{}s'_1\ \dots\ s'_k\texttt{\}}^{b} \Rightarrow \textbf{let } w \textbf{ be } \sum_{i=1}^{m}(w_i \textbf{ if } s_i \in \{s'_j\}_{j=1}^{k} \textbf{ else } 0) & \\
\qquad\qquad\qquad\qquad \textbf{in if } \bar{b} \textbf{ then } w \textbf{ else } 1 - w & \\
\mathbb{D}\llbracket \texttt{DistR(}F\, r_1\, r_2\texttt{)} \rrbracket v ::= \textbf{match } (\mathit{intersection}\ \texttt{((\#f}\, r_1\texttt{)}\ \texttt{(}r_2\, \texttt{\#f))}\ v) & \textsf{[DistReal]} \\
\quad \rhd \varnothing \mid \texttt{\{}r'_1 \dots r'_m\texttt{\}} \mid \texttt{\{}s'_1\ \dots\ s'_k\texttt{\}}^{b} \Rightarrow 0 & \\
\quad \rhd v_1 \amalg \cdots \amalg v_m \Rightarrow \sum_{i=1}^{m} \mathbb{D}\llbracket \texttt{DistR(}F\, r_1\, r_2\texttt{)} \rrbracket v_i & \\
\quad \rhd \texttt{((}b'_1\, r'_1\texttt{)}\ \texttt{(}r'_2\, b'_2\texttt{))} \Rightarrow \dfrac{F(r'_2) - F(r'_1)}{F(r_2) - F(r_1)} & \\
\mathbb{D}\llbracket \texttt{DistI(}F\, r_1\, r_2\texttt{)} \rrbracket v ::= \textbf{match } (\mathit{intersection}\ \texttt{((\#f}\, r_1\texttt{)}\ \texttt{(}r_2\, \texttt{\#f))}\ v) & \textsf{[DistInt]} \\
\quad \rhd \varnothing \mid \texttt{\{}s'_1\ \dots\ s'_k\texttt{\}}^{b} \Rightarrow 0 & \\
\quad \rhd v_1 \amalg \cdots \amalg v_m \Rightarrow \sum_{i=1}^{m} \mathbb{D}\llbracket \texttt{DistI(}F\, r_1\, r_2\texttt{)} \rrbracket v_i & \\
\quad \rhd \texttt{\{}r'_1 \dots r'_m\texttt{\}} \Rightarrow \dfrac{\displaystyle\sum_{i=1}^{m} \begin{bmatrix} \textbf{if } (r'_i = \lfloor r'_i \rfloor) \wedge (r_1 \leq r'_i \leq r_2) \\ \textbf{then } F(r') - F(r' - 1) \textbf{ else } 0 \end{bmatrix}}{F(\lfloor r_2 \rfloor) - F(\lceil r_1 \rceil - 1)} & \\
\quad \rhd \texttt{((}b'_1\, r'_1\texttt{)}\ \texttt{(}r'_2\, b'_2\texttt{))} \Rightarrow \textbf{let } \tilde{r}_1 \textbf{ be } \lfloor r'_1 \rfloor - \mathbf{1}\left[(r'_1 = \lfloor r'_1 \rfloor) \wedge \bar{b'_1}\right] & \\
\qquad\qquad\qquad\qquad\qquad\quad \textbf{in let } \tilde{r}_2 \textbf{ be } \lfloor r'_2 \rfloor - \mathbf{1}\left[(r'_2 = \lfloor r'_2 \rfloor) \wedge \bar{b'_2}\right] & \\
\qquad\qquad\qquad\qquad\qquad\quad \textbf{in } \dfrac{F(\tilde{r}_2) - F(\tilde{r}_1)}{F(\lfloor r_2 \rfloor) - F(\lceil r_1 \rceil - 1)} &
\end{array}
$$

(d) Primitive Distributions

Listing 7.2: Semantics of core calculus for sum-product domains.

$$\mathbb{P}_0 \llbracket S \rrbracket : \mathsf{SPE} \to \mathsf{Event} \to \mathsf{Natural} \times [0, \infty)$$

$$\mathbb{P}_0 \llbracket \mathtt{Leaf}\,\mathtt{(}x\,d\,\sigma\mathtt{)} \rrbracket \; \mathtt{(Id(}x\mathtt{)\;in\,\{}rs\mathtt{\})} ::= \mathbf{match}\, d \qquad \textsf{[Leaf]}$$

$$\quad \rhd \mathtt{DistR(}F\, r_1\, r_2\mathtt{)} \Rightarrow \; \mathbf{match}\, rs$$

$$\qquad \rhd r \Rightarrow (1, \mathbf{1}[r_1 \le r \le r_2] F'(r) / \left[F(r_2) - F(r_1)\right])$$

$$\qquad \rhd s \Rightarrow (1, 0)$$

$$\quad \rhd \mathbf{else} \Rightarrow \mathbf{let}\, w \,\mathbf{be}\, \mathbb{P} \llbracket \mathtt{Leaf}\,\mathtt{(}x\,d\,\sigma\mathtt{)} \rrbracket \, \mathtt{(Id(}x\mathtt{)\;in\,\{}rs\mathtt{\})} \,\mathbf{in}\, (\mathbf{1}[w = 0], w)$$

$$\mathbb{P}_0 \llbracket \mathtt{(}S_1\, w_1\mathtt{)} \oplus \cdots \oplus \mathtt{(}S_m\, w_m\mathtt{)} \rrbracket \; \sqcap_{i=1}^{\ell} \mathtt{(Id(}x_i\mathtt{)\;in\,\{}rs_i\mathtt{\})} ::=$$

$$\quad \mathbf{let}_{1 \le i \le m}\, (d_i, p_i) \,\mathbf{be}\, \mathbb{P}_0 \llbracket S_i \rrbracket \left(\sqcap_{i=1}^{\ell} \mathtt{(Id(}x_i\mathtt{)\;in\,\{}rs_i\mathtt{\})}\right) \qquad \textsf{[Sum]}$$

$$\quad \mathbf{in\,if}\, \forall_{1 \le i \le m}.\; p_i = 0 \;\mathbf{then}\; (1, 0)$$

$$\qquad \mathbf{else\,let}\, d^* \;\mathbf{be}\; \min\{d_i \mid 1 \le i \le m, 0 < p_i\}$$

$$\qquad\quad \mathbf{in}\, (d^*, \textstyle\sum_{i=1}^{m} \mathbf{1}[d_i = d^*] w_i p_i)$$

$$\mathbb{P}_0 \llbracket S_1 \otimes \cdots \otimes S_m \rrbracket \; \sqcap_{i=1}^{\ell} \mathtt{(Id(}x_i\mathtt{)\;in\,\{}rs_i\mathtt{\})} ::= \qquad \textsf{[Product]}$$

$$\quad \mathbf{let}_{1 \le i \le m}\, (d_i, p_i) \,\mathbf{be\,match}\; \{x_1, \dots, x_m\} \cap (\mathit{scope}\, S_i)$$

$$\qquad \rhd \{n_1, \dots, n_k\} \Rightarrow \mathbb{P}_0 \llbracket S_i \rrbracket \sqcap_{t=1}^{k} \mathtt{(Id(}x_{n_t}\mathtt{)\;in\,\{}rs_t\mathtt{\})}$$

$$\qquad \rhd \{\} \Rightarrow (0, 1)$$

$$\quad \mathbf{in}\; (\textstyle\sum_{i=1}^{n} d_i, \prod_{i=1}^{m} p_i)$$

(e) Sum-Product Expressions (Density Semantics)

$$\mathbb{P} : \mathsf{SPE} \to \mathsf{Event} \to [0, 1]$$

$$\mathbb{P} \llbracket \mathtt{Leaf}\,\mathtt{(}x\,d\,\sigma\mathtt{)} \rrbracket\, e ::= \mathbb{D} \llbracket d \rrbracket \, (\mathbb{E} \llbracket (\mathit{subsenv}\; e\; \sigma) \rrbracket\, x) \qquad \textsf{[Leaf]}$$

$$\mathbb{P} \llbracket \mathtt{(}S_1\, w_1\mathtt{)} \oplus \cdots \oplus \mathtt{(}S_m\, w_m\mathtt{)} \rrbracket\, e ::= \mathbf{let}\, Z \,\mathbf{be}\, \textstyle\sum_{i=i}^{m} w_i \qquad \textsf{[Sum]}$$

$$\qquad \mathbf{in}\; \textstyle\sum_{i=1}^{m} (\mathbb{P} \llbracket S_i \rrbracket\, e) w_i / Z$$

$$\mathbb{P} \llbracket S_1 \otimes \cdots \otimes S_m \rrbracket\, e ::= \mathbf{match}\; (\mathit{dnf}\; e) \qquad \textsf{[Product]}$$

$$\quad \rhd \mathtt{(}t\; \mathtt{in}\; v\mathtt{)} \Rightarrow \mathbf{let}\; n \;\mathbf{be}\; \min\{1 \le i \le m \mid (\mathit{vars}\, e) \subset (\mathit{scope}\, S_i)\}$$

$$\qquad \mathbf{in}\; \mathbb{P} \llbracket S_n \rrbracket\, e$$

$$\quad \rhd \mathtt{(}e_1 \sqcap \cdots \sqcap e_\ell\mathtt{)} \Rightarrow$$

$$\qquad \prod_{1 \le i \le m} \left[\begin{array}{l} \mathbf{match}\; \{1 \le j \le \ell \mid (\mathit{vars}\, e_j) \subset (\mathit{scope}\, S_i)\} \\ \rhd \{n_1, \dots, n_k\} \Rightarrow \mathbb{P} \llbracket S_i \rrbracket\, (e_{n_1} \sqcap \cdots \sqcap e_{n_k}) \\ \rhd \{\} \Rightarrow 1 \end{array} \right]$$

$$\quad \rhd \mathtt{(}e_1 \sqcup \cdots \sqcup e_\ell\mathtt{)} \Rightarrow \sum_{J \subset [\ell]} \left[(-1)^{|J|-1}\, \mathbb{P} \llbracket S_1 \otimes \cdots \otimes S_m \rrbracket\, (\sqcap_{i \in J}\, e_i) \right]$$

(f) Sum-Product Expressions (Distribution Semantics)

Listing 7.2: Semantics of core calculus for sum-product domains (continued).

```
Z ~ normal(0, 1)
if (Z <= 0):
  X ~ choice({"negative": 1}) # string-valued
elif (0 < Z < 4):
  X ~ 2 * exp(Z)              # real-valued
elif (4 <= Z):
  X ~ atomic(4)               # integer-valued
```

Listing 7.3: The random variable X in the SPPL program above is string-valued, integer-valued, or real-valued, depending on the stochastic execution path as determined by the random value of Z.

Appendix 7.C formalizes this domain and presents a robust semi-symbolic solver that implements *preimg* for each Transform, which enables exact probabilistic inferences about transformed variables. Example 7.8 further illustrates these concepts.

Events Listing 7.2c shows the semantics of the Event domain, which specifies predicates on variables. The valuation $\mathbb{E}\,[\![e]\!] : \mathsf{Var} \to \mathsf{Outcomes}$ of an Event takes a variable x and returns the set $v \in \mathsf{Outcomes}$ of elements that satisfy the predicate along dimension x, based on Eqs. (7.5) and (7.6).

Example 7.2. The predicate

$$\phi(X_1, X_2) ::= \{0 \le X_1 < 1\} \cup \{1/X_2 > 6\} \tag{7.7}$$

corresponds in the core calculus to

$$e = (\texttt{Id(X}_1\texttt{) in ((\#f 0) (1 \#t))}) \sqcup (\texttt{1/Id(X}_2\texttt{) in ((\#t 6) (}\infty\,\texttt{\#t))}) \in \mathsf{Event}. \tag{7.8}$$

Therefore $\mathbb{E}\,[\![e]\!]\,\texttt{X}_1 = \texttt{((\#f 0) (1 \#t))}$ and $\mathbb{E}\,[\![e]\!]\,\texttt{X}_2 = \texttt{((\#f }-\infty\texttt{) (6 \#f))}$. «

The Event domain is used to specify measurable sets of an n-dimensional distribution on variables $\{x_1, \dots, x_n\}$ as follows: let $\sigma_{\mathrm{gen}}(\{A_1, A_2, \dots\})$ be the sigma-algebra generated by $A_1, A_2, \dots$, and define

$$\mathcal{B}^n(\mathsf{Outcome}) ::= \sigma_{\mathrm{gen}}(\{\textstyle\prod_{i=1}^{n} U_i \mid \forall_{1 \le i \le n}.\ U_i \in \mathcal{B}(\mathsf{Outcomes})\}). \tag{7.9}$$

In other words, $\mathcal{B}^n(\mathsf{Outcome})$ is the n-fold product sigma-algebra generated by open rectangles of Outcomes. Any $e \in \mathsf{Event}$ specifies a measurable set $U \in \mathcal{B}^n(\mathsf{Outcome})$ whose i^{th} coordinate $U_i ::= \mathbb{E}\,[\![e]\!]\,x_i$ if $x_i \in (\mathit{vars}\, e)$ and $U_i ::= \mathsf{Outcomes}$ otherwise. Any Transform subexpression in e is solved and any Var that does not appear in e is marginalized, as illustrated in the next example.

Example 7.3. Let $\{\texttt{X}, \texttt{Y}, \texttt{Z}\}$ be elements of Var. Then

$$e ::= \texttt{Reciprocal(Id(X)) in ((\#f 1) (2 \#f))}$$

corresponds to the $\mathcal{B}^3(\mathsf{Outcome})$-measurable set

$$\{(\downarrow^{\mathsf{Real}}_{\mathsf{Outcome}} r) \mid 1/2 \le r \le 1\} \times \mathsf{Outcomes} \times \mathsf{Outcomes}. \quad «$$

As in Remark 7.1, mappings from Event to $[0, 1]$ induce probability distributions on $\mathcal{B}^n(\mathsf{Outcome})$.

Primitive Distributions Listing 7.2d presents the primitive distributions out of which multivariate distributions are constructed. The CDF domain contains cumulative distribution functions F, whose quantile function is denoted F^{-1} and derivative F'. From Billingsley [1995, Theorems 12.4, 14.1], (C1) Every leaf subexpression must have its label map to the identity transform

$$\forall\,\texttt{Leaf(}x\ d\ \sigma\texttt{)} \in \mathsf{Leaf}.\, x \in \sigma, \sigma(x) = \texttt{Id(}x\texttt{)}.$$

(C2) There are no cyclic dependencies between transformed variables

$$\begin{aligned}\forall\,\texttt{Leaf(}x\ d\ \sigma\texttt{)} \in \mathsf{Leaf}.\, \mathrm{dom}(\sigma) = \{x, x_1, \dots, x_m\} &\text{ for some } m > 0 \\ \implies \forall_{1\le t\le m}.\ (\mathit{vars}\ \sigma(x_t)) &\subset \{x, x_1, \dots, x_{t-1}\}.\end{aligned}$$

(C3) Children of sum nodes must contain identical variables

$$\forall\texttt{(}S_1\, w_1\texttt{)} \oplus \cdots \oplus \texttt{(}S_m\, w_m\texttt{)} \in \mathsf{Sum}.\ \forall_{1\le i\le n}.\ (\mathit{scope}\, S_i) = (\mathit{scope}\, S_1).$$

(C4) Children of product nodes must contain distinct variables

$$\forall\texttt{(}S_1 \otimes \cdots \otimes S_m\texttt{)} \in \mathsf{Product}.\ \forall_{1\le i<j\le n}.\ (\mathit{scope}\, S_i) \cap (\mathit{scope}\, S_j) = \varnothing.$$

(C5) Children of sum nodes must have at least one positive weight

$$\forall\texttt{(}S_1\, w_1\texttt{)} \oplus \cdots \oplus \texttt{(}S_m\, w_m\texttt{)} \in \mathsf{Sum}.\ \exists_{1\le i\le n}.\ w_i > 0.$$

Listing 7.4: Five technical conditions required for elements of the SPE domain to be well defined.

the CDF domain is in 1-1 correspondence with the collection of all probability distributions and the collection of all random variables on Real. The Distribution domain specifies continuous real, atomic real (on the integers) and nominal distributions. The denotation $\mathbb{D}\,[\![d]\!]$ of a Distribution is a distribution on Outcomes. For example, $\texttt{DistR(}F\, r_1\, r_2\texttt{)}$ is the restriction of F to the positive measure interval $[r_1, r_2]$. The distributions specified by DistR and DistI can be simulated using a variant of the integral probability transform, as described in the following proposition.

Proposition 7.4. *Let F be a* CDF *and r_1, r_2 real numbers such that $F(r_1) < F(r_2)$. Let $U \sim \mathrm{Uniform}(F(r_1), F(r_2))$ and define the random variable $X ::= F^{-1}(U)$. Then for all real numbers r,*

$$\widetilde{F}(r) ::= \Pr[X \le r] = \begin{cases} 0 & \textit{if } r < r_1 \\ \dfrac{F(r) - F(r_1)}{F(r_2) - F(r_1)} & \textit{if } r_1 \le r \le r_2 \\ 1 & \textit{if } r_2 < r. \end{cases} \qquad «$$

Proof. Immediate from $\Pr[X \le r] = \Pr[U \le F(r)]$ and the uniformity of U on $[r_1, r_2]$. ■

Sum-Product Expressions Listings 7.2e and 7.2f show the probability density and distribution semantics of the SPE domain. An element of SPE is well defined if and only if it satisfies the conditions in Listing 7.4. For Leaf, (C1) ensures that σ maps the leaf variable x to the Identity Transform and (C2) ensures there are no cyclic dependencies or undefined variables in Environment σ. Condition (C3) ensures the scopes of all children of a Sum are identical and (C4) ensures the scopes of all children of a Product are disjoint. which together ensure completeness and decomposability from sum-product networks [Poon and Domingos, 2011, Definitions 4 and 6].

In Listing 7.2f, the denotation $\mathbb{P}\,[\![S]\!]$ of $S \in$ SPE is a map from $e \in$ Event to its probability under the n-dimensional distribution defined by S, where $n ::= |\mathit{scope}\, S|$ is the number of variables in S. A terminal node $\texttt{Leaf(}x\, d\, \sigma\texttt{)}$ is comprised of $x \in$ Var, $d \in$ Distribution, and $\sigma \in$ Environment, where σ maps other variables to a $\mathsf{Transform}$ of x. An example mapping in σ is $\mathtt{Z} \mapsto \mathtt{Poly(Root(Id(X)\ 2)}\ [11,5])$.

When assessing the probability of e at a Leaf, *subsenv* (Listing 7.12 in Appendix 7.B) rewrites e as an Event e' on the base variable x. Therefore, probability of the $\mathsf{Outcomes}$ that satisfy e is precisely $\mathbb{D}\,[\![d]\!]\,(\mathbb{E}\,[\![e']\!]\,x)$. The *scope* function (Listing 7.11 in Appendix 7.B) returns the list of variables in S. For a Sum, the probability of e is a weighted average of the probabilities under each subexpression. For a $\mathsf{Product}$, the semantics are defined in terms of $(\mathit{dnf}\ e)$ (Listing 7.14 in Appendix 7.B), leveraging inclusion-exclusion.

In Listing 7.2e, the denotation $\mathbb{P}_0\,[\![S]\!]$ defines the density semantics of SPE, which correctly handles measure zero events such as $\{X = 3, Y = \pi, Z = \texttt{"foo"}\}$ under a mixed-type base measure that has both discrete and continuous components, as in Section 7.1.1. These semantics, which define the density as a pair, adapt the approximate inference “lexicographic likelihood-weighting” algorithm for discrete-continuous Bayes Nets [Wu et al., 2018] to exact “lexicographic enumeration” inference for the SPE domain.

7.3 Conditioning Sum-Product Expressions on Events

This section describes the main theoretical result for exact inference on probability distributions defined by an expression $S \in \mathsf{SPE}$ and presents the inference algorithm for conditioning on an Event (Listing 7.1d) in the core calculus.

Theorem 7.5 (Closure under conditioning)**.** *Let $S \in \mathsf{SPE}$ and $e \in \mathsf{Event}$ be given, where $\mathbb{P}\,[\![S]\!]\,e > 0$. There exists an algorithm which, given S and e, returns $S' \in \mathsf{SPE}$ such that, for all $e' \in \mathsf{Event}$, the probability of e' according to S' is equal to the conditional probability of e' given e according to S, i.e.,*

$$\mathbb{P}\,[\![S']\!]\,e' \equiv \mathbb{P}\,[\![S]\!]\,(e' \mid e) ::= \frac{\mathbb{P}\,[\![S]\!]\,(e \sqcap e')}{\mathbb{P}\,[\![S]\!]\,e}. \tag{7.10}$$

«

Theorem 7.5 is a structural conjugacy property [Diaconis and Ylvisaker, 1979] for the family of probability distributions defined by the SPE domain, where both the prior and posterior are elements of SPE. The closure property (7.10) is established constructively, using an algorithm *condition* : $\mathsf{SPE} \to \mathsf{Event} \to \mathsf{SPE}$ that satisfies

$$\mathbb{P}\,[\![(\mathit{condition}\,S\,e)]\!]\,e' = \mathbb{P}\,[\![S]\!]\,(e' \mid e) \qquad\qquad (e, e' \in \mathsf{Event}, \mathbb{P}\,[\![S]\!]\,e > 0). \tag{7.11}$$

The proof is given in Appendix 7.D. Figure 7.4 shows a conceptual example of how *condition* works, where the prior distribution is a $\mathsf{Product}$ S and the conditioned distribution is a $\mathsf{Sum\text{-}of\text{-}Product}$ S’.

Remark 7.6. Theorem 7.5 refers to a positive probability Event e. As with sum-product networks, SPE is also closed under conditioning on a single $\mathsf{Conjunction}$ of possibly measure zero equality constraints on non-transformed variables, which appear in many PPL interfaces [Saad and Mansinghka, 2016a, Cusumano-Towner et al., 2019, Molina et al., 2020]. Appendix 7.D.3 presents the $\mathit{condition}_0$ algorithm for conditioning on such events, leveraging the generalized mixed-type density semantics in Listing 7.2e.

«

The next result states a sufficient requirement for $(\mathit{condition}\,S\,e)$ to scale linearly in the size of S, which holds for both zero and positive measure events.

Theorem 7.7. *The runtime of $(\mathit{condition}\,S\,e)$ scales linearly in the number of nodes in the graph representing S whenever $e = (t_1\ \mathtt{in}\,v_1) \sqcap \cdots \sqcap (t_m\ \mathtt{in}\,v_m)$ is a single $\mathsf{Conjunction}$ of $\mathsf{Containment}$ constraints, where each t_i represents a non-transformed program variable.* «

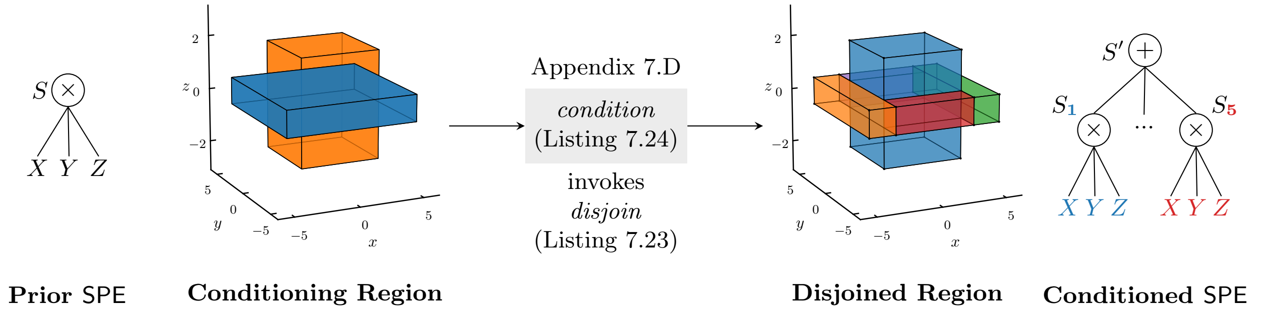


Figure 7.4: Conditioning a Product S on an Event e that defines a union of hyperrectangles in Real^3. The inference algorithm partitions the region into a disjoint union, in this case converting two overlapping regions into five disjoint regions. The result is a Sum-of-Product, where each child is the restriction of S to one of the disjoint hyperrectangles.

Example 7.8. Figure 7.5a shows an SPPL program that defines a pair of random variables (X, Z), where X is normally distributed; and $Z = -X^3 + X^2 + 6X$ if $X < 1$, otherwise $Z = 5\sqrt{X} + 1$. The first plot of Figure 7.5c shows the prior distribution of X; the middle plot shows the transformation t that defines $Z = t(X)$, which is a piecewise sum of t_{if} and t_{else}; and the final plot shows the distribution of $Z = t(X)$. Figure 7.5b shows the sum-product expression representing this program, where the root node is a sum whose left and right children have weights 0.691... and 0.309..., which corresponds to the prior probabilities of $\{X < 1\}$ and $\{1 \le X\}$. Nodes labeled $X \sim N(\mu, \sigma)$ with an incoming directed edge from a node labeled (r_1, r_2) denotes that the random variable is constrained to the interval (r_1, r_2). Deterministic transformations are denoted using red directed edges from a leaf node (i.e., X) to a numeric expression (e.g., $5\sqrt{X} + 11$), with the name of the transformed variable along the edge (i.e., Z). In Figure 7.5b, the environments at the leaves in the left and right subtrees are thus:

$$\sigma_{\text{left}} = \{\texttt{X} \mapsto \texttt{Id(X)},\ \texttt{Z} \mapsto \texttt{Poly(Id(X)}\ [0, 6, 1, -1]\texttt{)}\} \tag{7.12}$$

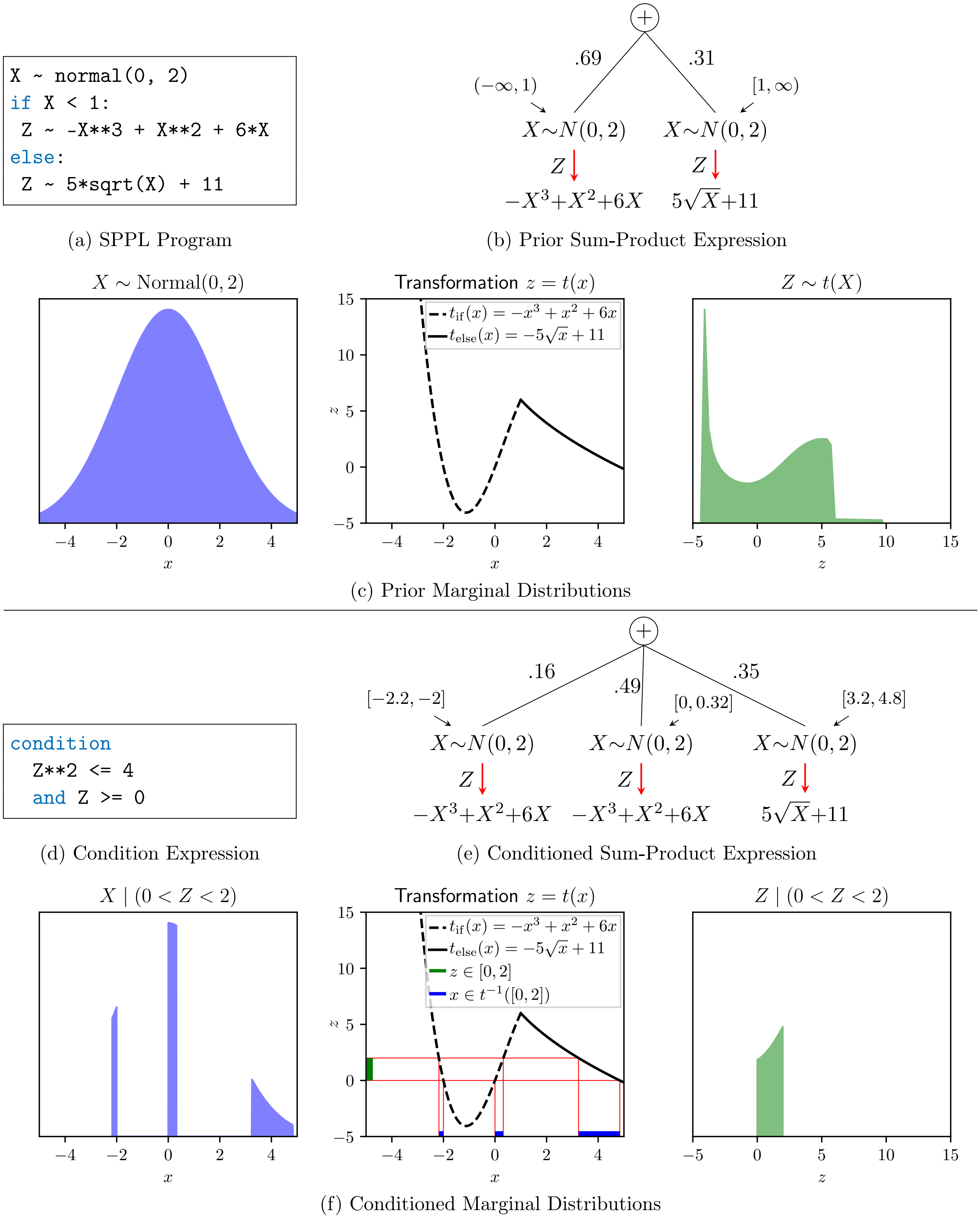
$$\sigma_{\text{right}} = \{\texttt{X} \mapsto \texttt{Id(X)},\ \texttt{Z} \mapsto \texttt{Poly(Root(Id(X) 2)}\ [11, 5]\texttt{)}\}. \tag{7.13}$$

Figure 7.5c shows the prior distribution induced by this program and Figure 7.5d shows an SPPL query that conditions the program on an event $\{Z^2 \le 4\} \cap \{Z \ge 0\}$ involving the transformed variable Z. The inference engine performs the following analysis on the query:

$$\begin{aligned}
&\{Z^2 \le 4\} \cap \{Z \ge 0\} && && (7.14)\\
&\equiv \{Z \in [0, 2]\} && \text{(simplifying the event)} && (7.15)\\
&\equiv \{X \in t^{-1}([0, 2])\} && \text{(using } Z ::= t(X)\text{)} && (7.16)\\
&\equiv \{X \in t_{\text{if}}^{-1}([0, 2])\} \cup \{X \in t_{\text{else}}^{-1}([0, 2])\} && \text{(inverting the event)} && (7.17)\\
&\equiv \underbrace{\{-2.174... \le X \le -2\} \cup \{0 \le X \le .321...\}}_{\text{constraints from left subtree}} \cup \underbrace{\{81/25 \le X \le 121/25\}}_{\text{constraint from right subtree}} && && (7.18)
\end{aligned}$$

Eq. (7.15) shows the first stage of inference, which solves all transformations in the conditioning event and yields $\{0 \le Z \le 2\}$. The conditional distribution of Z is shown in the final plot of Figure 7.5f. The next step is to dispatch the simplified event to the left and right subtrees. Each subtree will compute the constraint on X implied by the event under the transformation in that branch, as shown in Eq. (7.17). The middle plot of Figure 7.5f shows the preimage computation under t_{if} from the left subtree, which gives two intervals, and t_{else} from the right subtree, which gives one interval.

The final step is to transform the prior expression (Figure 7.5b) by conditioning each subtree on



```
X ~ normal(0, 2)
if X < 1:
 Z ~ -X**3 + X**2 + 6*X
else:
 Z ~ 5*sqrt(X) + 11
```

(a) SPPL Program

(b) Prior Sum-Product Expression

(c) Prior Marginal Distributions

```
condition
  Z**2 <= 4
  and Z >= 0
```

(d) Condition Expression

(e) Conditioned Sum-Product Expression

(f) Conditioned Marginal Distributions

Figure 7.5: Inference on a stochastic many-to-one transformation of a real random variable in SPPL.

$$\begin{gathered}
x \in \mathsf{Var} \quad y \in \mathsf{ArrayVar} \quad n \in \mathsf{Natural} \quad b \in \mathsf{Boolean} \quad r \in \mathsf{Real} \quad s \in \mathsf{String} \\
o_{\mathrm{arith}} \in \{\texttt{+}, \texttt{-}, \texttt{*}, \texttt{/}, \texttt{**}\} \quad o_{\mathrm{bool}} \in \{\texttt{and}, \texttt{or}\} \quad o_{\mathrm{neg}} \in \{\texttt{not}\} \quad o_{\mathrm{rel}} \in \{\texttt{<=}, \texttt{<}, \texttt{>}, \texttt{>=}, \texttt{==}, \texttt{in}\}
\end{gathered}$$

$$\begin{aligned}
D &\in \{\texttt{normal}, \texttt{poisson}, \texttt{choice}, \texttt{bernoulli}, \dots\} \\
E &\in \mathsf{Expr} ::= x \mid n \mid b \mid r \mid s \mid y\texttt{[}E\texttt{]} \mid D\texttt{(}E^*\texttt{)} \mid \texttt{(}E_1, \dots, E_m\texttt{)} \\
&\qquad \mid E_1\, o_{\mathrm{arith}}\, E_2 \mid o_{\mathrm{neg}}\, E \mid E_1\, o_{\mathrm{bool}}\, E_2 \mid E_1\, o_{\mathrm{rel}}\, E_2 \\
C &\in \mathsf{Command} ::= x \texttt{=} E \mid y\texttt{[}E_1\texttt{]} \texttt{=} E_2 \mid x \texttt{~} E \mid y\texttt{[}E_1\texttt{]} \texttt{~} E_2 \mid y \texttt{=} \texttt{array}\texttt{(}E\texttt{)} \\
&\qquad\qquad \mid \texttt{skip} \mid C_1 \texttt{;} C_2 \mid \texttt{if}\ E\ \texttt{\{}C_1\texttt{\}}\ \texttt{else}\ \texttt{\{}C_2\texttt{\}} \mid \texttt{condition}\texttt{(}E\texttt{)} \\
&\qquad\qquad \mid \texttt{for}\ x\ \texttt{in}\ \texttt{range}\texttt{(}E_1, E_2\texttt{)}\ \texttt{\{}C\texttt{\}} \mid \texttt{switch}\ x_1\ \texttt{cases}\ \texttt{(}x_2\ \texttt{in}\ E\texttt{)}\ \texttt{\{}C\texttt{\}}
\end{aligned}$$

Listing 7.5: Source syntax of SPPL.

the intervals in Eq. (7.18), which gives the posterior expression (Figure 7.5e). The left subtree in Figure 7.5b, which originally corresponded to $\{X < 1\}$, is split in Figure 7.5e into two subtrees that represent the events $\{-2.174... \le X \le -2\}$ and $\{0 \le X \le 0.321...\}$, respectively, and whose weights 0.159... and 0.494... are the renormalized probabilities of these regions under the prior distribution in Figure 7.5c. The right subtree in Figure 7.5b, which originally corresponded to $\{1 \le X\}$, is now restricted to $\{81/25 \le X \le 121/25\}$ in Figure 7.5e and its weight 0.347... is again the renormalized prior probability of the region. The graph in Figure 7.5e represents the distribution of (X, Z) conditioned on the query in Eq. (7.15). The new sum-product expression be used to run further queries, such as using `simulate` to generate random samples from the posterior distributions in Figure 7.5f. «

7.4 Translating Probabilistic Programs to Sum-Product Expressions

Probabilistic programs in SPPL are translated to sum-product expressions $S \in \mathsf{SPE}$ that symbolically represent the probability distribution over all program variables. As in Figure 7.1, the translated expression S is used to answer queries:

`simulate`$(x_1, \dots, x_n)$ using standard sampling semantics of Sum and Product from sum-product networks; the technique from Proposition 7.4 for constrained simulation at the Leaf nodes; and syntactic replacement for transformed variables.

`prob`(e) using the distribution semantics $\mathbb{P}\,[\![S]\!]\,e$ from Listing 7.2f or density semantics $\mathbb{P}_0\,[\![S]\!]\,e$ from Listing 7.2e, depending on the form of $e \in \mathsf{Event}$.

`condition`(e) using the *condition* operation (7.11) from Theorem 7.5.

Listing 7.5 shows the source syntax of SPPL, which contains standard constructs of an imperative language such as `array` data structures, `if`-`else` statements, and bounded `for` loops. The `switch`-`case` macro is defined in Eq. (7.4). Random variables are defined using "sample" (`~`) commands. The `condition`(E) command is used to constrain program executions to those for which $E \in \mathsf{Expr}$ evaluates to `#t`. Listing 7.6 defines a formal relation $\langle C, S\rangle \to_{\mathsf{SPE}} S'$ that translates the "current" expression $S \in \mathsf{SPE}$ and program fragment $C \in \mathsf{Command}$ into $S' \in \mathsf{SPE}$. The $\Downarrow$ relation evaluates $E \in \mathsf{Expr}$ to terminal values in other domains in the core calculus, using standard programming evaluation for arithmetic and logical expressions.

To ensure SPPL programs translate to well-defined elements of SPE as defined in (C1)–(C5), each program must satisfy the following syntactic properties:

$$\frac{E \Downarrow d;\, \text{where } x \notin \mathit{scope}\, S}{\langle x \sim E, S\rangle \to_{\mathsf{SPE}} S \otimes (x\ d\ \{x \mapsto \texttt{Id}(x)\})} \qquad (\textsc{Sample})$$

$$\frac{E \Downarrow t; \quad \text{where } \mathit{vars}\, t \in \mathrm{dom}(\sigma), x \notin \mathrm{dom}(\sigma)}{\langle x = E, \texttt{Leaf}(x'\ d\, \sigma)\rangle \to_{\mathsf{SPE}} \texttt{Leaf}(x'\ d\ (\sigma \cup \{x \mapsto t\}))} \qquad (\textsc{Transform-Leaf})$$

$$\frac{E \Downarrow t, \forall_{1 \leq i \leq m}.\langle x = E, S_i\rangle \to_{\mathsf{SPE}} S'_i}{\langle x = E, \oplus_{i=1}^{m} (S_i\, w_i)\rangle \to_{\mathsf{SPE}} \oplus_{i=1}^{m} (S'_i\, w_i)} \qquad (\textsc{Transform-Sum})$$

$$\frac{E \Downarrow t, \langle x = E, S_j\rangle \to_{\mathsf{SPE}} S'_j; \quad \text{where } j ::= \min\{i \mid (\mathit{vars}\, E) \in \mathit{scope}\, S_i\}}{\langle x = E, \otimes_{i=1}^{m} S_i\rangle \to_{\mathsf{SPE}} \otimes_{i=1, i \neq j}^{m} S_i \otimes S'_j} \qquad (\textsc{Transform-Prod})$$

$$\frac{E \Downarrow e}{\langle \texttt{condition}(E), S\rangle \to_{\mathsf{SPE}} \mathit{condition}\, S\, e} \qquad (\textsc{Condition})$$

$$\frac{\langle C_1, S\rangle \to_{\mathsf{SPE}} S_1, \langle C_2, S_1\rangle \to_{\mathsf{SPE}} S'}{\langle C_1 ; C_2, S\rangle \to_{\mathsf{SPE}} S'} \qquad (\textsc{Sequence})$$

$$\frac{E \Downarrow e, \langle C_1, \mathit{condition}\, S\, e\rangle \to_{\mathsf{SPE}} S_1, \langle C_2, \mathit{condition}\, S\, (\mathit{negate}\, e)\rangle \to_{\mathsf{SPE}} S_2}{\langle \texttt{if}\ E\ \{C_1\}\ \texttt{else}\ \{C_2\}, S\rangle \to_{\mathsf{SPE}} (S_1\, \mathbb{P}[\![S]\!]\, e) \oplus (S_2\, (1 - \mathbb{P}[\![S]\!]\, e))} \qquad (\textsc{If-Else})$$

$$\frac{E_1 \Downarrow n_1, E_2 \Downarrow n_2;\, \text{where } n_2 \leq n_1}{\langle \texttt{for}\ x\ \texttt{in}\ \texttt{range}(E_1, E_2)\ \{C\}, S\rangle \to_{\mathsf{SPE}} S} \qquad (\textsc{For-Exit})$$

$$\frac{E_1 \Downarrow n_1, E_2 \Downarrow n_2;\, \text{where } n_1 < n_2}{\begin{array}{l}\langle \texttt{for}\ x\ \texttt{in}\ \texttt{range}(E_1, E_2)\ \{C\}, S\rangle \\ \qquad \to_{\mathsf{SPE}} \langle C[x/n_1]\,;\ \texttt{for}\ x\ \texttt{in}\ \texttt{range}(n_1 + 1, E_2)\ \{C\}, S\rangle\end{array}} \qquad (\textsc{For-Repeat})$$

Listing 7.6: Translating an SPPL command C (Listing 7.5) to an element of SPE (Listing 7.2f).

(a) Invalid program (infinite-sized SPE)

```
mu ~ beta(a=4, b=3, scale=7)
num_loops ~ poisson(mu)       # invalid
for i in range(0, num_loops): # invalid
  [... commands ... ]
```

(b) Valid program (finite-sized SPE)

```
mu ~ beta(a=4, b=3, scale=7)
# binspace partitions [0,7] into 10 intervals
switch (mu) cases (m in binspace(0, 7, n=10)):
  num_loops ~ poisson(m.mean()) # discretization
condition (num_loops < 50)      # truncation
switch num_loops cases (n in range(50)):
  for i in range(0, n):
    [... commands ... ]
```

Listing 7.7: Examples of valid and invalid SPPL programs.

(R1) Variables defined by $x \sim E$ or $x = E$ must have fresh (new) names, to ensure (C1), (C2) and (C4) hold. The programs `X ~ normal(0,1); X ~ normal(0,2)` and `X ~ normal(0,1); X = X + 1` are thus forbidden.

(R2) Branches in an `if`-`else` statement must define identical variables, to ensure (C3) and (C5) hold. The program `if (X < 0) {Y ~ normal(0,1)} else {Z ~ normal(0,1)}` is thus forbidden.

(R3) Derived random variables are obtained via transformation of a single variable only. The program `X = Z + Y` is thus forbidden.

(R4) Parameters of distributions D or `range` must be either constants or random variables with finite support. The program `X ~ poisson(2); Y ~ normal(X, 1)` is thus forbidden.

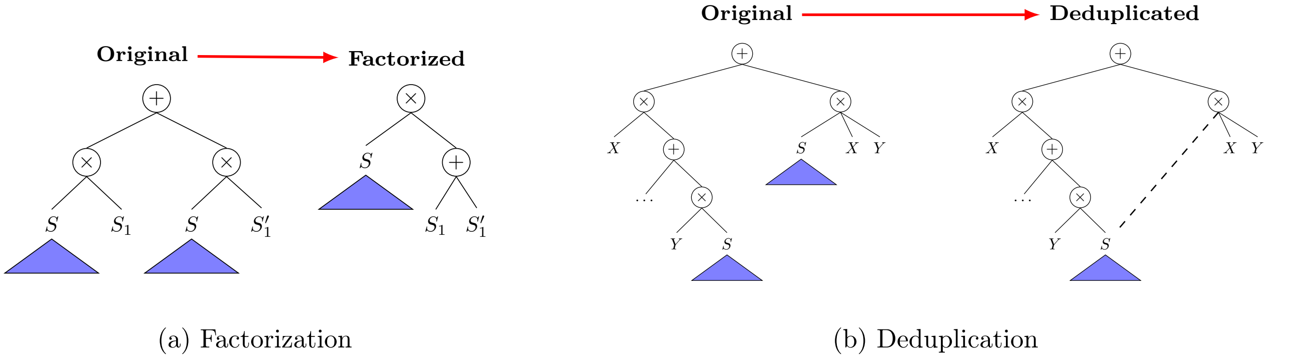
Restriction (R3) ensures that the program has a valid representation in SPE, since the Transform domain from the core calculus in Listing 7.1 does not contain multivariate transformations. Multivariate transformations are in general intractable and their distributions do not factor as Sum and Product expressions. Such transformations are also semantically ill-defined, even in superficially simple cases [Proschan and Presnell, 1998]. Restriction (R4) ensures that the translated SPE has finite-size, since distributional parameters with infinite support require integrals (uncountable support) or infinite series (countable support) to represent exactly. Listing 7.7a shows an example of an invalid SPPL program which is forbidden because the `mu` beta random variable is used to parameterize the mean of the `num_loops` Poisson random variable and the `num_loops` random variable is unbounded and used in the `for` loop. Listing 7.7b shows one modification of the model to work around these restrictions, where the `switch`-`cases` (defined in Eq. (7.4)) discretizes `mu` and `condition` sets an upper bound on `num_loops`.

7.4.1 Compiler Optimizations of Memory and Runtime

Are All SPPL Programs Tractable? No. As discrete Bayesian networks can be encoded as SPPL programs, inference queries are NP-Hard [Cooper, 1990]. Such programs correspond to the compiled SPE being exponentially large. For example, an SPPL program for an noisy-OR network over Boolean variables [Pearl, 1988] can be compactly specified, but exact inference scales exponentially in the number of variables. Static analysis techniques that report whether exact inference in an SPPL program is tractable are not possible either. The complexity of exact inference in Bayesian networks scales exponentially in the treewidth, which is the only structural restriction that can ensure tractability [Chandrasekeran et al., 2008]. As computing treewidth is NP-Complete [Arnborg et al., 1987], for fundamental theoretical reasons it is impossible to verify whether SPPL programs, even simple ones, admit tractable inference. However, many models contain conditional independence relationships [Koller and Friedman, 2009] that induce a compact factorization of the model into tractable subparts, as in Section 7.1.2. SPPL uses several optimization techniques to improve scalability of translation (Listing 7.6)

Table 7.1: Measurements of SPE graph size with and without the factorization and deduplication.

Benchmark	**Number of Nodes in Translated SPE**		**Data Compression Ratio (unopt/opt)**
	Unoptimized	Optimized	
Hiring [Albarghouthi et al., 2017]	33	27	1.2
Alarm [Nori et al., 2014]	58	45	1.3
Grass [Nori et al., 2014]	130	59	2.2
NoisyOR [Nori et al., 2014]	783	132	4.1
ClinicalTrial [Nori et al., 2014]	43761	4131	10.6
HeartDisease [Spiegelhalter et al., 1993]	1041235	6257	166.4
Hierarchical HMM (Section 7.1.2)	2.92×10^{16}	1787	1.64×10^{14}



(a) Factorization

(b) Deduplication

Figure 7.6: Exploiting independencies and repeated structure during translation of SPPL programs to build compact sum-product expressions. Blue subtrees are identical components.

and conditioning (Eq. (7.11)) by exploiting independencies and repeated structure, when they exist, to build compact sum-product expressions. Three optimizations are discussed next.

Factorization Using standard algebraic manipulations, a sum-product expression can be made smaller without changing its semantics by "factoring out" common terms, provided that the new expression satisfies (C1)–(C5). Factorization plays a central role in optimizing the (IF-ELSE) rule of $\to_{\mathsf{SPE}}$ in Listing 7.6: since all statements before the `if-else` statement are shared by the bodies of the `if` and `else` branches, statements outside the branch that are independent of statements inside the branch typically produce subexpressions that can be factored out.

Deduplication When a sum-product expression contains duplicate subexpressions that cannot be factored out without violating the definedness conditions, the duplicates can be represented using a single physical data structure. Figure 7.6b shows an example where the left and right components of the original expression contain an identical subexpression S (in blue), but factorization would lead to an invalid sum-product expression. The optimizer represents the computation graph of this expression using a single data structure S shared by the left and right subtrees (see also Figures 7.3c and 7.3d).

Memoization While deduplication reduces memory overhead, memoization reduces runtime overhead. Consider either SPE in Figure 7.6b: calling *condition* on the Sum root will dispatch the query to the left and right subexpressions. The values (*condition* S e) or $\mathbb{P}\,[\![S]\!]\,e$ are cached when S is visited by the parent in the left subtree, to avoid recomputing the value when S is visited again by the parent in the right subtree. Memoization delivers large runtime gains not only for solving queries but also for detecting duplicates returned by *condition* in the (IF-ELSE) translation step.

Table 7.2: Runtime measurements and speedup for 15 fairness verification tasks using SPPL, FairSquare [Albarghouthi et al., 2017], VeriFair [Bastani et al., 2019], and PSI [Gehr et al., 2016].

Decision Program	**Population Model**	**Lines of Code**	**Fairness Result**	**Wall-Clock Runtime (seconds)**				**SPPL Speedup Factor**	
				FairSquare	VeriFair	PSI	SPPL	Min	Max
DT_4	Independent	15	Unfair	1.4	16.0	22.7	0.01	**140x**	**2270x**
	Bayes Net. 1	25	Unfair	2.5	1.27	61.2	0.03	**40x**	**2040x**
	Bayes Net. 2	29	Unfair	6.2	0.91	1783	0.03	**30x**	**594500x**
DT_{14}	Independent	32	Fair	2.7	105	11.9	0.03	**90x**	**3500x**
	Bayes Net. 1	46	Fair	15.5	152	118	0.07	**221x**	**2171x**
	Bayes Net. 2	50	Fair	70.1	151	2069	0.08	**876x**	**25863x**
DT_{16}	Independent	36	Fair	4.1	13.6	57.7	0.03	**136x**	**1923x**
	Bayes Net. 1	49	Unfair	12.3	1.58	144	0.08	**20x**	**1800x**
	Bayes Net. 2	53	Unfair	30.3	2.02	2342	0.08	**25x**	**29275x**
$\mathrm{DT}^{\alpha}_{16}$	Independent	62	Fair	5.1	2.01	79.3	0.06	**35x**	**1322x**
	Bayes Net. 1	58	Fair	15.4	21.6	191	0.12	**127x**	**1591x**
	Bayes Net. 2	45	Fair	53.8	24.5	2284	0.12	**204x**	**29033x**
DT_{44}	Independent	93	Fair	15.6	23.1	15.2	0.05	**303x**	**644x**
	Bayes Net. 1	109	Unfair	264.1	19.8	68.3	0.09	**220x**	**2933x**
	Bayes Net. 2	113	Unfair	timeout	20.1	1651	0.09	**233x**	**18344x**

Measurements Table 7.1 shows measurements of performance gains delivered by the factorization and deduplication optimizations on seven benchmark programs. The compression ratio, which is the unoptimized size divided by the optimized size is highest in the presence of independence or repeated structure. In addition, deduplication and memoization together enable fast detection of duplicate subtrees by comparing logical memory addresses of internal nodes in $O(1)$ time, instead of computing hash functions that require an expensive subtree traversal.

7.5 Evaluation

The prototype implementation of SPPL (Section 1.4) was evaluated on benchmark problems from the literature. Section 7.5.1 compares the runtime for verifying fairness properties of decision trees using SPPL to three solvers: FairSquare [Albarghouthi et al., 2017], VeriFair [Bastani et al., 2019], and PSI [Gehr et al., 2016]. Section 7.5.2 compares the runtime of conditioning and querying probabilistic programs using SPPL to PSI, which is a state-of-the-art solver for exact symbolic inference. Section 7.5.3 compares the runtime of computing exact rare event probabilities in SPPL to sampling-based estimation in BLOG [Milch et al., 2005]. Experiments were run on Intel i7-8665U 1.9GHz CPU with 16GB RAM.

7.5.1 Fairness Benchmarks

Characterizing the fairness of classification algorithms is a growing application area in machine learning [Dwork et al., 2012]. Albarghouthi et al. [2017] precisely cast the problem of verifying the fairness of a classifier in terms of computing ratios of conditional probabilities in a pair of probabilistic programs that specify the data generating and classification processes. Briefly, if (i) D is a decision program that classifies whether applicant A should be hired; (ii) H is a population program that generates random applicants; and (iii) ϕ_{m} (resp. ϕ_{q}) is a predicate on A that is true if the applicant is a minority (resp.

qualified), then D is said to be ϵ-fair on H (where $\epsilon > 0$) if

$$\frac{\Pr_{A\sim H}\left[D(A) \mid \phi_{\mathrm{m}}(A) \wedge \phi_{\mathrm{q}}(A)\right]}{\Pr_{A\sim H}\left[D(A) \mid \neg\phi_{\mathrm{m}}(A) \wedge \phi_{\mathrm{q}}(A)\right]} > 1 - \epsilon. \tag{7.19}$$

Eq. (7.19) is a "group fairness" property comparing the probability of hiring a qualified minority applicant is to that of hiring a qualified non-minority applicant.

This evaluation compares the runtime needed by SPPL to obtain a fairness judgment (7.19) in a benchmark set of machine-learned decision and population programs against the FairSquare [Albarghouthi et al., 2017], VeriFair [Bastani et al., 2019], and PSI solvers. Fifteen benchmark problems for decision tree classifiers are adapted from Albarghouthi et al. [2017, Section 6.1], which are one-third of the full benchmark set. SPPL cannot express neural networks and support-vector machines because they contain multivariate transforms which do not have exact tractable solutions and are ruled out by the SPPL restriction (R3). FairSquare and VeriFair are able to solve these benchmarks using approximate inference.

Table 7.2 shows the runtime results. The first column shows the classification program, where DT_n means "decision tree" with n conditionals. The second column shows the population model used to generate data. The third column shows the lines of code (in SPPL). The fourth column shows the result of the fairness analysis, where all four systems produce the same judgment on the fifteen benchmarks. The remaining columns show the runtime and speedup factors. The measurements indicate that SPPL consistently delivers solutions in milliseconds, whereas the baselines can require tens, hundreds, or thousands of seconds. The SPPL speedup factors (20x–$594,500$x) are substantial. Furthermore, the runtime variance in the FairSquare, VeriFair, and PSI baselines is very high. For example, VeriFair uses rejection sampling to estimate Eq. (7.19) with a stopping rule to determine when the estimate is close enough, leading to unpredictable runtime (e.g., >100 seconds for DT_{14} but <1 second for DT_4, Bayes Net. 2). FairSquare, which uses symbolic volume computation and hyperrectangle sampling to approximate Eq. (7.19), is faster than VeriFair in some cases (e.g., DT_{14}), but times out in others (DT_{44}, Bayes Net. 2). PSI uses computer algebra whose scaling is extremely unpredictable, ranging from 11.9 seconds to 2284 seconds. In contrast, the runtime is SPPL does not vary significantly across the benchmark problems. The performance–expressiveness trade-off here is that SPPL computes exact probabilities and is substantially faster in verifying decision trees. FairSquare and VeriFair compute approximate probabilities that enable them to express more fairness problems, at the cost of a higher and less predictable runtime.

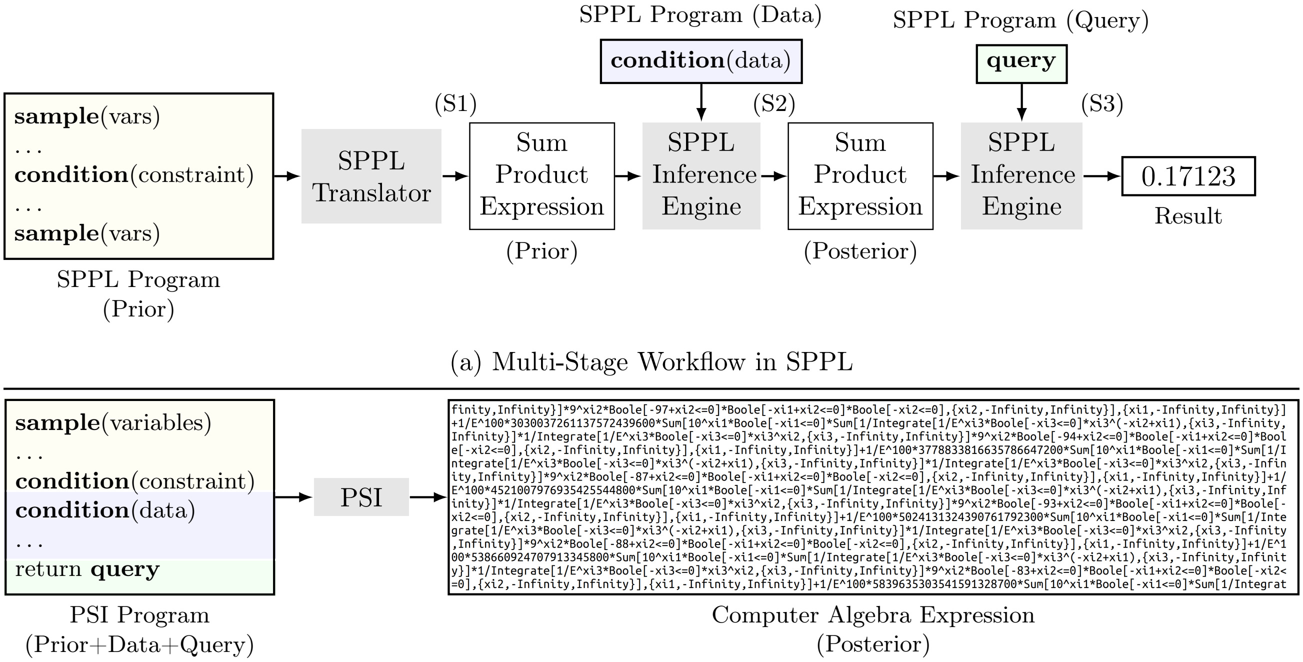
7.5.2 Comparison to Symbolic Integration

The second evaluation compares SPPL to the PSI [Gehr et al., 2016] symbolic solver on benchmark problems that include discrete, continuous, and transformed random variables. PSI can express more inference problems than SPPL because it uses general computer algebra and does not impose restrictions (R3) and (R4). As a result, SPPL can solve 14/21 benchmarks listed in Gehr et al. [2016, Table 1].

Workflow Comparison In SPPL, the multi-stage modeling and inference workflow demonstrated in Figure 7.7a involves three steps that reflect the key elements of a Bayesian inference problem:

(S1) Translating the model program into a prior SP S.
(S2) Conditioning S on data to obtain a posterior SP S'.
(S3) Querying S', using, e.g., `prob` or `simulate`.

An advantage of this multi-stage workflow is that multiple tasks can be run at a given stage without rerunning previous stages. For example, multiple datasets can be observed in (S2) without translating the prior expression in (S1) once per dataset; and, similarly, multiple queries can be run in (S3) without



(a) Multi-Stage Workflow in SPPL

```
finity,Infinity}]*9^xi2*Boole[-97+xi2<=0]*Boole[-xi1+xi2<=0]*Boole[-xi2<=0],{xi2,-Infinity,Infinity}],{xi1,-Infinity,Infinity}]
+1/E^100*3030037261137572439600*Sum[10^xi1*Boole[-xi1<=0]*Sum[1/Integrate[1/E^xi3*Boole[-xi3<=0]*xi3^(-xi2+xi1),{xi3,-Infinity,
Infinity}]*1/Integrate[1/E^xi3*Boole[-xi3<=0]*xi3^xi2,{xi3,-Infinity,Infinity}]*9^xi2*Boole[-94+xi2<=0]*Boole[-xi1+xi2<=0]*Bool
e[-xi2<=0],{xi2,-Infinity,Infinity}],{xi1,-Infinity,Infinity}]+1/E^100*3778833816635786647200*Sum[10^xi1*Boole[-xi1<=0]*Sum[1/I
ntegrate[1/E^xi3*Boole[-xi3<=0]*xi3^(-xi2+xi1),{xi3,-Infinity,Infinity}]*1/Integrate[1/E^xi3*Boole[-xi3<=0]*xi3^xi2,{xi3,-Infin
ity,Infinity}]*9^xi2*Boole[-87+xi2<=0]*Boole[-xi1+xi2<=0]*Boole[-xi2<=0],{xi2,-Infinity,Infinity}],{xi1,-Infinity,Infinity}]+1/
E^100*4521007976935425544800*Sum[10^xi1*Boole[-xi1<=0]*Sum[1/Integrate[1/E^xi3*Boole[-xi3<=0]*xi3^(-xi2+xi1),{xi3,-Infinity,Inf
inity}]*1/Integrate[1/E^xi3*Boole[-xi3<=0]*xi3^xi2,{xi3,-Infinity,Infinity}]*9^xi2*Boole[-93+xi2<=0]*Boole[-xi1+xi2<=0]*Boole[-
xi2<=0],{xi2,-Infinity,Infinity}],{xi1,-Infinity,Infinity}]+1/E^100*5024131324390761792300*Sum[10^xi1*Boole[-xi1<=0]*Sum[1/Inte
grate[1/E^xi3*Boole[-xi3<=0]*xi3^(-xi2+xi1),{xi3,-Infinity,Infinity}]*1/Integrate[1/E^xi3*Boole[-xi3<=0]*xi3^xi2,{xi3,-Infinity
,Infinity}]*9^xi2*Boole[-88+xi2<=0]*Boole[-xi1+xi2<=0]*Boole[-xi2<=0],{xi2,-Infinity,Infinity}],{xi1,-Infinity,Infinity}]+1/E^1
00*538660924707913345800*Sum[10^xi1*Boole[-xi1<=0]*Sum[1/Integrate[1/E^xi3*Boole[-xi3<=0]*xi3^(-xi2+xi1),{xi3,-Infinity,Infinit
y}]*1/Integrate[1/E^xi3*Boole[-xi3<=0]*xi3^xi2,{xi3,-Infinity,Infinity}]*9^xi2*Boole[-83+xi2<=0]*Boole[-xi1+xi2<=0]*Boole[-xi2<
=0],{xi2,-Infinity,Infinity}],{xi1,-Infinity,Infinity}]+1/E^100*5839635303541591328700*Sum[10^xi1*Boole[-xi1<=0]*Sum[1/Integrat
```

(b) Single-Stage Workflow in PSI

Figure 7.7: Comparison of multi-stage and single-stage workflows for exact probabilistic inference. In SPPL, modeling, observing data, and querying are separated into multiple distinct stages, which enables large efficiency gains from computation reuse across many datasets or queries. In PSI [Gehr et al., 2016], the workflow consists of a single stage that combines all these tasks into one large symbolic computation. Refer to Table 7.3 for performance comparison.

conditioning on data in (S2) once per query. In contrast, PSI adopts a single-stage workflow, shown in Figure 7.7b, where a single program contains the prior distribution over variables, "observe" (i.e., "condition") statements for conditioning on a dataset, and a "return" statement for the query. PSI converts the program into a symbolic expression for the distribution over the return value: if this expression is "complete" (i.e., it does not contain unevaluated symbolic integrals) then it can be used to obtain interpretable answers for plotting or tabulating; otherwise, the result is "partial" and is too complex to be used for practical purposes. A consequence of the single-stage workflow in systems like PSI is that the entire solution is recomputed from scratch on a per-dataset or per-query basis.

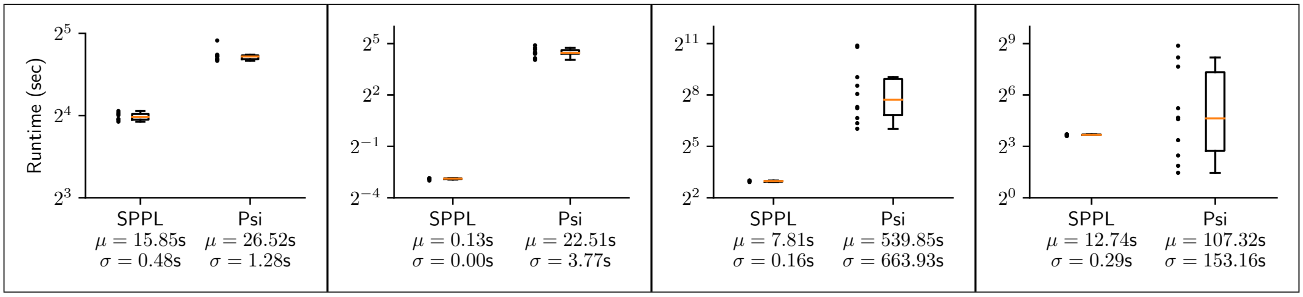
Runtime Comparison Table 7.3 compares the runtime of SPPL and PSI on seven benchmarks problems: Digit Recognition [Gehr et al., 2016]; TrueSkill [Laurel and Misailovic, 2020]; Clinical Trial [Gehr et al., 2016]; Gamma transforms (described below); Student Interviews [Laurel and Misailovic, 2020] (two variants); and Markov Switching (two variants, from Section 7.1.2); The second column shows the distributions in each benchmark, which include continuous, discrete, and transformed variables. The third column shows the number of datasets on which to condition the program. The next three columns show the time needed to translate the program (stage (S1)), condition the program on a dataset (stage (S2)), and query the posterior (stage (S3))—entries in the latter two columns are written as $n \times t$, where n is the number of datasets and t the average time per dataset. For PSI, (S1) and (S2) are implemented in a single stage, shown in the merged gray cell; and (S3) takes less than 0.01 seconds if the result is a simplified symbolic expression that can be evaluated to obtain a numeric answer, otherwise it times out. The last column shows the overall runtime for solving all n tasks.

For benchmarks that both systems solve completely, SPPL realizes speedups between 3x (Digit

Table 7.3: Comparison of PSI [Gehr et al., 2016] and SPPL on seven exact inference problems.

Benchmark	Distribution	Datasets	System	Wall-Clock Runtime of Inference Stages			Overall Time
				Translation (S1)	Conditioning (S2)	Querying (S3)	
Digit Recognition	$C\times B^{784}$	10	SPPL	6.9 sec	10 × 7.7 sec	10 × (<0.01 sec)	84 sec
			PSI	10 × 24.3 sec		10 × (<0.01 sec)	244 sec
TrueSkill	$P\times Bi^{2}$	2	SPPL	3.4 sec	2 × 0.7 sec	2 × 0.1 sec	4.9 sec
			PSI	2 × 41.60 sec		timeout	—
Clinical Trial	$B\times U^{3}$ $\times B^{50}\times B^{50}$	10	SPPL	9.5 sec	10 × 2.2 sec	10 × (<0.01 sec)	31 sec
			PSI	10 × 107.3 sec		10 × (<0.01 sec)	1073 sec
Gamma Transforms	$G\times T$ $\times(T+T)$	5	SPPL	0.02 sec	5 × 0.52 sec	5 × 0.03 sec	2.8 sec
			PSI	5 × 0.68 sec; i/e		timeout	—
Student Interviews$_2$	$P\times B^{2}\times Bi^{4}$ $\times(A+Be)^{2}$	10	SPPL	4.0 sec	10 × 0.7 sec	10 × 0.2 sec	13.5 sec
			PSI	10 × 540 sec		timeout	—
Student Interviews$_{10}$	$P\times B^{10}\times Bi^{20}$ $\times(A+Be)^{10}$	10	SPPL	24.6 sec	10 × 3.9 sec	10 × 1.2 sec	75 sec
			PSI	out of memory (64GB+)		—	—
Markov Switching$_3$	$B\times B^{3}$ $\times N^{3}\times P^{3}$	10	SPPL	0.05 sec	10 × (<0.01 sec)	10 × (<0.01 sec)	0.5 sec
			PSI	10 × 182.9 sec		10 × (<0.01 sec)	1829 sec
Markov Switching$_{100}$	$B\times B^{100}$ $\times N^{100}\times P^{100}$	10	SPPL	4.1 sec	10 × 6.5 sec	10 × 0.5 sec	74 sec
			PSI	out of memory (64GB+)		—	—

A:Atomic B:Bernoulli Be:Beta Bi:Binomial C:Categorical
N:Normal G:Gamma P: Poisson T: Transform U:Uniform



(a) Digit Recognition (b) Markov Switching$_3$ (c) Student Interviews$_2$ (d) Clinical Trial

Figure 7.8: Distribution of end-to-end runtime for four benchmark problems from Table 7.3 using SPPL and PSI. For each benchmark, one inference query is repeated over ten distinct datasets (dots).

Recognition) to 3600x (Markov Switching$_3$). The measurements also show the advantage of the multi-stage workflow in SPPL. For example, in TrueSkill, SPPL translation (3.4 seconds) is more expensive than both conditioning on data (0.7 seconds) and querying (0.1 seconds), which amortizes the translation cost over several datasets or queries. In PSI, solving TrueSkill takes 2 × 41.6 seconds, but the solution contains unsimplified integrals and is thus unusable. The Markov Switching and Student Interviews benchmarks show that PSI does not perform well in problems that contain many discrete random variables.

The Gamma Transform benchmark tests the robustness of many-to-one transformations (Listing 7.2b), where $X \sim \mathrm{Gamma}(3,1)$; $Y = 1/\exp X^2$ if $X < 1$ and $Y = 1/\ln X$ otherwise; and $Z = -Y^3 + Y^2 + 6Y$. Each of the $n = 5$ datasets specifies a different constraint $\phi(Z)$ and a query about the posterior $Y \mid \phi(Z)$, which needs to compute and integrate out $X \mid \phi(Z)$. PSI reports that there is an error in its answer for all five datasets, whereas SPPL, using the semi-symbolic transform solver from Appendix 7.C, solves all five problems effectively.

Figure 7.8 compares the runtime variance using SPPL and PSI for four of the benchmarks in

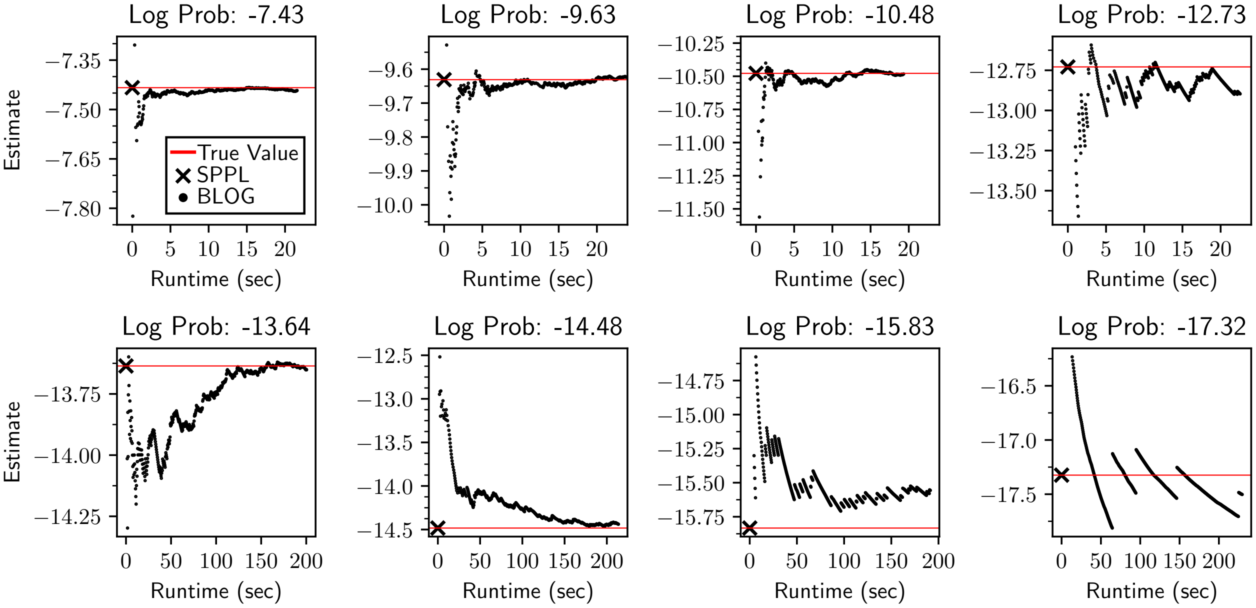


Figure 7.9: Runtime comparison for computing rare event probabilities using exact inference in SPPL and rejection sampling in BLOG in eight probabilistic programs from Table 4.2. As the probability of the event decreases, the runtime needed to obtain an accurate estimate using sampling-based inference in BLOG increases, whereas SPPL delivers exact answers in milliseconds for all events.

Table 7.3, where one query is repeated over 10 datasets. In all benchmarks, the SPPL variance is lower than that of PSI, with a maximum standard deviation $\sigma = 0.5$ sec. In contrast, the spread of PSI runtime is high for Student Interviews ($\sigma = 540$ sec, range 64–1890 sec) and Clinical Trial ($\sigma = 153$ sec, range 2.75–470 sec). In PSI, the analyses are sensitive to the numeric values in the observations, which leads to unpredictable runtime across different datasets even for a fixed query pattern. In SPPL, the runtime depends on the query pattern not the observed data and therefore behaves predictably across different observations.

As with the fairness benchmarks in Section 7.5.1, PSI trades off expressiveness with efficacy on tractable problems. The results show that its runtime and memory do not scale well and are unpredictable on benchmarks that SPPL solves very efficiently. Moreover, PSI may return unusable solutions to the user and it needs to recompute entire symbolic solutions from scratch for each new dataset or query. While SPPL is less expressive than PSI, it carries neither of these limitations.

7.5.3 Comparison to Sampling-Based Estimates

The final evaluation compares the runtime and accuracy of estimating probabilities of rare events in a canonical Bayesian network from Koller and Friedman [2009] using SPPL and BLOG [Milch et al., 2005]. As discussed in Koller and Friedman [2009, Section 12.13], rare events are the rule, not the exception, in many applications, as the probability of a predicate $\phi(X)$ decreases exponentially with the number of observed variables in X. Small estimation errors can magnify substantially when, e.g., taking ratios of probabilities, such as Eq. (7.19).

In Figure 7.9, each subplot shows the runtime and probability estimates for a low probability event ϕ. In BLOG, the rejection sampler estimates the probability of ϕ by computing the fraction of times it holds in a size n i.i.d. random sample from the prior. The horizontal red line shows the ground-truth probability. The x marker shows the runtime needed by SPPL to (exactly) compute the probability and the dots show the estimates from BLOG with increasing runtime (i.e., more samples n). SPPL consistently returns an exact answer in less than 2ms. The accuracy of BLOG estimates improve as the runtime increases: by the strong law of large numbers, these estimates converge to the true value for any sample path, but the fluctuations for any single run can be large (the standard error decays as $1/\sqrt{n}$). Each "jump" corresponds to a new sample $X^{(j)}$ that satisfies $\phi(X^{(j)})$, which increases the estimate. Without ground truth, it is hard to predict how much computation is needed for BLOG to obtain accurate results. For example, the estimates for predicates with $\log p = -12.73$ and $\log p = -17.32$ did not converge within the allotted time, while those for $\log p = -14.48$ converged after 180 seconds.

7.6 Related Work

SPPL is distinguished by being the first system to deliver exact symbolic inference by translating probabilistic programs to sum-product expressions, which are a new member of the class of probabilistic circuits [Darwiche, 2021] that extend and generalize sum-product networks. Several related approaches in the literature are discussed below.

Symbolic Integration Many systems aim to deliver exact inference by translating a probabilistic program and observed dataset into a symbolic expression that answers the query [Bhat et al., 2013, Narayanan et al., 2016, Gehr et al., 2016, Carette and Shan, 2016, Zhang and Xue, 2019]. The approach to exact inference in SPPL uses sum-product expressions instead of general computer algebra, which is less expressive but enables highly effective performance on the models and queries that can be expressed. The PSI solver, for example, can express many inference problems that SPPL cannot express due to restrictions (R1)–(R4), which include certain higher-order programs [Gehr et al., 2020]. However, comparisons on benchmarks that SPPL targets in Section 7.5.2 find PSI has less scalable and higher variance runtime, and can return unusable results with unsimplified symbolic integrals. In contrast, SPPL exploits conditional independencies when they exist to improve scalability (Section 7.4.1) and delivers usable answers to users. Moreover, SPPL's multi-stage workflow in Figure 7.7 allows expensive computations such as translation and conditioning to be amortized over multiple datasets or queries, whereas PSI recomputes the symbolic solution from scratch each time. Hakaru [Narayanan et al., 2016] is a symbolic solver that delivers exact inference in a multi-stage workflow based on program transformations and can disintegrate distributions against a variety of base measures [Narayanan and Shan, 2020]. SPPL is compared to PSI because the reference Hakaru implementation crashes or delivers incorrect or partial results on several benchmark problems [Gehr et al., 2016, Table 1] and does not support constructs such as arrays needed to support dozens or hundreds of observations.

Symbolic Execution and Volume Computation Several works have addressed the problem of computing the probability of a predicate by integrating a distribution defined by a program [Geldenhuys et al., 2012, Sankaranarayanan et al., 2013, Toronto et al., 2015, Albarghouthi et al., 2017]. While SPPL can model a variety of distributions, due to restriction (R3) it only supports predicates that specify rectangular regions, whereas several of the aforementioned systems can (approximately) handle non-rectangular regions. More specifically, predicates in SPPL may include combinations of nonlinear transforms, each of a single variable, which are then solved into linear expressions that specify unions of disjoint hyperrectangles (Appendix 7.C.2). Table 7.2 shows that SPPL delivers substantial speedup on the hyperrectangular regions from decision trees, which are central to many applications.

Sum-Product Networks The SPFlow library [Molina et al., 2020] is an object-oriented "graphical model toolkit" in Python for constructing and querying sum-product networks. SPPL leverages a new and more general sum-product representation (Listings 7.1 and 7.2) and solves probability and conditioning queries (Theorem 7.5) that are not supported by SPFlow. These queries include mixed-type random variables, numeric transforms, and logical predicates with set-valued constraints. In addition, SPPL uses a novel translation strategy (Section 7.4) that allows users to specify models as generative code in a PPL (using e.g., variables, arrays, arithmetic and logical expressions, loops, branches) without having to manually manipulate low-level data structures. "Factored sum-product networks" [Stuhlmüller and Goodman, 2012] have been used as intermediate representations for converting a probabilistic program and any functional interpreter into a system of equations whose solution is the marginal probability of the program's return value. These algorithms handle recursive procedures and leverage dynamic programming, but only apply to discrete variables, cannot handle transforms, and require solving fixed-points. Moreover, they have not been quantitatively evaluated on PPL benchmark problems.

Knowledge Compilation Weighted-model counting (WMC) and weighted model integration (WMI) are knowledge compilation techniques for solving probabilistic inference queries via algorithmic reductions [Darwiche and Marquis, 2002, Chavira and Darwiche, 2008, Fierens et al., 2011, Vlasselaer et al., 2015, Belle et al., 2015, Darwiche, 2021]. SPPL can be seen as an instance of knowledge compilation from probabilistic programs to sum-product expressions; its inference algorithms, however, are not based on WMC or WMI but instead leverage highly effective semi-symbolic solvers described in Appendices 7.B–7.D. Symbo [Zuidberg Dos Martires et al., 2019] and Dice [Holtzen et al., 2020] are two probabilistic programming systems that also compile programs to probabilistic circuits. Symbo delegates all continuous variables to the PSI solver, and thereby inherits many of the challenges with computer algebra shown in the evaluation from Section 7.5. As compared to Dice, SPPL uses a more expressive representation based on sum-product expressions rather than boolean decision diagrams, which let the system handle a broader range of modeling constructs such as continuous random variables, mixed-type random variables, and numeric many-to-one transforms.

7.A Representing Gaussian Process DSL Programs in SPPL

It is relatively straightforward to translate programs from the probabilistic DSLs for cross-sectional data tables (Chapter 4), multivariate time series (Chapter 5), and relational data (Chapter 6) into the standard SPPL source syntax (Listing 7.5), as the Likelihood semantics of expressions in these DSLs are based on hierarchical factorial mixtures. For example, the MultiMixture DSL from Chapter 4 admits a direct representation as a member of the SPE domain, as shown in Figure 4.1. Translating programs from the Gaussian process DSL for univariate time series (Chapters 2 and 3) into SPPL is less straightforward because the joint distribution of random variables generated by a multivariate normal with non-diagonal covariance (2.3) does not factor into sums and products. Representing Gaussian processes in SPPL is done by extending the core calculus to include multivariate primitives that generalize the univariate Distribution domain from Listing 7.1e. While the formal details are beyond the scope of this thesis, the reference implementation of SPPL listed in Section 1.4 provides a Gaussian process library for modeling covariance functions (Listing 2.1) and a Gaussian process leaf node. Listing 7.8 shows a representative Gaussian process probabilistic program in SPPL. Regarding the syntactic domains in Listing 7.5, the SPPL variable `X` in Listing 7.8 is neither a Var nor ArrayVar, but rather a ProcessVar which can be indexed at arbitrary numbers. For any $r \in \mathsf{Real}$, the variable `X[`r`]` represents the value of the time series at time point r and can be queried as a standard SPPL variable, for example in `prob` or `constrain` queries. While SPE elements with Gaussian process leaves are closed under the *condition*$_0$ operation (Appendix 7.D.3) for pointwise constraints, they are not closed under the more general *condition* operation (7.11). Further, it is not possible to draw random variables from truncated multivariate Gaussians in closed form using inverse CDF tricks as in Proposition 7.4, but can be performed exactly using rejection sampling or approximately using Gibbs sampling.

```
# Synthesized Program
covariance = ChangePoint(
    5, 0.5,
    Linear(1) * SquaredExponential(0.01),
    Periodic(10, 2))
epsilon = 0.147
X ~ GaussianProcess(
  mu=lambda x: 0,
  cov=covariance + WhiteNoise(epsilon))

# Observed Time Series Data
constrain({
  X[0.0] : -1.97,
  X[0.1] : -6.18,
  X[0.2] : -3.18,
  ...
  X[9.8] : 1.92,
  X[9.9] : 2.12,
  })
```

Listing 7.8: Example of synthesized Gaussian process probabilistic program in SPPL.

7.B Definitions of Auxiliary Functions

Section 7.2 refers to the following operations on the $\mathsf{Outcomes}$ domain:

$$\begin{aligned} \mathit{union} &: \mathsf{Outcomes}^* \to \mathsf{Outcomes}, && (7.20)\\ \mathit{intersection} &: \mathsf{Outcomes}^* \to \mathsf{Outcomes}, && (7.21)\\ \mathit{complement} &: \mathsf{Outcomes} \to \mathsf{Outcomes}, && (7.22) \end{aligned}$$

where any implementation satisfies the following invariants:

$$\begin{aligned} (v_1 \amalg \cdots \amalg v_m) = \mathit{union}\, v^* &\iff \forall i \neq j.\ (\mathit{intersection}\, v_i\, v_j) = \varnothing && (7.23)\\ (v_1 \amalg \cdots \amalg v_m) = \mathit{intersection}\, v^* &\iff \forall i \neq j.\ (\mathit{intersection}\, v_i\, v_j) = \varnothing && (7.24)\\ (v_1 \amalg \cdots \amalg v_m) = \mathit{complement}\, v &\iff \forall i \neq j.\ (\mathit{intersection}\, v_i\, v_j) = \varnothing. && (7.25) \end{aligned}$$

Listing 7.9 shows an implementation of the *complement* function, which operates separately on the Real and String components; *union* and *intersection* are implemented similarly. Listing 7.10 shows the *vars* function, which returns the variables in a Transform or Event expression. Listing 7.13 shows the *negate* function, which returns the logical negation of an Event.

7.C Transforms of Random Variables

This appendix describes the Transform domain in the core calculus, expanding Listings 7.2b and 7.1c, which expresses numerical transformations of real random variables.

$$
\begin{aligned}
\textit{complement}\ \texttt{\{}s_1 \ldots s_m\texttt{\}}^b &::= \texttt{\{}s_1 \ldots s_m\texttt{\}}^{\neg b} \\
\textit{complement}\ \texttt{((}b_1\ r_1\texttt{)}\ \texttt{(}r_2\ b_2\texttt{))} &::= \texttt{((\#f}\ {-\infty}\texttt{)}\ \texttt{(}r_1\ \neg b_1\texttt{))} \amalg \texttt{((}\neg b_2\ r_2\texttt{)}\ \texttt{(}\infty\ \texttt{\#f))} \\
\textit{complement}\ \texttt{\{}r_1 \ldots r_m\texttt{\}} &::= \texttt{((\#f}\ {-\infty}\texttt{)}\ \texttt{(}r_1\ \texttt{\#t))} \\
&\quad\ \amalg \left[\amalg_{j=2}^{m}\texttt{((\#t}\ r_{j-1}\texttt{)}\ \texttt{(}r_j\ \texttt{\#t))}\right] \\
&\quad\ \amalg \texttt{((\#t}\ r_m\texttt{)}\ \texttt{(}\infty\ \texttt{\#f))} \\
\textit{complement}\ \varnothing &::= \texttt{\{\}}^{\texttt{\#t}} \amalg \texttt{((\#f}\ {-\infty}\texttt{)}\ \texttt{(}\infty\ \texttt{\#f))}
\end{aligned}
$$

Listing 7.9: Implementation of *complement* on the sum domain Outcomes.

```
$\textit{vars} : (\mathsf{Transform} + \mathsf{Event}) \to \mathcal{P}(\mathsf{Vars})$
$\textit{vars te} = \textbf{match}\ \textit{te}$
    $\triangleright\ t \Rightarrow \textbf{match}\ t$
        $\triangleright\ \texttt{Id(}x\texttt{)} \Rightarrow \{x\}$
        $\triangleright\ \texttt{Root(}t'\ n\texttt{)} \mid \texttt{Exp(}t'\ r\texttt{)} \mid \texttt{Log(}t'\ r\texttt{)} \mid \texttt{Abs(}t'\texttt{)}$
            $\mid \texttt{Reciprocal(}t'\texttt{)} \mid \texttt{Poly(}t'\ r_0\ \ldots\ r_m\texttt{)}$
            $\Rightarrow \textit{vars}\ t'$
        $\triangleright\ \texttt{Piecewise((}t_i\ e_i\texttt{)}_{i=1}^{m}\texttt{)} \Rightarrow \cup_{i=1}^{m}((\textit{vars}\ t_i) \cup (\textit{vars}\ e_i))$
    $\triangleright\ \texttt{(}t\ \texttt{in}\ v\texttt{)} \Rightarrow \textit{vars}\ t$
    $\triangleright\ (e_1 \sqcap \cdots \sqcap e_m) \mid (e_1 \sqcup \cdots \sqcup e_m) \Rightarrow \cup_{i=1}^{m}\textit{vars}\ e_i$
```

Listing 7.10: Implementation of *vars*, which returns the variables in a Transform or Event.

$$
\begin{aligned}
&\textit{scope} : \mathsf{SPE} \to \mathcal{P}(\mathsf{Var}) \\
&\textit{scope}\ \texttt{(}x\,d\,\sigma\texttt{)} ::= \mathrm{dom}(\sigma) \\
&\textit{scope}\ \texttt{(}S_1 \otimes \cdots \otimes S_m\texttt{)} ::= \cup_{i=1}^{m}(\textit{scope}\ S_i) \\
&\textit{scope}\ \texttt{((}S_1\,w_1\texttt{)} \oplus \cdots \oplus \texttt{(}S_m\,w_m\texttt{))} ::= (\textit{scope}\ S_1)
\end{aligned}
$$

Listing 7.11: Implementation of *scope*, which returns the set of variables in an element of SPE.

```
$\textit{subsenv} : \mathsf{Event} \to \mathsf{Environment} \to \mathsf{Event}$
$\textit{subsenv}\ e\ \sigma$ ::= $\textbf{let}\ \{x, x_1, \ldots, x_m\} = \mathrm{dom}(\sigma)$
                 $\textbf{in}\,\textbf{let}\ e_1\ \textbf{be}\ \textit{subs}\ e\ x_m\ \sigma(x_m)$
                 $\ldots$
                 $\textbf{in}\,\textbf{let}\ e_m\ \textbf{be}\ \textit{subs}\ e_{m-1}\ x_1\ \sigma(x_1)$
                 $\textbf{in}\ e_m$
```

Listing 7.12: Implementation of *subsenv*, which rewrites e as an Event e' on one variable x.

```
$\mathit{negate} : \textsf{Event} \to \textsf{Event}$
$\mathit{negate}\,(t\,\texttt{in}\,v) ::= \textbf{match}\;(\mathit{complement}\;v)$
    $\rhd\; v_1 \amalg \cdots \amalg v_m \Rightarrow (t\,\texttt{in}\,v_1) \sqcup \cdots \sqcup (t\,\texttt{in}\,v_m)$
    $\rhd\; v \Rightarrow (t\,\texttt{in}\,v)$
$\mathit{negate}\,(e_1 \sqcap \cdots \sqcap e_m) ::= \sqcup_{i=1}^{m}(\mathit{negate}\,e_i)$
$\mathit{negate}\,(e_1 \sqcup \cdots \sqcup e_m) ::= \sqcap_{i=1}^{m}(\mathit{negate}\,e_i)$
```

Listing 7.13: Implementation of *negate*, which applies De Morgan's laws to an Event.

```
$\mathit{dnf} : \textsf{Event} \to \textsf{Event}$
$\mathit{dnf}\;(t\;\texttt{in}\;v) ::= (t\;\texttt{in}\;v)$
$\mathit{dnf}\;e_1 \sqcup \cdots \sqcup e_m ::= \sqcup_{i=1}^{m}(\mathit{dnf}\;e_i)$
$\mathit{dnf}\;e_1 \sqcap \cdots \sqcap e_m ::= \textbf{let}_{1 \le i \le m}\;(e'_{j1} \sqcap \cdots \sqcap e'_{j,k_i})\;\textbf{be}\;\mathit{dnf}\;e_i$
                    $\textbf{in}\;\bigsqcup_{\substack{1 \le j_1 \le k_1 \\ \cdots \\ 1 \le j_m \le k_m}}\;\prod_{i=1}^{m} e'_{i,j_i}$
```

Listing 7.14: *dnf* converts and Event to DNF (Definition 7.9).

```
$\mathit{disjoint?} : \textsf{Event} \times \textsf{Event} \to \textsf{Boolean}$
$\mathit{disjoint?}\,\langle e_1, e_2 \rangle ::= \textbf{match}\;\langle e_1, e_2 \rangle$
$\rhd\;\langle \sqcap_{i=1}^{m_1}(\texttt{Id}(x_{1,i})\;\texttt{in}\;v_{1,i}), \sqcap_{i=1}^{m_2}(\texttt{Id}(x_{2,i})\;\texttt{in}\;v_{2,i})\rangle$
$\;\Rightarrow\;[\exists_{1 \le i \le 2}.\exists_{1 \le j \le m_i}.v_{ij} = \varnothing)] \;\vee\; \begin{bmatrix} \textbf{let}\;\{\langle n_{1i}, n_{2i}\rangle\}_{i=1}^{k}\;\textbf{be}\;\{\langle i, j\rangle \mid x_{1,i} = x_{2,j}\} \\ \textbf{in}\;(\exists_{1 \le i \le k}.(\mathit{intersection}\;v_{1,n_{1,i}}\;v_{2,n_{2,i}}) = \varnothing) \end{bmatrix}$
$\rhd\;\textbf{else} \Rightarrow \textbf{undefined}$
```

Listing 7.15: *disjoint*? returns `#t` if two Events are disjoint (Definition 7.13).

$$
\begin{aligned}
\mathbb{T} &: \textsf{Transform} \to (\textsf{Real} \to \textsf{Real}) \\
\mathbb{T}\,[\![\texttt{Id(}x\texttt{)}]\!] &::= \lambda r'.\, r' \\
\mathbb{T}\,[\![\texttt{Reciprocal(}t\texttt{)}]\!] &::= \lambda r'.\, 1/\left(\mathbb{T}\,[\![t]\!]\,(r')\right) \\
\mathbb{T}\,[\![\texttt{Abs(}t\texttt{)}]\!] &::= \lambda r'.\, \left|\mathbb{T}\,[\![t]\!]\,(r')\right| \\
\mathbb{T}\,[\![\texttt{Root(}t\ n\texttt{)}]\!] &::= \lambda r'.\, \sqrt[n]{\mathbb{T}\,[\![t]\!]\,(r')} \\
\mathbb{T}\,[\![\texttt{Exp(}t\ r\texttt{)}]\!] &::= \lambda r'.\, r^{(\mathbb{T}[\![t]\!](r'))} && (\text{iff } 0 < r) \\
\mathbb{T}\,[\![\texttt{Log(}t\ r\texttt{)}]\!] &::= \lambda r'.\, \log_r\left(\mathbb{T}\,[\![t]\!]\,(r')\right) && (\text{iff } 0 < r) \\
\mathbb{T}\,[\![\texttt{Poly(}t\ r_0\ \ldots\ r_m\texttt{)}]\!] &::= \lambda r'.\, \textstyle\sum_{i=0}^{m} r_i\,(\mathbb{T}\,[\![t]\!]\,(r'))^i \\
\mathbb{T}\,[\![\texttt{Piecewise((}t_i\ e_i)_{i=1}^{m}\texttt{)}]\!] &::= \lambda r'.\, \textbf{if } \left[(\downarrow^{\textsf{Real}}_{\textsf{Outcome}} r') \in \mathbb{V}\,[\![\mathbb{E}\,[\![e_1]\!]\,x]\!]\right] \textbf{ then } \mathbb{T}\,[\![t_1]\!]\,r' \\
&\qquad\ \ \textbf{else if } \ldots \\
&\qquad\ \ \textbf{else if } \left[(\downarrow^{\textsf{Real}}_{\textsf{Outcome}} r') \in \mathbb{V}\,[\![\mathbb{E}\,[\![e_m]\!]\,x]\!]\right] \textbf{ then } \mathbb{T}\,[\![t_m]\!]\,r' \\
&\qquad\ \ \textbf{else undefined} \\
&\quad (\text{iff } (\mathit{vars}\ t_1) = \ldots = (\mathit{vars}\ t_m) \\
&\qquad\qquad = (\mathit{vars}\ e_1) = \cdots = (\mathit{vars}\ e_m) =: \{x\}
\end{aligned}
$$

Listing 7.16: Semantics of Transform.

$$
\begin{aligned}
\mathit{domainof} &: \textsf{Transform} \to \textsf{Outcomes} \\
\mathit{domainof}\ \texttt{Id(}x\texttt{)} &::= \texttt{((\#f } {-\infty}\texttt{) (}\infty\texttt{ \#f))} \\
\mathit{domainof}\ \texttt{Reciprocal(}t\texttt{)} &::= \texttt{((\#f } 0\texttt{) (}\infty\texttt{ \#f))} \\
\mathit{domainof}\ \texttt{Abs(}t\texttt{)} &::= \texttt{((\#f } {-\infty}\texttt{) (}\infty\texttt{ \#f))} \\
\mathit{domainof}\ \texttt{Root(}t\ n\texttt{)} &::= \texttt{((\#f } 0\texttt{) (}\infty\texttt{ \#f))} \\
\mathit{domainof}\ \texttt{Exp(}t\ r_0\texttt{)} &::= \texttt{((\#f } {-\infty}\texttt{) (}\infty\texttt{ \#f))} \\
\mathit{domainof}\ \texttt{Log(}t\ r_0\texttt{)} &::= \texttt{((\#f } 0\texttt{) (}\infty\texttt{ \#f))} \\
\mathit{domainof}\ \texttt{Poly(}t\ r_0\ \ldots\ r_m\texttt{)} &::= \texttt{((\#f } {-\infty}\texttt{) (}\infty\texttt{ \#f))} \\
\mathit{domainof}\ \texttt{Piecewise((}t_i\ e_i)_{i=1}^{m}\texttt{)} &::= \mathit{union}\ [(\mathit{intersection}\ (\mathit{domainof}\ t_i)\ (\mathbb{E}\,[\![e]\!]\,x)]_{i=1}^{m} \\
&\quad \text{where } \{x\} ::= \mathit{vars}\, t_1
\end{aligned}
$$

Listing 7.17: *domainof* returns the Outcomes on which a Transform is defined.

```
$\mathit{preimg}\ t\ v\ ::= \mathit{preimage}'\ t\ (\mathit{intersection}\ (\mathit{domainof}\ t)\ v)$
$\mathit{preimage}'\ \texttt{Id}\ v ::= v$
$\mathit{preimage}'\ t\ \varnothing ::= \varnothing$
$\mathit{preimage}'\ t\ (v_1 \amalg \cdots \amalg v_m) ::= \mathit{union}\ (\mathit{preimg}\ t\ v_1)\ \ldots\ (\mathit{preimg}\ t\ v_m)$
$\mathit{preimage}'\ t\ \texttt{\{}r_1 \ldots r_m\texttt{\}} ::= \mathit{preimg}\ t'\ (\mathit{union}\ (\mathit{finv}\ t\ r_1)\ \ldots\ (\mathit{finv}\ t\ r_m))$
$\mathit{preimage}'\ t\ \texttt{((}b_{\text{left}}\ r_{\text{left}}\texttt{)}\ \texttt{(}r_{\text{right}}\ b_{\text{right}}\texttt{))} ::= \textbf{match}\ t$
 $\triangleright\ \texttt{Radical(}t'\ n\texttt{)} \mid \texttt{Exp(}t'\ r\texttt{)} \mid \texttt{Log(}t'\ r\texttt{)} \Rightarrow \textbf{let}\ \texttt{\{}r'_{\text{left}}\texttt{\}}\ \textbf{be}\ \mathit{finv}\ t\ r_{\text{left}}$
                                                     $\textbf{in}\,\textbf{let}\ \texttt{\{}r'_{\text{right}}\texttt{\}}\ \textbf{be}\ \mathit{finv}\ t\ r_{\text{right}}$
                                                     $\textbf{in}\ \mathit{preimg}\ t'\ \texttt{((}b_{\text{left}}\ r'_{\text{left}}\texttt{)}\ \texttt{(}r'_{\text{right}}\ b_{\text{right}}\texttt{))}$
 $\triangleright\ \texttt{Abs(}t'\texttt{)} \Rightarrow \textbf{let}\ v'_{\text{pos}}\ \textbf{be}\ \texttt{((}b_{\text{left}}\ r_{\text{left}}\texttt{)}\ \texttt{(}r_{\text{right}}\ b_{\text{right}}\texttt{))}$
                 $\textbf{in}\,\textbf{let}\ v'_{\text{neg}}\ \textbf{be}\ \texttt{((}b_{\text{right}}\ {-r_{\text{right}}}\texttt{)}\ \texttt{(}{-r_{\text{left}}}\ b_{\text{left}}\texttt{))}$
                 $\textbf{in}\ \mathit{preimg}\ t'\ (\mathit{union}\ v'_{\text{pos}}\ v'_{\text{neg}})$
 $\triangleright\ \texttt{Reciprocal(}t'\texttt{)} \Rightarrow \textbf{let}\ \langle r'_{\text{left}}, r'_{\text{right}}\rangle\ \textbf{be}\ \textbf{if}\ (0 \le r_{\text{left}} < r_{\text{right}})$
                                                  $\textbf{then}\ \langle \textbf{if}\ (0 < r_{\text{left}})\ \textbf{then}\ 1/r_{\text{left}}\ \textbf{else}\ \infty,$
                                                        $\textbf{if}\ (r_{\text{right}} < \infty)\ \textbf{then}\ 1/r_{\text{right}}\ \textbf{else}\ 0\rangle$
                                                  $\textbf{else}\ \langle \textbf{if}\ (-\infty < r_{\text{left}})\ \textbf{then}\ 1/r_{\text{left}}\ \textbf{else}\ 0,$
                                                        $\textbf{if}\ (r_{\text{right}} < 0)\ \textbf{then}\ 1/r_{\text{right}}\ \textbf{else}\ -\infty\rangle$
                         $\textbf{in}\ \mathit{preimg}\ t'\ \texttt{((}b_{\text{right}}\ r'_{\text{right}}\texttt{)}\ \texttt{(}r'_{\text{left}}\ b_{\text{left}}\texttt{))}$
 $\triangleright\ \texttt{Polynomial(}t\ r_0\ \ldots\ r_m\texttt{)} \Rightarrow \textbf{let}\ v'_{\text{left}}\ \textbf{be}\ \mathit{polyLte}\ \neg b_{\text{left}}\ r_{\text{left}}\ r_0\ \ldots\ r_m$
                                          $\textbf{in}\,\textbf{let}\ v'_{\text{right}}\ \textbf{be}\ \mathit{polyLte}\ b_{\text{right}}\ r_{\text{right}}\ r_0\ \ldots\ r_m$
                                          $\textbf{in}\ \mathit{preimg}\ t'\ (\mathit{intersection}\ v'_{\text{right}}\ (\mathit{complement}\ v'_{\text{left}}))$
 $\triangleright\ \texttt{Piecewise((}t_i\ e_i\texttt{)}_{i=1}^{m}\texttt{)} \Rightarrow \textbf{let}_{1\le i\le m}\ v'_i\ \textbf{be}\ \mathit{preimg}\ t_i\ \texttt{((}b_{\text{left}}\ r_{\text{left}}\texttt{)}\ \texttt{(}b_{\text{right}}\ r_{\text{right}}\texttt{))}$
                                        $\textbf{in}\,\textbf{let}_{1\le i\le m}\ v_i\ \textbf{be}\ \mathit{intersection}\ v'_i\ (\mathbb{E}\,[\![e_i]\!]\,x),$
                                        $\textbf{in}\ \mathit{union}\ v_1\ \ldots\ v_m$      where $\{x\} ::= \mathit{vars}\, t_1$
```

Listing 7.18: *preimg* computes the generalized inverse of a many-to-one Transform.

```
                                 $\mathit{finv} : \mathsf{Transform} \to \mathsf{Real} \to \mathsf{Outcomes}$
                       $\mathit{finv}\ \texttt{Id(}x\texttt{)}\ r ::= \texttt{\{}r\texttt{\}}$
               $\mathit{finv}\ \texttt{Reciprocal(}t\texttt{)}\ r ::= \textbf{if}\ (r = 0)\ \textbf{then}\ \texttt{\{}-\infty\ \infty\texttt{\}}\textbf{else}\ \texttt{\{}1/r\texttt{\}}$
                      $\mathit{finv}\ \texttt{Abs(}t\texttt{)}\ r ::= \texttt{\{}-r\ r\texttt{\}}$
                   $\mathit{finv}\ \texttt{Root(}t\ n\texttt{)}\ r ::= \textbf{if}\ (0 \le r)\ \textbf{then}\ \texttt{\{}r^n\texttt{\}}\ \textbf{else}\ \varnothing$
                   $\mathit{finv}\ \texttt{Exp(}t\ r_0\texttt{)}\ r ::= \textbf{if}\ (0 \le r)\ \textbf{then}\ \texttt{\{}\log_{r_0}(r)\texttt{\}}\ \textbf{else}\ \varnothing$
                   $\mathit{finv}\ \texttt{Log(}t\ r_0\texttt{)}\ r ::= \texttt{\{}r_0^r\texttt{\}}$
      $\mathit{finv}\ \texttt{(Polynomial}\ t\ r_0\ \ldots\ r_m\texttt{)}\ r ::= \mathit{polySolve}\ r\ r_0\ r_1\ \ldots\ r_m$
         $\mathit{finv}\ \texttt{(Piecewise (}t_i\ e_i\texttt{)}_{i=1}^{m}\texttt{)} ::= \mathit{union}\ [(\mathit{intersection}\ (\mathit{finv}\ t_i\ r)\ (\mathbb{E}\,[\![e_i]\!]\,x))]_{i=0}^{m},$
                                   where $\{x\} ::= \mathit{vars}\, t_1$
```

Listing 7.19: *finv* computes the generalized inverse of a many-to-one transform at a single Real.

```
$polyLim$ : Real$^+$ → Real$^2$
$polyLim\ r_0$ ::= $\langle r_0, r_0 \rangle$
$polyLim\ r_0\ r_1\ \ldots\ r_m$ ::=
    let $n$ be $\max\{j \mid r_j > 0\}$
    in if ($even\ n$) then (if ($r_n > 0$) then $\langle \infty, \infty \rangle$ else $\langle -\infty, -\infty \rangle$)
                    else (if ($r_n > 0$) then $\langle -\infty, \infty \rangle$ else $\langle \infty, -\infty \rangle$)
```

Listing 7.20: *polyLim* computes the limits of a polynomial limits at the infinities.

```
$polySolve$ : Real → Real$^+$ → Set
$polySolve$ : $r\ r_0\ \ldots\ r_m$ ::= match $r$
    ▷ ($\infty \mid -\infty$)   ⇒ let $\langle r_{\text{neg}}, r_{\text{pos}} \rangle$   be $polyLim\ r_0\ \ldots\ r_m$
                       in let $f$           be $\lambda r'.$ if $(r = \infty)$ then $(r' = \infty)$ else $(r' = -\infty)$
                       in let $v_{\text{neg}}$         be if $(f\ r_{\text{neg}})$ then {$-\infty$} else $\varnothing$
                       in let $v_{\text{pos}}$         be if $(f\ r_{\text{pos}})$ then {$\infty$} else $\varnothing$
                       in                $union\ v_{\text{pos}}\ v_{\text{neg}}$
    ▷ else             ⇒ ($roots\ (r_0 - r)\ r_1\ \ldots\ r_m$)
```

Listing 7.21: *polySolve* computes the set of values at which a polynomial is equal to a specific value r.

```
$polyLte$ : Boolean → Real → Real$^+$ → Outcomes
$polyLte\ b\ r\ r_0\ \ldots\ r_m$ ::= match $r$
    ▷ $-\infty$    ⇒ if $b$ then $\varnothing$ else ($polySolve\ r\ r_0\ \ldots\ r_m$)
    ▷ $\infty$     ⇒ if $\neg b$ then ((#t $-\infty$) ($\infty$ #t))
                    else let $\langle r_{\text{left}}, r_{\text{right}} \rangle$ be $polyLim\ r_0\ \ldots\ r_m$
                         in let $\langle b_{\text{left}}, b_{\text{right}} \rangle$ be $\langle r_{\text{left}} = \infty, r_{\text{right}} = \infty \rangle$
                         in (($b_{\text{left}}$ $-\infty$) ($\infty b_{\text{right}}$))
    ▷ else    ⇒ let $[r_{\text{s},i}]_{i=1}^{k}$ be $roots\ (r_0 - r)\ r_1\ \ldots\ r_m$
               in let $[\langle r'_{\text{left},i}, r'_{\text{right},i} \rangle]_{i=0}^{k}$ be $[\langle -\infty, r_{\text{s},0} \rangle, \langle r_{\text{s},1}, r_{\text{s},2} \rangle, \ldots, \langle r_{\text{s},k-1}, r_{\text{s},k} \rangle, \langle r_{\text{s},k}, \infty \rangle]$
               in let $f_{\text{mid}}$ be $\lambda r r'.$ if       $(r = -\infty)$   then $r'$
                                      elseif   $(r' = \infty)$    then $r$
                                      else     $(r + r')/2$
               $t'$ be Poly(Id(x) $(r_0 - r)\ r_1\ \ldots\ r_m$)
               in $union$ $\left[\begin{array}{ll} \textbf{if } \mathbb{T}\ [\![t']\!]\ (f_{\text{mid}}\ r'_{\text{left},i}\ r'_{\text{right},i}) & \textbf{then } \texttt{((}b\ r'_{\text{left},i}\texttt{) (}r'_{\text{right},i}\ b\texttt{))} \\ & \textbf{else } \varnothing \end{array}\right]_{i=0}^{k}$
```

Listing 7.22: *polyLte* computes the set of values at which a polynomial is less than a given value r.

7.C.1 Valuation of Transforms

Listing 7.16 shows the valuation function $\mathbb{T}$ which defines each t as a Real function on Real. Each real function $[\![T]\!]\,t$ is defined on an input r' if and only if $\left(\downarrow^{\mathsf{Real}}_{\mathsf{Outcome}} r'\right) \in (\textit{domainof}\ t)$. Listing 7.17 shows the implementation of *domainof*.

7.C.2 Preimage Computation

Listing 7.18 shows the implementation of

$$\textit{preimg} : \mathsf{Transform} \to \mathsf{Outcomes} \to \mathsf{Outcomes}, \tag{7.26}$$

which, as discussed in Section 7.2, satisfies

$$\begin{aligned}
(\downarrow^{\mathsf{Real}}_{\mathsf{Outcome}} r) \in \mathbb{V}\,[\![\textit{preimg}\ t\ v]\!] &\iff \mathbb{T}\,[\![t]\!]\,(r) \in \mathbb{V}\,[\![v]\!]\,, && (7.27)\\
(\downarrow^{\mathsf{String}}_{\mathsf{Outcome}} s) \in \mathbb{V}\,[\![\textit{preimg}\ t\ v]\!] &\iff (t \in \mathsf{Identity}) \wedge (s \in \mathbb{V}\,[\![v]\!]). && (7.28)
\end{aligned}$$

The implementation of *preimg* uses several helper functions:

(Listing 7.19) *finv*: computes the preimage of each $t \in \mathsf{Transform}$ at a single Real.

(Listing 7.20) *polyLim*: computes the limits of a polynomial at the infinities.

(Listing 7.21) *polySolve*: computes the set of values at which a polynomial is equal to a given value (possibly positive or negative infinity).

(Listing 7.22) *polyLte*: computes the set of values at which a polynomial is less than or equal a given value.

In addition, a root finding algorithm *roots* : $\mathsf{Real}^{+} \to \mathsf{Real}^{*}$ (whose definition is not shown) is assumed to be available, which takes as input a list of $m+1$ coefficients and returns list of roots of the corresponding degree-m polynomial. In the reference implementation of SPPL, the *roots* function uses symbolic analysis for polynomials whose degree is at most two and a semi-symbolic solver for higher-order polynomials.

7.D Conditioning Sum-Product Expressions

This section presents algorithms for exact inference, that is, conditioning the distribution defined by an element of SPE from Listings 7.1f and 7.2f. Recall Theorem 7.5, which establishes that SPE is closed under conditioning on any positive probability Event.

Theorem 7.5 (Closure under conditioning)**.** *Let $S \in \mathsf{SPE}$ and $e \in \mathsf{Event}$ be given, where $\mathbb{P}\,[\![S]\!]\,e > 0$. There exists an algorithm which, given S and e, returns $S' \in \mathsf{SPE}$ such that, for all $e' \in \mathsf{Event}$, the probability of e' according to S' is equal to the conditional probability of e' given e according to S, i.e.,*

$$\mathbb{P}\,[\![S']\!]\,e' \equiv \mathbb{P}\,[\![S]\!]\,(e' \mid e) ::= \frac{\mathbb{P}\,[\![S]\!]\,(e \sqcap e')}{\mathbb{P}\,[\![S]\!]\,e}. \tag{7.10}$$

«

Appendix 7.D.2 presents an algorithm for conditioning on a positive probability Event (Listings 7.2c and 7.1d). Appendix 7.D.3 presents an algorithm for conditioning on a $\mathsf{Conjunction}$ of equality constraints involving only non-transformed variables, such as $\{X = 3\} \sqcap \{Y = 4\}$, as described in Remark 7.6. Before presenting the proofs, several preprocessing algorithms are needed.

7.D.1 Algorithms for Event Preprocessing

Normalizing an Event The *dnf* function (Listing 7.14) converts an Event e to DNF, defined below.

Definition 7.9. An Event e is said to be in disjunctive normal form (DNF) if and only if one of the following holds:

(7.9.1) $e \in \mathsf{Containment}$

(7.9.2) $e = e_1 \sqcap \cdots \sqcap e_m \in \mathsf{Conjunction} \implies \forall_{1 \le i \le m}.\ e_i \in \mathsf{Containment}$

(7.9.3) $e = e_1 \sqcup \cdots \sqcup e_m \in \mathsf{Disjunction} \implies \forall_{1 \le i \le m}.\ e_i \in \mathsf{Containment} \cup \mathsf{Conjunction}$

Terms e and e_i in items (7.9.1) and (7.9.2) are called “literals” and terms e_i in item (7.9.3) are called “clauses”. «

Definition 7.10. An Event e is in solved DNF if all the following conditions hold: (i) e is in DNF; (ii) all literals within a clause e_i of e have different variables; and (iii) each literal $(t\,\mathtt{in}\,v)$ of e satisfies $t \in \mathsf{Identity}$ and $v \notin \mathsf{Union}$. «

Example 7.11. Using informal notation, the solved DNF form of the event

$$\{X^2 \ge 9\} \cap \{|Y| < 1\} \tag{7.29}$$

is a disjunction with two conjunctive clauses:

$$[\{X \in (-\infty, -3)\} \cap \{Y \in (-1, 1)\}] \cup [\{X \in (3, \infty)\} \cap \{Y \in (-1, 1)\}]. \tag{7.30}$$

«

Listing 7.23a shows the *normalize* operation, which converts an Event e to solved DNF. In particular, predicates with (possibly nonlinear) arithmetic expressions are converted to predicates that contain only linear expressions, as in Eqs. (7.15)–(7.18). The next result, Proposition 7.12, follows from $\mathbb{E}\,[\![e]\!] = \mathbb{E}\,[\![\mathit{dnf}\ e]\!]$ and denotations of Union (Listing 7.2a) and $\mathsf{Disjunction}$ (Listing 7.2c).

Proposition 7.12. $\forall e \in \mathsf{Event}$, $\mathbb{E}\,[\![e]\!] \equiv \mathbb{E}\,[\![(\mathit{normalize}\, e)]\!]$. «

Disjoining an Event Suppose that $e \in \mathsf{Event}$ is in DNF and has $m \ge 2$ clauses. A key inference subroutine is to rewrite e in solved DNF (Definition 7.10) where all the clauses are disjoint.

Definition 7.13. Let $e \in \mathsf{Event}$ be in DNF. Two clauses e_i and e_j of e are said to be disjoint if both e_i and e_j are in solved DNF and at least one of the following conditions holds:

$$\exists x \in (\mathit{vars}\, e_i).\ \mathbb{E}\,[\![e_{ix}]\!]\, x \equiv \varnothing \tag{7.31}$$

$$\exists x \in (\mathit{vars}\, e_j).\ \mathbb{E}\,[\![e_{jx}]\!]\, x \equiv \varnothing \tag{7.32}$$

$$\exists x \in (\mathit{vars}\, e_i) \cap (\mathit{vars}\, e_j).\ \mathbb{E}\,[\![e_{ix} \sqcap e_{jx}]\!]\, x \equiv \varnothing \tag{7.33}$$

where e_{ix} denotes the unique literal of e_i that contains variable x (for $x \in \mathit{vars}\, e_i$), and similarly for e_j. «

Listing 7.15 shows the *disjoint*? procedure, which given a pair of clauses e_i and e_j that are in solved DNF (as produced by *normalize*), returns true if and only if one of the conditions in Definition 7.13 hold. Listing 7.23b presents the main algorithm *disjoin*, which decomposes an arbitrary Event e into solved DNF whose clauses are mutually disjoint. Proposition 7.14 establishes the correctness and worst-case complexity of *disjoin*.

```
$normalize : \mathsf{Event} \to \mathsf{Event}$
$normalize\ (t\ \mathtt{in}\ v) ::= \mathbf{match}\ preimg\ t\ v$
  $\triangleright\ v'_1 \amalg \cdots \amalg v'_m \Rightarrow \sqcup_{i=1}^{m} (\mathtt{Id}(x)\ \mathtt{in}\ v'_i)$
  $\triangleright\ v' \Rightarrow (\mathtt{Id}(x)\ \mathtt{in}\ v')$,        where $\{x\} ::= vars\ t$
$normalize\ (e_1 \sqcap \cdots \sqcap e_m) ::= dnf\ \sqcap_{i=1}^{m} (normalize\ e_i)$
$normalize\ (e_1 \sqcup \cdots \sqcup e_m) ::= dnf\ \sqcup_{i=1}^{m} (normalize\ e_i)$
```

(a) normalize

```
$disjoin : \mathsf{Event} \to \mathsf{Event}$
$disjoin\ e ::= \mathbf{let}\ (e_1 \sqcup \cdots \sqcup e_m)\ \mathbf{be}\ normalize\ e$                                   (7.34a)
  $\mathbf{in}\ \mathbf{let}_{2 \le i \le m}\ \tilde{e}\ \mathbf{be} \bigsqcap_{1 \le j < i \mid \neg(disjoint?\langle e_j, e_i \rangle)} (negate\ e_j)$     (7.34b)
  $\mathbf{in}\ \mathbf{let}_{2 \le i \le m}\ \tilde{e}_i\ \mathbf{be}\ (disjoin\ (e_i \sqcap \tilde{e}_i))$                          (7.34c)
  $\mathbf{in}\ e_1 \sqcup \tilde{e}_2 \sqcup \cdots \sqcup \tilde{e}_m$
```

(b) disjoin

Listing 7.23: Event preprocessing algorithms used by *condition*.

Proposition 7.14. *Let e be an Event with $h ::= |vars\ e|$ variables, and suppose that $e_1 \sqcup \cdots \sqcup e_m ::= (normalize\ e)$ has exactly $m \ge 1$ clauses. Put $\tilde{e} ::= (disjoin\ e)$. Then:*

(7.14.1) $\tilde{e}$ is in solved DNF.

(7.14.2) $\forall_{1 \le i \ne j \le \ell}.\ disjoint?\langle e_i, e_j \rangle$.

(7.14.3) $\mathbb{E}[\![e]\!] = \mathbb{E}[\![\tilde{e}]\!]$.

(7.14.4) The number ℓ of clauses in $\tilde{e}$ satisfies $\ell \le (2m-1)^h$.

«

Proof. Suppose first that $(normalize\ e)$ has $m = 1$ clause e_1. Then $\tilde{e} = e_1$, so (7.14.1) holds since $e_1 = normalize\ e$; (7.14.2) holds trivially; (7.14.3) holds by Proposition 7.12; and (7.14.4) holds since $\ell = (2-1)^h = 1$. Suppose now that $(normalize\ e)$ has $m > 1$ clauses. To employ set-theoretic reasoning, fix some $x \in \mathsf{Var}$ and define $\mathbb{E}'[\![e]\!] ::= \mathbb{V}[\![\mathbb{E}[\![e]\!]\,x]\!] \subset \mathsf{Outcome}$, for all $e \in \mathsf{Event}$. Then

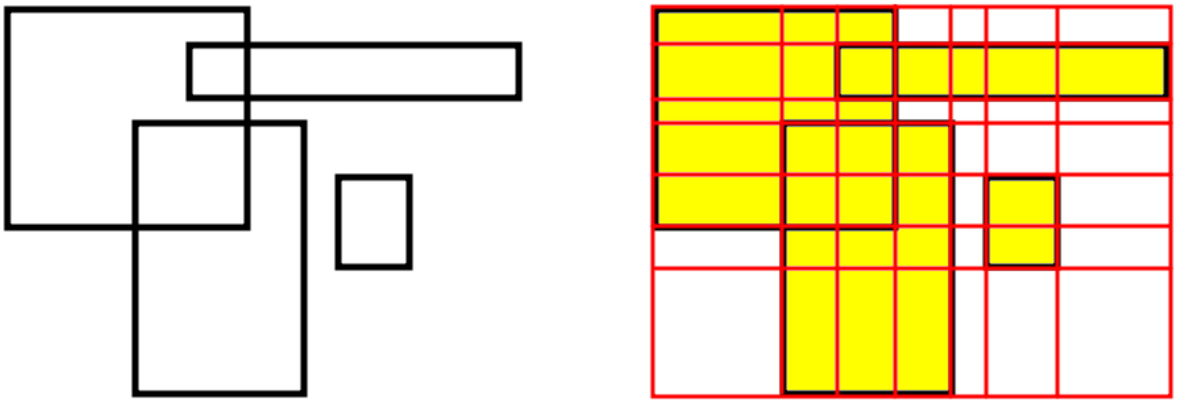
$$\begin{aligned}
&\mathbb{E}'[\![e_1 \sqcup \cdots \sqcup e_m]\!] && (7.35)\\
&= \cup_{i=1}^{m} \mathbb{E}'[\![e_i]\!] && (7.36)\\
&= \cup_{i=1}^{m} \left(\mathbb{E}'[\![e_i]\!] \cap \neg \left[\cup_{j=1}^{i-1} (\mathbb{E}'[\![e_j]\!])\right]\right) && (7.37)\\
&= \cup_{i=1}^{m} \left(\mathbb{E}'[\![e_i]\!] \cap \left[\cap_{j=1}^{i-1} (\neg \mathbb{E}'[\![e_j]\!])\right]\right) && (7.38)\\
&= \cup_{i=1}^{m} \left(\mathbb{E}'[\![e_i]\!] \cap \left[\cap_{j \in k(i)} (\neg \mathbb{E}'[\![e_j]\!])\right]\right) && (7.39)
\end{aligned}$$

where, for each $i = 1, \dots, m$,

$$k(i) ::= \left\{1 \le j \le i-1 \mid \mathbb{E}'[\![e_i]\!] \cap \mathbb{E}'[\![e_j]\!] \ne \varnothing\right\}.$$

Eq. (7.39) follows from the fact that for any $i = 1, \dots, m$ and $j < i$,

$$j \notin k(i) \implies \left[\left(\mathbb{E}'[\![e_i]\!] \cap \neg \mathbb{E}'[\![e_j]\!]\right) \equiv \mathbb{E}'[\![e_i]\!]\right]. \qquad (7.40)$$



(a) Conditioning Region

(b) Partition into Disjoint Rectangles

Figure 7.10: Illustration of the upper bound (7.14.4) on the number of disjoint rectangles in a worst-case partition of a conditioning region of axis-aligned rectangles that live in the two-dimensional Real plane.

As *negate* (Listing 7.13) computes set-theoretic complement $\neg$ in the Event domain and $j \notin k(i)$ if and only if $(\textit{disjoint?}\, e_j\, e_i)$, it follows that the Events $e'_i ::= e_i \sqcap \tilde{e}_i$ $(i = 2, \dots, m)$ in Eq. (7.34c) are pairwise disjoint and are also disjoint from e_1, so that $\mathbb{E}[\![e]\!] = \mathbb{E}[\![e_1 \sqcup e'_2 \sqcup \cdots \sqcup e'_m]\!]$. Thus, if *disjoin* halts, then all of (7.14.1)–(7.14.3) follow by induction.

It is next established that *disjoin* halts by proving an upper bound on the number ℓ of clauses returned by any call to *disjoin*. Recalling that $h ::= |\textit{vars}\, e|$, assume without loss of generality that all clauses e_i $(i = 1, \dots, n)$ in Eq. (7.34a) have the same variables $\{x_1, \dots, x_h\}$, by "padding" each e_i with vacuously true literals of the form $(\mathtt{Id}(x_i)\ \mathtt{in}\, \mathsf{Outcomes})$. Next, recall that clause e_i in Eq. (7.34a) is in solved DNF and has $m_i \geq 1$ literals $e_{ij} = (\mathtt{Id}(x_{ij})\ \mathtt{in}\, v_{ij})$ where $v_{ij} \notin \mathsf{Union}$ (Definition 7.10). Thus, e_i specifies exactly one hyperrectangle in h-dimensional space, where v_{ij} is the "interval" (possibly infinite) along the dimension specified by x_{ij} in literal e_{ij} $(i = 1, \dots, m; j = 1, \dots, m_i)$. A sufficient condition to produce the worst-case number of pairwise disjoint primitive sub-hyperrectangles that partition the region $e_1 \sqcup \cdots \sqcup e_m$ is when the previous clauses $e_1, \dots, e_{m-1}$ (i) are pairwise disjoint (Definition 7.13); and (ii) are strictly contained in e_m, i.e., $\forall x.\ \mathbb{E}[\![e_j]\!] \subsetneq \mathbb{E}[\![e_m]\!]$, $(j = 1, \dots, m-1)$. If these two conditions hold, then *disjoin* partitions the interior of the h-dimensional hyperrectangle specified by e_m into no more than $2(m-1)^h$ sub-hyperrectangles that do not intersect one another (and thus, produce no further recursive calls), thereby establishing (7.14.4). ■

Example 7.15. Figure 7.10a shows $m = 4$ rectangles in $\mathsf{Real} \times \mathsf{Real}$. Figure 7.10b shows a grid that induces $(2m-1)^2 = 49$ primitive rectangular regions that are pairwise disjoint from one another and whose union over-approximates the union of the 4 rectangles. In this case, 29 of these primitive rectangular regions are sufficient (but excessive) to exactly partition the union of the rectangles into a disjoint union. No more than 49 primitive rectangles are ever needed to partition any 4 rectangles in Real^2, and this bound is tight. The bound in (7.14.4) generalizes this idea to hyperrectangles that live in h-dimensional space. «

Remark 7.16. When defining $\tilde{e}$ in Eq. (7.34b) of *disjoin*, ignoring previous clauses that are disjoint from e_i is essential for *disjoin* to halt, so as to avoid recursing on a primitive sub-rectangle in the interior. That is, filtering out such clauses ensures that *disjoin* makes a finite number of recursive calls. «

7.D.2 Conditioning Sum-Product Expressions on Positive Measure Events

Having established the background results, Theorem 7.5 is proved next.

Proof of Theorem 7.5. Eq. (7.10) is shown by presenting the algorithm

$$\mathit{condition} : \mathsf{SPE} \to \mathsf{Event} \to \mathsf{SPE} \tag{7.41}$$

which satisfies

$$\mathbb{P}\llbracket(\mathit{condition}\, S\, e)\rrbracket\, e' = \frac{\mathbb{P}\llbracket S\rrbracket\,(e \sqcap e')}{\mathbb{P}\llbracket S\rrbracket\, e} \tag{7.42}$$

for all $e' \in \mathsf{Event}$ and $e \in \mathsf{Event}$ for which $\mathbb{P}\llbracket S\rrbracket\, e > 0$.

The *condition* algorithm is defined separately for each of the three domains Leaf, Sum, and Product from Listing 7.1f. The proof is by structural induction, where Leaf is the base case and Sum and Product are the recursive cases.

Conditioning Leaf Listing 7.24a shows the base cases of *condition*. The case of $d \in \mathsf{DistStr}$ is straightforward. For $d \in \mathsf{DistReal}$, if the intersection (defined in second line of Listing 7.24a) of v with the support of d is an interval $\texttt{((}b'_1\, r'_1\texttt{)}\ \texttt{(}r'_2, b'_2\texttt{))}$, then it suffices to return a Leaf restricting d to the interval. If the intersection is a Union $v_1 \amalg \cdots \amalg v_m$ (recall from Eq. (7.24) that *intersection* ensures the v_i are disjoint), then the conditioned SPE is a Sum, whose ith child is obtained by recursively calling *condition* on v_i and ith (relative) weight is the probability of v under d, since, for any new $v' \in \mathsf{Outcomes}$,

$$\frac{\mathbb{D}\llbracket d\rrbracket\,(\mathit{intersect}\, v'\,(v_1 \amalg \cdots \amalg v_m))}{\mathbb{D}\llbracket d\rrbracket\,(v_1 \amalg \cdots \amalg v_m)} = \frac{\mathbb{D}\llbracket d\rrbracket\, \amalg_{i=1}^{m}(\mathit{intersect}\, v'\, v_i)}{\sum_{i=1}^{m} \mathbb{D}\llbracket d\rrbracket\, v_i}. \tag{7.43}$$

Eq. (7.43) follows from the additivity of $\mathbb{D}\llbracket d\rrbracket$. The plots of X in Figures 7.5c and 7.5f illustrate the equality (7.43), where conditioning the unimodal normal distribution results in a mixture of three constrained Gaussian whose weights are given by the relative prior probabilities of the three regions.

For $d \in \mathsf{DistInt}$, if the positive probability Outcomes are $\texttt{\{}r_1 \dots r_m\texttt{\}}$, then the conditioned SPE is a Sum of "delta"-CDFs whose atoms are located on the integers r_i and weights are the (relative) probabilities $\mathbb{D}\llbracket d\rrbracket\,\texttt{\{}r_i\texttt{\}}$ $(i = 1, \dots, m)$. Since the atoms of F for DistInt are integers, it suffices to restrict F to the interval $(r_i - 1/2, r_i)$, for each r_i with a positive weight. Correctness again follows from Eq. (7.43), since finite sets are unions of disjoint singleton sets. For other positive probability Outcomes, the conditioning procedure DistInt is the same as that for DistReal.

Conditioning Sum Listing 7.24b shows *condition* for $S \in \mathsf{Sum}$. Recalling the denotation $\mathbb{P}\llbracket S\rrbracket$ for $S \in \mathsf{Sum}$ in Listing 7.2f, the correctness follows from

$$\begin{aligned}
\frac{\mathbb{P}\llbracket\texttt{(}S_1\ w_1\texttt{)} \oplus \cdots \oplus \texttt{(}S_m\ w_m\texttt{)}\rrbracket\,(e \sqcap e')}{\mathbb{P}\llbracket\texttt{(}S_1\ w_1\texttt{)} \oplus \cdots \oplus \texttt{(}S_m\ w_m\texttt{)}\rrbracket\, e} &= \frac{\sum_{i=1}^{m} w_i \mathbb{P}\llbracket S_i\rrbracket\,(e \sqcap e')}{\sum_{i=1}^{m} w_i \mathbb{P}\llbracket S_i\rrbracket\, e} && (7.44)\\
&= \frac{\sum_{i=1}^{m} w_i(\mathbb{P}\llbracket S_i\rrbracket\, e)\mathbb{P}\llbracket(\mathit{condition}\, S_i\, e)\rrbracket\, e'}{\sum_{i=1}^{m} w_i \mathbb{P}\llbracket S_i\rrbracket\, e} && (7.45)\\
&= \mathbb{P}\llbracket\oplus_{i=1}^{m}\texttt{(}(\mathit{condition}\, S_i\, e)\ , w_i \mathbb{P}\llbracket S_i\rrbracket\, e\texttt{)}\rrbracket\, e', && (7.46)
\end{aligned}$$

where Eq. (7.45) has applied Eq. (7.42) inductively for each S_i. Eqs. (7.44) and (7.45) assume for simplicity that $\mathbb{P}\llbracket S_i\rrbracket\, e > 0$ for each $i = 1, \dots, m$, whereas Listing 7.24a does not make this assumption.

Conditioning Product Listing 7.24c how *condition* operates on $S \in \mathsf{Product}$. The first step is to invoke *disjoin* to rewrite $(\mathit{dnf}\, e)$ as $\ell \geq 1$ disjoint clauses $e'_1 \sqcup \cdots \sqcup e'_\ell$ (recall from Proposition 7.14 that *disjoin* is semantics-preserving). The first pattern in the **match** statement corresponds $\ell = 1$, and the result is a new Product, where the ith child is conditioned on the literals of e_1 whose variables are contained in $\mathit{scope}\, S_i$ (if any). The second pattern returns a Sum of Product, since

$$\frac{\mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,(e \sqcap e')}{\mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,e} = \frac{\mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,((e_1 \sqcup \cdots \sqcup e_\ell) \sqcap e')}{\mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,(e_1 \sqcup \cdots \sqcup e_\ell)} \tag{7.47}$$

$$= \frac{\mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,((e_1 \sqcap e') \sqcup \cdots \sqcup (e_\ell \sqcap e'))}{\sum_{i=1}^{\ell} \mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,e_i} \tag{7.48}$$

$$= \frac{\sum_{i=1}^{\ell} \mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,(e_i \sqcap e')}{\sum_{i=1}^{\ell} \mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,e_i} \tag{7.49}$$

$$= \frac{\sum_{i=1}^{\ell} \mathbb{P}\,[\![S]\!]\,e_i\, \mathbb{P}\,[\![(\mathit{condition}\,(S_1 \otimes \cdots \otimes S_m)\, e_i)]\!]\,e'}{\sum_{i=1}^{\ell} \mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,e_i} \tag{7.50}$$

$$= \mathbb{P}\left[\!\!\left[\oplus_{i=1}^{\ell} ((\mathit{condition}\, S\, e_i)\ \mathbb{P}\,[\![S]\!]\,e_i)\right]\!\!\right] e'. \tag{7.51}$$

Eq. (7.50) follows from the induction hypothesis Eq. (7.42) and the idempotence $(\mathit{disjoin}\, e_i) \equiv e_i$, so that $(\mathit{disjoin}\, e_i \sqcap e') \equiv (\mathit{disjoin}\, e_i) \sqcap (\mathit{disjoin}\, e') \equiv e_i \sqcap (\mathit{disjoin}\, e')$.

Theorem 7.5 is thus established. ■

Figure 7.4 shows an example of the closure property from Theorem 7.5, where conditioning on a hyperrectangle changes the structure of the SPE from a Product into a Sum-of-Product. The algorithms in this section are the first to describe probabilistic inference and closure properties for conditioning an SPE on a query that involves transforms of random variables and predicates with set-valued constraints.

Theorem 7.7 is established next, which gives a sufficient condition for the runtime of *condition* (Listing 7.24) to scale linearly in the number of nodes in S. An identical result holds for computing Event probabilities ($\mathbb{P}\,[\![S]\!]\,e$, Listing 7.2f) and probability densities ($\mathbb{P}_0\,[\![S]\!]\,e$, Listing 7.2e).

Theorem 7.7. *The runtime of* $(\mathit{condition}\, S\, e)$ *scales linearly in the number of nodes in the graph representing* S *whenever* $e = (t_1\, \mathtt{in}\, v_1) \sqcap \cdots \sqcap (t_m\, \mathtt{in}\, v_m)$ *is a single* Conjunction *of* Containment *constraints, where each* t_i *represents a non-transformed program variable.* «

Proof. There are three cases to consider. Suppose $S \in \mathsf{Leaf}$. Then there are zero subcalls, independently of e. Suppose $S \in \mathsf{Sum}$ with m children. Then $(\mathit{condition}\, S\, e)$ makes no more than m subcalls to *condition*, one per child. Suppose $S \in \mathsf{Product}$ with m children. Since each node in S has exactly one parent, it will be shown that each node is visited exactly once by showing that the hypothesis on e implies there are makes at most m subcalls to *condition*. Since e is a single Conjunction $(t_1\, \mathtt{in}\, v_1) \sqcap \cdots \sqcap (t_m\, \mathtt{in}\, v_m)$ of Containment constraints on non-transformed variables, it must hold that $(\mathit{disjoin}\, e)$ returns a single Conjunction. Then the first pattern of the **match** statement in Listing 7.24c is matched, resulting in m subcalls to *condition*. ■

7.D.3 Conditioning Sum-Product Expressions on Equality Constraints

Recall from Remark 7.6 that SPE is also closed under conditioning on a Conjuction of possibly measure zero equality constraints of non-transformed variable, such as $\{X = 3, Y = \pi, Z = \texttt{"foo"}\}$. This section describes the conditioning algorithm for this case, which is implemented by

$$\mathit{condition}_0 : \mathsf{SPE} \to \mathsf{Event} \to \mathsf{SPE}, \tag{7.52}$$

where $e \in \mathsf{Event}$ satisfies the follows requirements with respect to $S \in \mathsf{SPE}$:

```
$\textit{condition}\ \texttt{Leaf(}x\,d\,\sigma\texttt{)}\ e\ ::= \textbf{let}\ v\ \textbf{be}\ \mathbb{E}\,[\![(\textit{subsenv}\ e\,\sigma)]\!]\,x\ \textbf{in match}\ d$
  $\triangleright\ \texttt{DistS((}s_i\,w_i)_{i=1}^{m}\texttt{)} \Rightarrow \textbf{match}\ v$
    $\triangleright\ \texttt{\{}s'_1 \dots s'_l\texttt{\}}^{b} \Rightarrow \textbf{let}_{1\le i\le m}\ w'_i\ \textbf{be if}\ \bar{b}\ \textbf{then}\ w_i\mathbf{1}[\exists_{1\le j\le \ell}.s'_j = s_i]$
                                        $\textbf{else}\ w_i\mathbf{1}[\forall_{1\le j\le \ell}.s'_j \neq s_i]$
                   $\textbf{in}\ \texttt{Leaf(}x\ \texttt{DistS((}s_i\ w'_i)_{i=1}^{m}\texttt{)}\ \sigma\texttt{)}$
    $\triangleright\ \textbf{else undefined}$
  $\triangleright\ \texttt{DistR(}F\ r_1\ r_2\texttt{)} \Rightarrow \textbf{match}\ (\textit{intersection}\ \texttt{((\#f}\,r_1\texttt{)}\ \texttt{(}r_2\,\texttt{\#f))}\ v)$
    $\triangleright\ \varnothing \mid \texttt{\{}r_1\ \dots\ r_m\texttt{\}} \Rightarrow \textbf{undefined}$
    $\triangleright\ \texttt{((}b_1\,r'_1\texttt{)}\ \texttt{(}r'_2\,b_2\texttt{))} \Rightarrow \texttt{Leaf(}x\ \texttt{DistR(}F\,r'_1\,r'_2\texttt{)}\ \sigma\texttt{)}$
    $\triangleright\ v_1 \amalg \cdots \amalg v_m \Rightarrow \textbf{let}_{1\le i\le m}\ w_i\ \textbf{be}\ \mathbb{D}\,[\![d]\!]\,v_i$
      $\textbf{in let}\ \{n_1,\dots,n_k\}\ \textbf{be}\ \{n \mid 0 < w_n\}$
      $\textbf{in let}_{1\le i\le k}\ S_i\ \textbf{be}\ (\textit{condition}\ \ \texttt{Leaf(}x\,d\,\sigma\texttt{)}\ \ \texttt{(Id(}x\texttt{)}\ \texttt{in}\,v_{n_i}\texttt{)})$
      $\textbf{in if}\ (k=1)\ \textbf{then}\ S_1\ \textbf{else}\ \oplus_{i=1}^{k}\ \texttt{(}S'_i\,w_{n_i}\texttt{)}$
  $\triangleright\ \texttt{DistI(}F\ r_1\ r_2\texttt{)} \Rightarrow \textbf{match}\ (\textit{intersection}\ \texttt{((\#f}\,r_1\texttt{)}\ \texttt{(}r_2\,\texttt{\#f))}\ v)$
    $\triangleright\ \texttt{\{}r_1\ \dots\ r_m\texttt{\}} \Rightarrow \textbf{let}_{1\le i\le m}\ w_i\ \textbf{be}\ \mathbb{D}\,[\![d]\!]\,\texttt{\{}r_i\texttt{\}}$
      $\textbf{in let}\ \{n_1,\dots,n_k\}\ \textbf{be}\ \{n \mid 0 < w_n\}$
      $\textbf{in let}_{1\le i\le k}\ S_i = \texttt{(}x\ \texttt{DistI(}F\ (r_{n_i}{-}1/2)\ r_{n_i}\texttt{)}\ \sigma\texttt{)}$
      $\textbf{in if}\ (k=1)\ \textbf{then}\ S_1\ \textbf{else}\ \oplus_{i=1}^{k}\ \texttt{(}S'_i\,w_{n_i}\texttt{)}$
    $\triangleright\ \textbf{else}$ // same as last two cases forDistR
```

(a) Conditioning Leaf

```
$\textit{condition}\ \texttt{((}S_1\ w_1\texttt{)} \oplus \cdots \oplus \texttt{(}S_m\ w_m\texttt{))}\ e ::= \textbf{let}_{1\le i\le m}\ w'_i\ \textbf{be}\ w_i\,(\mathbb{P}\,[\![S_i]\!]\,e)$
                                          $\textbf{in let}\ \{n_1,\dots,n_k\}\ \textbf{be}\ \{n \mid 0 < w'_n\}$
                                          $\textbf{in let}_{1\le i\le k}\ S'_i\ \textbf{be}\ (\textit{condition}\ S_{n_i}\ e)$
                                          $\textbf{in if}\ (k=1)\ \textbf{then}\ S'_1\ \textbf{else}\ \oplus_{i=1}^{k}\ \texttt{(}S'_i\,w'_{n_i}\texttt{)}$
```

(b) Conditioning Sum

```
$\textit{condition}\ (S_1 \otimes \cdots \otimes S_m)\ e ::= \textbf{match}\ \textit{disjoin}\ e$
                              //one $h$-dimensional hyperrectangle
                               $\triangleright\ e_1 \sqcap \cdots \sqcap e_h \Rightarrow$ //one $h$-dimensional hyperrectangle
                                  $\bigotimes_{1\le i\le m} \left[\begin{array}{l} \textbf{match}\ \{1 \le j \le h \mid (\textit{vars}\ e_j) \subset (\textit{scope}\ S_i)\} \\ \triangleright\ \{n_1,\dots,n_k\} \\ \quad \Rightarrow \textit{condition}\ S_i\ (e_{n_1} \sqcap \cdots \sqcap e_{n_k}) \\ \triangleright\ \{\} \Rightarrow S_i \end{array}\right]$
                              //$\ell \ge 2$ disjoint hyperrectangles
                               $\triangleright\ e_1 \sqcup \cdots \sqcup e_\ell \Rightarrow$
                                 $\textbf{let}_{1\le i\le \ell}\ w_i\ \textbf{be}\ \mathbb{P}\,[\![S_1 \otimes \cdots \otimes S_m]\!]\,e_i$
                                 $\textbf{in let}\ \{n_1,\dots,n_k\}\ \textbf{be}\ \{n \mid 0 < w_n\}$
                                 $\textbf{in let}_{1\le i\le k}\ S'_i\ \textbf{be}\ (\textit{condition}\,(S_1 \otimes \cdots \otimes S_m)\,e_{n_i})$
                                 $\textbf{in if}\ (k=1)\ \textbf{then}\ S'_1\ \textbf{else}\ \oplus_{i=1}^{k}\ \texttt{(}S'_i\,w_{n_i}\texttt{)}$
```

(c) Conditioning Product

Listing 7.24: Implementation of *condition* for Leaf, Sum, and Product expressions in SPE.

```
$condition_0$ Leaf ($x\,d\,\sigma$) (Id($x$) in {$rs$}) ::= match $d$
  ▷ DistR($F\,r_1\,r_2$) ⇒ match $rs$
    ▷ $r$ ⇒ match ($\mathbb{P}_0\,[\![$Leaf ($x\,d\,\sigma$)$]\!]$ (Id($x$) in {$rs$}))
      ▷ $(1, 0)$ ⇒ undefined
      ▷ else let $\tilde{F}$ be $\left(\lambda r'.\ \mathbf{1}\left[r \leq r'\right]\right)$ in DistI($\tilde{F}\,(r - 1/2)\,r$)
    ▷ $s$ ⇒ undefined
  ▷ else ⇒ $condition$ Leaf ($x\,d\,\sigma$) (Id($x$) in {$rs$})
```

(a) Conditioning Leaf

```
$condition_0$ (($S_1\ w_1$) $\oplus \cdots \oplus$ ($S_m\ w_m$)) $\left(\sqcap_{i=1}^{\ell}\right.$(Id($x_i$) in {$rs_i$})$\left.\right)$ ::=
let$_{1 \leq i \leq m}$ $(d_i, p_i)$ be $\mathbb{P}_0\,[\![S_i]\!]$ $\left(\sqcap_{i=1}^{\ell}\right.$(Id($x_i$) in {$rs_i$})$\left.\right)$
in if $\forall_{1 \leq i \leq m}.\ p_i = 0$ then undefined
   else let$_{1 \leq i \leq m}$ $w'_i$ be $w_i p_i$
      in let $d^*$ be $\min\{d_i \mid 1 \leq i \leq m, 0 < p_i\}$
      in let $\{n_1, \dots, n_k\}$ be $\{n \mid 0 < w'_n, d_i = d^*\}$
      in let$_{1 \leq i \leq k}$ $S'_i$ be $\left(condition_0\ S_{n_i}\ \left(\sqcap_{i=1}^{\ell}\right.\right.$(Id($x_i$) in {$rs_i$})$\left.\left.\right)\right)$
      in if $(k = 1)$ then $S'_1$ else $\oplus_{i=1}^{k}$ ($S'_i\ w_{n_i}$)
```

(b) Conditioning Sum

```
$condition_0$ ($S_1 \otimes \cdots \otimes S_m$) $\sqcap_{i=1}^{\ell}$ (Id($x_i$) in {$rs_i$}) ::=
  let$_{1 \leq i \leq m}$ $S'_i$ be match $\{x_1, \dots, x_m\} \cap (scope\ S_i)$
    ▷ $\{n_1, \dots, n_k\}$ ⇒ $condition_0\ S_i$ $\sqcap_{t=1}^{k}$ (Id($x_{n_t}$) in {$rs_t$})
    ▷ $\{\}$ ⇒ $S_i$
  in $S'_1 \otimes \cdots \otimes S'_m$
```

(c) Conditioning Product

Listing 7.25: Implementation of $condition_0$ for Leaf, Sum, and Product expressions in SPE.

1. Either $e \equiv$ (Id(x) ; in ; {rs}) or e is a Conjunction of such literals, where $\equiv$ here denote syntactic (not semantic) equivalence.

2. Every Var x in each literal of e is a non-transformed variable; i.e., for each Leaf expression S such that $x \in scope\, S$, it holds that $S \equiv$ Leaf ($x\,d\,\sigma$) for some d and σ.

With these requirements on e, Listing 7.25 presents the implementation of $condition_0$, leveraging the generalized density semantics from Listing 7.2e. The inference rules closely match those for standard sum-product networks, except for the fact that a density from $\mathbb{P}_0\,[\![S]\!]$ is a pair, whose first entry is the number of continuous distributions participating in the weight of the Event e which must be correctly accounted for by $condition_0$. In the reference implementation of SPPL, `condition` invokes $condition$ and `constrain` invokes $condition_0$. Analogously to the `prob` query, which returns probabilities using the distribution semantics $\mathbb{P}$ in Listing 7.1e, SPPL also includes the `density` query, which returns densities using the generalized semantics $\mathbb{P}_0$ in Listing 7.2e.

$$\frac{d \Uparrow D(E), t_1 \Uparrow E_1, \ldots, t_m \Uparrow E_m}{\begin{array}{r}(x\ d\ \{x \mapsto \mathtt{Id}(x), x_1 \mapsto t_1, \ldots, x_m \mapsto t_m\}) \\ \to_{\mathrm{SPPL}}\ x \,\mathtt{\sim}\, D(E)\,;x_1 = E_1\,;\ldots\,;x_m = E_m\end{array}} \qquad (\textsc{Leaf})$$

$$\frac{S_1 \to_{\mathrm{SPPL}} C_1, \ldots, S_m \to_{\mathrm{SPPL}} C_m}{\otimes_{i=1}^{m} S_i \to_{\mathrm{SPPL}} C_1\,;\ldots\,;C_m} \qquad (\textsc{Product})$$

$$\frac{S_1 \to_{\mathrm{SPPL}} C_1, \ldots, S_m \to_{\mathrm{SPPL}} C_m; \quad \text{where } b \text{ is a fresh } \mathsf{Var}}{\oplus_{i=1}^{m} (S_i\ w_i) \to_{\mathrm{SPPL}} \left[\begin{array}{l} b \,\mathtt{\sim}\, \mathtt{choice(\{'1':}w_1\mathtt{,}\ldots\mathtt{,'}m\mathtt{':}w_m\mathtt{\})} \\ \mathtt{if}\ (b \mathtt{==} \mathtt{'1'})\ \{C_1\} \\ \mathtt{elif}\ldots \\ \mathtt{elif}\ (b \mathtt{==} \mathtt{'}m\mathtt{'})\ \{C_m\} \end{array}\right]} \qquad (\textsc{Sum})$$

Listing 7.26: Translating an element of SPE (Listing 7.1f) to an SPPL command C (Listing 7.5).

7.E Translating Sum-Product Expressions to SPPL

Listing 7.6 in Section 7.4 presents the relation $\to_{\mathsf{SPE}}$, which translates $C \in$ Command (i.e., SPPL source syntax, Listing 7.5) to a sum-product expression $S \in$ SPE in the core language (Listing 7.1). Listing 7.26 defines a relation $\to_{\mathrm{SPPL}}$ that reverses the $\to_{\mathsf{SPE}}$ relation by converting expression $S \in$ SPE to $C \in$ Command. Briefly, (i) a Product is converted to a sequence Command; (ii) a Sum is converted to an `if`-`else` Command; and (iii) a Leaf is converted to a sequence of sample (~) and transform (=).

The symbol $\Uparrow$ (whose formal definition is omitted) in the Leaf rule converts semantic elements such as $d \in$ Distribution and $t \in$ Transform from the core calculus to an SPPL expression $E \in$ Expr in a straightforward way, e.g.,

$$(\mathtt{Poly(Id(X)}\ 1\ 2\ 3\mathtt{))} \Uparrow (\mathtt{1\ +\ 2*X\ +\ 3*X**2}). \tag{7.53}$$

Chaining $\to_{\mathsf{SPE}}$ (Listing 7.6) and $\to_{\mathrm{SPPL}}$ (Listing 7.26) for a given SPPL program does not preserve either SPPL or core syntax, that is[1]

$$((C \to^{*}_{\mathsf{SPE}} S) \to^{*}_{\mathrm{SPPL}} C') \qquad \text{does not imply } C = C' \tag{7.54}$$

$$((C \to^{*}_{\mathsf{SPE}} S) \to^{*}_{\mathrm{SPPL}} C') \to^{*}_{\mathsf{SPE}} S' \qquad \text{does not imply } S = S'. \tag{7.55}$$

Rather, $\to_{\mathrm{SPPL}}$ is a semantics-preserving inverse of $\to_{\mathsf{SPE}}$, in the sense that for all $e \in$ Event

$$((C \to^{*}_{\mathsf{SPE}} S) \to^{*}_{\mathrm{SPPL}} C') \to^{*}_{\mathsf{SPE}} S' \implies \mathbb{P}\,[\![S]\!]\,e = \mathbb{P}\,[\![S']\!]\,e. \tag{7.56}$$

Eq. (7.56) establishes a formal semantic correspondence between the SPPL language and the class of sum-product expressions. Each SPPL program admits a representation as an SPE, and each valid element of SPE that satisfies conditions (C1)–(C5) corresponds to some SPPL program.

The translation strategy in Listing 7.26 makes it possible to synthesize SPPL programs using a range of techniques for learning the structure and parameters of sum-product networks [Gens and Domingos, 2013, Vergari et al., 2019, Trapp et al., 2019]. SPPL integrates with these techniques by provides users a uniform representation language for sum-product networks as generative programs in a formal PPL.

[1] The symbol $C \to^{*}_{\mathsf{SPE}} S$ means $\langle C, S_{\varnothing}\rangle$ translates to S in zero or more steps of $\to_{\mathsf{SPE}}$, where $S_{\varnothing}$ is an "empty" SPE used for the initial translation step, and similarly for $\to^{*}_{\mathrm{SPPL}}$.

SPPL also lets users extend these learned programs with modeling extensions supported by the core calculus (Listing 7.2), such as predicates for decision trees and numeric transformations. Finally, SPPL, delivers exact answers to an extended set of probabilistic inference queries that involve predicates and mixed-type base random variables, for example, which other libraries for sum-product networks such SPFlow [Molina et al., 2020] do not support.

Part III

Statistical Estimation and Testing via Dynamic Program Analysis

Chapter 8

Estimators of Entropy and Information

> Seldom do more than a few of nature's secrets give way at one time.
>
> Claude E. Shannon

The Shannon entropy $H(Y) ::= -\mathbb{E}[\log p(Y)]$ of a random variable Y is a fundamental information-theoretic quantity that characterizes the inherent randomness contained in Y [Shannon, 1948]. Entropy is a basic building block of several information-theoretic quantities and has received widespread attention in information theory [Cover and Thomas, 1991] statistics [MacKay, 2003], as well as in diverse application domains that include quantitative finance [Zhou et al., 2013], systems biology [Borowska, 2016], computational neuroscience [Rieke et al., 1997] hydraulic engineering [Chiu, 1987], and image processing [Pun, 1980]. This chapter presents a new computational technique that addresses the fundamental problem of estimating the Shannon entropy $H(Y)$ in situations where its marginal distribution involves an intractable multidimensional integral over a known joint probability distribution:

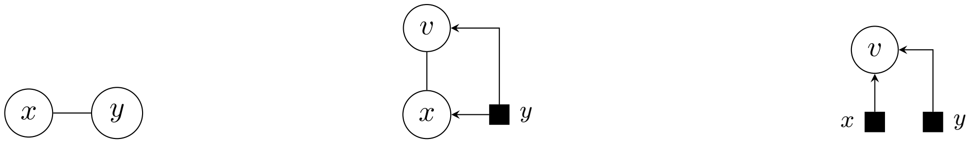
$$p(y) = \int_{\mathcal{X}} p(x, y)\, \mathrm{d}x \qquad (y \in \mathcal{Y}). \tag{8.1}$$

In Eq. (8.1), the term $p(x, y)$ refers to a probabilistic generative model that can be sampled from and whose joint density can be computed pointwise, as is common in a broad class of probabilistic systems including Bayesian networks [Pearl, 1988], deep generative models [Kingma and Welling, 2019], generative probabilistic programs, and all four probabilistic DSLs from Part I of this thesis. This class of queries are generally not tractable to solve exactly, even using the sum-product representations in SPPL from Chapter 7. The key challenge is that the expression

$$H(Y) = -\int_{\mathcal{Y}} \log \left[\int_{\mathcal{X}} p(x, y)\, \mathrm{d}x \right] p(y)\, \mathrm{d}y \tag{8.2}$$

contains an intractable integral inside the logarithm, which rules out the unbiased simple Monte Carlo estimator $-1/n \sum_{i=1}^{n} \log p(Y_i)$ (for $Y_i \sim p(y)$, $1 \le i \le n$).

To address these challenges, the chapter presents a new class of *estimators of entropy via inference* (EEVI) that return interval estimates of doubly intractable entropies as in Eq. (8.2). EEVI uses auxiliary-variable importance sampling constructs similar to those from pseudo-marginal methods [Andrieu and Roberts, 2009] to first compute unbiased estimates of the intractable quantities $p(y)$ and $1/p(y)$ for the inner integral. Under the log transform, these estimates become lower and upper bounds of $\log p(y)$, which are then embedded in a simple Monte Carlo estimator for the outer integral to form an interval estimate of $H(Y)$. In the limit of computation, the interval width can be driven to zero,



(a) Target Distribution $p(x, y)$ (b) Proposal Distribution $q(v, x; y)$ (c) Auxiliary Proposal Distribution $r(v; x, y)$

Figure 8.1: Target, proposal, and auxiliary proposal distributions. These distributions are inputs to the EEVI Algorithms 8.1 and 8.2, which provide compute bounds on $H(Y) = -\mathbb{E}[\log p(Y)]$, $Y \sim p(y)$.

squeezing the true entropy value at a rate that depends on the quality of the importance sampling proposal. The main contribution of this chapter is a family of entropy estimators that

(C1) Apply to arbitrary random variables in any probability distribution that can be sampled from and whose full joint density is tractable; no marginals or conditionals need to be tractable.

(C2) Guarantee upper and lower bounds in expectation, which can be composed (Figure 8.2) to squeeze many information quantities, e.g., Eqs. (8.12)–(8.16).

(C3) Leverage a broad family of proposal distributions that includes both variational and Monte Carlo inference for increasing accuracy as a function of computational effort.

The rest of the chapter is organized as follows: Sections 8.1 and 8.2 describe EEVI and explains how interval estimates of entropy can be composed to form interval estimates of several other information quantities, such as conditional mutual information and interaction information. Section 8.3 presents theoretical properties of importance sampling-based estimators of log marginal probabilities of the form given in Eq. (8.1), and gives examples of inference-based variational and Monte Carlo auxiliary-variable proposals to deliver accurate upper and lower bounds. Section 8.4 illustrates the scalability and efficacy of EEVI for two optimal-design tasks in a probabilistic expert system for diagnosing liver disorders and a dynamic model of carbohydrate metabolism in diabetic patients. Section 8.5 discusses related work.

8.1 Overview of EEVI

Let $p(z_1, \dots, z_d)$ be a d-dimensional probability density (with respect to an appropriate σ-finite measure) for which it is possible to sample $Z ::= (Z_1, \dots, Z_d) \sim p(z_1, \dots, z_d)$ and evaluate density values. Let $A \subset \{1, \dots, d\}$ be a subset of indexes and let $Y ::= \{Z_i, i \in A\}$ and $X ::= \{Z_i, i \notin A\}$ be the corresponding partition of variables in Z. The goal is to estimate the marginal entropy $H(Y)$ as defined in Eq. (8.2), where $\mathcal{Y}$ and $\mathcal{X}$ are the sets in which Y and X take values, respectively. As the partition A is arbitrary, neither the marginal densities $p(x)$ and $p(y)$ nor conditional densities $p(x \mid y)$ and $p(y \mid x)$ are assumed to tractable, which poses a key challenge for estimating $H(Y)$.

Suppose it is possible to compute two measurable real functions $w, w' : \mathcal{U} \times \mathcal{Y} \to \mathbb{R}$ such that for some random variables U, U' taking values in a set $\mathcal{U}$ and all $y \in \mathcal{Y}$ except for a p-measure zero set,

$$\mathbb{E}[w(U, y)] = p(y) \qquad \mathbb{E}\left[w'(U', y)\right] = 1/p(y). \tag{8.3}$$

If w and w' are nonnegative a.e., then concavity of log and Jensen's inequality gives bounds

$$\mathbb{E}[\log w(U, y)] \leq \log \mathbb{E}[w(U, y)] = \log p(y), \tag{8.4}$$

$$\mathbb{E}\left[\log w'(U', y)\right] \leq \log \mathbb{E}\left[w'(U', y)\right] = -\log p(y), \tag{8.5}$$

which together imply that

$$\mathbb{E}\left[\log w(U,y)\right] \leq \log p(y) \leq \mathbb{E}\left[-\log w'(U',y)\right]. \tag{8.6}$$

If the real functions $y \mapsto \mathbb{E}\left[\log w(U,y)\right]$ and $y \mapsto -\mathbb{E}\left[\log w'(U',y)\right]$ defined on $\mathcal{Y}$ are themselves both measurable then monotonicity of expectation gives

$$\mathbb{E}\left[\log w(U,Y)\right] \leq \mathbb{E}\left[\log p(Y)\right] \leq \mathbb{E}\left[-\log w'(U',Y)\right] \tag{8.7}$$

The expectations in (8.7) that squeeze $\mathbb{E}\left[\log p(Y)\right]$ can be estimated unbiasedly via simple Monte Carlo:

$$\mathcal{L}_{n,m} ::= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{m}\sum_{j=1}^{m}\log w(U_{ij},Y_i)\right], \tag{8.8}$$

$$\mathcal{T}_{n,m} ::= \frac{1}{n}\sum_{i=1}^{n}\left[\frac{1}{m}\sum_{j=1}^{m}-\log w'(U'_{ij},Y'_i)\right], \tag{8.9}$$

where Y_i, Y'_i are identically distributed to Y; U_{ij} identically to U; and U'_{ij} identically to U' ($i = 1,\dots,n; j = 1,\dots,m$).

Letting $\check{H}_Y ::= -\mathcal{T}_{n,m}$ and $\hat{H}_Y ::= -\mathcal{L}_{n,m}$, Eq. (8.7) implies that the means of $\check{H}_Y$ and $\hat{H}_Y$ satisfy

$$\mathbb{E}[\check{H}_Y] \leq H(Y) \leq \mathbb{E}[\hat{H}_Y]. \tag{8.10}$$

If using i.i.d. samples in Eqs. (8.8) and (8.9), under mild conditions the central limit theorem and Eq. (8.10) imply the interval estimator $[\check{H}_Y, \hat{H}_Y]$ has coverage probability

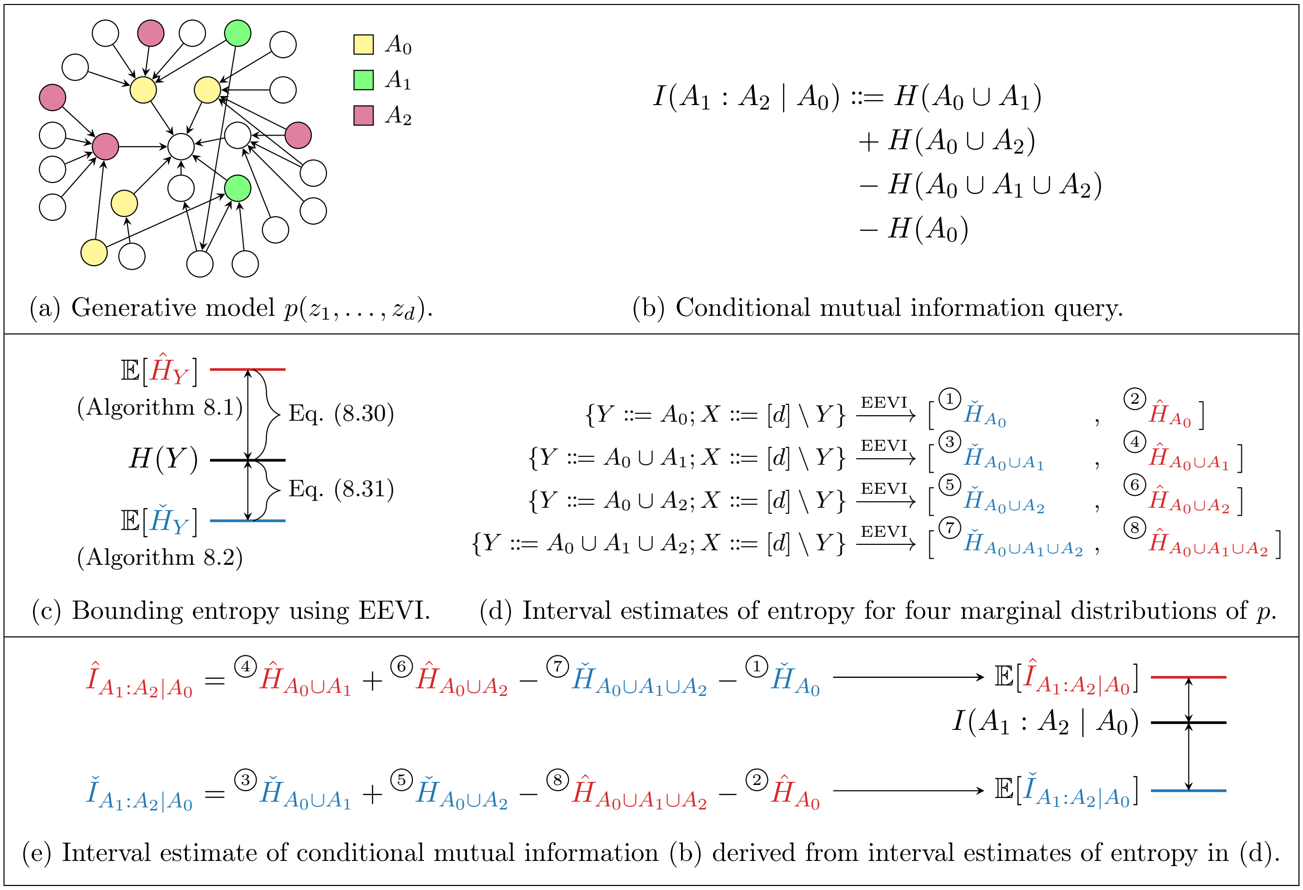
$$\Pr[\check{H}_Y \leq H(Y) \leq \hat{H}_Y] \approx \Phi\left(\sqrt{t}\check{B}/\check{\sigma}\right)\Phi\left(\sqrt{t}\hat{B}/\hat{\sigma}\right), \tag{8.11}$$

where Φ is the standard normal cumulative distribution function; $t = nm$; $\check{B} ::= \mathbb{E}\left[-\log w'(U',Y)\right] - \mathbb{E}\left[\log p(Y)\right]$ and $\hat{B} ::= \mathbb{E}\left[\log p(Y)\right] - \mathbb{E}\left[\log w(U,Y)\right]$ are the biases in Eq. (8.10); and $\check{\sigma}$ and $\hat{\sigma}$ are the standard deviations of the random variables $\log w'(U',Y)$ and $\log w(U,Y)$, respectively.

The preceding discussion has assumed access to functions w and w' that satisfy Eq. (8.3). Obtaining functions that satisfy these properties, however, is itself a challenging inference problem that must be solved for computing the estimators (8.8) and (8.9) Section 8.3 revisits this problem and shows how to construct functions w and w' using importance sampling-based techniques that deliver unbiased estimates of marginal likelihoods and their inverses.

8.2 Extending Entropy Bounds to Information-Theoretic Quantities

The lower and upper bounds on entropy in Eq. (8.10) can be composed to obtain bounds on several derived information-theoretic quantities that measure the degree of relationship between arbitrary subcollections of variables in a model, possibly conditioned on others. As discussed in Section 8.5, the class of queries that can be solved using these compositions is substantially more general than existing model-based estimators in the literature, since no assumptions about the tractability of marginals or conditionals are needed. Recall that the target generative model $p(z_1,\dots,z_d)$ has d variables. Define the shorthand notation $H(A) ::= H(\{Z_i, i \in A\})$ for $A \subset [d]$. By adding and subtracting upper and lower bounds on $H(A)$ it is possible to obtain interval estimators of the following quantities defined in terms of the Shannon entropy (in all cases, setting $A_0 ::= \varnothing$ gives unconditional versions):



🟨 A_0
🟩 A_1
🟥 A_2
$I(A_1 : A_2 \mid A_0) ::= H(A_0 \cup A_1) + H(A_0 \cup A_2) - H(A_0 \cup A_1 \cup A_2) - H(A_0)$
$\mathbb{E}[\hat{H}_Y]$
(Algorithm 8.1)
Eq. (8.30)
$H(Y)$
Eq. (8.31)
$\mathbb{E}[\check{H}_Y]$
(Algorithm 8.2)
$\{Y ::= A_0; X ::= [d] \setminus Y\} \xrightarrow{\text{EEVI}} [①\check{H}_{A_0}, ②\hat{H}_{A_0}]$
$\{Y ::= A_0 \cup A_1; X ::= [d] \setminus Y\} \xrightarrow{\text{EEVI}} [③\check{H}_{A_0 \cup A_1}, ④\hat{H}_{A_0 \cup A_1}]$
$\{Y ::= A_0 \cup A_2; X ::= [d] \setminus Y\} \xrightarrow{\text{EEVI}} [⑤\check{H}_{A_0 \cup A_2}, ⑥\hat{H}_{A_0 \cup A_2}]$
$\{Y ::= A_0 \cup A_1 \cup A_2; X ::= [d] \setminus Y\} \xrightarrow{\text{EEVI}} [⑦\check{H}_{A_0 \cup A_1 \cup A_2}, ⑧\hat{H}_{A_0 \cup A_1 \cup A_2}]$
$\hat{I}_{A_1:A_2|A_0} = ④\hat{H}_{A_0 \cup A_1} + ⑥\hat{H}_{A_0 \cup A_2} - ⑦\check{H}_{A_0 \cup A_1 \cup A_2} - ①\check{H}_{A_0} \longrightarrow \mathbb{E}[\hat{I}_{A_1:A_2|A_0}]$
$I(A_1 : A_2 \mid A_0)$
$\check{I}_{A_1:A_2|A_0} = ③\check{H}_{A_0 \cup A_1} + ⑤\check{H}_{A_0 \cup A_2} - ⑧\hat{H}_{A_0 \cup A_1 \cup A_2} - ②\hat{H}_{A_0} \longrightarrow \mathbb{E}[\check{I}_{A_1:A_2|A_0}]$

(a) Generative model $p(z_1, \dots, z_d)$.

(b) Conditional mutual information query.

(c) Bounding entropy using EEVI.

(d) Interval estimates of entropy for four marginal distributions of p.

(e) Interval estimate of conditional mutual information (b) derived from interval estimates of entropy in (d).

Figure 8.2: Composing interval estimators of entropy to obtain bounds on derived information measures.

- Conditional Entropy [Shannon, 1948]

$$H(A_1 \mid A_2) ::= H(A_1 \cup A_2) - H(A_2) \tag{8.12}$$

- Conditional Mutual Information [Shannon, 1948]

$$I(A_1 : A_2 \mid A_0) ::= H(A_1 \mid A_0) - H(A_1 \mid A_2, A_0) \tag{8.13}$$

- Conditional Total Correlation [Watanabe, 1960]

$$C(\{A_i\}_{i=1}^n \mid A_0) ::= \sum_{i=1}^{n} H(A_i \mid A_0) - H(\bigcup_{i=1}^{n} A_i \mid A_0) \tag{8.14}$$

- Conditional Interaction Information [Ting, 1962]

$$T(\{A_i\}_{i=1}^n \mid A_0) ::= \sum_{S \subset [n]} -1^{|S|} H(\cup_{i \in S} A_i \mid A_0) \tag{8.15}$$

- Conditional Dual Correlation [Han, 1978]

$$D(\{A_i\}_{i=1}^n \mid A_0) ::= H(\cup_{i=1}^n A_i \mid A_0) - \sum_{i=1}^{n} H(A_i \mid \cup_{j=0, j \neq i}^n A_i). \tag{8.16}$$

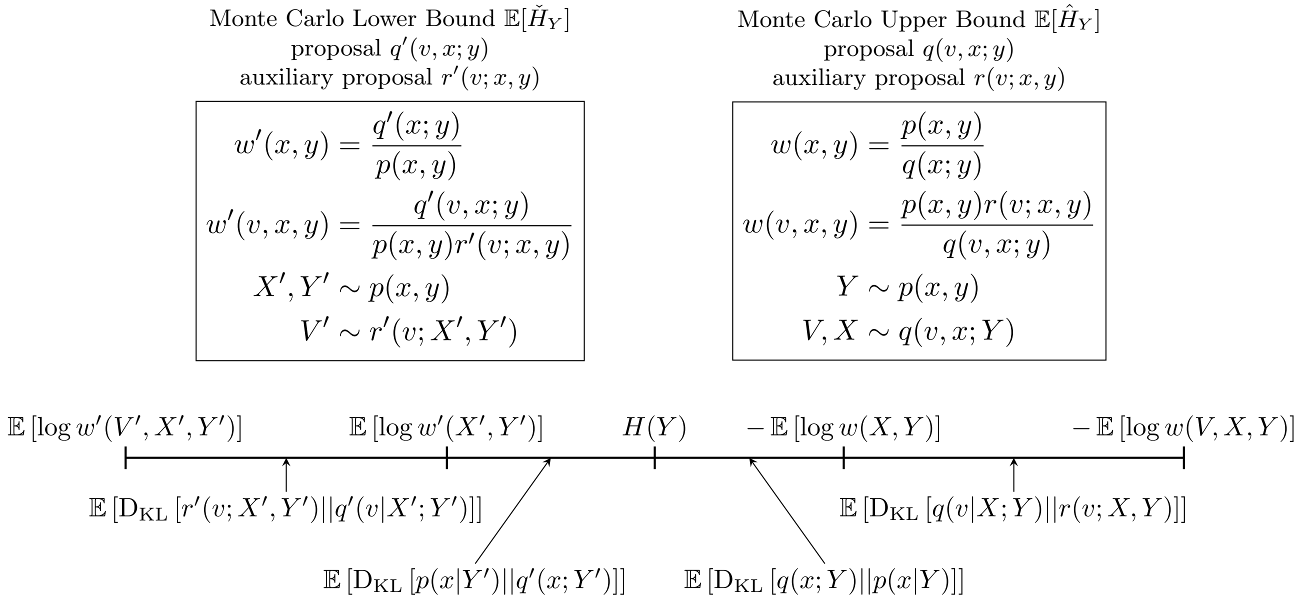

Monte Carlo Lower Bound $\mathbb{E}[\check{H}_Y]$
proposal $q'(v,x;y)$
auxiliary proposal $r'(v;x,y)$
$w'(x,y) = \frac{q'(x;y)}{p(x,y)}$
$w'(v,x,y) = \frac{q'(v,x;y)}{p(x,y)r'(v;x,y)}$
$X',Y' \sim p(x,y)$
$V' \sim r'(v;X',Y')$
Monte Carlo Upper Bound $\mathbb{E}[\hat{H}_Y]$
proposal $q(v,x;y)$
auxiliary proposal $r(v;x,y)$
$w(x,y) = \frac{p(x,y)}{q(x;y)}$
$w(v,x,y) = \frac{p(x,y)r(v;x,y)}{q(v,x;y)}$
$Y \sim p(x,y)$
$V,X \sim q(v,x;Y)$
$\mathbb{E}\left[\log w'(V',X',Y')\right]$
$\mathbb{E}\left[\log w'(X',Y')\right]$
$H(Y)$
$-\mathbb{E}\left[\log w(X,Y)\right]$
$-\mathbb{E}\left[\log w(V,X,Y)\right]$
$\mathbb{E}\left[\mathrm{D_{KL}}\left[r'(v;X',Y')||q'(v|X';Y')\right]\right]$
$\mathbb{E}\left[\mathrm{D_{KL}}\left[q(v|X;Y)||r(v;X,Y)\right]\right]$
$\mathbb{E}\left[\mathrm{D_{KL}}\left[p(x|Y')||q'(x;Y')\right]\right]$
$\mathbb{E}\left[\mathrm{D_{KL}}\left[q(x;Y)||p(x|Y)\right]\right]$

Figure 8.3: Estimation gaps for upper and lower bounds $\hat{H}_Y$ and $\check{H}_Y$ from EEVI Algorithms 8.1 and 8.2.

8.3 Sampling Bounds on Log Marginal Probabilities

Importance Sampling in Log Space Implementing the estimators $\hat{H}_Y, \check{H}_Y$ in Eq. (8.10) requires functions w, w' that satisfy Eq. (8.3). The starting point to obtaining these functions is an identity from importance sampling. Let h and g be two probability densities on a common set $\mathcal{X}$ such that h is absolutely continuous with respect to g (written $h \ll g$); i.e., $\int_A g(x)\,\mathrm{d}x = 0 \implies \int_A h(x)\,\mathrm{d}x = 0$ for all measurable A. Suppose $h(x) = \tilde{h}(x)/Z_h$, $g(x) = \tilde{g}(x)/Z_g$, where Z_h and Z_g are unknown normalizing constants needed to ensure that the densities are normalized. Then, for $X \sim g$,

$$\mathbb{E}\left[\tilde{h}(X)/\tilde{g}(X)\right] = Z_h/Z_g. \tag{8.17}$$

Under log transform, the ratio (8.17) is a lower bound on $\log(Z_h/Z_g)$ in expectation, with a gap equal to the KL divergence from h to g:

$$\mathbb{E}\left[\log\left(\tilde{h}(X)/\tilde{g}(X)\right)\right] = \log\left(Z_h/Z_g\right) - \mathrm{D_{KL}}\left[g||h\right] \tag{8.18}$$

Eq. (8.18) does not require $h \ll g$. However, the expectation is well defined only if $g \ll h$ and is finite only if $\mathrm{D_{KL}}\left[g||h\right] < \infty$. Moreover, the variance

$$\mathrm{Var}\left[\log(\tilde{h}(X)/\tilde{g}(X))\right] = \mathbb{E}\left[\log^2(h(X)/g(X))\right] - (\mathrm{D_{KL}}\left[g||h\right])^2 \tag{8.19}$$

is finite only if $\mathrm{D_{KL}}\left[g||h\right] < \infty$ and $\log^2(h(X)/g(X))$ has finite expectation. Applying Markov's inequality to Eq. (8.17) gives a right tail bound for $\log\tilde{h}(X)/\tilde{g}(X)$:

$$\Pr\left[\tilde{h}(X)/\tilde{g}(X) \geq e^t(Z_h/Z_g)\right] \leq e^{-t} \implies \Pr\left[\log(\tilde{h}(X)/\tilde{g}(X)) \geq t + \log(Z_h/Z_g)\right] \leq e^{-t} \tag{8.20}$$

Algorithm 8.1 EEVI upper bound.

```
Require: Target distribution $p(x,y)$;
         Proposal distribution $q(v,x;y)$;
         Aux. proposal distribution $r(v;x,y)$;
         Number of samples $n$, $m$.
Ensure: Monte Carlo upper bound $\hat{H}_Y$ on $H(Y)$
for $i = 1 \dots n$ do
    $(\widetilde{X}, Y) \sim p(x,y)$                ▷ $\widetilde{X}$ is discarded
    for $j = 1 \dots m$ do
        $(V, X) \sim q(v,x;Y)$
        $u_j \leftarrow \log \dfrac{p(X,Y)r(V;X,Y)}{q(V,X;y)}$
    $t_i \leftarrow \frac{1}{m}\sum_{j=1}^{m} u_j$
return $-\frac{1}{n}\sum_{i=1}^{n} t_i$
```

Algorithm 8.2 EEVI lower bound.

```
Require: Target distribution $p(x,y)$;
         Proposal distribution $q'(v,x;y)$;
         Aux. proposal distribution $r'(v;x,y)$;
         Number of samples $n$, $m$.
Ensure: Monte Carlo lower bound $\check{H}_Y$ on $H(Y)$
for $i = 1 \dots n$ do
    $(X'_1, Y) \sim p(x,y)$
    $(X'_{2:m}) \sim$ Markov chain targeting $p(x \mid Y)$
                starting at $X'_1$ (optional step)
    for $j = 1 \dots m$ do
        $V' \sim r'(v; X'_j, Y)$
        $u_j \leftarrow \log \dfrac{q'(V', X'_j; Y)}{p(X'_j, Y)r'(V'; X'_j, Y)}$
    $t_i \leftarrow -\frac{1}{m}\sum_{j=1}^{m} u_j$
return $-\frac{1}{n}\sum_{i=1}^{n} t_i$
```

for any $t > 0$. The mean absolute deviation satisfies

$$\mathbb{E}\left[\left|\log(\tilde{h}(X)/\tilde{g}(X)) - \mu\right|\right] \leq 2 + 2\,\mathrm{D}_{\mathrm{KL}}\left[g||h\right], \tag{8.21}$$

where $\mu ::= \mathbb{E}[\log(\tilde{h}(X)/\tilde{g}(X))]$. This upper bound is two plus twice the bias in Eq. (8.18), which decreases as g more closely matches h.

Interval Estimators of Entropy Recalling the distribution $p(x,y)$ from Section 8.1, suppose that $q(x;y)$ and $q'(x;y)$ are normalized proposal densities over $\mathcal{X}$ parameterized by $\mathcal{Y}$. From Eq. (8.17), for fixed $y \in \mathcal{Y}$, setting $\tilde{h}(x) = p(x,y)$ and $g(x) = q(x;y)$ gives an unbiased estimate of $Z_h \equiv p(y)$;

$$\mathbb{E}\left[\frac{p(X,y)}{q(X;y)}\right] = p(y). \tag{8.22}$$

Similarly, setting $h(x) = q'(x;y)$ and $\tilde{g}(x) = p(x,y)$ gives an unbiased estimate of $1/Z_g \equiv 1/p(y)$:

$$\mathbb{E}\left[\frac{q'(X';y)}{p(X',y)}\right] = 1/p(y). \tag{8.23}$$

These expressions provide the estimators needed for Eq. (8.3), by defining $w(x,y) ::= p(x,y)/q(x;y)$ and $w'(x,y) ::= q'(x;y)/p(x,y)$ and letting $X \sim q(x;y)$ be U and $X' \sim p(x \mid y)$ be U'. Then Eq. (8.8) yields the Monte Carlo upper bound $\hat{H}_Y$ in Eq. (8.10); and Eq. (8.9) yields the Monte Carlo lower bound $\check{H}_Y$ in Eq. (8.10). While sampling $X' \sim p(x \mid y)$ given a fixed value y is generally intractable under the stated assumptions on p, since $H(Y)$ is the expectation of random values $-\log p(Y)$ for $Y \sim p$, it suffices to use joint samples $(X', Y') \sim p(x,y)$ to obtain the lower bound. In particular,

$$\begin{aligned} \int_{\mathcal{X}\times\mathcal{Y}} \log\left[q'(x;y)/p(x,y)\right] p(x,y)\,\mathrm{d}x\,\mathrm{d}y &= \int_{\mathcal{Y}} \left[\int_{\mathcal{X}} \log\left[q'(x;y)/p(x,y)\right] p(x \mid y)\,\mathrm{d}x\right] p(y)\,\mathrm{d}y && (8.24) \\ &\leq \int_{\mathcal{Y}} \left[-\log p(y)\right] p(y)\,\mathrm{d}y = H(Y). && (8.25) \end{aligned}$$

The second line follows from Schervish [1995, Theorems B.46 and B.52] and third line from Eq. (8.18). Given an initial sample $(X'_1, Y) \sim p(x, y)$, an additional $m-1$ samples $X'_2, \dots, X'_m$ from $p(x \mid Y')$ to use for $\mathcal{T}_{n,m}$ in Eq. (8.9) can be obtained by simulating a Markov chain initialized at X'_1 that leaves $p(x \mid Y')$ invariant. These samples reduce $\operatorname{Var}[\mathcal{T}_{n,m}]$ if and only if $\Pr[X'_i \neq X'_1] > 0$ for some i.

8.3.1 Constructing Accurate Proposals

Estimators of normalizing constants and their inverses in direct space as in Eqs. (8.22) and (8.23) can suffer from notoriously high variance, especially when using proposals that do not closely match the target [Neal, 2008]. However, Eq. (8.20) suggests that log space estimators can be more stable. By Eqs. (8.18) and (8.21), the quality of the entropy bounds obtained via importance sampling depends on constructing proposal distributions $q(x; y)$ and $q'(x; y)$ that have small expected biases (over $Y \sim p$):

$$\mathbb{E}[\hat{H}_Y] - H(Y) = \mathbb{E}\left[\mathrm{D}_{\mathrm{KL}}\left[q(x;Y) || p(x \mid Y)\right]\right], \tag{8.26}$$

$$H(Y) - \mathbb{E}[\check{H}_Y] = \mathbb{E}\left[\mathrm{D}_{\mathrm{KL}}\left[p(x \mid Y) || q'(x;Y)\right]\right]. \tag{8.27}$$

Two families of approaches that leverage probabilistic inference algorithms to construct accurate proposals are described next.

Amortized Variational Inference Proposals $q(x; y)$ and $q'(x; y)$ for use in Eqs. (8.22) and (8.23) can be trained from a dataset $\{(x_i, y_i)\}_{i=1}^n$ simulated from p. In particular, variational inference can be used to learn recognition networks $q_\phi(x; y)$ and $q'_\varphi(x; y)$ that each specify a family of distributions over $\mathcal{X}$, parametrized by $y \in \mathcal{Y}$ and ϕ, φ, respectively. Training q via "exclusive" amortized variational inference, as in variational autoencoders [Kingma and Welling, 2013], minimizes both the bias and an upper bound on the mean absolute deviation (MAD) of $\check{H}_Y$ in Algorithm 8.1. Similarly, training q' via "inclusive" amortized variational inference, as in the "sleep" phase of the wake-sleep algorithm [Hinton et al., 1995], minimizes both the bias and an upper bound on the MAD of $\hat{H}_Y$ in Algorithm 8.2. Thus, EEVI can leverage advances in training neural networks via stochastic gradient descent to improve estimation accuracy. Refer to Figure 8.4 for an example of variationally trained proposals.

Auxiliary Variable Monte Carlo Another approach to constructing accurate proposals, which can be composed with variational learning [Salimans et al., 2015], is Monte Carlo methods such as annealed importance sampling [AIS; Neal, 2001] and sequential Monte Carlo [SMC; Del Moral et al., 2006] that define proposal distributions on extended state-spaces. These methods yield state-of-the-art estimates of normalizing constants. In particular, the proposals $q(v, x; y)$ are defined over an extended space $\mathcal{V} \times \mathcal{X}$, such that the marginal density $q(x; y) = \int_{\mathcal{V}} q(v, x; y)\, \mathrm{d}v$ is an integral auxiliary random variables v sampled by q. As the ratios in Eqs. (8.22) and (8.23) can no longer be evaluated, a tractable "auxiliary proposal distribution" $r(v; x, y)$ over $\mathcal{V}$ parameterized by $\mathcal{X} \times \mathcal{Y}$ (Figure 8.1) is used instead, such that

$$w(v, x, y) ::= \left[p(x, y) r(v; x, y)\right] / q(v, x; y), \tag{8.28}$$

$$w'(v, x, y) ::= q'(v, x; y) / \left[p(x, y) r'(v; x, y)\right]. \tag{8.29}$$

By Eq. (8.17), these weight functions satisfy Eq. (8.3) by letting $(V, X) \sim q(v, x; y)$ serve as U and $(X', V') \sim p(x \mid y) r(v; x, y)$ as U'. From Eq. (8.18), the gap when lower bounding $\log p(y)$ using these extended weights w and w' must account for the accuracy of the auxiliary proposals r and r'

$$\mathbb{E}[\hat{H}_Y] - H(Y) = \mathbb{E}\left[\mathrm{D}_{\mathrm{KL}}\left[q(x;Y) || p(x|Y)\right]\right] + \mathbb{E}\left[\mathrm{D}_{\mathrm{KL}}\left[q(v \mid X;Y) || r(v;X,Y)\right]\right], \tag{8.30}$$

$$H(Y) - \mathbb{E}[\check{H}_Y] = \mathbb{E}\left[\mathrm{D}_{\mathrm{KL}}\left[p(x \mid Y) || q'(x;Y)\right]\right] + \mathbb{E}\left[\mathrm{D}_{\mathrm{KL}}\left[r(v';X',Y) || q'(v \mid X';Y)\right]\right]. \tag{8.31}$$

Algorithm 8.3 EEVI upper bound using SIR proposal without auxiliary variables.

```
Require: Target distribution $p(x, y)$
         Base proposal distribution $q_0(x; y)$
         Number of samples $n$, $m$, $P$
Ensure: Monte Carlo upper bound $\hat{H}_Y$ on $H(Y)$
for $i = 1 \dots n$ do
    $(\widetilde{X}, Y) \sim p(x, y)$
    for $j = 1 \dots m$ do
        for $k = 1 \dots P$ do
            $X \sim q_0(x; Y)$
            $\xi_k \leftarrow \dfrac{p(X, Y)}{q_0(X; Y)}$
        $u_j \leftarrow \log \left[ \frac{1}{P} \sum_{k=1}^{P} \xi_k \right]$
    $t_i \leftarrow \frac{1}{m} \sum_{j=1}^{m} u_j$
return $-\frac{1}{n} \sum_{i=1}^{n} t_i$
```

Algorithm 8.4 EEVI lower bound using SIR proposal without auxiliary variables.

```
Require: Target distribution $p(x, y)$
         Base proposal distribution $q_0(x; y)$
         Number of samples $n$, $m$, $P$
Ensure: Monte Carlo lower bound $\hat{H}_Y$ on $H(Y)$
for $i = 1 \dots n$ do
    $(X_1, Y) \sim p(x, y)$
    $(X_{2:m}) \sim$ Markov chain targeting $p(x \mid Y)$
                    starting at $X'_1$ (optional step)
    for $j = 1 \dots m$ do
        $X'_1 \leftarrow X_j$
        for $k = 2 \dots P$ do
            $X'_k \sim q_0(x; Y)$
        for $k = 1 \dots P$ do
            $\xi_k \leftarrow \dfrac{p(X'_k, Y)}{q_0(X'_k; y)}$
        $u_j \leftarrow \log \left[ \dfrac{1}{\frac{1}{P} \sum_{k=1}^{P} \xi_k} \right]$
    $t_i \leftarrow -\frac{1}{m} \sum_{j=1}^{m} u_j$
return $-\frac{1}{n} \sum_{i=1}^{n} t_i$
```

Figure 8.3 shows the estimation gaps of EEVI when using the standard weights (8.26) and (8.27) and extended weights (8.30) and (8.31) Algorithms 8.1 and 8.2 give interval estimators $[\check{H}_Y, \hat{H}_Y]$ that implement Eqs. (8.8) and (8.9) with extended weights w and w' (8.28) and (8.29). Standard proposals without auxiliary variables are a special case, where $\mathcal{V} = \{\omega\}$ is a singleton and $r(v; x, y) = \delta(v; \omega)$.

Example 8.1 (SIR Proposals)**.** To fix ideas, consider a base proposal $q_0(x; y)$ that has no auxiliary variables, which may have been hand constructed or trained variationally. The proposal q_0 can be embedded in a sampling-importance-resampling (SIR) scheme that generates P variables $x_{1:P}$ i.i.d. from q_0 and a selection index k taking value i with relative probability $p(x_i, y)/q_0(x_i; y)$ (i.e., the auxiliary variables $v ::= (x_{1:P}, k)$), then sets $x \leftarrow x_k$:

$$q((x_{1:P}, k), x; y) = \prod_{j=1}^{P} q_0(x_j) \left[\frac{\frac{p(x_k, y)}{q_0(x_k; y)}}{\sum_{i=1}^{P} \frac{p(x_i, y)}{q_0(x_i; y)}} \right] \delta(x; x_k).$$

The task of the auxiliary proposal $r((x_{1:P}, k); x, y)$ is to infer v for an (x, y) pair as follows:

$$r((x_{1:P}, k); x, y) = \prod_{\substack{j=1 \\ j \neq k}}^{P} q_0(x_j; y) \left[\frac{1}{P} \right] \delta(x_k; x).$$

The weight (8.28) is then precisely the usual SIR estimate of the marginal density of y,

$$\frac{p(x, y) r(v; x, y)}{q(v, x; y)} = \frac{1}{P} \sum_{j=1}^{P} \frac{p(x_j, y)}{q_0(x_j; y)}. \tag{8.32}$$

By Burda et al. [2015, Theorem 1], if $p(x, y)/q_0(x; y)$ is bounded then $\mathrm{D}_{\mathrm{KL}} \left[q(v, x; y) || p(x \mid y) r(x) \right] \to 0$

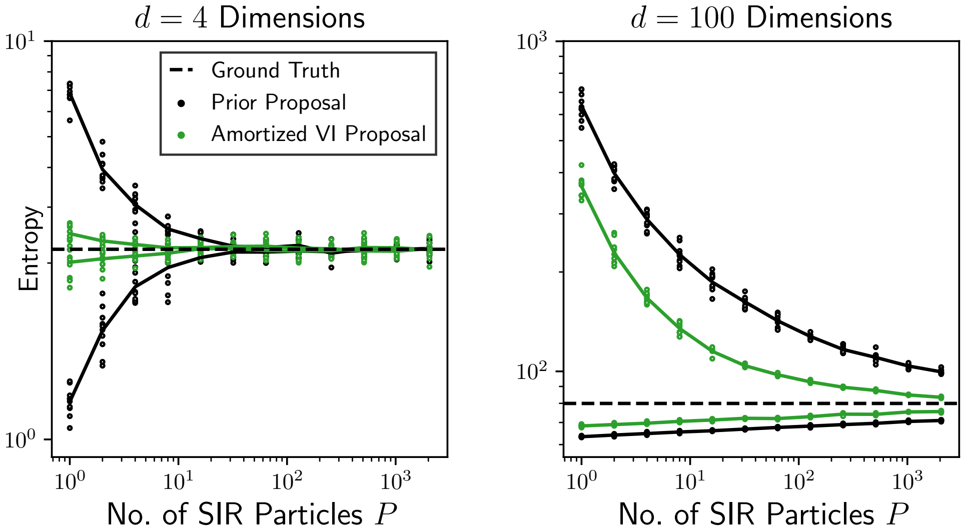


Figure 8.4: Lower and upper bounds on the entropy of $d/2$ dimensions Y of a d-dimensional Gaussian (X, Y) using the SIR scheme from Example 8.1 and Algorithms 8.3 and 8.4. The base proposals $q_0(x; y)$ are the prior and an amortized variational approximation to the posterior that specifies a separate regression for each dimension of X given Y. While the bounds converge to the ground truth (known in closed form for Gaussians) as the number P of SIR particles increases using both proposals, the variational proposal is closer in "exclusive" and "inclusive" KL to the posterior, resulting in a higher accuracy at each P. At $d = 100$, the lower bounds exhibit much lower bias and variance as compared to the upper bounds, especially for small P.

(i.e., the bias in Eq. (8.18)) as $P \to \infty$. Algorithms 8.3 and 8.4 present EEVI with SIR, which are special cases of Algorithms 8.1 and 8.2, respectively. «

Example 8.2 (SMC Proposals)**.** The SIR proposal from Example 8.1 can be generalized to the setting of a sequence $\{\tilde{p}_t(x; y)\}_{t=0}^{T}$ with T intermediate (unnormalized) target densities such that $\tilde{p}_T(x, y) = p(x, y)$. Such a sequence might arise in dynamic state-space models (as partial posteriors up to some time point) [Doucet and Johansen, 2011] or in static models (via tempering or sequential Bayesian updating) [Neal, 2001, Del Moral et al., 2006]. Algorithm 8.5 shows the proposal $q(v, x; y)$ from a run of SMC with initial kernel $q_0(x_0; y)$; forward kernels $q_t(x_t; x_{t-1}, y)$ $(t = 1 \dots T)$; backward kernels $l_t(x_t; x_{t+1}, y)$ $(t = 0 \dots T-1)$; and P particles. Here, $v ::= (I_T, a_{1:T}^{1:P}, x_{0:T}^{1:P})$ contains all auxiliary variables and $x \sim \delta(x_T^{I_T})$ is the selected final particle. Algorithm 8.6 shows the auxiliary proposal $r(v; x, y)$, which infers v given (x, y) using generalized "conditional SMC" [Andrieu et al., 2010, Cusumano-Towner and Mansinghka, 2017]. Simplifying the weights (8.28) and (8.29) gives

$$w(v, x, y) = \prod_{t=0}^{T} \left[\frac{1}{P} \sum_{j=1}^{P} w_t^j \right] \tag{8.33}$$

$$w'(v, x, y) = \frac{1}{w(v, x, y)}, \tag{8.34}$$

where w_t^j terms are defined in line 6 of Algorithm 8.5 for $w(v, x, y)$ and line 12 of Algorithm 8.6 for $w'(v, x, y)$. It also possible to learn the kernels q_t using variational inference [Maddison et al., 2017]. «

Example 8.3 (SMC+SIR Proposals)**.** Multiple runs of the SMC proposals in Example 8.2 can be embedded within the SIR scheme described in Example 8.1. In this case, the base proposal $q_0(v, x; y)$

Algorithm 8.5 SMC proposal $q(v, x; y)$.

```
Require: Target distribution $p(x, y)$
         Observation $y$
         Unnormalized target densities
           $\{\tilde{p}_t(x_t; y)\}_{t=0}^{T}$ such that
           $\tilde{p}_T(x; y) = p(x, y)$
         Initial kernel $q_0(x_0; y)$
         Kernels $q_t(x_t; x_{t-1}, y)$ $(1 \leq t \leq T)$
         Kernels $l_t(x_t; x_{t+1}, y)$ $(0 \leq t \leq T-1)$
         Number of particles $P$
         ESS threshold $\gamma$
Ensure: Approximate sample $x$ from $p_T(x; y)$ and
    record $v$ of all the sampled auxiliary random
    variables.
$x_0^i \sim q_0(-; y)$ $(i = 1 \dots P)$
$w_0^i \leftarrow \tilde{p}_0(x_0^i; y)/q_0(x_0^i; y)$ $(i = 1 \dots P)$
for $t = 1 \dots T$ do
    $a_t^i \leftarrow$ Categorical$(w_{t-1}^{1:P})$ $(i = 1 \dots P)$
    $x_t^i \sim q_t(-; x_{t-1}^{a_t^i}, y)$ $(i = 1 \dots P)$
    $w_t^i \leftarrow \frac{\tilde{p}_t(x_t^i; y) l_{t-1}(x_{t-1}^{a_t^i}; x_t^i, y)}{\tilde{p}_{t-1}(x_t^{a^i}; y) q_t(x_t^i; x_{t-1}^{a_t^i}, y)}$ $(i = 1 \dots P)$
$I_T \sim$ Categorical$(w_T^{1:P})$
return $(v, x) ::= ((I_T, a_{1:T}^{1:P}, x_{0:T}^{1:P}), x^{I_T})$
```

Algorithm 8.6 Auxiliary SMC proposal $r(v; x, y)$.

```
Require: Target distribution $p(x, y)$
         Observation $(x, y)$
         Unnormalized target densities
           $\{\tilde{p}_t(x_t; y)\}_{t=0}^{T}$ such that
           $\tilde{p}_T(x; y) = p(x, y)$
         Initial kernel $q_0(x_0; y)$
         Kernels $q_t(x_t; x_{t-1}; y)$ $(1 \leq t \leq T)$
         Kernels $l_t(x_t; x_{t+1}; y)$ $(0 \leq t \leq T-1)$
         Number of particles $P$
         ESS Threshold $\gamma$
Ensure: Approximate sample $v$ of all auxiliary
    variables generated by a run of Algorithm 8.5
    that returned $x$.
$I_T \sim$ Uniform$(1 \dots P)$
$x_T^{I_T} \leftarrow x$
for $t = T-1 \dots 0$ do
    $I_t \sim$ Uniform$(1 \dots P)$
    $x_t^{I_t} \sim l_t(-; x_{t+1}^{I_{t+1}}, y)$
    $a_{t+1}^{I_{t+1}} \leftarrow I_t$
$x_0^i \sim q_0(-; y)$ $(i = 1 \dots P; i \neq I_0)$
$w_0^i \leftarrow \tilde{p}_0(x_0^i; y)/q_0(x_0^i; y)$ $(i = 1 \dots P)$
for $t = 1 \dots T$ do
    $a_t^i \leftarrow$ Categorical$(w_{t-1}^{1:P})$ $(i = 1 \dots P, i \neq I_t)$
    $x_t^i \sim q_t(-; x_{t-1}^{a_t^i})(i = 1 \dots P; i \neq I_t)$
    $w_t^i \leftarrow \frac{\tilde{p}_t(x_t^i; y) l_{t-1}(x_{t-1}^{a_t^i}; x_t^i, y)}{\tilde{p}_{t-1}(x_t^{a^i}; y) q_t(x_t^i; x_{t-1}^{a_t^i}, y)}$ $(i = 1 \dots P)$
return $v ::= (I_T, a_{1:T}^{1:P}, x_{0:T}^{1:P})$
```

for SIR has auxiliary variables v and the corresponding base auxiliary proposal is denoted $r_0(v; x, y)$. If q_0 and r_0 are from the SMC scheme in Example 8.2 and Algorithms 8.5 and 8.6, then the overall SIR scheme corresponds to P independent runs of SMC and conditional SMC, which themselves use P' particles internally. The resampling step selects one of these P independent runs. The overall proposal $q(v, x; y)$ and auxiliary proposal $r(v; x, y)$ on the extended space are then

$$q(v_{1:P}, x; y) = \frac{1}{P} \sum_{k=1}^{P} q_0(v_k, x; y) \prod_{\substack{t=1 \\ t \neq k}}^{P} r_0(v_t; x, y), \tag{8.35}$$

$$r(v_{1:P}; x, y) = \prod_{t=1}^{P} r_0(v_t; x, y). \tag{8.36}$$

Algorithm 8.7 EEVI upper bound using SIR proposal with auxiliary variables.

```
Require: Target distribution $p(x, y)$
         Base proposal distribution $q_0(v, x; y)$
         Base auxiliary proposal dist $r_0(v; x, y)$
         Number of samples $n$, $m$, $P$
 1: for $i = 1 \dots n$ do
 2:     $(\tilde{X}, Y) \sim p(x, y)$
 3:     for $j = 1 \dots m$ do
 4:         $(V_1, X) \sim q_0(v, x; Y)$
 5:         for $k = 2 \dots P$ do
 6:             $V_k \sim r_0(v; X, Y)$
 7:         for $k = 1 \dots P$ do
 8:             $\xi_k \leftarrow \dfrac{q(V_k, X; Y)}{r(V_k; X, Y)}$
 9:         $u_j \leftarrow \log \dfrac{p(X, Y)}{\frac{1}{P}\sum_{k=1}^{P} \xi_k}$
10:     $t_i \leftarrow \frac{1}{m}\sum_{j=1}^{m} u_j$
11: return $-\frac{1}{n}\sum_{i=1}^{n} t_i$
```

Algorithm 8.8 EEVI lower bound using SIR proposal with auxiliary variables.

```
Require: Target distribution $p(x, y)$
         Base proposal distribution $q_0(v, x; y)$
         Base auxiliary proposal dist $r_0(v; x, y)$
         Number of samples $n$, $m$, $P$
 1: for $i = 1 \dots n$ do
 2:     $(X_1, Y) \sim p(x, y)$
 3:     $(X_{2:m}) \sim \mathrm{MCMC}_{X_1}$ targeting $p(x \mid Y)$
 4:     for $j = 1 \dots m$ do
 5:         for $k = 1 \dots P$ do
 6:             $V_k \sim r(v; X_j, Y)$
 7:             $\xi_k \leftarrow \dfrac{q(V_k, X_j; Y)}{r(V_k; X_j, Y)}$
 8:         $u_j \leftarrow \log \dfrac{\frac{1}{P}\sum_{k=1}^{P} \xi_k}{p(X_j, Y)}$
 9:     $t_i \leftarrow -\frac{1}{m}\sum_{j=1}^{m} u_j$
10: return $\frac{1}{N}\sum_{i=1}^{N} t_i$
```

The extended weight Eq. (8.28), which appears in line 9 of Algorithm 8.7, is then

$$w(v_{1:P}, x; y) = \frac{p(x, y) \displaystyle\prod_{k=1}^{P} r_0(v_k; x, y)}{\dfrac{1}{P} \displaystyle\sum_{k=1}^{P} q_0(v_k, x; y) \prod_{\substack{t=1 \\ t \neq k}}^{P} r_0(v_t; x, y)} = \frac{p(x, y)}{\dfrac{1}{P} \displaystyle\sum_{k=1}^{P} \dfrac{q_0(v_k, x; y)}{r(v_k; x, y)}}. \tag{8.37}$$

The extended weight (8.29), which appears in line 8 of Algorithm 8.8, is the reciprocal of the weight Eq. (8.37). The extended proposal $q(v_{1:P}, x; y)$ generates samples $(V_{1:P}, X)$ as follows:

- sample $(V_0, X) \sim q_0(v, x; y)$;
- sample selection index $k \sim \mathrm{Uniform}(1 \dots P)$;
- set $V_k \leftarrow V_0$;
- sample $V_j \sim r_0(v; X, y)$ from the base auxiliary proposal for $j = 1, \dots, k-1, k+1, \dots, P$.

The extended auxiliary proposal $r(v_{1:P}; x, y)$ generates P i.i.d. samples from $r_0(v; x, y)$. «

8.4 Applications to Optimal Data Acquisition

Information estimates delivered by EEVI are evaluated on two challenging data acquisition problems. Section 8.4.1 shows how to rank medical tests in an expert system for liver disorders according to their conditional mutual information with diseases of interest, given a pattern of symptoms and patient attributes. Section 8.4.2 analyzes a dynamic insulin model for diabetes and shows how to compute optimal times to take blood glucose measurements that maximize information about a patient's insulin sensitivity, given their insulin intake and meal schedule.

8.4.1 HEPAR Liver Disease Network

HEPAR [Lucas et al., 1989] is a medical expert system that helps physicians diagnose complex disorders in the liver and biliary tract. This evaluation analyzes the Bayesian network variant of HEPAR [Oniśko, 2003] shown in Figure 8.5. The model contains three types of nodes, which represents a patient's (i) attributes, such as age and obesity; (ii) latent liver diseases, such as PBC and cirrhosis; and (iii) symptoms, such as nausea and blood pressure. Consider the following inference problem:

> *Given a patient with a set $\{o_i\}$ of observed attributes and symptoms, which medical tests $\{t_j\}$ for symptoms should the physician conduct to maximize information about the presence of a disease d?*

This problem can be formalized as computing a ranking of the tests $\{t_j\}$ according to the their conditional mutual information (CMI) with d, which is defined as

$$I(d : t_j \mid \{o_i\}) = H(d \mid \{o_i\}) - H(d \mid t_j, \{o_i\}). \tag{8.38}$$

As $H(d \,|\, \{o_i\})$ in Eq. (8.38) is the same for all tests t_j, it can be ignored for ranking. For computational efficiency, tests are ranked by increasing conditional entropy $H(d \,|\, t_j, \{o_i\})$ as defined in Eq. (8.12).

Results Figure 8.5 shows a pattern of 20 observed nodes (red), 31 medical tests for symptoms (yellow), and two liver diseases (blue). Table 8.1 shows the top 10 tests ranked by $H(d \mid t_j, \{o_i\})$ for the PBC and cirrhosis diseases. In Tables 8.1a and 8.1b, the first and second columns show the top 10 most informative tests and conditional entropy values for PBC and cirrhosis, respectively The conditional entropies are computed as the midpoint of interval estimates from EEVI (Algorithms 8.1 and 8.2), using SIR auxiliary variable proposal (Example 8.1) with ancestral sampling base proposal. The number of particles P is sufficient to drive the interval width to below 10^{-3} nats. To assess how useful these rankings might be to a physician, the third columns in Tables 8.1a and 8.1b show median prediction errors for each disease in 5000 random patients, obtained from conditioning on the symptoms, attributes, and one additional medical test. That is, for each candidate test t_j, the prediction error is defined as the log loss between the posterior $p(d \mid t_j, \{o_i\})$ and the ground-truth label. For reference, prediction errors using $p(d \mid \{o_i\})$ (denoted "no test") and $p(d)$ (denoted "no test or obs") are also shown. Tables 8.1a and 8.1b confirm that tests with higher information values produce lower errors. Conditional entropy estimates from EEVI can thus serve as a useful decision-making tool in this expert system.

Runtime Figure 8.6 compares runtime versus accuracy profiles of EEVI to exact inference using SPPL (Chapter 7) and the nonparametric estimator of Kraskov et al. [2004]. The query is estimating the joint entropy $H(\{o_i\}_{i=1}^{k})$ of k-dimensional random variables in the HEPAR network (k = 10, 15, 20, 40). For these queries, upper and lower bounds from EEVI converge in 1–10 seconds for each k. Unsurprisingly, the runtime needed for the bounds to meet grows as k increases. In contrast, the estimator of Kraskov et al. [2004] converges slower, as it is "model-free" and estimates log probabilities from simulated data without leveraging model structure. Exact inference in SPPL does not scale well beyond 15 dimensions for computing entropies in the HEPAR network for two reasons: first, computing exact probabilities of observations at the leaves of Figure 8.5 requires summing over all possible parent configurations given the lack of conditional independencies; second, computing the outer expectation for entropy requires enumerating over exponentially many variable states. The plots also highlight a key feature of EEVI: the width of the interval quantifies the accuracy of the estimate at any given level of computation and can squeeze the true value. In contrast, the nonparametric estimator provides point estimates (typically lower bounds) whose accuracy at a given level of computation is unknown.

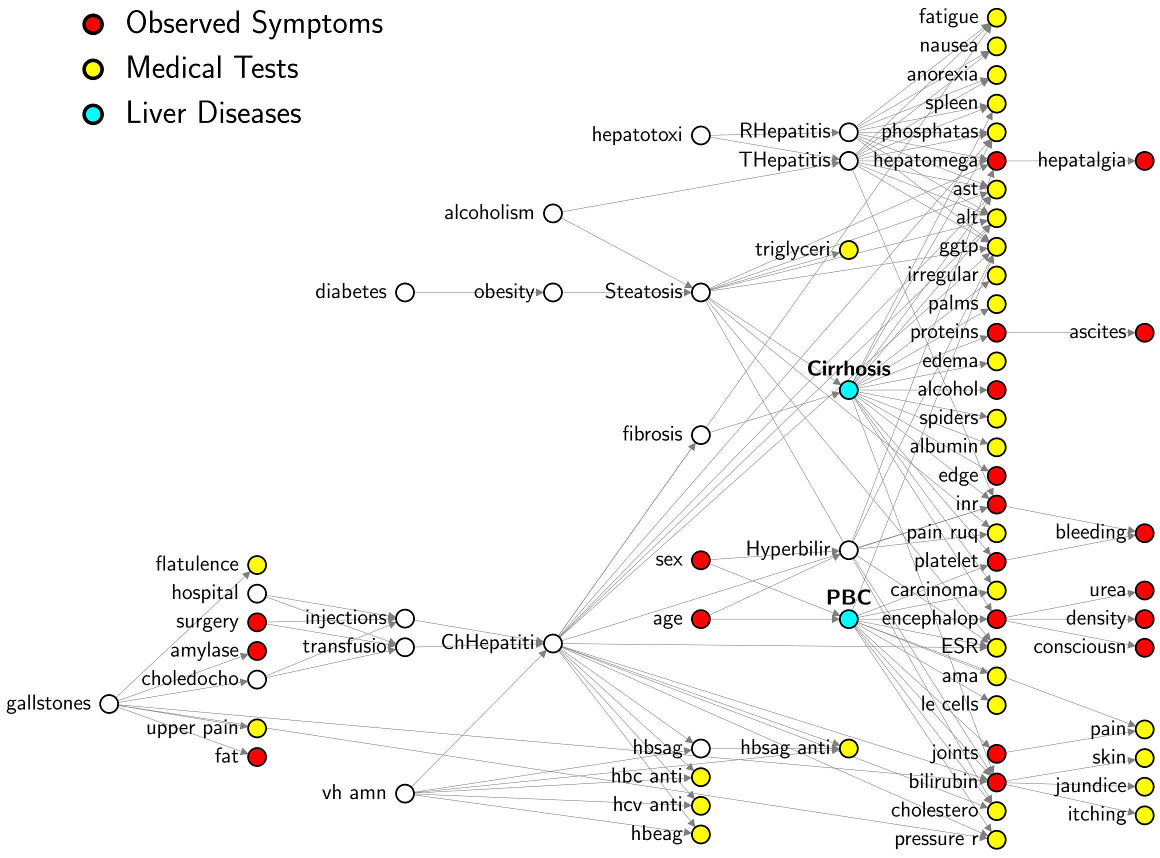


Figure 8.5: HEPAR liver disease model [Oniśko, 2003].

Table 8.1: Using EEVI to rank diagnostic medical tests (yellow) in the HEPAR liver disease network by how informative they are about diseases (blue) given a pattern of observations (red). For both the PBC disease in (a) and cirrhosis diseases in (b), conducting medical tests that give lower conditional entropy $H(d \mid t, \{o_i\}_{i=1}^{20})$ of the disease (i.e., higher conditional mutual information) results in lower prediction errors about its presence or absence.

Decreasing Information Value

(a) Top 10 tests for disease $d =$ PBC.

Medical Test t	$H(d \mid t, \{o_i\}_{i=1}^{20})$	Prediction Error
ama	0.274	0.030
ESR	0.346	0.094
cholesterol	0.385	0.126
ggtp	0.397	0.131
carcinoma	0.402	0.144
pain	0.404	0.148
pressure ruq	0.410	0.174
le cells	0.411	0.172
irregular liver	0.413	0.173
edge	0.415	0.175
no test	0.418	0.175
no test or obs	0.663	0.486

(b) Top 10 tests for disease $d =$ Cirrhosis.

Medical Test t	$H(d \mid t, \{o_i\}_{i=1}^{20})$	Prediction Error
irregular liver	0.225	0.035
edge	0.247	0.036
spiders	0.253	0.037
spleen	0.256	0.043
palms	0.261	0.043
carcinoma	0.261	0.049
edema	0.262	0.056
triglycerides	0.264	0.056
albumin	0.268	0.068
phosphatase	0.272	0.068
no test	0.290	0.070
no test or obs	0.320	0.080

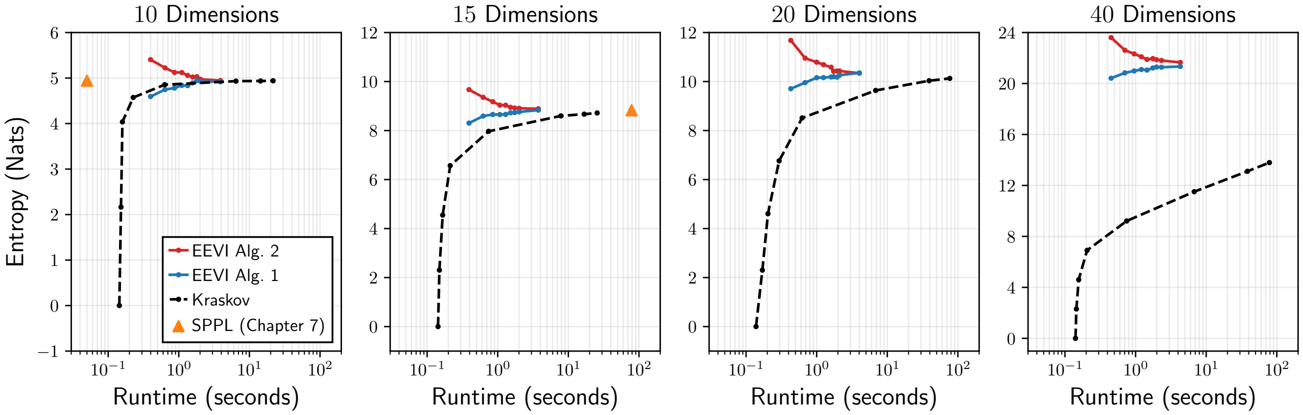


Figure 8.6: Runtime for estimating entropies in the HEPAR network using EEVI Algorithms 8.1 and 8.2, exact inference in SPPL (Chapter 7) and the baseline nonparametric estimator of Kraskov et al. [2004]. For EEVI, the runtime increases with number P of SIR samples for EEVI (Eq. (8.32)). For the Kraskov et al. estimator, the runtime increases with number of observations from $p(x, y)$. For SPPL, the runtime is constant, but exact inference scales poorly beyond 15 dimensions because computing the exact marginal probabilities is expensive and entropy is a sum over exponentially many variable configurations.

Variance Control Following Eq. (8.12), bounds on the conditional entropy $H(d \mid t_j, \{o_i\})$ in Figure 8.5 are a difference of two marginal entropy bounds (see also Figure 8.2)

$$\check{H}(d \,|\, t_j, \{o_i\}) = \check{H}(d, t_j, \{o_i\}) - \hat{H}(t_j, \{o_i\}), \tag{8.39}$$

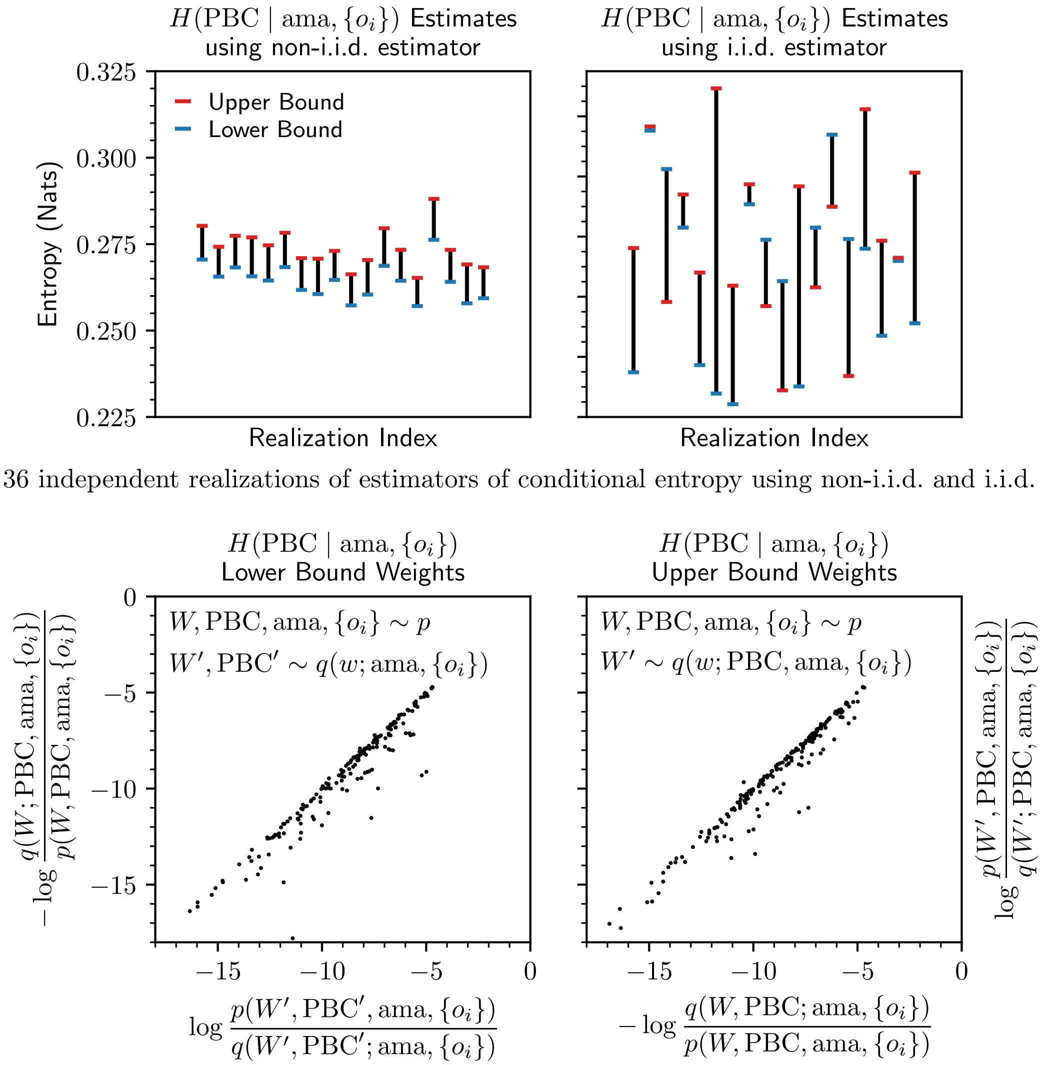
$$\hat{H}(d \,|\, t_j, \{o_i\}) = \hat{H}(d, t_j, \{o_i\}) - \check{H}(t_j, \{o_i\}). \tag{8.40}$$

Figure 8.7a shows 18 realizations of interval estimates of conditional entropy using shared samples (non-i.i.d. estimator) and independent samples (i.i.d. estimator) for the random random variables used to estimate each of the two terms in the right-hand sides of Eqs. (8.39) and (8.40). The non-i.i.d. estimator has lower variance as compared to the i.i.d. estimator and ensures that the realized lower bound is smaller than the realized upper bound. Figure 8.7b explains this behavior in terms of the correlation of the random weights Eqs. (8.28) and (8.29) used to estimate the four marginal entropies in Eqs. (8.39) and (8.40) (recall that $\mathrm{Var}[A - B] = \mathrm{Var}[A] + \mathrm{Var}[B] - 2\mathrm{Cov}[A, B]$ for any pair of real random variables A and B). It is recommended that practitioners empirically assess the variance and width of the interval estimators as in Figure 8.7b, as well as inspect scatter plots of the weights when adding/subtracting EEVI bounds to bound derived information measures.

8.4.2 Dynamic Insulin Model for Diabetes

Adjusting the insulin dosage for diabetic patients is a challenging clinical process. While most medications have two or three standard dosing options, insulin dose is highly individualized to each patient. Finding the correct dose is an iterative process that must account for the patient's insulin response to a test dose, their latent insulin sensitivities, and changing clinical conditions. Commercial software such as Glytec and EndoTool for insulin management have been developed to aid clinicians with this nuanced and costly process. The application in this section shows how to use EEVI for solving a data acquisition task in a differential equation model of carbohydrate metabolism [Andreassen et al., 1991] for insulin adjustment in diabetic patients.

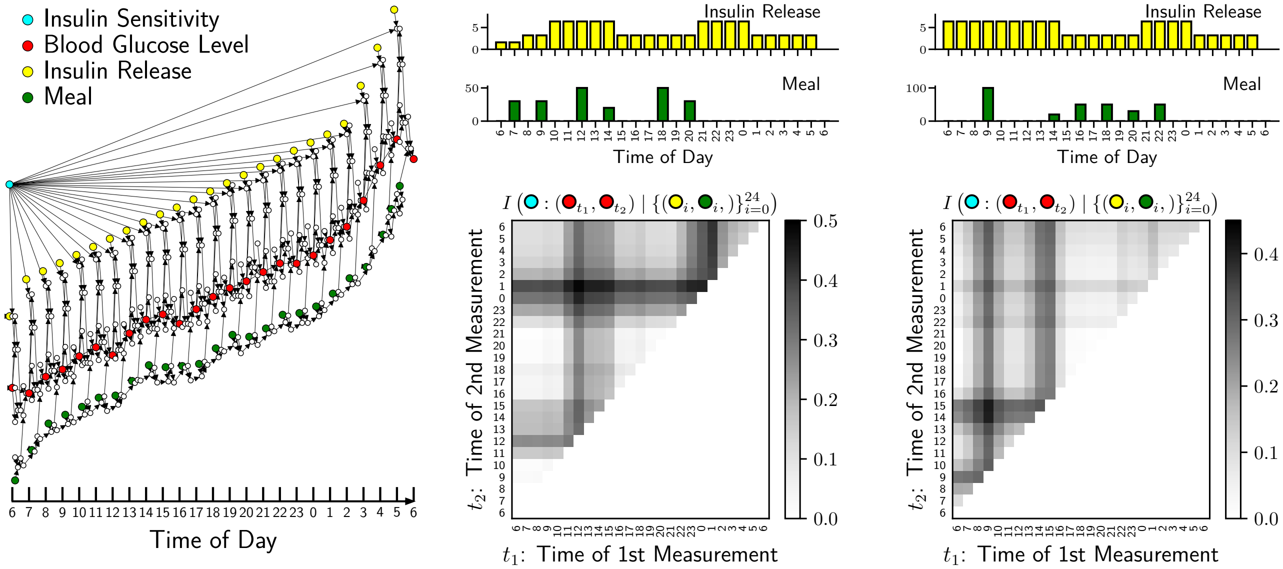
Figure 8.8a shows a dynamic Bayesian network that represents a discretization of the differential equation. The "insulin sensitivity" node (blue) is a global latent parameter that dictates how effectively the patient converts released insulin (e.g., from oral medications or injections) into biologically usable



(a) 36 independent realizations of estimators of conditional entropy using non-i.i.d. and i.i.d. sampling.

(b) Scatter plot of random log importance weights. Positive covariance reduces variance of non-i.i.d. sampling.

Figure 8.7: (a) Reducing the variance of interval estimators of $H(\text{PBC} \mid \text{ama}, \{o_i\})$ (Table 8.1a, row 1) by non-i.i.d. sampling. (b) As conditional entropy bounds Eqs. (8.39) and (8.40) are average differences (x-axes minus y-axes) of random weights that here are positively correlated, computing differences using non-i.i.d. samples reduces variance by twice the covariance. The variable W refers to all other variables in the HEPAR network from Figure 8.5.



(a) Carbohydrate metabolism model [Andreassen et al., 1991]

(b) Meal and Insulin Scenario 1

(c) Meal and Insulin Scenario 2

Figure 8.8: Inferring optimal pairs of times to measure blood glucose level (red) that maximize information about a patient's latent insulin sensitivity (blue). Each heatmap in (b)–(c) shows estimates from EEVI of the conditional mutual information of insulin sensitivity with a pair of blood glucose measurements for all pairs of times, under a certain scenario of the patient's insulin release (yellow) and meal (green) schedule. Each scenario has a different optimal pair of times to measure blood glucose: 12pm/1am and 9am/3pm, respectively.

insulin. The "meal" (green) and "insulin release" (yellow) nodes are intervention variables based on the patient's victual and medication intake over a 25-hour period. At time t, the blood glucose level is a noisy function of insulin sensitivity and several variables at $t-1$, namely: blood glucose level, meal, insulin release, and 14 intermediate biological latent variables (white nodes). Refer to Andreassen et al. [1991] for full details about the model. Suppose a diabetic patient is having their insulin adjusted and the physician is interested in the following problem:

> *Given the patient's meal and insulin release schedule, at which pairs of times should blood glucose level be measured to maximize information about insulin sensitivity?*

More formally, the problem is to rank pairs of times (t_1, t_2) by decreasing CMI values with insulin sensitivity (for $0 \leq t_1 < t_2 \leq 24$), given the meal and insulin release schedule:

$$I(\text{ins sens} : (\text{BG}_{t_1}, \text{BG}_{t_2}) \mid \{(\text{meal}_t, \text{ins rel}_t)\}_{t=0}^{24}), \tag{8.41}$$

where BG_t denotes blood glucose measured at time t and "ins sense" denotes the latent insulin sensitivity. Given the temporal structure in the model, entropy bounds are computed using the SMC proposals in Example 8.2 and Algorithms 8.5 and 8.6, with a number P of particles that squeezes the interval width to 10^{-2} nats. Figures 8.8b and 8.8c show CMI estimates for all pairs of time points under two different scenarios of meal and insulin release. Assuming that the model of Andreassen et al. [1991] is accurate, the optimal times from these heatmaps can provide valuable insights to the physician, as insulin sensitivity relates to blood glucose level through complex dynamics of carbohydrate metabolism that are challenging and costly to assess heuristically. EEVI enables quantitative analysis of information values of time points by probabilistic inference in the model for any meal and insulin schedule.

8.5 Related Work

Model Based Entropy Estimation Several works have derived variational bounds of information measures for a known target distribution [Barber and Agakov, 2003, Alemi and Fischer, 2018, Foster et al., 2019, Poole et al., 2019]. Unlike EEVI, these estimators do not not apply to arbitrary subsets of random variables in a generative model and do not report interval estimates. The proposed EEVI method can compute two-sided bounds and solve queries not handled previously. For example, the upper bounds on $I(X : Y)$ in Poole et al. [2019, Figure 1] and Foster et al. [2019, Eq. (9)] assume that $p(y \mid x)$ is tractable. In contrast, Algorithms 8.1 and 8.2 can be used to obtain interval estimates of $I(X : Y)$ even when $p(y \mid x)$ and $p(x \mid y)$ are both intractable, as in Figure 8.2. The applications in Section 8.4 solve such queries for analyzing medical data.

Model-Free Entropy Estimation Many nonparametric entropy estimators given i.i.d. data from an unknown distribution have also been developed [Kozachenko and Leonenko, 1987, Paninski, 2003, Kraskov et al., 2004, Pérez-Cruz, 2008, Belghazi et al., 2018, Goldfeld et al., 2020]. In contrast to these methods, EEVI is a "model-based" estimator in the sense that it requires that the target distribution is known. Nonparametric estimators can typically be used in model-based settings by applying them to i.i.d. data simulated from the model. A drawback of using nonparametric estimators in this way, however, is that they ignore the known model structure. Figure 8.6 suggests that EEVI scales better in these cases, because the model structure is used to build a suitable proposal. A second advantage of EEVI is that the interval width indicates the quality of the estimate at a given level of computational effort, whereas nonparametric methods do not deliver two-sided bounds. The flip side is that building accurate proposals for EEVI needs more expertise as compared to purely nonparametric methods.

Nested and Recursive Monte Carlo Rainforth et al. [2018] give a thorough treatment of consistency and convergence properties of a very general class of nested Monte Carlo estimators. The expressions in Eqs. (8.8) and (8.9) used for EEVI are instances of nested Monte Carlo, where the inner expectation is obtained via pseudo-marginal methods [Andrieu and Roberts, 2009] and the nonlinear mapping is log. The recursive auxiliary-variable inference (RAVI) method from Lew et al. [2022] can be used to construct EEVI algorithms with deeper levels of nesting than just a pair (q, r) of a proposal and auxiliary proposal for a given model p (Figure 8.1). Such extensions, however, must be accompanied with a careful characterization of the underlying runtime and accuracy trade-offs.

Bounds on Marginal Probabilities Grosse et al. [2015] introduced the idea of using annealed importance sampling or sequential harmonic mean to "sandwich" marginal log probabilities. Grosse et al. [2016] and Wu et al. [2016] apply these estimators to diagnose MCMC inference algorithms and analyze deep generative models, respectively. EEVI builds on log probability bounds for the new problem of forming interval estimators of entropy and composing the estimates to bound many derived information-theoretic quantities. Cusumano-Towner and Mansinghka [2017] use a similar family of auxiliary-variable importance samplers from Section 8.3.1 to upper bound symmetric KL divergences between a pair of distributions—in that setting, the normalizing constants in Eq. (8.18) are irrelevant as they cancel out so the weights Eqs. (8.28) and (8.29) are only needed up to normalizing constants. In contrast, the normalizing constants in EEVI cannot be ignored as they are the essential quantities needed to bound entropies.

Probabilistic Programming All EEVI experiments were implemented in the Gen probabilistic programming system [Cusumano-Towner et al., 2019, Cusumano-Towner, 2020], by representing Algorithms 8.1 and 8.2 as meta-programs that operate on probabilistic model programs (Sections 1.4 and 8.6). These implementations make it easy to apply EEVI to a broad set of generative models in

Gen, provided that the target random variables correspond to random choices at addresses that exist in each execution of the program. Moreover, EEVI is more widely applicable than previous probabilistic programming-based estimators for information-theoretic quantities, such as Saad and Mansinghka [2017, Algorithm 2a], Gehr et al. [2020, Figure 11], and Narayanan and Shan [2020, Section 8.2]. These estimators assume that the probabilistic programming system can exactly compute any marginal or conditional density, which is rarely possible except in languages that restrict modeling expressiveness to enable exact inference [Saad and Mansinghka, 2016a, Gehr et al., 2016, Narayanan et al., 2016, Saad et al., 2021]. Even when the marginal probabilities can be computed exactly and relatively efficiently, the outer integral in the entropy expression (8.2) remains intractable, so a (nonexact) simple Monte Carlo estimator is required.

To increase the level of automation and accuracy of EEVI, it may be worthwhile to leverage probabilistic programming methods for learning amortized proposal distributions [Paige and Wood, 2016, Ritchie et al., 2016, Le et al., 2017], as discussed in Section 8.3.1. Another direction is using EEVI to extend the class of models and queries that can be handled by existing probabilistic programming-based frameworks for tasks such as searching structured databases [Saad et al., 2017] and optimal experimental design [Ouyang et al., 2018], which both need accurate estimates of entropy and information.

8.6 Implementation as Probabilistic Meta-Programs in Gen

Listings 8.1 and 8.2 show how to implement EEVI upper and lower bounds (Algorithms 8.1 and 8.2) with custom proposal distributions as meta-programs in Gen. These functions take as input a `model` program and `proposal` program, both of type `GenerativeFunction`. The `targets` input is a selection of the trace addresses in the `model` program whose marginal entropy is being queried. The `proposal` program takes as input a `ChoiceMap` whose addresses are precisely those specified by the `targets` variable. The `proposal` must make random choices at all trace addresses in `model` other than those in `targets`. The user is responsible for ensuring that `proposal` is valid with respect to `model` in terms of the absolute continuity requirements discussed in Section 8.3.

In terms of the notation in Figure 8.1, the `model` is $p(x, y)$, the `targets` are Y, and the `proposal` is $q(x, v; y)$. While `proposal` may make auxiliary random choices v that are not part of `model`, the Gen specification for a `GenerativeFunction` type ensures that these choices are transparent to the user. In particular, the `proposal` object carries a default internal proposal $r(v; x, y)$ that furnishes the required terms for computing the log weight ratios (8.28) and (8.29). In Listing 8.2, the (optional) MCMC kernel in line 30 is resimulation Metropolis-Hastings from the prior, though it is possible for the kernel to instead be customized by the user [Cusumano-Towner, 2020, Section 3.4.2].

EEVI can also be fully automated in Gen without custom proposals. In this case, the proposal $q(x; y)$ is the default internal proposal of `model`, which uses ancestral sampling. To increase accuracy, the default internal proposal is used as the base proposal within the SIR scheme (Algorithms 8.3 and 8.4) discussed in Example 8.1. Listings 8.3 and 8.4 show the resulting Gen implementations of the fully automated EEVI bounds. The user only needs to provide the `model` probabilistic program, the `targets` whose marginal entropy is being queried, the number `N` of outer samples, the number `M` of inner samples, and the number `P` of SIR particles. In line 30 of Listing 8.3, an error is thrown when the `model` program is such that the ancestral sampling proposal $q(x; y)$ is not absolutely continuous with respect the posterior $p(x \mid y)$, which might happen when the observed value $\{Y = y\}$ has zero conditional probability given some setting of $\{X = x\}$. Programs where the support of a random choice at a given address varies across different executions are referred to as "undisciplined" in Gen parlance. While users are able to write undisciplined programs, they are not supported by the formal semantics of the Gen language and using can result in undefined program behavior.

```
import Gen

"""Entropy upper bound using custom proposal."""
function entropy_upper_bound(
    model::Gen.GenerativeFunction,
    model_args::Tuple,
    proposal::Gen.GenerativeFunction,
    proposal_args::Tuple,
    targets::Gen.Selection,
    N::Integer,
    M::Integer
    )
  wi_list = []
  for i=1:N
      # Sample observations from model.
      trace_p = Gen.simulate(model, model_args) : model_traces[i]
      observations = Gen.get_selected(Gen.get_choices(trace_p), targets)
      wj_list = []
      for j=1:M
          # Sample latents from proposal and compute score.
          trace_q = Gen.simulate(proposal, (proposal_args..., observations,))
          log_q = Gen.get_score(trace_q)
          # Compute score of latents + observations under model.
          choices = merge(observations, Gen.get_choices(trace_q))
          _, log_p = Gen.generate(model, model_args, choices)
          # Compute importance weight.
          log_w = log_p - log_q
          push!(wj_list, log_w)
      end
      wj_avg = mean(wj_list)
      push!(wi_list, wj_avg)
  end
  return -mean(wi_list)
end
```

Listing 8.1: Gen implementation of EEVI upper bound using custom proposal (Algorithm 8.1).

```
import Gen

"""Entropy lower bound using custom proposal."""
function entropy_lower_bound(
    model::Gen.GenerativeFunction,
    model_args::Tuple,
    proposal::Gen.GenerativeFunction,
    proposal_args::Tuple,
    targets::Gen.Selection,
    N::Integer,
    M::Integer
    )
  wi_list = []
  for i=1:N
    # Sample latents + observations from model and compute score.
    trace_p = Gen.simulate(model, model_args)
    wj_list = []
    for j=1:M
        # Compute score of latents + observations under model.
        log_p = Gen.get_score(trace_p)
        # Compute score of latents under proposal.
        observations = Gen.get_selected(Gen.get_choices(trace_p), targets)
        latents = Gen.get_selected(Gen.get_choices(trace_p), Gen.complement(targets))
        _, log_q = Gen.generate(proposal, (proposal_args..., observations,), latents)
        # Compute importance weight.
        log_w = log_q - log_p
        push!(wj_list, log_w)
        # MCMC step iterating the latents in trace_p
        if j < M
          trace_p = Gen.metropolis_hastings(trace_p, Gen.complement(targets);
              observations=observations)
        end
    end
    wj_avg = -mean(wj_list)
    push!(wi_list, wj_avg)
end
return -mean(wi_list)
end
```

Listing 8.2: Gen implementation of EEVI lower bound using custom proposal (Algorithm 8.2).

```
import Gen

"""Entropy upper bound using default ancestral sampling proposal."""
function entropy_upper_bound(
    model::Gen.GenerativeFunction,
    model_args::Tuple,
    targets::Gen.Selection,
    N::Integer,
    M::Integer,
    P::Integer
  )
  wi_list = []
  for i=1:N
      # Sample observations from model.
      trace_p = Gen.simulate(model, model_args)
      observations = Gen.get_selected(Gen.get_choices(trace_p), targets)
      wj_list = []
      for j=1:M
          wk_list::Vector{Float64} = []
          for k=1:P
              # Sample latents (particle k) from ancestral sampling proposal
              # and retrieve weight w_k,
              # which is p(observations | latents)
              trace_q, log_w = Gen.generate(model, model_args, observations)
              push!(wk_list, log_w)
          end
          # Compute overall log importance weight,
          # which is log [1/P sum w_k]
          wk_avg = (Gen.logsumexp(wk_list) - log(length(wk_list)))
          !isinf(wk_avg) || throw("Invalid proposal (infinite log weight)")
          push!(wj_list, wk_avg)
      end
      wj_avg = mean(wj_list)
      push!(wi_list, wj_avg)
  end
  return -mean(wi_list)
end
```

Listing 8.3: Gen implementation of EEVI upper bound using SIR with default proposal (Algorithm 8.3).

```julia
import Gen

"""Entropy lower bound using default ancestral sampling proposal."""
function entropy_lower_bound(
        model::Gen.GenerativeFunction,
        model_args::Tuple,
        targets::Gen.Selection,
        N::Integer,
        M::Integer,
        P::Integer)
    wi_list = []
    for i=1:N
        # Sample latents + observations from model.
        trace_p = Gen.simulate(model, model_args) : model_traces[i]
        observations = Gen.get_selected(Gen.get_choices(trace_p), targets)
        wj_list = []
        for j=1:M
            trace_pk = trace_p
            wk_list::Vector{Float64} = []
            # Retrieve weight of exact posterior sample (particle 1)
            # using ancestral sampling proposal, which is
            # p(observations | latents)
            log_w = Gen.project(trace_pk, targets)
            push!(wk_list, log_w)
            for k=2:P
                # Sample latents (particle k) from ancestral sampling proposal
                # and retrieve weight w_k,
                # which is p(observations | latents)
                trace_pk, log_w = Gen.generate(model, model_args, observations)
                push!(wk_list, log_w)
            end
            # Compute overall log importance weight, which is
            # log [1 / (1/P sum w_k)] = - log [1/P sum w_k]
            wk_avg = -(Gen.logsumexp(wk_list) - log(length(wk_list)))
            push!(wj_list, wk_avg)
            # MCMC step iterating latents in trace_p.
            if j < M
              trace_p = Gen.metropolis_hastings(trace_p, Gen.complement(targets);
                observations=observations)
            end
        end
        wj_avg = -mean(wj_list)
        push!(wi_list, wj_avg)
    end
    return -mean(wi_list)
end
```

Listing 8.4: Gen implementation of EEVI lower bound using SIR with default proposal (Algorithm 8.4).

Chapter 9

Goodness-of-Fit Tests

> It has been said that "all models are wrong but some models are useful." In other words, any model is at best a useful fiction—there never was, or ever will be, an exactly normal distribution or an exact linear relationship. Nevertheless, enormous progress has been made by entertaining such fictions and using them as approximations.
>
> George E. P. Box

This chapter presents a new method to test whether a dataset of observed samples $y_1, \dots, y_m$ is likely to have been drawn from a candidate probability distribution $\mathbf{p}$, where the only interface to $\mathbf{p}$ is generating random variables via stochastic simulation. Distributions that match these assumption arise in many settings, which include: (i) probabilistic programs, where probability distributions are represented as executable stochastic simulators; (ii) simulation-based algorithms for approximate posterior inference such as MCMC (Section 3.3.1) and SMC (Section 3.3.2), where a forward run of the algorithm produces an output whose marginal probability is intractable to compute. The general problem of testing whether a set of samples was drawn from a given distribution is known as goodness-of-fit testing. This fundamental problem has applications in a variety of fields including Bayesian statistics [Gelman et al., 1996, Talts et al., 2018], high-energy physics [Williams, 2010], astronomy [Peacock, 1983], genetic association studies [Lewis and Knight, 2009], and psychometrics [Andersen, 1973].

The family of rank-based tests are a common approach to testing goodness-of-fit of data to a stochastic simulator [Lehmann and D'Abrera, 1975]. However, most existing rank-based tests operate with continuous distributions in Euclidean space [Lehmann and Romano, 2005, VI.8]. Analogous methods for distributions over combinatorially large discrete spaces that are theoretically rigorous, customizable using domain knowledge, and practical to implement remain relatively less investigated.

The method presented in this chapter makes a new connection between rank-based tests and distributions on high-dimensional discrete spaces. By algorithmically specifying an ordering on the data domain, the practitioner can quantitatively assess how typical the observed samples are with respect to simulated data from the candidate distribution. This ordering is leveraged by the test to effectively surface distributional differences between the observed data and candidate distribution. More specifically, the proposed method tests whether observations $y_1, \dots, y_n$ taking values in a set $\mathcal{T}$ were drawn from a given distribution $\mathbf{p}$ on the basis of the rank of each y_i with respect to m i.i.d. samples $x_1, \dots, x_m \sim \mathbf{p}$. If y_i was drawn from $\mathbf{p}$ then its rank should be uniformly distributed over $0, 1, \dots, m$. When the ranks show a deviation from uniformity, it is unlikely that the y_i were drawn from $\mathbf{p}$. For distributions with discrete atoms, however, the ranks are no longer uniform. A key step in the proposed method is to

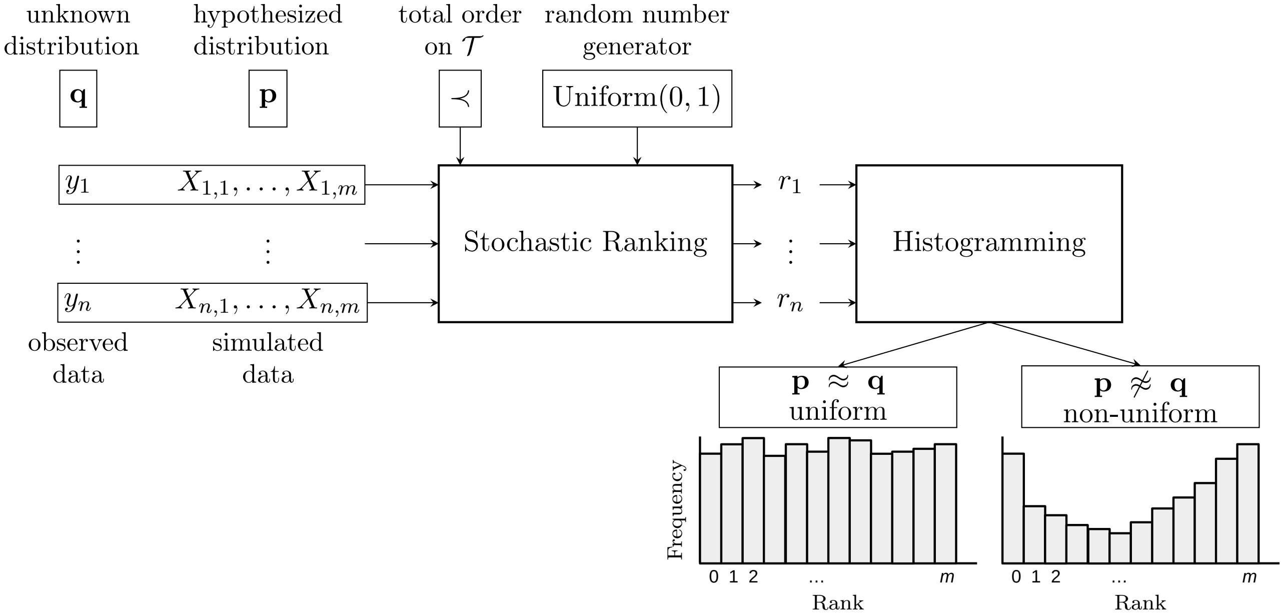

unknown distribution
hypothesized distribution
total order on $\mathcal{T}$
random number generator
$\mathbf{q}$
$\mathbf{p}$
$\prec$
Uniform(0, 1)
y_1 $X_{1,1}, \ldots, X_{1,m}$
y_n $X_{n,1}, \ldots, X_{n,m}$
observed data
simulated data
Stochastic Ranking
r_1
r_n
Histogramming
$\mathbf{p} \approx \mathbf{q}$ uniform
$\mathbf{p} \not\approx \mathbf{q}$ non-uniform
Frequency
0 1 2 ... m
Rank
0 1 2 ... m
Rank

Figure 9.1: Overview of the proposed goodness-of-fit test. Step 1: Observations $y_1, \ldots, y_n$ are assumed to be drawn i.i.d. from an unknown discrete distribution $\mathbf{q}$ over a set $\mathcal{T}$. Step 2: For each y_i ($i = 1, \ldots, n$), m samples $X_{i,1}, \ldots, X_{i,m}$ are simulated i.i.d. from the hypothesized distribution $\mathbf{p}$ over $\mathcal{T}$. Step 3: Given a total order $\prec$ on $\mathcal{T}$ and the observed and simulated data, a stochastic ranking procedure returns the rank r_i of each y_i within $X_{i,1}, \ldots, X_{i,m}$, using uniform random numbers to ensure the ranks are unique. Step 4: The histogram of the ranks $r_1, \ldots, r_n$ is analyzed for uniformity over $0, 1, \ldots, m$.

use continuous uniform random variables U_i that are paired with each y_i and x_i to break ties when computing the ranks. The resulting statistic is called the Stochastic Rank Statistic (SRS) and has multiple desirable properties when used for testing goodness-of-fit:

1. The SRS is distribution-free, meaning that its sampling distribution under the null hypothesis does not depend on $\mathbf{p}$. There is thus no need to construct approximate tables or use expensive Monte Carlo estimates of rejection regions.

2. The test is consistent against all alternatives. In particular, the SRS is distributed as a discrete uniform if and only if $y_1, \ldots, y_n \sim^{\text{iid}} \mathbf{p}$.

3. The exact (non-asymptotic) sampling distribution of the SRS is a discrete uniform. This property obviates the need to apply asymptotic approximations in small sample or sparse data regimes.

4. The test gives the practitioner flexibility to specify the properties of the data that are checked to agree with samples from $\mathbf{p}$. This flexibility comes from the ordering on the domain used to compute the ranks.

5. The test is readily implemented using a procedure that is linear time in the number of observations. The test is simulation-based and does not require explicitly computing $\mathbf{p}(x)$, which is essential for distributions with intractable probabilities.

While the test is consistent for any linear ordering $\prec$ over the domain $\mathcal{T}$, the power of the test depends on the choice of $\prec$. Strategies to obtain useful orderings are demonstrated for a variety of domains, which include defining procedures that traverse discrete data structures; composing probe statistics that reflect numerical properties of the data; and using randomization to generate orderings.

The remainder of the chapter is organized as follows. Section 9.1 reviews the goodness-of-fit problem and discusses related work. Section 9.2 presents the proposed test and establishes several theoretical properties. Section 9.3 presents simulation studies for distributions over integers, binary strings, and partitions. Section 9.4 applies the SRS to compare approximate Bayesian sampling algorithms over mixture assignments in a Dirichlet process mixture model and to assess the sample quality of random lattices produced by MCMC samplers for the Ising model.

9.1 Background: The Goodness-of-Fit Problem

Problem Statement The goodness-of-fit problem considered in this chapter can be informally described as follows: *Let* $\mathbf{p}$ *be a discrete distribution over a finite or countably infinite domain* $\mathcal{T}$. *Given observations* $y_1, \dots, y_n$ *drawn i.i.d. from an unknown distribution* $\mathbf{q}$ *over* $\mathcal{T}$, *is there sufficient evidence to reject the hypothesis* $\mathbf{p} = \mathbf{q}$*?*

More formally, the parlance of statistical testing, the null and alternative hypotheses are

$$\mathsf{H}_0 ::= [\mathbf{p} = \mathbf{q}] \qquad\qquad \mathsf{H}_1 ::= [\mathbf{p} \neq \mathbf{q}]. \tag{9.1}$$

A statistical test $\phi_n \colon \mathcal{T}^n \to \{\mathsf{reject}, \mathsf{not\ reject}\}$ maps each size n dataset to a decision as to whether the null hypothesis H_0 should be rejected or accepted. The significance level

$$\alpha ::= \Pr\left[\phi_n(Y_{1:n}) = \mathsf{reject} \mid \mathsf{H}_0\right] \tag{9.2}$$

is the probability of incorrectly declaring reject. For a given significance level α, the performance of the test ϕ_n is characterized by its power

$$\beta ::= \Pr\left[\phi_n(Y_{1:n}) = \mathsf{reject} \mid \mathsf{H}_1\right], \tag{9.3}$$

which is the probability of correctly declaring reject.

Related Work There are many goodness-of-fit tests for nominal (unordered) data, which include the multinomial test [Horn, 1977]; Pearson chi-square test [Pearson, 1900]; likelihood-ratio test [Williams, 1976]; nominal Kolmogorov–Smirnov test [Hoeffding, 1965, Pettitt and Stephens, 1977]; and power-divergence statistics [Read and Cressie, 1988]. For ordinal data, goodness-of-fit test statistics include the ordinal Watson, Cramér–von Mises, and Anderson–Darling tests [Choulakian et al., 1994] as well as the ordinal Kolmogorov–Smirnov test [Conover, 1972, Arnold and Emerson, 2011]. These methods have several challenges for large domains; for example, many of them assume that $\mathbf{p}(x)$ is tractable to compute, which is rarely possible in modern machine learning applications. These tests also require that each discrete outcome $x \in \mathcal{T}$ has a non-negligible expectation $n\mathbf{p}(x)$ [Maydeu-Olivares and Garcia-Forero, 2010], which can require a very large number of observations n. Moreover, the rejection regions of the statistics used in these tests are either distribution-dependent, which requires estimating the region for each new candidate distribution $\mathbf{p}$, or only known asymptotically, which gives inexact results for finite-sample data. The Mann–Whitney U [Mann and Whitney, 1947], which is also a rank-based test that bears some similarity to the SRS, is only consistent under median shift, whereas the SRS is consistent under arbitrary distributional differences.

Results in the theoretical computer science literature have established computational and sample complexity bounds for testing approximate equality of discrete distributions Batu et al. [2000]. These methods have been primarily studied from a theoretical perspective and have not been applied to yield practical goodness-of-fit tests, nor have they attained widespread adoption among practitioners or implemented in statistical software packages. For instance, the test in Acharya et al. [2015] is based on a variant of Pearson chi-square. It requires some form of enumeration over the domain $\mathcal{T}$ and representing

Algorithm 9.1 Exact goodness-of-fit test using the Stochastic Rank Statistic.

```
Require: { simulator for candidate distribution $\mathbf{p}$ over $\mathcal{T}$;
         { i.i.d. samples $\{y_1, y_2, \dots, y_n\}$ from distribution $\mathbf{q}$;
         { strict total order $\prec$ on $T$, of any order type;
         { number $m \geq 1$ of datasets to resimulate;
         { significance level $\alpha$ of hypothesis test;
Ensure: Decision to reject the null hypothesis $\mathsf{H}_0\colon \mathbf{p} = \mathbf{q}$ versus alternative hypothesis $\mathsf{H}_1 : \mathbf{p} \neq \mathbf{q}$.
for $i = 1, 2, \dots, n$ do
    $U_0 \sim \text{Uniform}(0, 1)$
    $r_i \leftarrow 0$
    for $j = 1, \dots, m$ do
        $X \sim \mathbf{p}$
        if $X = y_i$ then
            $U \sim \text{Uniform}(0, 1)$
            if $U < U_0$ then
                $r_i \leftarrow r_i + 1$
        else if $X \prec y_i$ then
            $r_i \leftarrow r_i + 1$
Use a hypothesis test to compute the $p$-value of data $r_1, \dots, r_n$ under a Uniform($\{0, \dots, m\}$) null.
return reject if $p \leq \alpha$ else not reject.
```

$\mathbf{p}(x)$ explicitly. The test in Valiant and Valiant [2011] requires specifying and solving a linear program. While these algorithms may obtain asymptotically sample-optimal limits, they do not account for any structure in the domain $\mathcal{T}$ that can be leveraged by the practitioner to effectively surface distributional differences that are of interest in a given application.

Permutation and bootstrap resampling of test statistics give another family of approaches for goodness-of-fit testing [Good, 2004]. Principled and consistent tests can be obtained using kernel methods, for example, including the maximum mean discrepancy [Gretton et al., 2012] and discrete Stein discrepancy [Yang et al., 2018]. Since the null distribution is unknown, rejection regions are estimated by bootstrap resampling, which may be inexact due to discreteness of the data. Instead of bootstrapping, the method proposed in this chapter can be used with kernel methods to obtain an exact, distribution-free test by defining an ordering using the norm induced by the kernel. That is, if the observation space $\mathcal{T}$ has a norm $\|\cdot\|$, then a strict partial ordering $\tilde{\prec}$ can be defined by $x \mathrel{\tilde{\prec}} x'$ if and only if $\|x\| < \|x'\|$ for $x, x' \in \mathcal{T}$. The ordering $\tilde{\prec}$ can then be elevated to an almost-surely linear ordering $\prec$ by using a lazily generated stochastic process $U_x, x \in \mathcal{T}$ of i.i.d. Uniform$(0, 1)$ random variables, where $x \prec x'$ holds if and only if $U_x < U_{x'}$ for all $x, x' \in \mathcal{T}$ such that $\|x\| = \|x'\|$.

9.2 The Stochastic Rank Statistic

This section describes the Stochastic Rank Statistics and shows how it can be used for goodness-of-fit testing. The proposed test combines (i) the intuition from existing methods for ordinal data [Choulakian et al., 1994] that the deviation between the expected CDF and empirical CDF of the sample serves as a good signal for goodness-of-fit, with (ii) the flexibility of probe statistics in Monte Carlo-based resampling tests [Good, 2004] to define, using an linear ordering $\prec$ on $\mathcal{T}$, characteristics of the distribution that are of interest to the experimenter. Figure 9.1 shows the step-by-step workflow of the proposed test and Algorithm 9.1 formally describes the testing procedure.

The method does not assume that $\mathbf{p}(x)$ is tractable to compute and is not based on comparing the expected frequency of each $x \in \mathcal{T}$ with its observed frequency. Furthermore, the stochastic ranks r_i Algorithm 9.1 have an exact and distribution-free sampling distribution. The following theorem establishes that the r_i are uniformly distributed if and only if $\mathbf{p} = \mathbf{q}$. Proofs of all results are given in Sections 9.2.1–9.2.5.

Theorem 9.1. *Let $\mathcal{T}$ be a finite or countably infinite set, $\prec$ be a strict total order on $\mathcal{T}$, $\mathbf{p}$ and $\mathbf{q}$ be two probability distributions on $\mathcal{T}$, and m be a positive integer. Consider the following random variables:*

$$X_0 \sim \mathbf{q} \tag{9.4}$$

$$X_1, X_2, \dots, X_m \sim^{\text{iid}} \mathbf{p} \tag{9.5}$$

$$U_0, U_1, U_2, \dots, U_m \sim^{\text{iid}} \text{Uniform}(0,1) \tag{9.6}$$

$$R = \textstyle\sum_{j=1}^{m} \mathbf{1}[X_j \prec X_0] + \mathbf{1}[X_j = X_0, U_j < U_0]. \tag{9.7}$$

Then $\mathbf{p} = \mathbf{q}$ if and only if for all $m \geq 1$, the rank R is distributed as a discrete uniform random variable on the set of integers $[m+1] ::= \{0, 1, \dots, m\}$. «

Note that the r_i in Algorithm 9.1 are n i.i.d. samples of the random variable R in Eq. (9.7), which is the rank of $X_0 \sim \mathbf{q}$ within a size m sample $(X_1, \dots, X_m) \sim^{\text{iid}} \mathbf{p}$. Theorem 9.1 only holds if ties are broken by pairing each X_i with a uniform random variable U_i, as opposed to, for example, breaking each tie independently with probability 1/2, as demonstrated by the next example.

Example 9.2. Let $\mathcal{T}$ contain a single element. Then all the X_i (for $0 \leq i \leq m$) are equal almost surely. Break each tie between X_0 and X_j by flipping a fair coin. Then R is binomially distributed with m trials and weight 1/2, not uniformly distributed over $[m+1]$. «

In the case where the X_i are almost-surely distinct, the forward direction of Theorem 9.1, which establishes that if $\mathbf{p} = \mathbf{q}$ then the rank R is uniform for all $m \geq 1$, is well known in the statistics literature [Ahsanullah et al., 2013]. However, to the best of found knowledge, no existing methods in the literature use rank-based goodness-of-fit tests based on an embedding of atomic distributions into continuous ones using stochastic ranks. Nor do they establish that $\mathbf{p} = \mathbf{q}$ is a *necessary* condition for uniformity of R across all m beyond some integer, and can therefore be used as the basis of a consistent goodness-of-fit test. The next result is a straightforward consequence of Theorem 9.1.

Corollary 9.3. *If $\mathbf{p} \neq \mathbf{q}$, then there is some $M \geq 1$ such that R is not uniformly distributed on $[M+1]$.* «

The next theorem significantly strengthens Corollary 9.3 by showing that if $\mathbf{p} \neq \mathbf{q}$, the rank statistic is nonuniform for *all but finitely many m*.

Theorem 9.4. *Let $\mathbf{p} \neq \mathbf{q}$ and M be defined as in Corollary 9.3. Then for all $m \geq M$, the rank R is not uniformly distributed on $[m+1]$.* «

In fact, unless $\mathbf{p}$ and $\mathbf{q}$ satisfy an adversarial symmetry relationship under the selected ordering $\prec$, the rank is nonuniform for *all* $m \geq 1$.

Corollary 9.5. *Let $\lhd$ denote the lexicographic order on $\mathcal{T} \times [0,1]$ induced by $(\mathcal{T}, \prec)$ and $([0,1], <)$. Suppose $\Pr[(X, U_1) \lhd (Y, U_0)] \neq 1/2$ for $Y \sim \mathbf{q}$, $X \sim \mathbf{p}$, and $U_0, U_1 \sim^{\text{iid}} \mathsf{Uniform}(0,1)$. Then for all $m \geq 1$, the rank R is not uniformly distributed on $[m+1]$.* «

The next theorem establishes the existence of a linear ordering on $\mathcal{T}$ that satisfies the hypothesis of Corollary 9.5.

Theorem 9.6. *If $\mathbf{p} \neq \mathbf{q}$, then there is an ordering $\prec^*$ whose associated rank statistic R is nonuniform for $m = 1$ (and hence by Theorem 9.4 for all $m \geq 1$).* «

Intuitively, the ordering $\prec^*$ sets elements $x \in \mathcal{T}$ that have a high probability under $\mathbf{q}$ to be "small" in the linear order, and elements $x \in \mathcal{T}$ that have a high probability under $\mathbf{p}$ to be "large" in the linear order. More precisely, $\prec^*$ maximizes the sup-norm distance between the induced cumulative distribution functions $\tilde{\mathbf{p}}$ and $\tilde{\mathbf{q}}$ of $\mathbf{p}$ and $\mathbf{q}$, respectively (Figure 9.3). Under a slight variant of this ordering, for finite $\mathcal{T}$, the next theorem establishes the sample complexity required to obtain exponentially high power in terms of the statistical distance $L_\infty(\mathbf{p}, \mathbf{q}) = \sup_{x\in\mathcal{T}}|\mathbf{p}(x) - \mathbf{q}(x)|$ between $\mathbf{p}$ and $\mathbf{q}$.

Theorem 9.7. *Given significance level $\alpha = 2\Phi(-c)$ for $c > 0$, there is an ordering for which the proposed test with $m = 1$ achieves power $\beta \geq 1 - \Phi(-c)$ using*

$$n \approx 4c^2/L_\infty(\mathbf{p}, \mathbf{q})^4 \tag{9.8}$$

samples from $\mathbf{q}$, where Φ is the cumulative distribution function of a standard normal. «

Theorem 9.7 is independent of the domain size and establishes a lower bound for any ordering $\prec$ because it is based on the provably optimal ordering $\prec^*$. The next theorem derives the exact sampling distribution for any pair of distributions $(\mathbf{p}, \mathbf{q})$, which is useful for simulation studies, such as the one in Figure 9.3, that characterize the power of the SRS against known alternatives.

Theorem 9.8. *The distribution of R is given by*

$$\Pr[R = r] = \sum_{x\in\mathcal{T}} H(x, m, r)\,\mathbf{q}(x) \tag{9.9}$$

for $0 \leq r \leq m$ where

$$H(x,m,r) ::= \begin{cases} \displaystyle\sum_{e=0}^{m} \left\{ \left[\sum_{j=0}^{e} \binom{m-e}{r-j} \left[\frac{\tilde{\mathbf{p}}(x)}{1-\mathbf{p}(x)}\right]^{r-j} \left[1 - \frac{\tilde{\mathbf{p}}(x)}{1-\mathbf{p}(x)}\right]^{(m-e)-(r-j)} \right.\right. & \text{if } 0 < \mathbf{p}(x) < 1, \\ \qquad\qquad \left.\left. \left(\frac{1}{e+1}\right) \right] \binom{m}{e} [\mathbf{p}(x)]^m \, [1-\mathbf{p}(x)]^{e-m} \right\} & \\ \displaystyle\binom{r}{m} [\tilde{\mathbf{p}}(x)]^r \, [1-\tilde{\mathbf{p}}(x)]^{m-r} & \text{if } \mathbf{p}(x) = 0, \\ \displaystyle\frac{1}{m+1} & \text{if } \mathbf{p}(x) = 1, \end{cases}$$

and $\tilde{\mathbf{p}}(x) ::= \sum_{x' \prec x} \mathbf{p}(x)$ is the cumulative distribution function of $\mathbf{p}$. «

Sections 9.2.1–9.2.5 establish several background results and prove Theorems 9.1, 9.4 and 9.6–9.8.

9.2.1 Proof: Uniformity of Rank

Before proving Theorem 9.1, several supporting lemmas are established. The first lemma proves that an i.i.d. sequence yields a uniform rank distribution.

Lemma 9.9. *Let $T_0, T_1, \dots, T_m$ be an i.i.d. sequence of random variables. If $\Pr[T_i = T_j] = 0$ for all distinct i and j, then the rank statistics $S_i ::= \sum_{j=0}^{m} \mathbf{1}[T_j \prec T_i]$ for $0 \leq i \leq m$ are each uniformly distributed on $[m+1]$.* «

Proof. Since $T_0, T_1, \dots, T_m$ is i.i.d., it is a finitely exchangeable sequence, and so the rank statistics $S_0, \dots, S_m$ are identically (but not independently) distributed. Fix an arbitrary $k \in [m+1]$. Then $\Pr[S_i = k] = \Pr[S_j = k]$ for all $i, j \in [m+1]$. By hypothesis, $\Pr[T_i = T_j] = 0$ for distinct i and j. Therefore the rank statistics are almost surely distinct, and the events $\{S_i = j\}$ (for $0 \leq i \leq m$) are mutually exclusive and exhaustive. Since these events partition the outcome space, their probabilities sum to 1, and so $\Pr[S_i = k] = 1/(m+1)$ for all $i \in [m+1]$. As k was arbitrary, S_i is uniformly distributed on $[m+1]$ for all $i \in [m+1]$. ■

The next result states a convergence property for discrete uniform random variables to continuous uniform random variables.

Lemma 9.10. *Let $(V_m)_{m\geq 1}$ be a sequence of discrete random variables such that V_m is uniformly distributed on $\{0, 1/m, 2/m, \ldots, 1\}$, and let U be a continuous random variable uniformly distributed on the interval $[0,1]$. Then $(V_m)_{m\geq 1}$ converges in distribution to U, i.e., for all $u \in [0,1]$,*

$$\lim_{m\to\infty} \Pr[V_m < u] = \Pr[U < u] = u. \tag{9.10}$$

Furthermore, the convergence (9.10) *is uniform in u.* «

Proof. Let $\epsilon > 0$. The distribution function F_m of V_m is given by

$$F_m(u) = \begin{cases} 1/(m+1) & u \in [0, 1/m) \\ 2/(m+1) & u \in [1/m, 2/m) \\ \cdots & \\ (a+1)/(m+1) & u \in [a/m, (a+1)/m) \\ \cdots & \\ m/(m+1) & u \in [(m-1)/m, 1) \\ 1 & u = 1. \end{cases} \tag{9.11}$$

For $0 \leq a < m$, the value $F_m(u)$ lies in the interval $[a/m, (a+1)/m)$ because $(a/m) < (a+1)/(m+1) < (a+1)/m$. Since u also lies in this interval,

$$|F_m(u) - u| \leq (a+1)/m - a/m = 1/m < \epsilon \tag{9.12}$$

whenever $m > 1/\epsilon$, for all u. ■

The next lemma gives an intermediate value property for step functions on the rationals.

Lemma 9.11. *Let $p\colon (\mathbb{Q} \cap [0,1]) \to [0,1]$ be a function satisfying $p(0) = 0$ and $\sum_{x\in\mathbb{Q}\cap[0,1]} p(x) = 1$. Then for each $\delta \in (0,1)$, there is some $w \in \mathbb{Q} \cap [0,1]$ such that*

$$\sum_{x\in\mathbb{Q}\cap(0,w)} p(x) \leq \delta \leq \sum_{x\in\mathbb{Q}\cap(0,w]} p(x). \tag{9.13}$$

«

Remark 9.12. The infinite sums in Lemma 9.11 whose index x ranges over a subset of the rationals can be formally defined as follows: Consider an arbitrary enumeration $\{q_1, q_2, \ldots, q_n, \ldots\}$ of $\mathbb{Q} \cap [0,1]$, and define the summation over the integer-valued index $n \geq 1$. Since the series consists of positive terms, it converges absolutely, and so all rearrangements of the enumeration converge to the same sum [Rudin, 1976, Theorem 3.55]. One can show that the Cauchy criterion holds in this setting. Namely, suppose that a sum $\sum_{a<x<c} p(x)$ of non-negative terms converges. Then for all $\epsilon > 0$ there is some rational $b \in (a,c)$ such that $\sum_{a<x\leq b} p(x) < \epsilon$. «

Proof of Theorem 9.1. As $\mathcal{T}$ is countable, by a standard Cantor "back-and-forth" argument, the total order $(\mathcal{T}, \prec)$ is isomorphic to $(B, <)$ for some set $B \subseteq \mathbb{Q} \cap (0,1)$. Without loss of generality, it will be assumed that $\mathcal{T} = \mathbb{Q} \cap [0,1]$ and that $\mathbf{p}(0) = \mathbf{p}(1) = 0$.

Consider the unit square $[0,1]^2$ equipped with the dictionary order $\lhd_{\mathrm{d}}$, which gives a total order with the least upper bound property. For each $i \in [m+1]$, let $T_i ::= (X_i, U_i)$, which is a random pair that takes values in $[0,1]^2$. The rank R in Eq. (9.7) of Theorem 9.1 is therefore equivalent to the rank $\sum_{i=0}^{m} \mathbf{1}[T_i \lhd_{\mathrm{d}} T_0]$ of T_0 taken according to the dictionary order.

Necessity Suppose $\mathbf{p} = \mathbf{q}$. Then $T_0, \ldots, T_m$ are independent and identically distributed. Since $U_0, \ldots, U_m$ are continuous random variables, $\Pr[T_i = T_j] = 0$ for all $i \neq j$. Apply Lemma 9.9.

Sufficiency Suppose that for all $m > 0$, the rank R is uniformly distributed on $\{0, 1, 2, \ldots, m\}$. Let $\tilde{\mathbf{p}} \colon [0,1] \to [0,1]$ be the "left-closed right-open" cumulative distribution function of $\mathbf{p}$, defined by

$$\tilde{\mathbf{p}}(x) ::= \sum_{y \in \mathbb{Q} \cap [0,x)} \mathbf{p}(y) \qquad (x \in [0,1]). \tag{9.14}$$

Define $\mathbf{p}'$ to be the probability measure on $[0,1]$ that is equal to $\mathbf{p}$ on subsets of $\mathbb{Q} \cap [0,1]$ and zero elsewhere. Define distribution function $F_{\mathbf{p}} \colon [0,1]^2 \to [0,1]$ on S by

$$F_{\mathbf{p}}(x,u) ::= \tilde{\mathbf{p}}(x) + u\mathbf{p}'(x) \qquad ((x,u) \in [0,1]^2). \tag{9.15}$$

To establish that $F_{\mathbf{p}}$ is a valid probability distribution function, it is sufficient to show that (i) its range is $[0,1]$; (ii) it is monotonically non-decreasing in each of its variables; and (iii) it is right-continuous in each of its variables. It is immediate that $F_{\mathbf{p}}(0,0) = 0$ and $F_{\mathbf{p}}(1,1) = 1$. To establish that $F_{\mathbf{p}}$ is monotonically non-decreasing, put $x < y$ and $u < v$. Then

$$\begin{aligned}
F_{\mathbf{p}}(x,u) &= \tilde{\mathbf{p}}(x) + u\mathbf{p}'(x) && (9.16)\\
&\leq \tilde{\mathbf{p}}(x) + \mathbf{p}'(x) && (9.17)\\
&\leq \textstyle\sum_{z \in \mathbb{Q} \cap [0,y)} \mathbf{p}'(z) && (9.18)\\
&= \tilde{\mathbf{p}}(y) && (9.19)\\
&\leq F_{\mathbf{p}}(y,u) && (9.20)
\end{aligned}$$

and

$$\begin{aligned}
F_{\mathbf{p}}(x,u) &= \tilde{\mathbf{p}}(x) + u\mathbf{p}'(x) && (9.21)\\
&\leq \tilde{\mathbf{p}}(x) + v\mathbf{p}'(x) && (9.22)\\
&= F_{\mathbf{p}}(x,v). && (9.23)
\end{aligned}$$

It remains to establish right-continuity. For fixed x, the function $u \mapsto F_{\mathbf{p}}(x,u)$ is linear in u and so continuity is immediate. For fixed u, $F_{\mathbf{p}}(x,u)$ has been shown to be nondecreasing so it is sufficient to show that for any x and for any $\epsilon > 0$ there exists $x' > x$ such that

$$\begin{aligned}
\epsilon &> F(x',u) - F(x,u) && (9.24)\\
&= \tilde{\mathbf{p}}(x') + u\mathbf{p}'(x') - \tilde{\mathbf{p}}(x) - u\mathbf{p}(x) && (9.25)\\
&= \tilde{\mathbf{p}}(x') + u\mathbf{p}'(x') - \tilde{\mathbf{p}}(x) - u\mathbf{p}(x) && (9.26)\\
&= \sum_{y \in \mathbb{Q} \cap [x,x']} \mathbf{p}(y), && (9.27)
\end{aligned}$$

which is immediate from the Cauchy criterion.

Finally, Lemma 9.11 and the continuity of $F_{\mathbf{p}}$ for fixed x and variable u together imply that $F_{\mathbf{p}}$ obtains all intermediate values. That is, for any $\delta \in [0,1]$ there is some (x,u) such that $F(x,u) = \delta$.

Next define the inverse $F_{\mathbf{p}}^{-1}\colon [0,1] \to [0,1]^2$, also known as the quantile function of $\mathbf{p}$, by

$$F_{\mathbf{p}}^{-1}(s) ::= \inf\{(x,u) \mid F_{\mathbf{p}}(x,u) = s\} \qquad (s \in [0,1]), \tag{9.28}$$

where the infimum is taken with the respect to the dictionary order $\lhd_{\mathrm{d}}$. The set in Eq. (9.28) is nonempty since $F_{\mathbf{p}}$ obtains all values in $[0,1]$. Moreover, $F_{\mathbf{p}}^{-1}(s) \in [0,1]^2$ since $\lhd_{\mathrm{d}}$ has the least upper bound property. This "generalized" notion of an inverse is required since $F_{\mathbf{p}}$ is one-to-one only under the stronger assumption that $\mathbf{p}(x) > 0$ for all $x \in \mathbb{Q} \cap (0,1)$. Analogously define $F_{\mathbf{q}}$ in terms of $\mathbf{q}$.

Define the rank function

$$r(a_0, \{a_1, \dots, a_m\}) ::= \sum_{i=0}^{m} \mathbf{1}[a_i < a_0], \tag{9.29}$$

so that $R \equiv r(T_0, \{T_1, \dots, T_m\})$. By the hypothesis of Theorem 9.1, $r(T_0, \{T_1, \dots, T_m\})/m$ is uniformly distributed on $\{0, 1/m, 2/m, \dots, 1\}$ for all $m > 0$. Applying Lemma 9.10 gives, for any $s \in [0,1]$,

$$\lim_{m\to\infty} \Pr\left[\frac{1}{m}\tilde{r}(T_0, \{T_1, \dots, T_m\}) < s\right] = \Pr\left[U_0 < s\right] = s. \tag{9.30}$$

For any $t \in [0,1]$ and $m \geq 1$, the random variable $\hat{F}_{\mathbf{p}}^{m}(t) ::= \tilde{r}(t, \{T_1, \dots, T_m\})/m$ is the empirical distribution of $F_{\mathbf{p}}$. By the Dehardt [1971, Corollary of Theorem 4], which generalizes the Glivenko–Cantelli theorem to empirical distribution functions on k-dimensional Euclidean space, the random sequence $(\hat{F}_{\mathbf{p}}^{m}(t))_{m\geq 1}$ converges almost surely to the real number $F_{\mathbf{p}}(t)$ uniformly in t. Thus, the random sequence $(\hat{F}_{\mathbf{p}}^{m}(T_0))_{m\geq 1}$ converges almost surely to the random variable $\hat{F}_{\mathbf{p}}(T_0)$, so that for any $s \in [0,1]$,

$$\begin{aligned}
\lim_{m\to\infty} \Pr\left[\frac{1}{m}\tilde{r}(T_0, \{T_1, \dots, T_m\}) < s\right] &= \lim_{m\to\infty} \Pr\left[\hat{F}_{\mathbf{p}}^{m}(T_0) < s\right] && (9.31)\\
&= \Pr\left[F_{\mathbf{p}}(T_0) < s\right] && (9.32)\\
&= \Pr\left[T_0 \lhd_{\mathrm{d}} F_{\mathbf{p}}^{-1}(s)\right] && (9.33)\\
&= F_{\mathbf{q}}(F_{\mathbf{p}}^{-1}(s)). && (9.34)
\end{aligned}$$

The interchange of the limit and the probability in Eq. (9.32) follows from the bounded convergence theorem, since $\hat{F}_{\mathbf{p}}^{m}(T_0) \to F_{\mathbf{p}}(T_0)$ almost surely, and for all $m \geq 1$, $|\hat{F}_{\mathbf{p}}^{m}(T_0)| \leq 1$ almost surely.

Combining Eqs. (9.30) and (9.34), for each $s \in (0,1)$,

$$F_{\mathbf{q}}(F_{\mathbf{p}}^{-1}(s)) = s \implies F_{\mathbf{p}}^{-1}(s) = F_{\mathbf{q}}^{-1}(s). \tag{9.35}$$

As $0 \leq F_{\mathbf{p}}(x,u) \leq 1$, for each $(x,u) \in [0,1]^2$, it follows that

$$\begin{aligned}
F_{\mathbf{q}}^{-1}(F_{\mathbf{p}}(x,u)) &= F_{\mathbf{p}}^{-1}(F_{\mathbf{p}}(x,u)) && (9.36)\\
&= F_{\mathbf{q}}^{-1}(F_{\mathbf{q}}(x,u)) && (9.37)\\
&= (x,u). && (9.38)
\end{aligned}$$

Therefore, $F_{\mathbf{p}}(x,u) = F_{\mathbf{q}}(x,u)$ for all $(x,u) \in [0,1]^2$. Fixing $u = 0$ gives

$$\tilde{\mathbf{p}}(x) = F_{\mathbf{p}}(x,0) = F_{\mathbf{q}}(x,0) = \tilde{\mathbf{q}}(x) \qquad (x \in [0,1]). \tag{9.39}$$

Assume towards a contradiction that $\mathbf{p} \neq \mathbf{q}$. Let a be any rational such that $\mathbf{p}(a) \neq \mathbf{q}(a)$, and suppose without loss of generality that $\mathbf{q}(a) < \mathbf{p}(a)$. By the Cauchy criterion in Lemma 9.11, there exists some $b > a$ such that

$$\sum_{a<x<b} \mathbf{q}(x) < \mathbf{p}(a) - \mathbf{q}(a).$$

Then

$$\begin{aligned}
\tilde{\mathbf{q}}(b) &= \tilde{\mathbf{q}}(a) + \mathbf{q}(a) + \sum_{x \in \mathbb{Q} \cap (a,b)} \mathbf{q}(x) \\
&= \tilde{\mathbf{p}}(a) + \mathbf{q}(a) + \sum_{x \in \mathbb{Q} \cap (a,b)} \mathbf{q}(x) \\
&< \tilde{\mathbf{p}}(a) + \mathbf{q}(a) + (\mathbf{p}(a) - \mathbf{q}(a)) \\
&= \tilde{\mathbf{p}}(a) + \mathbf{p}(a) \\
&\leq \tilde{\mathbf{p}}(b),
\end{aligned}$$

and so $\tilde{\mathbf{p}} \neq \tilde{\mathbf{q}}$, a contradiction to Eq. (9.39). ■

Before proving Theorem 9.4, a supporting lemma is needed.

Lemma 9.13. *Let $Z_1, \ldots, Z_{m+1}$ be a finitely exchangeable sequence of Bernoulli random variables. If*

$$S_m ::= \sum_{i=1}^{m} Z_i \tag{9.40}$$

is not uniformly distributed on $[m+1]$, then

$$S_{m+1} ::= \sum_{i=1}^{m+1} Z_i \tag{9.41}$$

is not uniformly distributed on $[m+2]$. «

Proof. By finite exchangeability, there is some $r \in [0,1]$ such that the distribution of every Z_i is Bernoulli(r). There are two cases to consider.

In the first case, $r \neq 1/2$. Then for any $\ell \geq 1$,

$$\mathbb{E}[S_\ell] = \mathbb{E}\left[\sum_{i=1}^{\ell} Z_i\right] = \sum_{i=1}^{\ell} \mathbb{E}[Z_i] = \ell r \neq r/2 = \mathbb{E}[U_\ell], \tag{9.42}$$

and so S_ℓ is not uniformly distributed on $[\ell+1]$. In particular, Eq. (9.42) holds for ℓ equal to m or $m+1$.

In the second case, $r = 1/2$. The lemma is established by proving the contrapositive. Suppose that S_{m+1} is uniformly distributed on $[m+1]$.

Assume S_{m+1} is uniform and fix $k \in [m+1]$. By total probability,

$$\Pr[S_m = k] = \Pr[S_m = k \text{ and } Z_{m+1} = 0] + \Pr[S_m = k \text{ and } Z_{m+1} = 1]. \tag{9.43}$$

The two events on the right-hand side of Eq. (9.43) are analyzed separately separately.

First, the event $\{S_m = k \text{ and } Z_{m+1} = 0\}$ in Eq. (9.43) is the union over all $\binom{m}{k}$ assignments of $(Z_1, \ldots, Z_m)$ that have exactly k ones and $Z_{m+1} = 0$. All such assignments are disjoint events. Let

$$A ::= \{Z_1 = \cdots = Z_k = 1 \text{ and } Z_{k+1} = \cdots = Z_m = Z_{m+1} = 0\}. \tag{9.44}$$

By finite exchangeability, each assignment has probability $\Pr[A]$, and so

$$\Pr[S_m = k \text{ and } Z_{m+1} = 0] = \binom{m}{k} \Pr[A]. \tag{9.45}$$

The event $\{S_{m+1} = k\}$ is the union of all $\binom{m+1}{k}$ assignments of $(Z_1, \dots, Z_{m+1})$ that have exactly k ones. Since all the assignments are disjoint events and each has probability $\Pr[A]$,

$$\Pr[S_{m+1} = k] = \binom{m+1}{k} \Pr[A] = \frac{1}{m+2}. \tag{9.46}$$

Second, $\{S_m = k \text{ and } Z_{m+1} = 1\}$ in Eq. (9.43) is the union over all $\binom{m}{k}$ assignments of $(Z_1, \dots, Z_m)$ that have exactly k ones and also $Z_{m+1} = 1$. All such assignments are disjoint events. Let

$$B ::= \{Z_1 = \dots = Z_k = Z_{m+1} = 1 \text{ and } Z_{k+1} = \dots = Z_m = 0\}. \tag{9.47}$$

Again by finite exchangeability, each assignment has probability $\Pr[B]$, and so

$$\Pr[S_m = k \text{ and } Z_{m+1} = 1] = \binom{m}{k} \Pr[B]. \tag{9.48}$$

The event $\{S_{m+1} = k+1\}$ is the union of all $\binom{m+1}{k+1}$ assignments of $(Z_1, \dots, Z_{m+1})$ that have exactly $k+1$ ones. Since the assignments are disjoint events and each has probability $\Pr[B]$,

$$\Pr[S_{m+1} = k+1] = \binom{m+1}{k+1} \Pr[B] = \frac{1}{m+2}. \tag{9.49}$$

Take Eq. (9.43), divide by $1/(m+2)$, and replace terms using Eqs. (9.45), (9.46), (9.48) and (9.49) to obtain

$$\begin{aligned}
\frac{\Pr[S_m = k]}{1/(m+2)} &= \frac{\Pr[S_m = k \text{ and } Z_{m+1} = 0]}{1/(m+2)} + \frac{\Pr[S_m = k \text{ and } Z_{m+1} = 1]}{1/(m+2)} && (9.50)\\
&= \frac{\binom{m}{k}\Pr[A]}{\binom{m+1}{k}\Pr[A]} + \frac{\binom{m}{k}\Pr[B]}{\binom{m+1}{k+1}\Pr[B]} && (9.51)\\
&= \frac{m!}{k!(m-k)!}\frac{k!(m+1-k)!}{(m+1)!} + \frac{m!}{k!(m-k)!}\frac{(k+1)!(m+1-(k+1))!}{(m+1)!} && (9.52)\\
&= \frac{m+1-k}{m+1} + \frac{k+1}{m+1} && (9.53)\\
&= \frac{m+2}{m+1} && (9.54)\\
&= \frac{1/(m+1)}{1/(m+2)}, && (9.55)
\end{aligned}$$

which proves that $\Pr[S_m = k] = 1/(m+1)$. ■

Proof of Theorem 9.4. Suppose that $\mathbf{p} \neq \mathbf{q}$. By Corollary 9.3, there exists $M \geq 1$ such that the rank statistic $R = \sum_{i=1}^{M} \mathbf{1}[T_i \prec T_0]$ is nonuniform over $[M+1]$. The rank statistic for $m = M+1$ is given by $\sum_{i=1}^{M+1} \mathbf{1}[T_i \prec T_0]$. Now, each indicator $Z_i ::= \mathbf{1}[T_i \prec T_0]$ is a Bernoulli random variable, and they are identically distributed since $(T_1, \dots, T_{M+1})$ is an i.i.d. sequence. Furthermore the random sequence

$(Z_1, \dots, Z_{M+1})$ is finitely exchangeable since the Z_i are conditionally independent given T_0. Then the sequence of indicators $(\mathbf{1}[T_1 \prec T_0], \mathbf{1}[T_2 \prec T_0], \dots, \mathbf{1}[T_{M+1} \prec T_0])$ satisfy the hypothesis of Lemma 9.13, and so the rank statistic for $M+1$ is nonuniform. By induction, the rank statistic is nonuniform for all $m \geq M$. ■

Proof of Corollary 9.5. If $\Pr[(X, U_1) \lhd (Y, U_0)] \neq 1/2$ then R is nonuniform for $m = 1$. To complete the proof, apply Theorem 9.4. ■

9.2.2 Proof: An Ordering Witnessing $\mathbf{p} \neq \mathbf{q}$ for $m = 1$

Toward establishing Theorem 9.6, this section presents an ordering $\prec$ that satisfies $\Pr[R = 0] > 1/2$ when $m = 1$. Let

$$A ::= \{x \in \mathcal{T} \mid \mathbf{q}(x) > \mathbf{p}(x)\} \tag{9.56}$$

be the set of all elements of $\mathcal{T}$ that have a greater probability according to $\mathbf{q}$ than according to $\mathbf{p}$, and let A^c denote its complement. Let $\mathbf{h}_{\mathbf{p},\mathbf{q}}$ be the signed measure given by the difference $\mathbf{h}_{\mathbf{p},\mathbf{q}}(x) ::= \mathbf{q}(x) - \mathbf{p}(x)$ between $\mathbf{q}$ and $\mathbf{p}$. For the rest of this section, $\mathbf{h}_{\mathbf{p},\mathbf{q}}$ is denoted $\mathbf{h}$. Let $\prec$ be any total order on $\mathcal{T}$ satisfying

$$\mathbf{h}(x) > \mathbf{h}(x') \implies x \prec x' \tag{9.57}$$

$$\mathbf{h}(x) < \mathbf{h}(x') \implies x \succ x \tag{9.58}$$

The linear ordering $\prec$ may be defined arbitrarily for all pairs x and x' such that $\mathbf{h}(x) = \mathbf{h}(x')$. As an immediate consequence, if $x \in A$ and $x' \in A^c$ then $x \prec x'$. Intuitively, the ordering is designed to ensure that elements $x \in A$ are "small", and are ordered by decreasing value of $\mathbf{q}(x) - \mathbf{p}(x)$ (with ties broken arbitrarily); elements $x \in A^c$ are "large" and are ordered by increasing value of $\mathbf{p}(x) - \mathbf{q}(x)$, with ties again broken arbitrarily. The smallest element in $\mathcal{T}$ maximizes $\mathbf{q}(x) - \mathbf{p}(x)$ and the largest element in $\mathcal{T}$ maximizes $\mathbf{p}(x) - \mathbf{q}(x)$.

Supporting lemmas are first established.

Lemma 9.14. *$A = \varnothing$ if and only if $\mathbf{p} = \mathbf{q}$.* «

Proof. Immediate. ■

Lemma 9.15.

$$\sum_{x \in A} [\mathbf{q}(x) - \mathbf{p}(x)] = \sum_{x \in A^c} [\mathbf{p}(x) - \mathbf{q}(x)]. \tag{9.59}$$

«

Proof.

$$\sum_{x \in A} [\mathbf{q}(x) - \mathbf{p}(x)] - \sum_{x \in A^c} [\mathbf{p}(x) - \mathbf{q}(x)] = \sum_{x \in \mathcal{T}} \mathbf{q}(x) - \sum_{x \in \mathcal{T}} \mathbf{p}(x) = 0, \tag{9.60}$$

■

Given a probability distribution $\mathbf{r}$, define its cumulative distribution function $\tilde{\mathbf{r}}$ by $\tilde{\mathbf{r}}(x) ::= \sum_{y \prec x} \mathbf{r}(y)$.

Lemma 9.16. *The inequality $\tilde{\mathbf{q}}(x) > \tilde{\mathbf{p}}(x)$ holds for all $x \in \mathcal{T}$.* «

Proof. Let $\mathcal{T}_x ::= \{y \in \mathcal{T} \mid y \prec x\}$. If $x \in A$ then $\mathcal{T}_x \subseteq A$, and so

$$\tilde{\mathbf{q}}(x) - \tilde{\mathbf{p}}(x) = \sum_{y \in \mathcal{T}_x} [\mathbf{q}(y) - \mathbf{p}(y)] > 0,$$

since all terms in the sum are positive. Otherwise, $y \in A$ for all $y \prec x$, and so $A \subseteq \mathcal{T}_x$. Let $A_x^c ::= \{y \in A^c \mid y \prec x\}$. Then

$$\begin{aligned}
\tilde{\mathbf{q}}(x) - \tilde{\mathbf{p}}(x) &= \sum_{y \prec x} [\mathbf{q}(y) - \mathbf{p}(y)] && (9.61) \\
&= \sum_{y \in A} [\mathbf{q}(y) - \mathbf{p}(y)] + \sum_{y \in A_x^c} [\mathbf{q}(y) - \mathbf{p}(y)] && (9.62) \\
&= \sum_{y \in A_x} [\mathbf{q}(y) - \mathbf{p}(y)] - \sum_{y \in A_x^c} [\mathbf{p}(y) - \mathbf{q}(y)] && (9.63) \\
&> \sum_{y \in A_x} [\mathbf{q}(y) - \mathbf{p}(y)] - \sum_{y \in A^c} [\mathbf{p}(y) - \mathbf{q}(y)] && (9.64) \\
&= 0, && (9.65)
\end{aligned}$$

establishing the lemma. ■

The probability $\Pr[R = 0]$ is now analyzed for the case that $m = 1$. In this case, Y is written in place of X_1, so that Eqs. (9.4)–(9.7) reduces to

$$\begin{aligned}
X_{\mathbf{p}} &\sim \mathbf{p} && (9.66) \\
Y_{\mathbf{q}} &\sim \mathbf{q} && (9.67) \\
R_{\mathbf{p},\mathbf{q}} \mid X_{\mathbf{p}}, Y_{\mathbf{q}} &\sim \begin{cases} 0 & \text{if } X_{\mathbf{p}} \succ Y_{\mathbf{q}}, \\ 1 & \text{if } X_{\mathbf{p}} \prec Y_{\mathbf{q}}, \\ \text{Bernoulli}(1/2) & \text{if } X_{\mathbf{p}} = Y_{\mathbf{q}}. \end{cases} && (9.68)
\end{aligned}$$

The generative process (9.66)–(9.68) samples $X_{\mathbf{p}} \sim \mathbf{p}$ and $Y_{\mathbf{q}} \sim \mathbf{q}$ independently. Given these values, it then sets $R_{\mathbf{p},\mathbf{q}}$ to be 0 if $X_{\mathbf{p}} \succ Y_{\mathbf{q}}$, to be 1 if $X_{\mathbf{p}} \prec Y_{\mathbf{q}}$, and the outcome of an independent fair coin flip otherwise. These random variables will be referred to as X, Y, and R when $\mathbf{p}$ and $\mathbf{q}$ are clear from context. The subscripts will be reintroduced when it is necessary to emphasize different settings of distributions $\mathbf{p}$ and $\mathbf{q}$.

Proof of Theorem 9.6. Total probability and independence of X and Y gives

$$\begin{aligned}
\Pr[R = 0] &= \sum_{x,y \in \mathcal{T}} \Pr[R = 0 \mid X = x, Y = y] \Pr[Y = y] \Pr[X = x] && (9.69) \\
&= \sum_{x,y \in \mathcal{T}} \Pr[R = 0 \mid X = x, Y = y]\, \mathbf{q}(y)\mathbf{p}(x) && (9.70) \\
&= \sum_{x \in \mathcal{T}} \Pr[R = 0 \mid X = x, Y = x]\, \mathbf{q}(x)\mathbf{p}(x) && (9.71) \\
&\quad + \sum_{y \prec x \in \mathcal{T}} \Pr[R = 0 \mid X = x, Y = y]\, \mathbf{q}(y)\mathbf{p}(x) \\
&\quad + \sum_{x \prec y \in \mathcal{T}} \Pr[R = 0 \mid X = x, Y = y]\, \mathbf{q}(y)\mathbf{p}(x) \\
&= \frac{1}{2} \sum_{x \in \mathcal{T}} \mathbf{q}(x)\mathbf{p}(x) + 1 \sum_{y \prec x \in \mathcal{T}} \mathbf{q}(y)\mathbf{p}(x) + 0 \sum_{x \prec y \in \mathcal{T}} \mathbf{q}(y)\mathbf{p}(x) && (9.72) \\
&= \frac{1}{2} \sum_{x \in \mathcal{T}} \mathbf{p}(x)\mathbf{q}(x) + \sum_{x \in \mathcal{T}} \tilde{\mathbf{q}}(x)\mathbf{p}(x). && (9.73)
\end{aligned}$$

An identical argument establishes that

$$\Pr\left[R=1\right] = \frac{1}{2}\sum_{x\in\mathcal{T}} \mathbf{p}(x)\mathbf{q}(x) + \sum_{x\in\mathcal{T}} \tilde{\mathbf{p}}(x)\mathbf{q}(x). \tag{9.74}$$

Since $\Pr\left[R{=}0\right] + \Pr\left[R=1\right] = 1$, it suffices to establish that $\Pr\left[R=0\right] > \Pr\left[R=1\right]$:

$$\begin{aligned}
\Pr\left[R=0\right] - \Pr\left[R=1\right] &= \sum_{x\in\mathcal{T}} \tilde{\mathbf{q}}(x)\mathbf{p}(x) - \sum_{x\in\mathcal{T}} \tilde{\mathbf{p}}(x)\mathbf{q}(x) &(9.75)\\
&> \sum_{x\in\mathcal{T}} \tilde{\mathbf{p}}(x)\mathbf{p}(x) - \sum_{x\in\mathcal{T}} \tilde{\mathbf{p}}(x)\mathbf{q}(x) &(9.76)\\
&= \sum_{x\in\mathcal{T}} \tilde{\mathbf{p}}(x)[\mathbf{p}(x) - \mathbf{q}(x)] &(9.77)\\
&= \sum_{x\in A^c} \tilde{\mathbf{p}}(x)[\mathbf{p}(x) - \mathbf{q}(x)] - \sum_{x\in A} \tilde{\mathbf{p}}(x)[\mathbf{q}(x) - \mathbf{p}(x)] &(9.78)\\
&\geq \sum_{x\in A^c} \big(\max_{y\in A} \tilde{\mathbf{p}}(y)\big)[\mathbf{p}(x) - \mathbf{q}(x)] - \sum_{x\in A} \tilde{\mathbf{p}}(x)[\mathbf{q}(x) - \mathbf{p}(x)] &(9.79)\\
&= \sum_{x\in A} \big(\max_{y\in A} \tilde{\mathbf{p}}(y)\big)[\mathbf{q}(x) - \mathbf{p}(x)] - \sum_{x\in A} \tilde{\mathbf{p}}(x)[\mathbf{q}(x) - \mathbf{p}(x)] &(9.80)\\
&= \sum_{x\in A} \big(\max_{y\in A} \tilde{\mathbf{p}}(y) - \tilde{\mathbf{p}}(x)\big)[\mathbf{q}(x) - \mathbf{p}(x)] &(9.81)\\
&> 0. &(9.82)
\end{aligned}$$

Eq. (9.76) follows from Lemma 9.16; Eq. (9.79) follows from monotonicity of $\tilde{\mathbf{p}}$; Eq. (9.80) follows from Lemma 9.15; and Eq. (9.82) follows from since all terms in the sum are positive. ■

9.2.3 Proof: A Tighter Bound in Terms of $L_\infty(\mathbf{p}, \mathbf{q})$

Section 9.2.2 exhibited an ordering such that $\Pr\left[R=0\right] > 1/2$ whenever $\mathbf{p} \neq \mathbf{q}$ and $m = 1$. This section proves a tighter lower bound on this probability in terms of the L_∞ distance between $\mathbf{p}$ and $\mathbf{q}$. In this section and the following, the domain $\mathcal{T}$ is assumed to be finite. The following lemma is immediate.

Lemma 9.17. *Let $B, C \subseteq \mathcal{T}$. For all $\mathbf{p}, \mathbf{q}$ and all $\delta > 0$ there is an $\epsilon > 0$ such that for all distributions $\mathbf{p}'$ on $\mathcal{T}$ with $\sup_{x\in\mathcal{T}} |\mathbf{p}(x) - \mathbf{p}'(x)| < \epsilon$,*

$$\left|\Pr\left[R_{\mathbf{p},\mathbf{q}} = 0 \,|\, X_{\mathbf{p}} \in B,\ Y_{\mathbf{q}} \in C\right] - \Pr\left[R_{\mathbf{p}',\mathbf{q}} = 0 \,|\, X_{\mathbf{p}'} \in B,\ Y_{\mathbf{q}} \in C\right]\right| < \delta. \tag{9.83}$$

«

Definition 9.18. The distribution $\mathbf{p}$ is said to be ϵ-discrete (with respect to $\mathbf{q}$) if for all $a, b \in \mathcal{T}$,

$$\left|\mathbf{h}_{\mathbf{p},\mathbf{q}}(a) - \mathbf{h}_{\mathbf{p},\mathbf{q}}(b)\right| \geq \epsilon. \tag{9.84}$$

«

Lemma 9.17 implies the following result

Lemma 9.19. *For all $\mathbf{p}, \mathbf{q}$ and all $\delta > 0$ there is an $\epsilon > 0$ and an ϵ-discrete distribution $\mathbf{p}_\epsilon$ on $\mathcal{T}$ such that for all $B, C \subseteq \mathcal{T}$,*

$$\left|\Pr\left[R_{\mathbf{p},\mathbf{q}} = 0 \,|\, X_{\mathbf{p}} \in B,\ Y_{\mathbf{q}} \in C\right] - \Pr\left[R_{\mathbf{p}_\epsilon,\mathbf{q}} = 0 \,|\, X_{\mathbf{p}_\epsilon} \in B,\ Y_{\mathbf{q}} \in C\right]\right| < \delta. \tag{9.85}$$

«

The next lemma is needed to prove a lower bound on $\Pr[R_{\mathbf{p},\mathbf{q}} = 0]$.

Lemma 9.20. *Let $\mathbf{p}_0$ and $\mathbf{p}_1$ be probability measures on $\mathcal{T}$, and let $\lhd$ be a total order on $\mathcal{T}$ such that if $\mathbf{h}_{\mathbf{p}_0,\mathbf{q}}(x) > \mathbf{h}_{\mathbf{p}_0,\mathbf{q}}(x')$ then $x \lhd x'$ and if $\mathbf{h}_{\mathbf{p}_0,\mathbf{q}}(x) < \mathbf{h}_{\mathbf{p}_0,\mathbf{q}}(x')$ then $x \rhd x'$. Suppose that if $\mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(x) > 0$ and $\mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(y) \le 0$, then $x \lhd y$. Then $\Pr[R_{\mathbf{p}_0,\mathbf{q}} = 0] \ge \Pr[R_{\mathbf{p}_1,\mathbf{q}} = 0]$.* «

Proof. Note that

$$\begin{aligned}
\Pr[R_{\mathbf{p}_1,\mathbf{q}} = 0 \mid Y_{\mathbf{q}} = y] &= \sum_{x \rhd y} \mathbf{p}_1(x) + \frac{1}{2}\mathbf{p}_1(y) && (9.86)\\
&= \sum_{x \rhd y} \mathbf{p}_0(x) + \mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(x) + \frac{1}{2}[\mathbf{p}_0(y) + \mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(y)] && (9.87)\\
&= \Pr[R_{\mathbf{p}_0,\mathbf{q}} = 0 \mid Y_{\mathbf{q}} = y] + \sum_{x \rhd y} \mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(x) + \frac{1}{2}\mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(y) && (9.88)\\
&= \Pr[R_{\mathbf{p}_0,\mathbf{q}} = 0 \mid Y_{\mathbf{q}} = y] - \sum_{x \lhd y} \mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(x) - \frac{1}{2}\mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(y), && (9.89)
\end{aligned}$$

where the last equality holds because $\sum_{x \in \mathcal{T}} \mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(x) = 0$. However, from the assumption, the term $\sum_{x \lhd y} \mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(x) + \frac{1}{2}\mathbf{h}_{\mathbf{p}_0,\mathbf{p}_1}(y)$ is non-negative and so $\Pr[R_{\mathbf{p}_1,\mathbf{q}} = 0 \mid Y_{\mathbf{q}} = y] \le \Pr[R_{\mathbf{p}_0,\mathbf{q}} = 0 \mid Y_{\mathbf{q}} = y]$, from which the conclusion follows. ■

The next result gives a lower bound on $\Pr[R_{\mathbf{p},\mathbf{q}} = 0]$. Its proof is beyond the scope of this thesis and can be found in Saad et al. [2019b, Appendix A.3].

Proposition 9.21. *For $R_{\mathbf{p},\mathbf{q}}$ in Eq. (9.68), the following lower bound holds:*

$$\Pr[R_{\mathbf{p},\mathbf{q}} = 0] \ge \frac{1}{2} + \frac{1}{2}\max_{x \in \mathcal{T}} \mathbf{h}_{\mathbf{p},\mathbf{q}}(x)^2. \tag{9.90}$$

«

Proposition 9.21 implies the following theorem, which is needed for the proof of Theorem 9.7.

Theorem 9.22. *Given probability measure $\mathbf{p}, \mathbf{q}$ on $\mathcal{T}$ there is a linear ordering $\sqsubset$ of $\mathcal{T}$ such that if $X_{\mathbf{p}}$ and $Y_{\mathbf{q}}$ are sampled independently from $\mathbf{p}$ and $\mathbf{q}$ respectively then*

$$\Pr[X_{\mathbf{q}} \sqsubset Y_{\mathbf{p}}] \ge \frac{1}{2} + \frac{1}{2}L_\infty(\mathbf{p},\mathbf{q})^2. \tag{9.91}$$

«

Proof. First,

$$L_\infty(\mathbf{p},\mathbf{q}) = \max\{\max_{x \in \mathcal{T}} \mathbf{h}_{\mathbf{p},\mathbf{q}}(x), \max_{x \in \mathcal{T}} \mathbf{h}_{\mathbf{q},\mathbf{p}}(x)\}. \tag{9.92}$$

If $L_\infty(\mathbf{p},\mathbf{q}) = \max_{x \in \mathcal{T}} \mathbf{h}_{\mathbf{p},\mathbf{q}}(x)$, then the theorem follows from Proposition 9.21 using the ordering $x \sqsubset y$ if and only if $\mathbf{h}_{\mathbf{p},\mathbf{q}}(x) > \mathbf{h}_{\mathbf{p},\mathbf{q}}(y)$. If, however, $L_\infty(\mathbf{p},\mathbf{q}) = \max_{x \in \mathcal{T}} \mathbf{h}_{\mathbf{q},\mathbf{p}}(x)$, then the conclusion follows from Proposition 9.21 by interchanging $\mathbf{p}$ and $\mathbf{q}$, that is, using the ordering $\sqsubset$ defined by the rule $x \sqsubset y$ if and only if $\mathbf{h}_{\mathbf{q},\mathbf{p}}(x) > \mathbf{h}_{\mathbf{q},\mathbf{p}}(y)$. ■

9.2.4 Proof: Sample Complexity of SRS Test

The result in Theorem 9.22 can be amplified using repeated trials to obtain a bound on the sample complexity of the main algorithm for determining whether $\mathbf{p} = \mathbf{q}$. In particular, the following proof of Theorem 9.7 uses the linear ordering $\sqsubset$ defined in Theorem 9.22.

Proof of Theorem 9.7. Assume without loss of generality that the order $\sqsubset$ from Theorem 9.22 is such that $L_\infty = \max_{s\in\mathcal{T}}(\mathbf{q}(x) - \mathbf{p}(x))$. Let $(Y_1, \dots, Y_n) \sim^{\text{iid}} \mathbf{q}$ be the n samples from $\mathbf{q}$. With $m = 1$, the testing procedure generates n samples $(X_1, \dots, X_n) \sim^{\text{iid}} \mathbf{p}$, and $2n$ uniform random variables $(U_1^Y, \dots, U_n^Y, U_1^X, \dots, U_n^X) \sim^{\text{iid}}$ Uniform$(0, 1)$ to break ties. Let $\lhd$ denote the lexicographic order on $\mathcal{T} \times [0, 1]$ induced by $(\mathcal{T}, \lhd)$ and $([0, 1], <)$. Define $W_i ::= \mathbf{1}\big[(Y_i, U_i^Y) \lhd (X_i, U_i^X)\big]$, for $1 \le i \le n$, to be the rank of the i^{th} observation from $\mathbf{q}$.

Under the null hypothesis H_0, each rank W_i has distribution Bernoulli$(1/2)$ by Lemma 9.9. Testing whether the ranks have a uniform distribution on $\{0, 1\}$ is equivalent to testing whether a coin is unbiased given the i.i.d. flips $\{W_1, \dots, W_n\}$. Let $\hat{B} ::= \sum_{i=1}^n (1 - W_i)/n$ denote the empirical proportion of zeros. By the central limit theorem, for sufficiently large n, $\hat{B}$ is approximately normally distributed with mean $1/2$ and standard deviation $1/(2\sqrt{n})$. For the given significance level $\alpha = 2\Phi(-c)$, the two-sided reject region $F = (-\infty, \gamma) \cup (\gamma, \infty)$ is such that critical value γ satisfies

$$c = \frac{\gamma - 1/2}{1/(2\sqrt{n})} = 2\sqrt{n}(\gamma - 1/2). \tag{9.93}$$

Replacing n in Eq. (9.8) gives

$$\begin{aligned}\gamma &= 1/2 + c/(2\sqrt{n}) && (9.94)\\ &= 1/2 + c/(2(2c/L_\infty(\mathbf{p}, \mathbf{q})^2)) && (9.95)\\ &= 1/2 + L_\infty(\mathbf{p}, \mathbf{q})^2/4. && (9.96)\end{aligned}$$

This construction ensures that $\Pr[\mathsf{reject} \mid \mathsf{H}_0] = \alpha$.

It remains to be shown that the test with this rejection region has power $\beta \ge \Pr[\mathsf{reject} \mid \mathsf{H}_1] = 1 - \Phi(-c)$. Under the alternative hypothesis H_1, each W_i has (in the worst case) distribution Bernoulli$(1/2 + L_\infty(\mathbf{p}, \mathbf{q})^2/2)$ by Theorem 9.22. Hence, the empirical proportion $\hat{B}$ is approximately normally distributed with mean at least $1/2 + L_\infty(\mathbf{p}, \mathbf{q})^2/2$ and standard deviation at most $1/(2\sqrt{n})$. Under the alternative distribution of $\hat{B}$, the standard score c' of the critical value γ is

$$\begin{aligned}c' &= \frac{\gamma - (1/2 + L_\infty(\mathbf{p}, \mathbf{q})^2/2)}{1/(2\sqrt{n})} && (9.97)\\ &= 2\sqrt{n}((1/2 + L_\infty(\mathbf{p}, \mathbf{q})^2/4) - (1/2 + L_\infty(\mathbf{p}, \mathbf{q})^2/2)) && (9.98)\\ &= -2\sqrt{n}(L_\infty(\mathbf{p}, \mathbf{q})^2/4) && (9.99)\\ &= -\sqrt{n}L_\infty(\mathbf{p}, \mathbf{q})^2/2 && (9.100)\\ &= -c, && (9.101)\end{aligned}$$

where the second equality follows from Eq. (9.96). The $\mathsf{not\ reject}$ region $E = [-\gamma, \gamma] \subset (-\infty, \gamma]$, and so the probability $\hat{B} \in e$ is at most the probability that $\hat{B} < \gamma$, which by Eq. (9.101) is equal to $\Phi(-c)$. It follows that $\beta \ge 1 - \Phi(-c)$. ■

Corollary 9.23. *As the significance level α varies, the test with ordering $\sqsubset$ and $m = 1$ achieves an overall error*

$$\frac{\alpha + (1 - \beta)}{2} \le \frac{3\Phi(-c)}{2} \tag{9.102}$$

using $n = 4c^2/L_\infty(\mathbf{p}, \mathbf{q})^4$ samples. «

9.2.5 Proof: Distribution of SRS under Alternative Hypothesis

This section derives the distribution of R under the alternative hypothesis $\mathbf{p} \neq \mathbf{q}$. As before, the cumulative distribution of $\mathbf{p}$ is denoted $\tilde{\mathbf{p}}(x) ::= \sum_{x'<x} \mathbf{p}(x)$.

Proof of Theorem 9.8. Define the following random variables:

$$L ::= \sum_{i=1}^{m} \mathbb{I}\left[X_i \prec X_0\right], \tag{9.103}$$

$$E ::= \sum_{i=1}^{m} \mathbb{I}\left[X_i = X_0\right], \tag{9.104}$$

$$G ::= \sum_{i=1}^{m} \mathbb{I}\left[X_i \succ X_0\right]. \tag{9.105}$$

The variables L, E, and G are referred to as "bins", where L is the "less than" bin, E is the "equal to" bin, and G is the "greater than" bin (all with respect to X_0). Total probability gives

$$\Pr\left[R = r\right] = \sum_{x \in \mathcal{T}} \Pr\left[R = r, X_0 = x\right] = \sum_{\substack{x \in \mathcal{T} \\ \mathbf{q}(x) > 0}} \Pr\left[R = r \mid X_0 = x\right] \mathbf{q}(x). \tag{9.106}$$

Fix $x \in \mathcal{T}$ such that $\mathbf{q}(x) > 0$. Consider $\Pr\left[R = r \mid X_0 = s\right]$. The counts in bins L, E, and G are binomial random variables with m trials, where the bin L has success probability $\tilde{\mathbf{p}}(x)$, the bin E has success probability $\mathbf{p}(x)$, and the bin G has success probability $1 - (\tilde{\mathbf{p}}(x) + \mathbf{p}(x))$.

There are three cases to consider.

Case 1. $\mathbf{p}(x) = 0$. The event $\{E = 0\}$ occurs with probability one since each X_i, for $1 \leq i \leq m$, cannot possibly be equal to x. Therefore, conditioned on $\{X_0 = x\}$, the event $\{R = r\}$ occurs if and only if $\{L = r\}$. Since L is binomially distributed,

$$\Pr\left[R = r \mid X_0 = x\right] = \Pr\left[L = r \mid X_0 = x\right] = \binom{m}{r} \left[\tilde{\mathbf{p}}(x)\right]^r \left[1 - \tilde{\mathbf{p}}(x)\right]^{m-r}. \tag{9.107}$$

Case 2. $\mathbf{p}(x) = 1$. Then the event $\{E = m\}$ occurs with probability one since each X_i, for $1 \leq i \leq m$, can only equal s. The uniform numbers $U_0, \dots, U_m$ used to break the ties will determine the rank R of X_0. Let B be the rank of U_0 among the m other uniform random variables $U_1, \dots, U_m$. The event $\{R = r\}$ occurs if and only if $\{B = r\}$. Since the U_i are i.i.d., B is uniformly distributed over $\{0, 1, 2, \dots, m\}$ by Lemma 9.9. Therefore,

$$\Pr\left[R = r \mid X_0 = x\right] = \Pr\left[B = r \mid X_0 = x\right] = \frac{1}{m+1}. \tag{9.108}$$

Case 3. $0 < \mathbf{p}(x) < 1$. By total probability,

$$\Pr\left[R = r \mid X_0 = x\right] = \sum_{e=0}^{m} \Pr\left[R = r \mid X_0 = x, E = e\right] \Pr\left[E = e \mid X_0 = x\right]. \tag{9.109}$$

Since E is binomially distributed,

$$\Pr\left[E = e \mid X_0 = x\right] = \binom{m}{e} \left[\mathbf{p}(x)\right]^e \left[1 - \mathbf{p}(x)\right]^{m-e}. \tag{9.110}$$

The event $\{R = r \,|\, X_0 = x, E = e\}$ is analyzed next. The uniform numbers $U_0, \ldots, U_m$ used to break the ties will determine the rank R of X_0. Define B to be the rank of U_0 among the e other uniform random variables assigned to bin E, i.e., those U_i for $1 \le i \le m$ such that $X_i = s$. The random variable B is independent of all the X_i, but is dependent on E. Given $\{E = e\}$, B is uniformly distributed on $\{0, 1, \ldots, e\}$. By total probability,

$$\Pr[R = r \,|\, X_0 = x, E = e] = \sum_{b=0}^{e} [\Pr[R = r \,|\, X_0 = x, E = e, B = b] \Pr[B = b \,|\, E = e]] \tag{9.111}$$

$$= \sum_{b=0}^{e} \Pr[R = r \,|\, X_0 = x, E = e, B = b] \frac{1}{e+1}. \tag{9.112}$$

Conditioned on $\{E = e\}$ and $\{B = 0\}$, the event $\{R = r\}$ occurs if and only if $\{L = r\}$, since exactly 0 random variables in bin E "are less" than X_0, so exactly r random variables in bin L are needed to ensure that the rank of X_0 is r. By the same reasoning, conditioning on $\{E = e, B = b\}$ gives $\{R = r\}$ if and only if $\{L = r - b\}$ (for $0 \le b \le e$). Now, conditioned on $\{E = e\}$, there are $m - e$ remaining assignments to be split among bins L and G. Let i be such that $X_i \ne x$. Then the relative probability that X_i is assigned to bin L is $\tilde{\mathbf{p}}(x)$ and to bin G is $1 - (\tilde{\mathbf{p}}(x) + \mathbf{p}(x))$. By renormalizing these probabilities, it is seen that L is conditionally (given $\{E = e\}$) a binomial random variable with $m - e$ trials and success probability

$$\frac{\tilde{\mathbf{p}}(x)}{\tilde{\mathbf{p}}(x) + (1 - (\tilde{\mathbf{p}}(x) + \mathbf{p}(x)))} = \frac{\tilde{\mathbf{p}}(x)}{1 - \mathbf{p}(x)}. \tag{9.113}$$

Therefore,

$$\Pr[R = r \,|\, X_0 = x, E = e, B = b] \tag{9.114}$$

$$= \Pr[L = r - b \,|\, X_0 = x, E = e] \tag{9.115}$$

$$= \binom{m-e}{r-j} \left[\frac{\tilde{\mathbf{p}}(x)}{1 - \mathbf{p}(x)}\right]^{r-j} \left[1 - \frac{\tilde{\mathbf{p}}(x)}{1 - \mathbf{p}(x)}\right]^{(m-e)-(r-j)}, \tag{9.116}$$

which completes the proof. ■

Remark 9.24. The sum in Eq. (9.9) of Theorem 9.8 converges since $H(x, m, r) \le 1$. «

Remark 9.25. Theorem 9.8 shows that it is not the case that $\mathbf{p} = \mathbf{q}$ whenever there exists some m for which the rank R is uniform on $[m + 1]$. For example, let $m = 1$, $\mathcal{T} ::= \{0, 1, 2, 3\}$, $\prec$ be the usual order $<$ on the integers. Put

$$\mathbf{p} ::= \frac{1}{2}\delta_0 + \frac{1}{2}\delta_3 \tag{9.117}$$

$$\mathbf{q} ::= \frac{1}{2}\delta_1 + \frac{1}{2}\delta_2 \tag{9.118}$$

Let $X \sim \mathbf{p}$ and $Y \sim \mathbf{q}$. Then $\Pr[R = 0] = \Pr[X > Y] = 1/2 = \Pr[Y < X] = \Pr[R = 1]$. Rather, Theorem 9.1 merely states if R is not uniform on $\{0, \ldots, m\}$ for *some* m, then $\mathbf{p} \ne \mathbf{q}$. For the distributions in Eqs. (9.117) and (9.118), the setting $m = 2$ provides such a witness, as does any $m > 2$ by Theorem 9.4. «

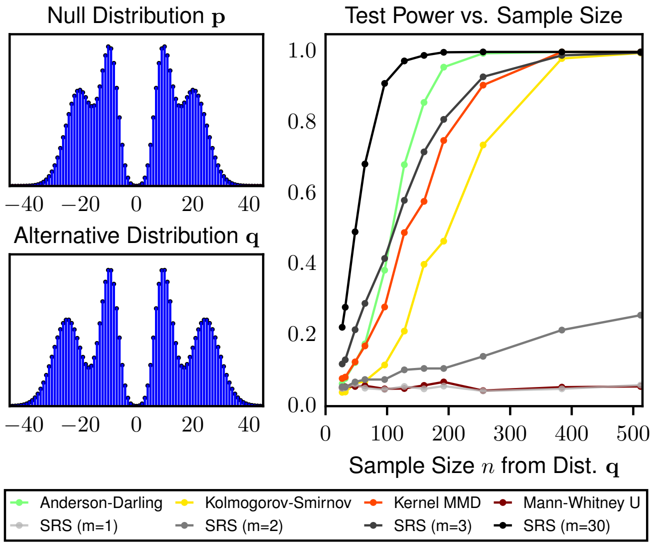


Figure 9.2: The left panel shows a pair $(\mathbf{p}, \mathbf{q})$ of reflected, bimodal Poisson distributions with slight location shift. The right plot compares the power of testing $\mathbf{p} = \mathbf{q}$ using the SRS for various m and several baseline methods.

9.3 Simulation Studies

This section applies the proposed goodness-of-fit in Algorithm 9.1 to data from a countably infinite domain and two finite high-dimensional domains, illustrating a power comparison and how distributional differences can be detected when the number of observations is much smaller than the domain size. The Pearson chi-square test is used to assess uniformity of the SRS in line 12 of Algorithm 9.1; refer to Steele and Chaseling [2006] for a comparison of alternative techniques to test for a discrete uniform.

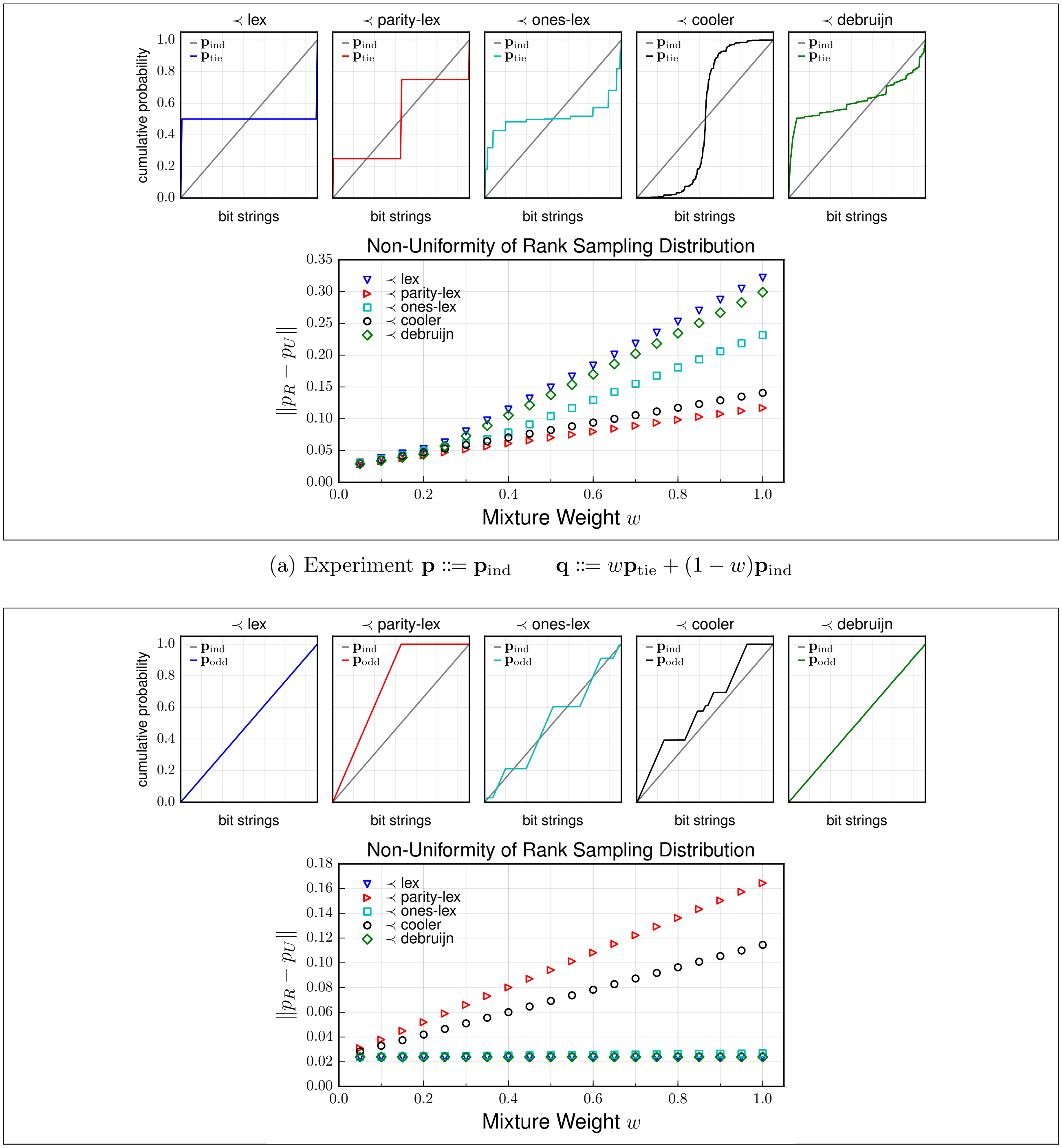
9.3.1 Bimodal Symmetric Poisson

For $x \in \mathbb{Z}$, define the discrete distribution

$$\mathbf{f}(x; \lambda_1, \lambda_2) ::= \frac{1}{2}\left(\frac{1}{2}\text{Poisson}(|x|; \lambda_1) + \frac{1}{2}\text{Poisson}(|x|; \lambda_2)\right), \tag{9.119}$$

which specifies a mixture of Poisson distributions with rates λ_1 and λ_2, reflected symmetrically about $x = 0$. Setting $\mathbf{p}(x) ::= \mathbf{f}(x; 10, 20)$ and $\mathbf{q}(x) ::= \mathbf{f}(x; 10, 25)$ produces an alternative distribution $\mathbf{q}$ that is location-shifted from $\mathbf{p}$ in two of the four modes, as shown in the left panel of Figure 9.2.

The right plot of Figure 9.2 compares the power of tests for $\mathbf{p} = \mathbf{q}$ for various sample sizes n from $\mathbf{q}$ using the SRS test ($m =, 1, 2, 3, 30$, shown in increasing shades of gray) and several baselines (shown in color). The baselines are used to assess goodness-of-fit by performing a two-sample test on n samples from $\mathbf{q}$ with samples drawn i.i.d. from $\mathbf{p}$. The power at level $\alpha = 0.05$ is estimated as the fraction of correct answers over 1024 independent trials. The Mann–Whitney U, which is also based on rank statistics with a correction for ties, has no power for all n as it can only detect median shift, as does the SRS with $m = 1$ (Corollary 9.5). The SRS becomes nonuniform for $m = 2$ although this choice results in low power. The SRS with $m = 3$ has comparable power to the Anderson-Darling (AD) test and Maximum Mean Discrepancy (MMD) kernel test. The SRS with $m = 30$ is the most powerful, although it requires more computational effort and samples from $\mathbf{p}$, as Algorithm 9.1 scales as $O(mn)$.



(a) Experiment $\mathbf{p} ::= \mathbf{p}_{\mathrm{ind}} \qquad \mathbf{q} ::= w\mathbf{p}_{\mathrm{tie}} + (1-w)\mathbf{p}_{\mathrm{ind}}$

(b) Experiment $\mathbf{p} ::= \mathbf{p}_{\mathrm{ind}} \qquad \mathbf{q} ::= w\mathbf{p}_{\mathrm{odd}} + (1-w)\mathbf{p}_{\mathrm{ind}}$

Figure 9.3: In each of the panels (a) and (b) above, the first plot compares the cumulative distribution function of the uniform distribution $\mathbf{p} ::= \mathbf{p}_{\mathrm{ind}}$ on $\{0,1\}^{16}$ (diagonal gray line) with an alternative distribution (colored)—(a) $\mathbf{p}_{\mathrm{tie}}$ and (b) $\mathbf{p}_{\mathrm{odd}}$—for six different orderings on the binary strings. The bottom plot shows the sup-norm distance between the sampling distribution of the stochastic rank statistic and discrete uniform distribution over $\{1,\dots,6\}$ for alternative distributions of the form $\mathbf{q} ::= w\mathbf{p}_{\mathrm{alt}} + (1-w)\mathbf{p}_{\mathrm{ind}}$, $w \in [0,1]$. Orderings that induce a greater distance between the cumulative distribution functions of the null and alternative result in more power to detect the alternative.

9.3.2 Binary Strings

Let $\mathcal{T} ::= \{0,1\}^k$ be the set of all length k binary strings. Define the following distributions to be uniform over all strings $x = (x_1, \dots, x_k) \in \{0,1\}^k$ which satisfy the given predicates:

$$\mathbf{p}_{\text{ind}} : \text{ uniform on all strings} \tag{9.120}$$
$$\mathbf{p}_{\text{odd}} : \ \textstyle\sum_{i=1}^{k} x_i \equiv 1 \pmod 2 \tag{9.121}$$
$$\mathbf{p}_{\text{tie}} : \ x_1 = x_2 = \dots = x_{k/2}. \tag{9.122}$$

Each of these distributions assigns marginal probability 1/2 to each bit x_i (for $1 \leq i \leq k$), so all deviations from the uniform distribution $\mathbf{p}_{\text{ind}}$ are captured by higher-order relationships. The five orderings used for comparing binary strings are

$$\prec_{\text{lex}} : \text{ Lexicographic (dictionary) ordering} \tag{9.123}$$
$$\prec_{\text{par}} : \text{ Parity of ones, ties broken using } \prec_{\text{lex}} \tag{9.124}$$
$$\prec_{\text{one}} : \text{ Number of ones, ties broken using } \prec_{\text{lex}} \tag{9.125}$$
$$\prec_{\text{coo}} : \text{ Cooler ordering (randomly generated) [Stevens and Williams, 2012]} \tag{9.126}$$
$$\prec_{\text{dbj}} : \text{ De Bruijn sequence ordering.} \tag{9.127}$$

The null distributions and alternative distributions

$$\mathbf{p} ::= \mathbf{p}_{\text{ind}} \tag{9.128}$$
$$\mathbf{q} ::= w\mathbf{p}_{\text{c}} + (1-w)\mathbf{p}_{\text{ind}}, \tag{9.129}$$

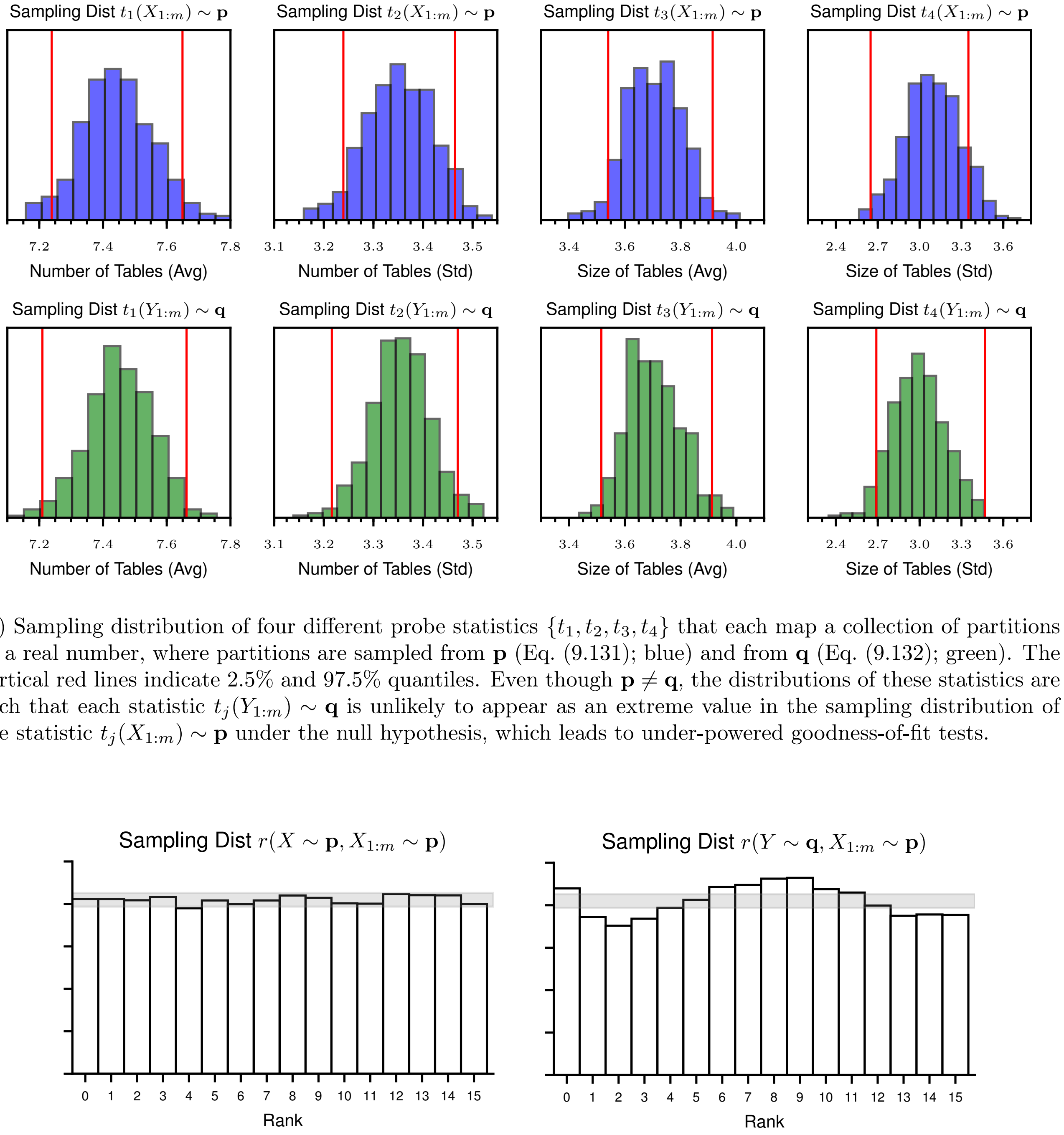
for $w \in [0,1]$ $c \in \{\text{odd}, \text{tie}\}$. That is, the alternative distributions are probabilistic mixtures of $\mathbf{p}_{\text{ind}}$ with the other two distributions. The bit string length $k = 16$ with $n = 256$ observations are used. Therefore, $|\mathcal{T}| = 65,536$ and the n observations comprise only 0.4% of the domain size.

Figure 9.3 shows how the nonuniformity of the SRS with $m = 6$ varies for each of the two alternatives and five orderings. Each ordering induces a different CDF over $\{0,1\}^k$ for the alternative distribution, shown in the right panel of Figure 9.3 for $w = 1$. Orderings with a greater maximum vertical distance between the null and alternative CDF attain greater rank nonuniformity. No single ordering is more powerful than all others in both test cases. However, in each case, some ordering detects the difference even at low weights w, despite the sparse observation set.

The alternative $\mathbf{q} = \mathbf{p}_{\text{odd}}$ in Figure 9.3b is especially challenging since in any string $x \in \{0,1\}^k$, all substrings (not necessarily contiguous) of a given length $1 \leq j < k$ are equally likely. Even though the SRS is nonuniform for all orderings, the powers vary significantly. For example, using the lexicographic ordering $\prec_{\text{lex}}$ does not effectively distinguish between $\mathbf{p}_{\text{ind}}$ and $\mathbf{p}_{\text{odd}}$, as strings with an odd number of ones are lexicographically evenly interspersed within the set of all strings. The parity ordering $\prec_{\text{par}}$, which is optimal for this alternative, and the randomly generated cooler ordering $\prec_{\text{coo}}$ both have increasing power with w.

9.3.3 Partition Testing

The final example applies the SRS to test distributions on the space of partitions of the set $\{1, 2, \dots, N\}$. Let Π_N denote the set of all such partitions. The two-parameter Chinese Restaurant Process (CRP) [Buntine and Hutter, 2010, Section 5.1] can be used to define a distribution on Π_N. Letting $(x|y)_N ::= (x)(x+y)\cdots(x+(N-1)y)$ denote the rising factorial, the probability of a partition $\pi ::= \{\pi_1, \dots, \pi_k\} \in$



(a) Sampling distribution of four different probe statistics $\{t_1, t_2, t_3, t_4\}$ that each map a collection of partitions to a real number, where partitions are sampled from $\mathbf{p}$ (Eq. (9.131); blue) and from $\mathbf{q}$ (Eq. (9.132); green). The vertical red lines indicate 2.5% and 97.5% quantiles. Even though $\mathbf{p} \neq \mathbf{q}$, the distributions of these statistics are such that each statistic $t_j(Y_{1:m}) \sim \mathbf{q}$ is unlikely to appear as an extreme value in the sampling distribution of the statistic $t_j(X_{1:m}) \sim \mathbf{p}$ under the null hypothesis, which leads to under-powered goodness-of-fit tests.

(b) Monte Carlo simulation of the sampling distribution of the SRS illustrates that it has a uniform distribution under the null hypothesis $\mathbf{p} = \mathbf{q}$ and a non-uniform distribution under the alternative hypothesis $\mathbf{p} \neq \mathbf{q}$, leading to a more powerful goodness-of-fit test.

Figure 9.4: Comparison of bootstrap and SRS goodness-of-fit tests for detecting distributional differences between two Chinese restaurant processes $\mathbf{q}$ and $\mathbf{q}$ on $N = 20$ customers. Panel (a) shows the sampling distribution of various bootstrapped probe statistics and panel (b) shows the sampling distribution the stochastic rank statistic.

Algorithm 9.2 Total order $\prec$ on the set Π_N of partitions over $\{1, \dots, N\}$.

```
Require: { Partition $\pi ::= \{\pi_1, \pi_2, \dots, \pi_k\} \in \Pi_N$ with $k$ blocks.
         { Partition $\nu ::= \{\nu_1, \nu_2, \dots, \nu_l\} \in \Pi_N$ with $l$ blocks.
Ensure: LT if $\pi \prec \nu$; GT if $\pi \succ \nu$; EQ if $\pi = \nu$.
if $k < l$ then
    return LT                                                    ▷ $\nu$ has more blocks
if $k > l$ then
    return GT                                                    ▷ $\pi$ has more blocks
$\tilde{\pi} \leftarrow$ blocks of $\pi$ sorted by value of least element in the block
$\tilde{\nu} \leftarrow$ blocks of $\nu$ sorted by value of least element in the block
for $b = 1, 2, \dots, l$ do
    if $|\tilde{\pi}_b| < |\tilde{\nu}_b|$ then
        return LT                                                ▷ $\tilde{\nu}_b$ has more elements
    if $|\tilde{\pi}_b| > |\tilde{\nu}_b|$ then
        return GT                                                ▷ $\tilde{\pi}_b$ has more elements
    $\pi'_b \leftarrow$ values in $\tilde{\pi}_b$ sorted in ascending order
    $\nu'_b \leftarrow$ values in $\tilde{\nu}_b$ sorted in ascending order
    for $i = 1, 2, \dots, |\pi'_b|$ do
        if $\pi'_{b,i} < \nu'_{b,i}$ then
            return LT                                            ▷ $\pi'_b$ has smallest element
        if $\pi'_{b,i} > \nu'_{b,i}$ then
            return GT                                            ▷ $\nu'_b$ has smallest element
return EQ
```

Π_N with k blocks (also referred to as tables) is

$$\mathsf{CRP}(\pi; a, b) ::= \begin{cases} \dfrac{(b|a)_k}{(b|1)_N} \displaystyle\prod_{i=1}^{k} (1-a)_{c_k - 1} & (\text{if } a > 0) \\ \dfrac{b^k}{(b|1)_N} \displaystyle\prod_{i=1}^{k} (c_k - 1)! & (\text{if } a = 0), \end{cases} \tag{9.130}$$

where c_i is the number of integers (customers) assigned to block (table) π_i $(i = 1, \dots, k)$. Simulating a CRP proceeds by sequentially assigning customers to tables [Buntine and Hutter, 2010, Definition 7].

Even though the probability of any partition can be directly computed, the cardinality of Π_N grows exponentially in N; for example, $|\Pi_{20}| \approx 5.17 \times 10^{13}$. The expected frequency of any partition is essentially zero for sample size $n \ll |\Pi_N|$, so Pearson chi-square or likelihood-ratio tests on the raw data are inappropriate. Algorithm 9.2 defines a procedure that induces a linear order on Π_N based on traversing the partitions and using various statistics to determine precedence.

Consider The following pair of distributions:

$$\mathbf{p} ::= \mathsf{CRP}(0.26, 0.76)/2 + \mathsf{CRP}(0.19, 5.1)/2 \tag{9.131}$$

$$\mathbf{q} ::= \mathsf{CRP}(0.52, 0.52). \tag{9.132}$$

These distributions are designed to ensure that partitions from $\mathbf{p}$ and $\mathbf{q}$ have similar distributions on the number and sizes of tables. Figure 9.4 compares the quality of goodness-of-fit tests using bootstrap resampling [Good, 2004] of various probe statistics and the SRS test with the ordering in Algorithm 9.2.

In Figure 9.4a, each probe statistic takes a size m dataset $(X_1, \dots, X_m)$, where each X_i is itself a

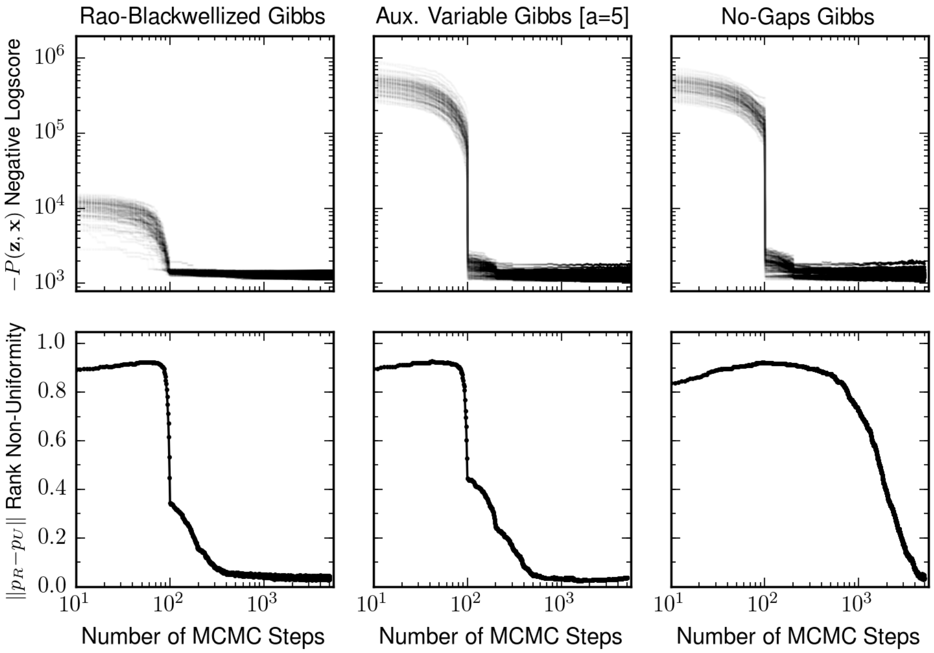


Figure 9.5: The uniformity of the SRS (bottom row) captures convergence behavior of MCMC sampling algorithms for Dirichlet process mixture models that are not captured by common diagnostics such as the logscore (top row), which is the joint probability density of latent variables and observed data..

partition over $[N]$, and produces a numerical summary such as the average of the number of tables in each sample. Bootstrap resampling with these probe statistics will report with high probability that an observed statistic $t(Y_1, \dots, Y_m) \sim \mathbf{q}$ drawn from the alternative distribution is a non-extreme value in the null distribution $t(X_1, \dots, X_m) \sim \mathbf{p}$, which is indicated by alignment of the quantiles (red lines) in Figure 9.4a. These resampling tests will therefore have insufficient evidence to reject $\mathbf{p} = \mathbf{q}$. In contrast, Figure 9.4b shows that when ranked using the linear ordering from Algorithm 9.2, which is based on a multivariate combination of the univariate probe statistics in Figure 9.4a, a partition $Y \sim \mathbf{q}$ is more likely to lie in the center of a dataset $(X_1, \dots, X_m) \sim^{\text{iid}} \mathbf{p}$. The gray band in Figure 9.4b shows 99% variation for a uniform histogram, confirming that the rank distribution under the alternative hypothesis is statistically nonuniform.

9.4 Applications to Convergence Analysis of Approximate Samplers

A key application of the SRS is assessing the sample quality of random data structures produced by approximate sampling algorithms over combinatorially large domains with intractable probabilities. Sections 9.4.1 and 9.4.2 use the SRS to diagnose the convergence of approximate sampling algorithms for Dirichlet process mixture models and Ising models, respectively. A third example can be found in Lin [2022, Section 5.1.2], who combine the SRS and simulation-based calibration as in Section 9.4.1 to assess the sample quality of structured expressions in the Gaussian process DSL (Chapter 2) obtained via sequential Monte Carlo learning (Section 3.3.2). The example in [Lin, 2022] is enabled by defining a linear ordering on the underlying space of primitive and composite Gaussian process covariance kernels in the DSL from Listing 2.1, analogously to the procedurally-specified ordering on integer partitions from Algorithm 9.2.

9.4.1 Dirichlet Process Mixture Models

Talts et al. [2018] describe simulation-based calibration (SBC), a procedure for validating samples from algorithms that generate posterior samples in Bayesian models. More specifically, for a prior $\pi(z)$ over latent variables z and likelihood function $\pi(x \mid z)$ over data x, integrating the posterior over the joint distribution returns the prior distribution:

$$\pi(z) = \int \pi(z, x')\mathrm{d}x' = \int \pi(z|x')\pi(x')\mathrm{d}x' = \int \left[\int \pi(z|x')\pi(x'|z')\mathrm{d}x'\right] \pi(z')\mathrm{d}z', \qquad (9.133)$$

where the final equality uses the fact that $\pi(x') = \int \pi(x'|z')\pi(z')\,\mathrm{d}z'$. Eq. (9.133) indicates for n i.i.d. datasets $x_1, \dots, x_n \sim \pi(x')$, the i.i.d. samples $\hat{z}_i \sim \pi(z|x_i)$ for $i = 1, \dots, n$ are equivalent to samples from the prior $\pi(z)$. An approximate sampler can be thus be diagnosed by performing a goodness-of-fit test to check whether $\hat{z}_1, \dots, \hat{z}_n$ are distributed according to $\pi(z)$. Talts et al. [2018] apply this idea to test the goodness-of-fit of one-dimensional marginals of a continuous parameter vector $z \in \mathbb{R}^d$. The SRS is used to extend SBC for testing discrete latent variables z that take values in a large domain.

A sample size of $n = 1000$ independent datasets $x_1, \dots, x_n$ were simulated from a Dirichlet process mixture model (Section 5.1.1). Each dataset x_i has $k = 100$ observations and each observation $x_i \in \mathbb{R}^{k\times 5}$ is five-dimensional with a Gaussian likelihood, so that the latent state $z \in \Pi_k$ with $|\Pi_k| \approx 10^{115}$, From Eq. (9.133), the samples $\hat{z}_1, \dots, \hat{z}_n$ of the mixture assignment vector should be distributed according to the CRP prior $\pi(z)$. The top row of Figure 9.5 shows trace plots of the logscore (unnormalized posterior) of approximate samples from Rao–Blackwellized Gibbs, Auxiliary Variable Gibbs, and No-Gaps Gibbs samplers, which correspond to Algorithms 3, 8, and 4 from Neal [2000], respectively. Each line corresponds to an independent run of MCMC inference. The bottom row shows the evolution of the uniformity of the SRS using $m = 64$ and the linear ordering on partitions from Algorithm 9.2.

The logscores stabilize after 100 MCMC steps, which is one epoch through all observations in a dataset, and suggest little difference across the three samplers. In contrast, the SRS shows that Rao-Blackwellized Gibbs is slightly more efficient than Auxiliary Variable Gibbs and that the sample quality from No-Gaps Gibbs is inferior to those from the other two algorithms, up until roughly $5,000$ steps. These results are consistent with the observation from Neal [2000] that No-Gaps has inefficient mixing, as it excessively rejects proposals on singleton clusters.

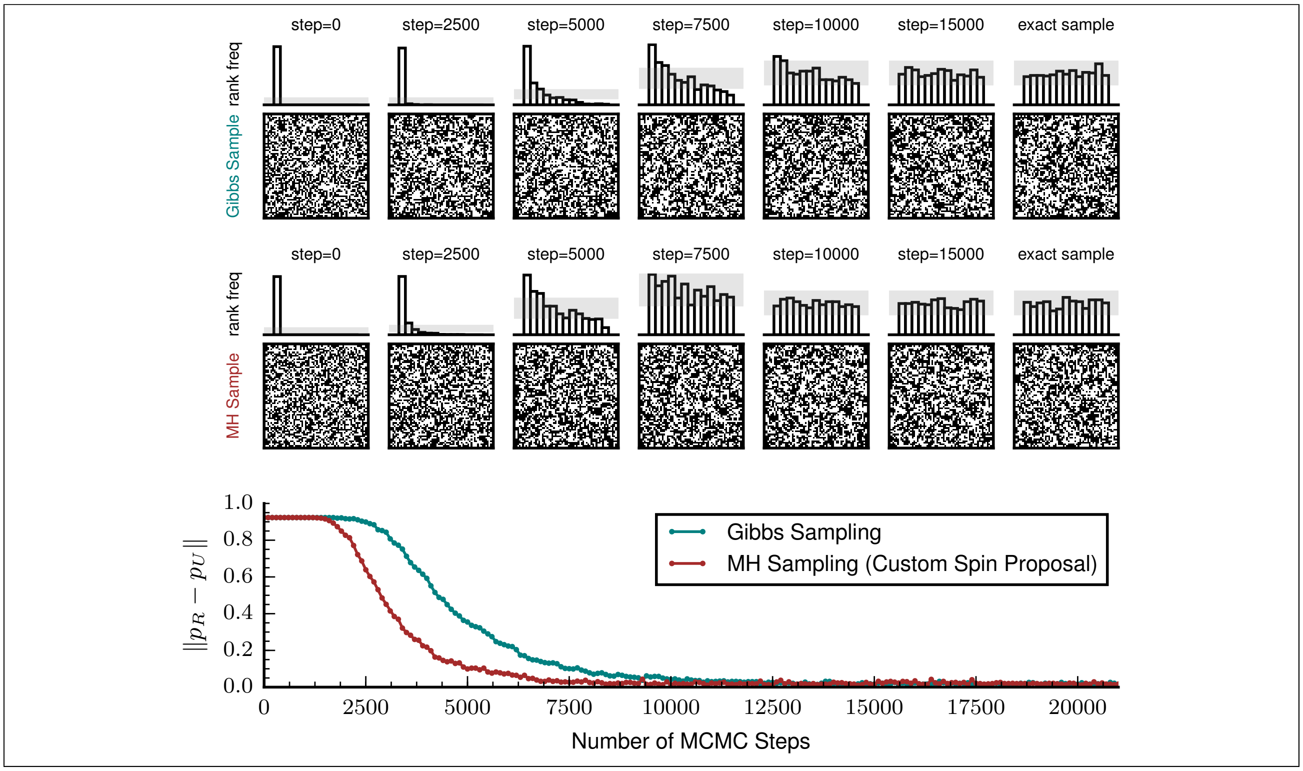
9.4.2 Ising Models

Generating simulations of Ising models is a fundamental problem in statistical mechanics [Hughes, 1999]. For a ferromagnetic $k \times k$ lattice with temperature T, the probability of a spin configuration x is

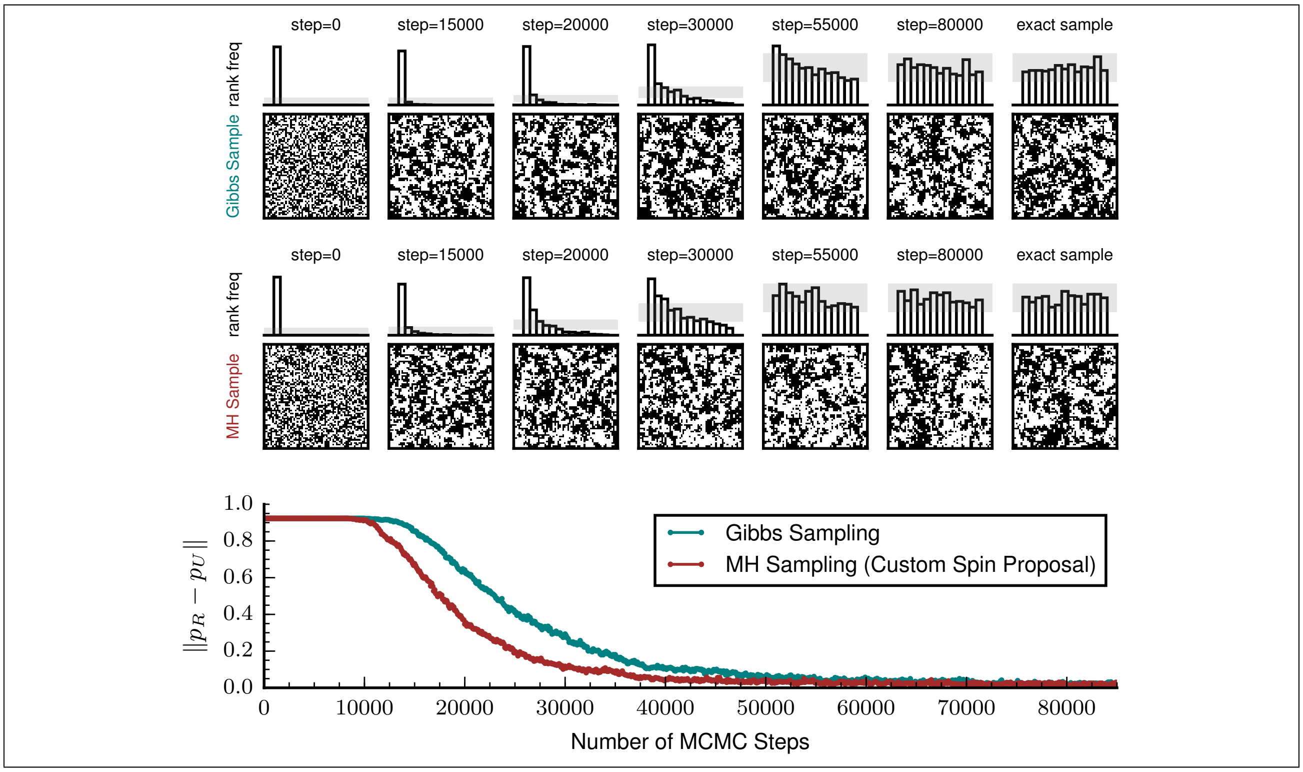
$$P(x) \propto \exp\left(-1/T \textstyle\sum_{i,j} x_i x_j\right) \qquad\qquad (x \in \{-1, +1\}^{k\times k}). \qquad (9.134)$$

While the normalizing constant of Eq. (9.134) is intractable to compute for any x, coupling-from-the-past [Propp and Wilson, 1996] is a popular MCMC technique which can tractably obtain exact samples from the Ising model. For a 64×64 Ising model (whose domain size is $2^{64\times 64}$), 650 exact samples were obtained via coupling-from-the-past and used as the "ground-truth" samples to assess the goodness-of-fit of approximate samples obtained via Gibbs sampling and Metropolis-Hastings (MH) sampling. The proposal for MH is based on the method described in MacKay [2003, Section 31.1].

For each temperature $T \in \{3, 8\}$, a total of 7,800 approximate samples were simulated from Gibbs and MH. The first two rows of Figure 9.6 each show the evolution of one particular sample (top: Gibbs; bottom: MH). Two exact samples are shown in the final column of each panel. All approximate and exact samples are independent of one another and were obtained by running parallel Markov chains (Figure 3.2). The SRS of the exact samples with respect to the approximate samples was taken at



(a) Temperature $T = 8$



(b) Temperature $T = 3$

Figure 9.6: Using the SRS to assess the goodness-of-fit of approximate samples of a 64×64 Ising model using Gibbs sampling and Metropolis–Hastings sampling.

checkpoints of 100 MCMC steps, using $m = 12$ and a linear ordering based on the Hamiltonian energy, spin magnetization, and connected components. The SRS histograms and 99% variation bands evolving at various steps of inference are shown above the Ising model renderings.

The SRS is nonuniform, including in regimes where the difference between approximate and exact samples is too fine-grained to be detected visually, at early steps and more uniform at higher steps. The plots show that MH is a more efficient sampler than Gibbs at moderate temperatures, as its sample quality improves more rapidly. This property was conjectured by MacKay [2003], who noted that the MH sampler "has roughly double the probability of accepting energetically unfavourable moves, so may be a more efficient sampler [than Gibbs]". In addition, the plots suggest that the samples become close to exact—with respect to the joint energy, magnetization, and connected components properties used to define the linear ordering—after 20,000 steps for $T = 8$ and 100,000 steps for $T = 3$. Exact sampling via coupling-from-the-past requires 500,000—1,000,000 MCMC steps for both temperatures.

9.5 Implementation as a Probabilistic Meta-Program in Gen

The SRS can be used to perform goodness-of-fit tests for arbitrary generative processes specified as probabilistic programs. Listing 9.1 shows an implementation of the SRS goodness-of-fit test (Algorithm 9.1) as a meta-program in the Gen probabilistic programming system [Cusumano-Towner et al., 2019]. The function `simulate_stochastic_rank_statistic` takes as input an arbitrary probabilistic program called `model`, whose type is a `GenerativeFunction`. The `targets` input is a selection of the trace addresses in the `model` probabilistic program whose marginal distribution is being tested. The `observations` input is a list of `ChoiceMap` objects, where each element is a map from trace addresses to observed values. The addresses in each observation are required to exactly match the selected addresses in `targets`. The `less_than` input is a function that defines the user-specified linear ordering on the `ChoiceMap` domain, which returns `true` if and only if the first argument is less than the second argument. The return value in line 32 is the list of realizations of the SRS. These ranks can then be assessed for uniformity in many ways [Steele and Chaseling, 2006]. One such approach is shown in line 35 which returns the p-value of the rank histogram under the Pearson chi-square test.

```
import Gen
import HypothesisTests

function simulate_stochastic_rank_statistic(
        model::Gen.GenerativeFunction,
        model_args::Tuple,
        targets::Gen.Selection,
        observations::Vector{T},
        m::Integer,
        less_than::Function,
        )
        where T <: Gen.ChoiceMap
    ranks = Vector{Integer}(undef, length(observations))
    for (i, observation) in enumerate(observations)
        U = rand(Gen.Distributions.Uniform(0,1))
        r = 0
        for j=1:m
            trace_p = Gen.simulate(model, model_args)
            choices = Gen.get_choices(trace_p)
            simulation = Gen.get_selected(choices, targets)
            if simulation == observation
                V = rand(Gen.Distributions.Uniform(0,1))
                if V < U
                    r += 1
                end
            elseif less_than(observation, simulation)
                r += 1
            end
        ranks[i] = r
        end
    end
    return ranks
end

function test_stochastic_rank_statistic_uniformity(m::Integer, ranks::Vector{Integer})
  histogram = [sum(ranks .== i) for i=0:m]
  return HypothesisTests.ChisqTest(histogram)
end
```

Listing 9.1: Gen implementation of SRS goodness-of-fit test (Algorithm 9.1) for probabilistic programs.

Part IV

Conclusion

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 10

Future Directions

We began this thesis with two motivating questions:

> *How can we build systems that automatically discover probabilistic models of noisy empirical data?*
> *How can we build systems that return principled answers to queries about probabilistic models?*

Part I precisely cast the problem of model discovery given noisy data as a hierarchical Bayesian inference over structured expressions in domain-specific data modeling languages; presented sequential Monte Carlo algorithms for online Bayesian structure learning; and applied the approach to automatically discover models in four domains: univariate time series, multivariate time series, cross-sectional data tables, and relational systems. Parts II and III showed how to solve challenging queries about probabilistic models by representing them as probabilistic programs and using symbolic analyses for exact Bayesian inference and dynamic analyses for statistical estimation and testing. We identified six scalability principles in Section 1.1 for engineering software systems that achieve these capabilities, and through a large collection of quantitative evaluations showed they deliver solutions that match or outperform widely used techniques across a spectrum of real-world data science problems. We close with a discussion of some limitations, open questions, and opportunities for future work that this research program invites.

10.1 Human-In-The-Loop Model Discovery

In many data modeling problems, practitioners have good intuitions about how to model parts of the data and might wish to apply automated model discovery to learn structure in the unknown parts. Examples include the presence of a holiday effect or changepoint in a certain time window (for the time series DSL in Chapter 2); constraints about which variables must be modeled as dependent (for the cross-sectional data DSL in Chapter 4); or known functional relationships that dictate direct causal dependencies between observations (for the relational data DSL in Chapter 6). The formalism of "Bayesian Synthesis" described in Chapter 3 does not easily enable such knowledge to be integrated, as it requires the user to manually redesign the DSL syntax, prior semantics, or likelihood semantics. New structure learning algorithms and high-level "meta-modeling" languages are needed to let practitioners more naturally specify constraints on the class of DSL expressions under consideration for a given problem. A related capability is building user interfaces that let practitioners inspect and refine the synthesized models, since the raw probabilistic programs in the learned ensemble, while individually interpretable, are often too low-level and numerous to be edited by hand. These capabilities could then be incorporated in a feedback system that informs practitioners whether their assumptions are supported by or highly implausible given the observed data; for example, by comparing the predictive quality of customized models to default models on functional metrics of interest, or comparing marginal likelihood estimates obtained as a byproduct of sequential Monte Carlo structure learning.

10.2 Extracting Causal Structure from Phenomenological Models

A fundamental challenge in using observational data to discover exact causal relationships between variables is that different causal systems may give rise to identical statistical associations [Pearl, 2009]. Despite this challenge, it is still possible to use automatically discovered probabilistic models to rule in or rule out certain causal relationships. For example, the three DSLs in Chapters 4–6 each include inductive biases to discover completely independent subsystems of variables that have neither statistical nor causal interactions (variables in the same subsystem, on the other hand, may be marginally independent while having a causal path such as a common effect that renders them conditionally dependent). If a pair of variables is inferred to be structurally independent across many DSL expressions in the synthesized ensemble (e.g., 95%), it is reasonable to conclude that there is unlikely to be a strong causal path between them. Solving conditional mutual information queries via dynamic analysis of probabilistic programs (Chapter 8) can also be used to test for more refined local causal structures, such as common causes, Markov chains, or common effects. One advantage of testing for causal relationships using mutual information queries about models from Bayesian synthesis is that they correspond to coherent posterior inferences with respect to a joint generative model over structure, parameters, observed data, and future data. There is thus no need to perform a large number of multiple testing or false discovery rate corrections as in frequentist estimation, although systematic power studies and coverage analyses will be needed to establish efficacy of these methods on challenging real-world datasets.

10.3 Is Sampling Superior to Optimization?

A central theme in our approach to probabilistic structure learning is jointly sampling from the posterior distribution over model structures and parameters to characterize uncertainty about all unknown quantities. This approach differs from greedy methods that aim to find a single structure or parameter setting by optimizing a posterior density or loss function. Is it really necessary to sample rather than optimize, and what are the practical advantages of doing so? Figure 2.6 shows an example where sampling structures and parameters handily outperforms optimization: the latter approach converges to poor solutions even when the correct structure has been specified. There are compelling theoretical results in the literature that substantiate the scaling benefits of sampling. Ma et al. [2019] disprove the folk wisdom that sampling is necessarily slower than optimization by showing that, for a class of non-convex objective functions that arise in mixture modeling and multistable systems, the computational complexity of sampling algorithms scales linearly with the model dimension whereas that of optimization algorithms scales exponentially. Mou et al. [2019] show that MCMC with polynomial-time mixing is possible in a restricted class of Bayesian Gaussian mixture models. Is it possible to obtain positive results for sampling complexity and convergence that go beyond inferring parameters in fixed-dimensional Euclidean spaces to structure learning in discrete spaces of probabilistic model structures?

10.4 Metalinguistic Abstractions for Building DSLs

As mentioned in Section 1.3, the DSLs presented in Part I are all manually designed. Given our present belief that it remains computationally infeasible to learn entire DSLs, especially for the type of online structure learning and real-time inference problems considered in this thesis, there is a pressing need for new abstractions that simplify the process of designing DSLs. A positive result from Chapter 3 is that, when the DSL is generated by a probabilistic context-free grammar (PCFG), sound Bayesian synthesis can be automated using default MCMC and SMC learning algorithms. That said, even though PCFGs are a relatively expressive metalanguage, they cannot express context-sensitive DSLs such as those in Chapters 4–6, which require custom synthesis approaches. This limitation motivates us to consider richer metalanguages such as macro grammars [Fischer, 1968] and attribute grammars [Knuth, 1968] for specifying DSLs using similar formal constructions to those for PCFGs in Section 3.4. Systematically developing DSLs using these metalanguages will improve our ability to verify properties such as prior normalization and design default synthesis algorithms, type checkers, and optimizations that exploit incremental computation. We take both inspiration and caution from deterministic program synthesis, where researchers have developed frameworks such as syntax-guided synthesis [Alur et al., 2013] and semantics-guided synthesis [Kim et al., 2021] for defining synthesis problems in an implementation agnostic way. While these unifying abstractions are theoretically appealing, extending them to probabilistic programs must aid, and not hinder, scalability of the systems implementation.

10.5 Probabilistic Programming Abstractions for Structure Learning

Implementing correct and scalable algorithms for Bayesian structure learning is challenging. It involves carefully deriving forward and reverse proposal probabilities, computing Jacobian corrections, or implementing resample-move SMC with hand-designed rejuvenation kernels. These challenges have impeded wider research progress in Bayesian structure learning. While probabilistic programming has been successfully used for Bayesian structure learning in a few settings, which include Gen implementations of the univariate time series DSL from Chapter 2 and the 3D scene perception system of Gothoskar et al. [2021], more mature tooling is needed for these systems to be as easily applicable as automatic differentiation frameworks, for example, which have enabled great advances for differentiable machine learning models. One promising direction is to build on the probabilistic programming-based methods described in Cusumano-Towner [2020], who introduce a family of "involutive MCMC" techniques that integrate structure learning and differentiable programming. Even though involutive MCMC is extremely flexible, the implementation complexity remains somewhat too high, even for advanced practitioners. Developing new abstractions that further specialize involutive MCMC to probabilistic DSLs as opposed to probabilistic programs in a general-purpose language will help researchers explore new applications of Bayesian structure learning in problem areas where the software tooling does not yet exist. Such frameworks might also enable new research in learning effective inference algorithms for probabilistic structure learning, toward the broader goal of synthesizing not only probabilistic model programs for a given dataset but also probabilistic inference programs for a given model program.

10.6 Theorem Proving and Verification for Probabilistic Programs

Probabilistic programs invite several opportunities to leverage formal methods for establishing the reliability, security, and safety of complex probabilistic modeling and inference workflows. For example, is it possible to mechanize the SPPL correctness proofs from Chapter 7 by expressing the formal syntax and semantics in a proof assistant such as Coq? Addressing this problem is likely to require novel extensions of interactive proof assistants that integrate real analysis and measure theoretic techniques for probabilistic reasoning. Further examples grounded in this thesis include:

1. Automating static analyses for the proofs of Proposition 3.53 to ensure that a PCFG defining a probabilistic DSL specifies a normalized prior semantics and that the associated likelihood semantics are bounded and nonnegative.

2. Designing type systems to ensure that the model and proposal probabilistic programs used in the EEVI estimators from Section 8.6 are not "undisciplined" and satisfy the absolute continuity requirements with respect to one another.

3. Using dynamic checks of irreflexivity, transitivity, and connectedness to ensure that the ordering in the SRS test from Section 9.5 is total on the observations and simulated data.

Extending probabilistic programming systems with built-in verification and testing will help practitioners more rapidly surface issues with the model or inference algorithms and help them iterate on solutions, toward developing the type of robust Bayesian workflows advocated by Gelman et al. [2020].

10.7 Broader Applications to the Social and Natural Sciences

For all the sophistication of modeling and inference at the frontiers of academic research, domain specialists within a field continue to leverage relatively simple data modeling techniques, for example:

- Gaussian graphical models in psychometrics [Borsboom et al., 2021];
- power laws in behavioral economics [Tversky and Kahneman, 1992];
- multiple regression in comparative social policy [Mabbett and Bolderson, 1999];
- multilevel regression in sociology [DiPrete and Forristal, 1994];
- hidden Markov models in bioinformatics [Yoon, 2009];
- logistic regression in clinical medicine [Bagley et al., 2001];
- GARCH models in computational finance [Francq and Zakoïan, 2019].

These approaches are widely used because they have interpretable semantics, perform reasonably well across many datasets, and have accessible and efficient software implementations. However, as data becomes more abundant and complex, traditional approaches that make rigid structural assumptions will inhibit our ability to discover novel scientific theories from data, which has been discussed at length by many influential researchers in the aforesaid fields [Fried and Cramer, 2017, Peterson et al., 2021, Shalev, 2007, Gelman, 2006, Bagley et al., 2001, Francq et al., 2011].

The systems for model discovery and probabilistic programming listed in Section 1.4 and described throughout this thesis are now mature enough to be used in close collaboration with domain specialists to help address limitations of conventional approaches. A representative example in the life sciences can be found in Bolton et al. [2019]. How can we enable hundreds or thousands of domain experts to more independently leverage these systems across more research areas? Achieving this vision will at minimum require the same level of usability and transparency that draws practitioners to their current modeling methods of choice. We also need to make strategic investments in release engineering, documentation, and teaching material to establish the credibility needed for sensitive applications in public policy or clinical medicine that have substantial implications for society's well-being.

We remain a long way from fully automating the process of building interpretable and accurate models given noisy empirical data, which humans perform by leveraging expert knowledge acquired over many years of individual experience and centuries of collective experience. Even further from our grasp are universal reasoning engines that can efficiently and soundly solve queries about arbitrarily rich probabilistic models. The techniques in this thesis constitute a design proposal for how to carefully scope these problems to ensure they are relatively tractable and automatable, along with systems implementations that deliver highly effective solutions in a range of structured data domains. Building on these ideas will continue to improve the quality of predictions and discoveries that machines can help us draw about complex phenomena we observe in the world around us.

Bibliography

Jayadev Acharya, Constantinos Daskalakis, and Gautam Kamath. Optimal testing for properties of distributions. In *Proceedings of the 29th Conference on Neural Information Processing Systems*, volume 28 of *Advances in Neural Information Processing Systems 28*, pages 3591–3599. Curran Associates, Inc., 2015. (Cited on page 185)

Ryan P. Adams, Hanna Wallach, and Zoubin Ghahramani. Learning the structure of deep sparse graphical models. In *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 1–8. PMLR, 2010. (Cited on page 67)

Amr Ahmed and Eric Xing. Dynamic non-parametric mixture models and the recurrent Chinese restaurant process: With applications to evolutionary clustering. In *Proceedings of the 2008 SIAM International Conference on Data Mining*, pages 219–230. SIAM, 2008. doi:10.1137/1.9781611972788.20. (Cited on page 97)

Mohammad Ahsanullah, Valery B. Nevzorov, and Mohammad Shakil. *An Introduction to Order Statistics*. Number 3 in Atlantis Studies in Probability and Statistics. Atlantis Press, Paris, 2013. doi:10.2991/978-94-91216-83-1. (Cited on page 187)

Omar Y. Al-Jarrah, Paul D. Yoo, Sami Muhaidat, George K. Karagiannidis, and Kamal Taha. Efficient machine learning for big data: A review. *Big Data Research*, 2(3):87–93, 2015. doi:10.1016/j.bdr.2015.04.001. (Cited on page 16)

Aws Albarghouthi, Loris D'Antoni, Samuel Drews, and Aditya V. Nori. FairSquare: Probabilistic verification of program fairness. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA): 80.1–80.30, 2017. doi:10.1145/3133904. (Cited on pages 119, 120, 136, 137, 138, and 142)

David J. Aldous. Exchangeability and related topics. In P. L. Hennequin, editor, *École d'Été de Probabilités de Saint-Flour XIII*, volume 1117 of *Lecture Notes in Mathematics*, pages 1–198. Springer, 1985. doi:10.1007/BFb0099421. (Cited on pages 82 and 101)

Alexander A. Alemi and Ian Fischer. GILBO: One metric to measure them all. In *Proceedings of the 32nd Conference on Neural Information Processing Systems*, volume 31 of *Advances in Neural Information Processing Systems*, pages 7037–7046. Curran Associates, Inc., 2018. (Cited on page 177)

Rajeev Alur, Rastislav Alur, Garvit Juniwal, Milo M. K. Juniwal, Mukund Raghothaman, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. Syntax-guided synthesis. In *Proceedings of the 13th International Conference on Formal Methods in Computer-Aided Design*, pages 1–8. IEEE Press, 2013. doi:10.1109/FMCAD.2013.6679385. (Cited on pages 67 and 215)

Mauricio Alvarez and Neil D. Lawrence. Sparse convolved Gaussian processes for multi-output regression. In *Proceedings of the 22nd Conference on Neural Information Processing Systems*, volume 21 of *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2008. (Cited on page 96)

Erling B. Andersen. A goodness of fit test for the Rasch model. *Psychometrika*, 38(1):123–140, 1973. doi:10.1007/BF02291180. (Cited on page 183)

Steen Andreassen, Roman Hovorka, Jonathan Benn, Kristian G. Olesen, and Ewart R. Carson. A model-based approach to insulin adjustment. In *Proceedings of the 3rd Conference on Artificial Intelligence in Medicine*, volume 44 of *Lecture Notes in Informatics*, pages 239–248. Springer, 1991. doi:10.1007/978-3-642-48650-0_19. (Cited on pages 174 and 176)

Christophe Andrieu and Gareth O. Roberts. The pseudo-marginal approach for efficient Monte Carlo computations. *The Annals of Statistics*, 37(2):697–725, 2009. doi:10.1214/07-AOS574. (Cited on pages 161 and 177)

Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342, 2010. doi:10.1111/j.1467-9868.2009.00736.x. (Cited on page 169)

Stefan Arnborg, Derek G Corneil, and Andrzej Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM Journal on Algebraic Discrete Methods*, 8(2):277–284, 1987. doi:10.1137/0608024. (Cited on page 135)

Taylor B. Arnold and John W. Emerson. Nonparametric goodness-of-fit tests for discrete null distributions. *The R Journal*, 3(2):34–39, 2011. doi:10.32614/RJ-2011-016. (Cited on page 185)

Steven C. Bagley, Halbart White, and Beatrice A. Golomb. Logistic regression in the medical literature: Standards for use and reporting, with particular attention to one medical domain. *Journal of Clinical Epidemiology*, 54(10):979–985, 2001. doi:10.1016/S0895-4356(01)00372-9. (Cited on page 216)

Matej Balog, Alexander L. Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. *arXiv*, 1611.01989, 2017. doi:10.48550/arXiv.1611.01989. (Cited on page 67)

David Barber. Expectation correction for smoothed inference in switching linear dynamical systems. *Journal of Machine Learning Research*, 7(89):2515–2540, 2006. (Cited on page 83)

David Barber and Felix Agakov. The IM algorithm: A variational approach to information maximization. In *Proceedings of the 17th Conference on Neural Information Processing Systems*, volume 16 of *Advances in Neural Information Processing Systems*, pages 201–208. MIT Press, 2003. (Cited on page 177)

Osbert Bastani, Xin Zhang, and Armando Solar-Lezama. Probabilistic verification of fairness properties via concentration. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):118.1–118.27, 2019. doi:10.1145/3360544. (Cited on pages 119, 120, 137, and 138)

Tuğkan Batu, Lance Fortnow, Ronitt Rubinfeld, Warren D. Smith, and Patrick White. Testing that distributions are close. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, pages 259–269. IEEE Press, 2000. doi:10.1109/SFCS.2000.892113. (Cited on page 185)

Atilim Güneş Baydin, Lei Shao, Wahid Bhimji, Lukas Heinrich, Lawrence Meadows, Jialin Liu, Andreas Munk, Saeid Naderiparizi, Bradley Gram-Hansen, Gilles Louppe, Mingfei Ma, Xiaohui Zhao, Philip Torr, Victor Lee, Kyle Cranmer, Prabhat, and Frank Wood. Etalumis: Bringing probabilistic programming to scientific simulators at scale. In *Proceedings of the International Conference for*

High Performance Computing, Networking, Storage and Analysis, pages 29.1–29.24. Association for Computing Machinery, 2019. doi:10.1145/3295500.3356180. (Cited on page 18)

Mohamed Ishmael Belghazi, Aristide Baratin, Sai Rajeshwar, Sherjil Ozair, Yoshua Bengio, Aaron Courville, and Devon Hjelm. Mutual information neural estimation. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 531–540. PMLR, 2018. (Cited on page 177)

Vaishak Belle, Andrea Passerini, and Guy Van den Broeck. Probabilistic inference in hybrid domains by weighted model integration. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 2770–2776. International Joint Conferences on Artificial Intelligence, 2015. (Cited on page 143)

José M. Bernardo and Adrian F. M. Smith. *Bayesian Theory.* Wiley Series in Probability & Statistics. John Wiley & Sons, New York, 1994. doi:10.1002/9780470316870. (Cited on page 84)

Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. Julia: A fresh approach to numerical computing. *SIAM Review*, 59(1):65–98, 2017. doi:10.1137/141000671. (Cited on page 42)

Sooraj Bhat, Johannes Borgström, Andrew D. Gordon, and Claudio Russo. Deriving probability density functions from probabilistic functional programs. In *Proceedings of the 19th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, volume 7795 of *Lecture Notes in Computer Science*, pages 508–522. Springer, 2013. doi:10.1007/978-3-642-36742-7_35. (Cited on pages 117 and 142)

Patrick Billingsley. *Probability and Measure.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, 3rd edition, 1995. (Cited on pages 55 and 128)

Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Obermeyer Fritz, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep universal probabilistic programming. *Journal of Machine Learning Research*, 20(28):1–6, 2019. (Cited on page 20)

David M. Blei and John D. Lafferty. Dynamic topic models. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 113–120. Association for Computing Machinery, 2006. doi:10.1145/1143844.1143859. (Cited on page 97)

David M. Blei, Thomas L. Griffiths, and Michael I. Jordan. The nested Chinese restaurant process and Bayesian nonparametric inference of topic hierarchies. *Journal of the ACM*, 57(2):7.1–7.30, 2010. doi:10.1145/1667053.1667056. (Cited on page 115)

Anrew D. Bolton, Martin Haesemeyer, Josua Jordi, Ulrich Schaechtle, Feras A. Saad, Vikash K. Mansinghka, Joshua B. Tenenbaum, and Florian Engert. Elements of a stochastic 3D prediction engine in larval zebrafish prey capture. *eLife*, 8:e51975, 2019. doi:10.7554/eLife.51975. (Cited on page 216)

Taylor L. Booth and Richard A. Thompson. Applying probability measures to abstract languages. *IEEE Transactions on Computers*, C-22(5):442–450, 1973. doi:10.1109/T-C.1973.223746. (Cited on page 56)

Marta Borowska. Entropy-based algorithms in the analysis of biomedical signals. *Studies in Logic, Grammar and Rhetoric*, 43(1):21–32, 2016. doi:10.1515/slgr-2015-0039. (Cited on page 161)

Denny Borsboom et al. Network analysis of multivariate data in psychological science. *Nature Reviews Methods Primers*, 1(58):1–18, 2021. doi:10.1038/s43586-021-00055-w. (Cited on page 216)

Creagh Briercliffe. Poisson process infinite relational model: A Bayesian nonparametric model for transactional data. Master's thesis, University of British Columbia, 2016. (Cited on page 114)

Tamara A. Broderick. *Clusters and Features from Combinatorial Stochastic Processes.* PhD thesis, University of California, Berkeley, 2014. (Cited on page 16)

Stephen P. Brooks and Andrew Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7(4):434–455, 1998. doi:10.1080/10618600.1998.10474787. (Cited on page 117)

Wray Buntine and Marcus Hutter. A Bayesian view of the Poisson–Dirichlet process. *arXiv*, 1007.0296, 2010. doi:arXiv.1007.0296. (Cited on pages 203 and 205)

Yuri Burda, Roger B. Grosse, and Ruslan Salakhutdinov. Importance weighted autoencoders. *arXiv*, 1509.00519, 2015. doi:10.48550/arXiv.1509.00519. (Cited on page 168)

Richard M. Burstall and John Darlington. A transformation system for developing recursive programs. *Journal of the ACM*, 24(1):44–67, 1977. doi:10.1145/321992.321996. (Cited on page 67)

Jacques Carette and Chung-chieh Shan. Simplifying probabilistic programs using computer algebra. In *Proceedings of the 18th International Symposium on Practical Aspects of Declarative Languages*, volume 9585 of *Lecture Notes in Computer Science*, pages 135–152. Springer, 2016. doi:10.1007/978-3-319-28228-2_9. (Cited on pages 117 and 142)

François Caron, Manuel Davy, Arnaud Doucet, Emmanuel Duflos, and Philippe Vanheeghe. Bayesian inference for linear dynamic models with Dirichlet process mixtures. *IEEE Transactions on Signal Processing*, 56(1):71–84, 2008. doi:10.1109/TSP.2007.900167. (Cited on page 97)

Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 76(1):1–32, 2017. doi:10.18637/jss.v076.i01. (Cited on pages 20, 66, and 117)

Peter J. Carrington, John Scott, and Stanley Wasserman, editors. *Models and Methods in Social Network Analysis.* Number 27 in Structural Analysis in the Social Sciences. Cambridge University Press, Cambridge, UK, 2005. doi:10.1017/CBO9780511811395. (Cited on page 99)

Carlos M. Carvalho, Michael S. Johannes, Hedibert F. Lopes, and Nicholas G. Polson. Particle learning and smoothing. *Statistical Science*, 25(1):88–106, 2010. doi:10.1214/10-STS325. (Cited on page 89)

Venkat Chandrasekeran, Nathan Srebro, and Prahladh Harsha. Complexity of inference in graphical models. In *Proceedings of the 24th Annual Conference on Uncertainty in Artificial Intelligence*, pages 70–78. AUAI Press, 2008. doi:10.48550/arXiv.1206.3240. (Cited on page 135)

Sarah Chasins and Phitchaya M. Phothilimthana. Data-driven synthesis of full probabilistic programs. In *Proceedings of the 29th International Conference on Computer Aided Verification*, volume 10426 of *Lecture Notes in Computer Science*, pages 279–304. Springer, 2017. doi:10.1007/978-3-319-63387-9_14. (Cited on pages 66 and 74)

Surajit Chaudhuri, Umeshwar Dayal, and Vivek Narasayya. An overview of business intelligence technology. *Communications of the ACM*, 54(8):88–98, 2011. doi:10.1145/1978542.1978562. (Cited on page 99)

Swarat Chaudhuri, Kevin Ellis, Oleksandr Polozov, Rishabh Singh, Armando Solar-Lezama, and Yisong Yue. Neurosymbolic programming. *Foundations and Trends in Programming Languages*, 7(3):158–243, 2021. doi:10.1561/2500000049. (Cited on page 67)

Mark Chavira and Adnan Darwiche. On probabilistic inference by weighted model counting. *Artificial Intelligence*, 172(6):772–799, 2008. doi:10.1016/j.artint.2007.11.002. (Cited on page 143)

Jie Cheng, Christos Hatzis, Hisashi Hayashi, Mark-A. Krogel, Shinichi Morishita, David Page, and Jun Sese. KDD Cup 2001 report. *SIGKDD Explorations Newsletter*, 3(2):47–64, 2002. doi:10.1145/507515.507523. (Cited on pages 100 and 112)

Chao-Lin Chiu. Entropy and probability concepts in hydraulics. *Journal of Hydraulic Engineering*, 113 (5):583–599, 1987. doi:10.1061/(ASCE)0733-9429(1987)113:5(583). (Cited on page 161)

Nicolas Chopin. Central limit theorem for sequential Monte Carlo methods and its application to Bayesian inference. *The Annals of Statistics*, 32(6):2385–2411, 2004. doi:10.1214/009053604000000698. (Cited on page 52)

Vartan Choulakian, Richard A. Lockhart, and Michael A. Stephens. Cramér–von Mises statistics for discrete distributions. *Canadian Journal of Statistics*, 22(1):125–137, 1994. doi:10.2307/3315828. (Cited on pages 185 and 186)

William J. Conover. A Kolmogorov goodness-of-fit test for discontinuous distributions. *Journal of the American Statistical Association*, 67(339):591–596, 1972. doi:10.1080/01621459.1972.10481254. (Cited on page 185)

Gregory F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2):393–405, 1990. doi:10.1016/0004-3702(90)90060-D. (Cited on page 135)

Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley Series in Telecommunications and Signal Processing. Wiley, Hoboken, 1991. (Cited on page 161)

Robert Cowell. Introduction to inference for bayesian networks. In Michael I. Jordan, editor, *Learning in Graphical Models*, Adaptive Computation and Machine Learning Series, pages 9–26. MIT Press, 1999. (Cited on page 15)

Stephen Cranefield and Ashish Dhiman. Identifying norms from observation using MCMC sampling. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 118–124. International Joint Conferences on Artificial Intelligence Organization, 2021. doi:10.24963/ijcai.2021/17. (Cited on page 65)

Marco Cusumano-Towner, Alexander K. Lew, and Vikash K. Mansinghka. Automating involutive MCMC using probabilistic and differentiable programming. *arXiv*, 2007.09871, 2020. doi:arXiv.2007.09871. (Cited on page 20)

Marco F. Cusumano-Towner. *Gen: A High-Level Programming Platform for Probabilistic Inference*. PhD thesis, Massachusetts Institute of Technology, 2020. (Cited on pages 20, 42, 177, 178, and 215)

Marco F. Cusumano-Towner and Vikash K. Mansinghka. AIDE: An algorithm for measuring the accuracy of probabilistic inference algorithms. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, volume 30 of *Advances in Neural Information Processing Systems*, pages 3004–3014. Curran Associates, Inc., 2017. (Cited on pages 169 and 177)

Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Design and Implementation*, pages 221–236. Association for Computing Machinery, 2019. doi:10.1145/3314221.3314642. (Cited on pages 20, 39, 42, 117, 130, 177, and 209)

Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993. doi:10.1016/0004-3702(93)90036-B. (Cited on page 117)

Rönän Daly, Qiang Shen, and Stuart Aitken. Learning Bayesian networks: Approaches and issues. *The Knowledge Engineering Review*, 26(2):99–157, 2011. doi:10.1017/S0269888910000251. (Cited on pages 69 and 114)

Adnan Darwiche. Tractable Boolean and arithmetic circuits. In Pascal Hitzler and Md Kamruzzaman Sarker, editors, *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, chapter 6, pages 146–172. IOS Press Ebooks, 2021. doi:10.3233/FAIA210353. (Cited on pages 69, 118, 142, and 143)

Adnan Darwiche and Pierre Marquis. A knowledge compilation map. *Journal of Artificial Intelligence Research*, 17:229–264, 2002. doi:10.1613/jair.989. (Cited on page 143)

John Dehardt. Generalizations of the Glivenko-Cantelli theorem. *The Annals of Mathematical Statistics*, 42(6):2050–2055, 1971. doi:10.1214/aoms/1177693073. (Cited on page 191)

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006. doi:10.1111/j.1467-9868.2006.00553.x. (Cited on pages 52, 167, and 169)

Pierre Del Moral, Ajay Jasra, Anthony Lee, Christopher Yau, and Xiaole Zhang. The alive particle filter and its use in particle Markov chain Monte Carlo. *Stochastic Analysis and Applications*, 33(6):943–974, 2015. doi:10.1080/07362994.2015.1060892. (Cited on page 50)

Dua Dheeru and Casey Graff. UCI Machine Learning Repository, 2017. URL http://archive.ics.uci.edu/ml. (Cited on page 74)

Persi Diaconis and Donald Ylvisaker. Conjugate priors for exponential families. *Annals of Statistics*, 7 (2):269–281, 1979. doi:10.1214/aos/1176344611. (Cited on page 130)

Thomas A. DiPrete and Jerry D. Forristal. Multilevel models: Methods and substance. *Annual Review of Sociology*, 20(1):331–357, 1994. doi:10.1146/annurev.so.20.080194.001555. (Cited on page 216)

Arnaud Doucet and Adam M. Johansen. Tutorial on particle filtering and smoothing: Fifteen years later. In Dan Crisan and Boris Rozovskii, editors, *The Oxford Handbook of Nonlinear Filtering*, chapter 24, pages 656–704. Oxford University Press, 2011. (Cited on page 169)

David B. Dunson, Natesh Pillai, and Ju-Hyun Park. Bayesian density regression. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(2):163–183, 2007. doi:10.1111/j.1467-9868.2007.00582.x. (Cited on pages 84 and 89)

David Duvenaud, James Lloyd, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Structure discovery in nonparametric regression through compositional kernel search. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1166–1174. PMLR, 2013. (Cited on pages 67 and 68)

David K. Duvenaud. *Automatic Model Construction with Gaussian Processes.* PhD thesis, University of Cambridge, 2014. (Cited on page 29)

Sašo Džeroski and Nada Lavrač, editors. *Relational Data Mining.* Springer, Berlin, 2001. doi:10.1007/978-3-662-04599-2. (Cited on page 99)

Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 214–226. Association for Computing Machinery, 2012. doi:10.1145/2090236.2090255. (Cited on page 137)

Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. Unsupervised learning by program synthesis. In *Proceedings of the 29th Conference on Neural Information Processing Systems*, volume 28 of *Advances in Neural Information Processing Systems*, pages 973–981. Curran Associates, Inc., 2015. (Cited on page 65)

Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. Sampling for Bayesian program learning. In *Proceedings of the 30th Conference on Neural Information Processing Systems*, volume 29 of *Advances in Neural Information Processing Systems*, pages 1297–1305. Curran Associates, Inc., 2016. (Cited on pages 66 and 67)

Michael D. Escobar and Mike West. Bayesian density estimation and inference using mixtures. *Journal of the American Statistical Association*, 90(430):577–588, 1995. doi:10.1080/01621459.1995.10476550. (Cited on page 72)

Mordecai Ezekiel. *Methods of Correlation Analysis.* John Wiley & Sons, Inc., New York, 1930. (Cited on page 74)

Xuhui Fan, Bin Li, and Scott A. Sisson. The binary space partitioning-tree process. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1859–1867. PMLR, 2018. (Cited on page 99)

Xuhui Fan, Bin Li, Caoyuan Li, Scott Sisson, and Ling Chen. Scalable deep generative relational model with high-order node dependence. In *Proceedings of the 33rd Conference in Neural Information Processing Systems*, volume 32 of *Advances in Neural Information Processing Systems*, pages 12658–12668. Curran Associates, Inc., 2019. (Cited on page 114)

Xuhui Fan, Bin Li, Ling Luo, and Scott A. Sisson. Bayesian nonparametric space partitions: A survey. In *Proceedings of the 30th International Joint Conference on Artificial Intelligence*, pages 4408–4415, 2021. doi:10.24963/ijcai.2021/602. (Cited on page 99)

Julian J. Faraway and Nicole H. Augustin. When small data beats big data. *Statistics & Probability Letters*, 136(C):142–145, 2018. doi:10.1016/j.spl.2018.02.031. (Cited on page 16)

John K. Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 229–239. Association for Computing Machinery, 2015. doi:10.1145/2737924.2737977. (Cited on page 67)

Daan Fierens, Guy Van den Broeck, Ingo Thon, Bernd Gutmann, and Luc De Raedt. Inference in probabilistic logic programs using weighted CNF's. In *Proceedings of the 27th Annual Conference on Uncertainty Artificial Intelligence*, pages 211–220. AUAI Press, 2011. doi:10.48550/arXiv.1202.3719. (Cited on page 143)

Michael J. Fischer. Grammars with macro-like productions. In *Proceedings of the 9th Annual Symposium on Switching and Automata Theory*, pages 131–142. IEEE Computer Society, October 1968. doi:10.1109/SWAT.1968.12. (Cited on page 215)

Adam Foster, Martin Jankowiak, Elias Bingham, Paul Horsfall, Yee Whye Teh, Thomas Rainforth, and Noah Goodman. Variational Bayesian optimal experimental design. In *Proceedings of the 33rd Conference in Neural Information Processing Systems*, volume 32 of *Advances in Neural Information Processing Systems*, pages 14059–14070. Curran Associates, Inc., 2019. (Cited on page 177)

Emily Fox, Erik B. Sudderth, Michael I. Jordan, and Alan S. Willsky. Nonparametric Bayesian learning of switching linear dynamical systems. In *Proceedings of the 22nd Conference on Neural Information Processing Systems*, volume 21 of *Advances in Neural Information Processing Systems*, pages 457–464. Curran Associates, Inc., 2008. (Cited on page 97)

Christian Francq and Jean-Michel Zakoïan. *GARCH Models: Structure, Statistical Inference and Financial Applications*. John Wiley & Sons, Hoboken, 2019. doi:10.1002/9781119313472. (Cited on page 216)

Christian Francq, Lajos Horváth, and Jean-Michel Zakoïan. Merits and drawbacks of variance targeting in GARCH models. *Journal of Financial Econometrics*, 9(4):619–656, 2011. doi:10.1093/jjfinec/nbr004. (Cited on page 216)

David H. Fremlin. *Measure Theory: Volume 2, Broad Foundations*. Torres Fremlin, Colchester, 2009. (Cited on page 44)

Eiko I. Fried and Angélique O. J. Cramer. Moving forward: Challenges and directions for psychopathological network theory and methodology. *Perspectives on Psychological Science*, 12(6):999–1020, 2017. doi:10.1177/1745691617705892. (Cited on page 216)

Nir Friedman and Daphne Koller. Being Bayesian about network structure. a Bayesian approach to structure discovery in Bayesian networks. *Machine Learning*, 50(1):95–125, 2003. doi:10.1023/A:1020249912095. (Cited on pages 67 and 69)

Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1300–1307. International Joint Conferences on Artificial Intelligence Organization, 1999. (Cited on page 114)

Ben D. Fulcher and Nick S. Jones. Highly comparative feature-based time-series classification. *IEEE Transactions on Knowledge and Data Engineering*, 26(12):3026–3037, 2014. doi:10.1109/TKDE.2014.2316504. (Cited on page 81)

Gapminder Foundation. Gapminder, 2022. URL https://gapminder.org. (Cited on pages 81 and 93)

Roland Gecse and Attila Kovács. Consistency of stochastic context-free grammars. *Mathematical and Computer Modelling*, 52(3):490–500, 2010. doi:10.1016/j.mcm.2010.03.046. (Cited on pages 56, 62, and 63)

Timon Gehr, Sasa Misailovic, and Martin Vechev. PSI: Exact symbolic inference for probabilistic programs. In *Proceedings of the 28th International Conference on Computer Aided Verification*, volume 9779 of *Lecture Notes in Computer Science*, pages 62–83. Springer, 2016. doi:10.1007/978-3-319-41528-4_4. (Cited on pages 117, 119, 120, 137, 138, 139, 140, 142, and 178)

Timon Gehr, Sasa Misailovic, Petar Tsankov, Laurent Vanbever, Pascal Wiesmann, and Martin Vechev. Bayonet: Probabilistic inference for networks. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 586–602. Association for Computing Machinery, 2018. doi:10.1145/3192366.3192400. (Cited on page 18)

Timon Gehr, Samuel Steffen, and Martin Vechev. λPSI: Exact inference for higher-order probabilistic programs. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 883–897. Association for Computing Machinery, 2020. doi:10.1145/3385412.3386006. (Cited on pages 142 and 178)

Jaco Geldenhuys, Matthew B. Dwyer, and Willem Visser. Probabilistic symbolic execution. In *Proceedings of the 2012 International Symposium on Software Testing and Analysis*, pages 166–176. Association for Computing Machinery, 2012. doi:10.1145/2338965.2336773. (Cited on page 142)

Andrew Gelman. Multilevel (hierarchical) modeling: What it can and cannot do. *Technometrics*, 48 (3):432–435, 2006. doi:10.1198/004017005000000661. (Cited on page 216)

Andrew Gelman, Xiao-Li Meng, and Hal Stern. Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sinica*, 6(4):733–807, 1996. (Cited on page 183)

Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis*. Chapman & Hall/CRC Texts in Statistical Science. CRC Press, Boca Raton, 3rd edition, 2014. doi:10.1201/9780429258411. (Cited on page 18)

Andrew Gelman, Aki Vehtari, Daniel Simpson, Charles C. Margossian, Bob Carpenter, Yuling Yao, Lauren Kennedy, Jonah Gabry, Paul-Christian Bürkner, and Martin Modrák. Bayesian workflow. *arXiv*, 2011.01808, 2020. doi:10.48550/arxiv.2011.01808. (Cited on page 216)

Robert Gens and Pedro Domingos. Learning the structure of sum-product networks. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 873–880. PMLR, 2013. (Cited on pages 69, 108, and 158)

Lise Getoor, Nir Friedman, Daphne Koller, Avi Pfeffer, and Ben Pfeffer. Probabilistic relational models. In Lise Getoor and Ben Taskar, editors, *Introduction to Statistical Relational Learning*, Adaptive Computation and Machine Learning Series. MIT Press, 2007. (Cited on page 114)

Zoubin Ghahramani. Probabilistic machine learning and artificial intelligence. *Nature*, 521(7553):452–459, 2015. doi:10.1038/nature14541. (Cited on page 18)

Walter R. Gilks and Carlo Berzuini. Following a moving target–Monte Carlo inference for dynamic Bayesian models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 63(1): 127–146, 2001. doi:10.1111/1467-9868.00280. (Cited on pages 49 and 52)

Walter R. Gilks, Andrew Thomas, and David J. Spiegelhalter. A language and program for complex Bayesian modelling. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 43(1): 169–177, 1994. doi:10.2307/2348941. (Cited on pages 20 and 117)

Geof H. Givens, Adrian E. Raftery, and Judith E. Zeh. A reweighting approach for sensitivity analysis within the Bayesian synthesis framework for population assessment modeling. *Report of the International Whaling Commission*, 44:377–384, 1994. (Cited on page 64)

Sharad Goel, Jake M. Hofman, Sébastien Lahaie, David M. Pennock, and Duncan J. Watts. Predicting consumer behavior with Web search. *Proceedings of the National Academy of Sciences*, 107(41): 17486–17490, 2010. doi:10.1073/pnas.1005962107. (Cited on page 17)

Ziv Goldfeld, Kristjan Greenewald, Jonathan Niles-Weed, and Yury Polyanskiy. Convergence of smoothed empirical measures with applications to entropy estimation. *IEEE Transactions on Information Theory*, 66(7):4368–4391, 2020. doi:10.1109/TIT.2020.2975480. (Cited on page 177)

Phillip I. Good. *Permutation, Parametric, and Bootstrap Tests of Hypotheses*. Springer Series in Statistics. Springer, New York, 3rd edition, 2004. doi:10.1007/b138696. (Cited on pages 186 and 205)

Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *Proceedings of the 24th Annual Conference on Uncertainty in Artificial Intelligence*, pages 220–229. AUAI Press, 2008a. (Cited on pages 65 and 117)

Noah D. Goodman, Joshua B. Tenenbaum, Jacob Feldman, and Thomas L. Griffiths. A rational analysis of rule-based concept learning. *Cognitive Science*, 32(1):108–154, 2008b. doi:10.1080/03640210701802071. (Cited on page 65)

Nishad Gothoskar, Marco Cusumano-Towner, Ben Zinberg, Matin Ghavamizadeh, Falk Pollok, Austin Garrett, Joshua B. Tenenbaum, Dan Gutfreund, and Vikash K. Mansinghka. 3DP3: 3D scene perception via probabilistic programming. In *Proceedings of the 35th Conference in Neural Information Processing Systems*, volume 34 of *Advances in Neural Information Processing Systems*, pages 9600–9612. Curran Associates, Inc., 2021. (Cited on pages 18 and 215)

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing machines. *arXiv*, 1410.5401, 2014. doi:10.48550/arXiv.1410.5401. (Cited on page 67)

Edwin J. Green, David W. MacFarlane, and Harry T. Valentine. Bayesian synthesis for quantifying uncertainty in predictions from process models. *Tree Physiology*, 20(5-6):415–419, 2000. doi:10.1093/treephys/20.5-6.415. (Cited on page 64)

Peter J. Green. Reversible jump Markov chain Monte Carlo computation and Bayesian model determination. *Biometrika*, 82(4):711–732, 1995. (Cited on page 90)

Arthur Gretton, Karsten M. Borgwardt, Malte J. Rasch, Bernhard Schölkopf, and Alexander Smola. A kernel two-sample test. *Journal of Machine Learning Research*, 13(25):723–773, 2012. (Cited on page 186)

Thomas L. Griffiths, Charles Kemp, and Joshua B. Tenenbaum. Bayesian models of cognition. In Ron Sun, editor, *The Cambridge Handbook of Computational Psychology*, Cambridge Handbooks in Psychology, pages 59–100. Cambridge University Press, 2008. doi:10.1017/CBO9780511816772.006. (Cited on page 18)

Roger Grosse, Ruslan Salakhutdinov, William Freeman, and Joshua B. Tenenbaum. Exploiting compositionality to explore a large space of model structures. In *Proceedings of the 28th Annual Conference on Uncertainty in Artificial Intelligence*, pages 306–315. AUAI Press, 2012. (Cited on page 67)

Roger B. Grosse, Zoubin Ghahramani, and Ryan P. Adams. Sandwiching the marginal likelihood using bidirectional Monte Carlo. *arXiv*, 1511.02543, 2015. doi:10.48550/arXiv.1511.02543. (Cited on page 177)

Roger B. Grosse, Siddharth Ancha, and Daniel M. Roy. Measuring the reliability of MCMC inference with bidirectional Monte Carlo. In *Proceedings of the 30th Conference on Neural Information Processing Systems*, volume 29 of *Advances in Neural Information Processing Systems*, pages 2459–2467. Curran Associates, Inc., 2016. (Cited on page 177)

Lutz F. Gruber and Mike West. Bayesian online variable selection and scalable multivariate volatility forecasting in simultaneous graphical dynamic linear models. *Econometrics and Statistics*, 3:3–22, 2017. doi:10.1016/j.ecosta.2017.03.003. (Cited on page 81)

Roger Guimerá, Ignasi Reichardt, Antoni Aguilar-Mogas, Francesco A. Massucci, Manuel Miranda, Jordi Pallarés, and Marta Sales-Pardo. A Bayesian machine scientist to aid in the solution of challenging scientific problems. *Science Advances*, 6(5):eaav6971, 2020. doi:10.1126/sciadv.aav6971. (Cited on page 66)

Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 317–330. Association for Computing Machinery, 2011. doi:10.1145/1926385.1926423. (Cited on page 67)

Sumit Gulwani, Susmit Jha, Ashish Tiwari, and Ramarathnam Venkatesan. Synthesis of loop-free programs. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 62–73. Association for Computing Machinery, 2011. doi:10.1145/1993498.1993506. (Cited on page 67)

Sumit Gulwani, Oleksandr Polozov, and Rishabh Singh. Program synthesis. *Foundations and Trends in Programming Languages*, 4(1–2):1–119, 2017. doi:10.1561/2500000010. (Cited on page 15)

Te Sun Han. Nonnegative entropy measures of multivariate symmetric correlations. *Information and Control*, 36(2):133–156, 1978. doi:10.1016/S0019-9958(78)90275-9. (Cited on page 164)

David J. Hand. Classifier technology and the illusion of progress. *Statistical Science*, 21(1):1–14, 2006. doi:10.1214/088342306000000060. (Cited on page 17)

Lauren A. Hannah, David M. Blei, and Warren B. Powell. Dirichlet process mixtures of generalized linear models. *Journal of Machine Learning Research*, 12(54):1923–1953, 2011. (Cited on page 115)

David Heckerman. A tutorial on learning with Bayesian networks. In Michael I. Jordan, editor, *Learning in Graphical Models*, Adaptive Computation and Machine Learning Series, pages 301–354. MIT Press, 1999. (Cited on page 15)

David Heckerman, Christopher Meek, and Daphne Koller. Probabilistic models for relational data. Technical Report MSR-TR-2004-30, Microsoft Research, 2004. (Cited on page 114)

Steven S. Henley, Richard M. Golden, and T. Michael Kashner. Statistical modeling methods: Challenges and strategies. *Biostatistics & Epidemiology*, 4(1):105–139, 2020. doi:10.1080/24709360.2019.1618653. (Cited on page 20)

James G. Hershberg. The United States, Brazil, and the Cuban missile crisis, 1962 (Part 1). *Journal of Cold War Studies*, 6(2):3–20, 2004. doi:10.1162/152039704773254740. (Cited on page 112)

Geoffrey E. Hinton, Peter Dayan, and Radford M. Frey, Brendan J. Neal. The "wake-sleep" algorithm for unsupervised neural networks. *Science*, 268(5214):1158–1161, 1995. doi:10.1126/science.7761831. (Cited on page 167)

Wassily Hoeffding. Asymptotically optimal tests for multinomial distributions. *The Annals of Mathematical Statistics*, 36(2):369–401, 1965. doi:10.1214/aoms/1177700150. (Cited on page 185)

Steven Holtzen, Guy Van den Broeck, and Todd Millstein. Scaling exact inference for discrete probabilistic programs. *Proc. ACM Program. Lang.*, 4(OOPSLA):140.1–140.31, 2020. doi:10.1145/3133904. (Cited on page 143)

James Honaker, Gary King, and Matthew Blackwell. Amelia II: A program for missing data. *Journal of Statistical Software*, 45(7):1–47, 2011. doi:10.18637/jss.v045.i07. (Cited on pages 81 and 94)

Susan Dadakis Horn. Goodness-of-fit tests for discrete data: A review and an application to a health impairment scale. *Biometrics*, 33(1):237–247, 1977. doi:10.2307/2529319. (Cited on page 185)

R. I. G. Hughes. The Ising model, computer simulation, and universal physics. In Mary S. Morgan and Margaret Morrison, editors, *Models as Mediators: Perspectives on Natural and Social Science*, number 52 in Ideas in Context, chapter 5, pages 97–145. Cambridge University Press, 1999. doi:10.1017/CBO9780511660108.006. (Cited on page 207)

Irvin Hwang, Andreas Stuhlmüller, and Noah D. Goodman. Inducing probabilistic programs by Bayesian program merging. *arXiv*, 1110.5667, 2011. doi:10.48550/arXiv.1110.5667. (Cited on page 65)

Rob J. Hyndman and Yeasmin Khandakar. Automatic time series forecasting: The forecast package for R. *Journal of Statistical Software*, 27(3):1–22, 2008. doi:10.18637/jss.v027.i03. (Cited on pages 40 and 96)

Katsuhiko Ishiguro, Naonori Ueda, and Hiroshi Sawada. Subset infinite relational models. In *Proceedings of the 15th International Conference on Artificial Intelligence and Statistics*, volume 22 of *Proceedings of Machine Learning Research*, pages 547–555. PMLR, 2012. (Cited on page 114)

Hemant Ishwaran and Lancelot F. James. Generalized weighted Chinese restaurant processes for species sampling mixture models. *Statistica Sinica*, 13(4):1211–1235, 2003. doi:10.2307/24307169. (Cited on page 96)

Ali Jahanian, Xavier Puig, Yonglong Tian, and Phillip Isola. Generative models as a data source for multiview representation learning. *arXiv*, 2106.05258, 2021. doi:10.48550/arxiv.2106.05258. (Cited on page 77)

Sonia Jain and Radford M. Neal. A split-merge Markov chain Monte Carlo procedure for the Dirichlet process mixture model. *Journal of Computational and Graphical Statistics*, 13(1):158–182, 2004. doi:10.1198/1061860043001. (Cited on page 106)

Frederick Jelinek, John D. Lafferty, and Robert L. Mercer. Basic methods of probabilistic context free grammars. In Pietro Laface and Renato De Mori, editors, *Speech Recognition and Understanding*, volume 75 of *NATO ASI Series, Sub-Series F: Computer and Systems Sciences*, pages 345–360. Springer-Verlag, 1992. doi:10.1007/978-3-642-76626-8_35. (Cited on page 53)

Mark Johnson, Thomas L. Griffiths, and Sharon Goldwater. Adaptor grammars: A framework for specifying compositional nonparametric Bayesian models. In *Proceedings of the 20th Conference on Neural Information Processing Systems*, volume 19 of *Advances in Neural Information Processing Systems*, pages 641–648. MIT Press, 2006. (Cited on page 66)

Matthew J. Johnson and Alan S. Willsky. Bayesian nonparametric hidden semi-Markov models. *Journal of Machine Learning Research*, 14:673–701, 2013. (Cited on pages 40, 96, and 97)

Philip N. Johnson-Laird. Deductive reasoning. *Annual Review of Psychology*, 50:109–135, 1999. doi:10.1146/annurev.psych.50.1.109. (Cited on page 15)

Eric Jonas and Konrad Kording. Automatic discovery of cell types and microcircuitry from neural connectomics. *eLife*, 4:e04250, 2015. doi:10.7554/eLife.04250. (Cited on page 114)

Charles Kemp, Joshua B. Tenenbaum, Thomas L. Griffiths, Takeshi Yamada, and Naonori Ueda. Learning systems of concepts with an infinite relational model. In *Proceedings of the 21st National Conference on Artificial Intelligence*, pages 381–388. AAAI Press, 2006. (Cited on pages 67, 99, 101, 102, 104, and 108)

Dae I. Kim, Prem Gopalan, David M. Blei, and Erik B. Sudderth. Efficient online inference for Bayesian nonparametric relational models. In *Proceedings of the 27th Conference on Neural Information Processing Systems*, volume 26 of *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2013. (Cited on page 99)

Jinwoo Kim, Qinheping Hu, Loris D'Antoni, and Thomas Reps. Semantics-guided synthesis. *Proceedings of the ACM on Programming Languages*, 5(POPL):30.1–30.32, 2021. doi:10.1145/3434311. (Cited on pages 67 and 215)

Chuck Kincaid. Guidelines for selecting the covariance structure in mixed model analysis. In *Proceedings of SAS Users Group International 30*, pages 198–30:1–8. SAS Institute Inc., 2005. (Cited on page 29)

Diederik P. Kingma and Max Welling. Auto-encoding variational Bayes. *arXiv*, 1312.6114, 2013. doi:10.48550/arXiv.1312.6114. (Cited on page 167)

Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019. doi:10.1561/2200000056. (Cited on page 161)

Thomas N. Kipf and Max Welling. Variational graph auto-encoders. *arXiv*, 1611.07308, 2016. doi:10.48550/arXiv.1611.07308. (Cited on page 114)

Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968. doi:10.1007/BF01692511. (Cited on page 215)

Stanley Kok and Pedro Domingos. Learning the structure of Markov logic networks. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 441–448. Association for Computing Machinery, 2005. doi:10.1145/1102351.1102407. (Cited on page 114)

Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, 2009. (Cited on pages 135 and 141)

Daphne Koller and Avi Pfeffer. Object-oriented Bayesian networks. In *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 302–313. AUAI Press, 1997. doi:10.48550/arXiv.1302.1554. (Cited on page 114)

Gary M. Koop. Forecasting with medium and large Bayesian VARS. *Journal of Applied Econometrics*, 28(2):177–203, 2013. doi:10.1002/jae.1270. (Cited on page 81)

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, 1992. (Cited on page 67)

John R. Koza, Forest H. Bennett III, Andre David, Martin A. Keane, and Frank Dunlap. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, 1(2):109–128, 1997. doi:10.1109/4235.687879. (Cited on page 67)

Lyudmyla F. Kozachenko and Nikolai N. Leonenko. A statistical estimate for the entropy of a random vector. *Problemy Peredachi Informatsii*, 23(2):9–16, 1987. (Cited on page 177)

Alexander Kraskov, Harald Stögbauer, and Peter Grassberger. Estimating mutual information. *Physical Review E*, 69(6):066138, 2004. doi:10.1103/PhysRevE.69.066138. (Cited on pages 172, 174, and 177)

Tejas D. Kulkarni, Pushmeet Kohli, Joshua B. Tenenbaum, and Vikash Mansinghka. Picture: A probabilistic programming language for scene perception. In *Proceedings of the 2015 IEEE Conference on Computer Vision and Pattern Recognition*, pages 4390–4399. IEEE Press, 2015. doi:10.1109/CVPR.2015.7299068. (Cited on page 18)

Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015. doi:10.1126/science.aab3050. (Cited on page 66)

Brenden M. Lake, Tomer D. Ullman, Joshua B. Tenenbaum, and Samuel J. Gershman. Building machines that learn and think like people. *Behavioral and Brain Sciences*, 40(e253):1–72, 2017. doi:10.1017/S0140525X16001837. (Cited on page 18)

Jürgen Landes. Bayesian epistemology. In Duncan Pritchard, editor, *Oxford Bibliographies in Philosophy*. Oxford University Press, 2021. doi:10.1093/OBO/9780195396577-0417. (Cited on page 18)

Jacob Laurel and Sasa Misailovic. Continualization of probabilistic programs with correction. In *Proceedings of the 29th European Symposium on Programming*, volume 12075 of *Lecture Notes in Computer Science*, pages 366–393. Springer, 2020. doi:10.1007/978-3-030-44914-8_14. (Cited on pages 119 and 139)

David Lazer, Ryan Kennedy, Gary King, and Alessandro Vespignani. The parable of Google flu: Traps in big data analysis. *Science*, 343(6176):1203–1205, 2014. doi:10.1126/science.1248506. (Cited on page 16)

Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference compilation and universal probabilistic programming. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1338–1348. PMLR, 2017. (Cited on page 178)

Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. Accelerating search-based program synthesis using learned probabilistic models. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 436–449. Association for Computing Machinery, 2018. doi:10.1145/3296979.3192410. (Cited on page 67)

Erich L. Lehmann and Howard J. D'Abrera. *Nonparametrics: Statistical Methods Based on Ranks*. Holden-Day Series in Probability and Statistics. Holden-Day, San Francisco, 1975. (Cited on page 183)

Erich L. Lehmann and Joseph P. Romano. *Testing Statistical Hypotheses*. Springer Texts in Statistics. Springer, New York, 3rd edition, 2005. doi:10.1007/0-387-27605-X. (Cited on page 183)

Emmanuel Lesaffre and Andrew B. Lawson. *Bayesian Biostatistics*. Statistics in Practice. John Wiley & Sons, Ltd., Chichester, 2012. doi:10.1002/9781119942412. (Cited on page 18)

Alexander Lew, Marco Cusumano-Towner, and Vikash Mansinghka. Recursive Monte Carlo and variational inference with auxiliary variables. *arXiv*, 2203.02836, 2022. doi:10.48550/arXiv.2203.02836. (Cited on page 177)

Alexander K. Lew, Marco F. Cusumano-Towner, Benjamin Sherman, Michael Carbin, and Vikash K. Mansinghka. Trace types and denotational semantics for sound programmable inference in probabilistic languages. *Proceedings of the ACM on Programming Languages*, 4(POPL):19.1–19.32, 2020. doi:10.1145/3371087. (Cited on page 117)

Cathryn M. Lewis and Jo Knight. Introduction to genetic association studies. In Ammar Al-Chalabi and Laura Almasy, editors, *Genetics of Complex Human Diseases: A Laboratory Manual*. Cold Spring Harbor Laboratory Press, 2009. doi:10.1101/pdb.top068163. (Cited on page 183)

Percy Liang, Michael I. Jordan, and Dan Klein. Learning programs: A hierarchical Bayesian approach. In *Proceedings of the 27th International Conference on Machine Learning*, pages 639–646. Omnipress, 2010. doi:10.5555/3104322.3104404. (Cited on page 66)

Alexander Lin, Yingzhuo Zhang, Jeremy Heng, Stephen A. Allsop, Kay M. Tye, Pierre E. Jacob, and Demba Ba. Clustering time series with nonlinear dynamics: A Bayesian non-parametric and particle-based approach. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 2476–2484. PMLR, 2019. (Cited on page 97)

Gloria Z. Lin. Bayesian active structure learning for Gaussian Process probabilistic programs. Master's thesis, Massachusetts Institute of Technology, 2022. (Cited on page 206)

Ramon C. Littell, Jane Pendergast, and Ranjini Natarajan. Modelling covariance structure in the analysis of repeated measures data. *Statistics in Medicine*, 19(13):1793–1819, 2000. doi:10.1002/1097-0258(20000715)19:13<1793::AID-SIM482>3.0.CO;2-Q. (Cited on page 29)

James R. Lloyd. jamesrobertlloyd/gpss-research: Kernel Structure Discovery Research Code, 2014. URL https://github.com/jamesrobertlloyd/gpss-research/tree/master/data. (Cited on pages 39 and 66)

Albert Y. Lo. On a class of Bayesian nonparametric estimates: I. Density estimates. *The Annals of Statistics*, 12(1):351–357, 1984. doi:10.1214/aos/1176346412. (Cited on pages 81, 108, and 115)

P. J. F. Lucas, R. W. Segaar, and A. R. Janssens. HEPAR: An expert system for the diagnosis of disorders of the liver and biliary tract. *Liver*, 9(5):266–275, 1989. doi:10.1111/j.1600-0676.1989.tb00410.x. (Cited on page 172)

Yi-An Ma, Yuansi Chen, Chi Jin, Nicolas Flammarion, and Michael I. Jordan. Sampling can be faster than optimization. *Proceedings of the National Academy of Sciences*, 116(42):20881–20885, 2019. doi:10.1073/pnas.1820003116. (Cited on page 214)

Deborah Mabbett and Helen Bolderson. Theories and methods in comparative social policy. In Jochen Clasen, editor, *Comparative Social Policy: Concepts, Theories, and Methods*, chapter 3. Blackwell Publishing, 1999. (Cited on page 216)

David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press, Cambridge, UK, 2003. (Cited on pages 18, 161, 207, and 209)

Chris J. Maddison, John Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Whye Teh. Filtering variational objectives. In *Proceedings of the 31st Conference on Neural Information Processing Systems*, volume 30 of *Advances in Neural Information Processing Systems*, pages 6576–6586. Curran Associates, Inc., 2017. (Cited on page 169)

Henry B. Mann and Donald R. Whitney. On a test of whether one of two random variables is stochastically larger than the other. *The Annals of Mathematical Statistics*, 18(1):50–60, 1947. doi:10.1214/aoms/1177730491. (Cited on page 185)

Zohar Manna and Richard Waldinger. Synthesis: Dreams to programs. *IEEE Transactions on Software Engineering*, 5(4):294–328, 1979. doi:10.1109/TSE.1979.234198. (Cited on page 67)

Zohar Manna and Richard Waldinger. A deductive approach to program synthesis. *ACM Transactions on Programming Languages and Systems*, 2(1):90–121, 1980. doi:10.1145/357084.357090. (Cited on page 67)

Vikash Mansinghka, Charles Kemp, Thomas Griffiths, and Joshua B. Tenenbaum. Structured priors for structure learning. In *Proceedings of the 22nd Conference Annual Conference on Uncertainty in Artificial Intelligence*, pages 324–331. AUAI Press, 2006. doi:10.48550/arXiv.1206.6852. (Cited on page 67)

Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: A higher-order probabilistic programming platform with programmable inference. *arXiv*, 1404.0099, 2014. doi:10.48550/arXiv.1404.0099. (Cited on page 20)

Vikash Mansinghka, Richard Mansinghka, Jay Baxter, Shafto Pat, and Baxter Eaves. BayesDB: a probabilistic programming system for querying the probable implications of data. *arXiv*, 1512.05006, 2015. doi:10.48550/arXiv.1512.05006. (Cited on page 71)

Vikash Mansinghka, Patrick Shafto, Eric Jonas, Cap Petschulat, Max Gasner, and Joshua B. Tenenbaum. CrossCat: A fully Bayesian nonparametric method for analyzing heterogeneous, high dimensional data. *Journal of Machine Learning Research*, 17(138):1–49, 2016. (Cited on pages 67, 71, 72, 97, and 115)

Vikash K. Mansinghka, Ulrich Schaechtle, Shivam Handa, Alexey Radul, Yutian Chen, and Martin Rinard. Probabilistic programming with programmable inference. In *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 603–616. Association for Computing Machinery, 2018. doi:10.1145/3192366.3192409. (Cited on pages 42 and 117)

Marvin Marcus and Henryk Minc. *A Survey of Matrix Theory and Matrix Inequalities.* Dover Books on Mathematics. Dover Publications, New York, 1992. (Cited on page 63)

Sandra Marcus, editor. *Automating Knowledge Acquisition for Expert Systems*, volume 57 of *The Springer International Series in Engineering and Computer Science.* Kluwer Academic Publishers, Boston, 1988. doi:10.1007/978-1-4684-7122-9. (Cited on page 15)

Alberto Maydeu-Olivares and Carlos Garcia-Forero. Goodness-of-fit testing. In Penelope Peterson, Eva Baker, and Barry McGaw, editors, *International Encyclopedia of Education*, pages 190–196. Elsevier, 3rd edition, 2010. doi:10.1016/B978-0-08-044894-7.01333-6. (Cited on page 185)

Kenichiro McAlinn and Mike West. Dynamic Bayesian predictive synthesis in time series forecasting. *Journal of Econometrics*, 210(1):155–169, 2019. doi:10.1016/j.jeconom.2018.11.010. (Cited on page 64)

Nikhil Mehta, Lawrence Carin, and Piyush Rai. Stochastic blockmodels meet graph neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 4466–4474. PMLR, 2019. (Cited on page 114)

David Merrell and Anthony Gitter. Inferring signaling pathways with probabilistic programming. *Bioinformatics*, 36(26):i822–i830, 2020. doi:10.1093/bioinformatics/btaa861. (Cited on page 18)

Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Sontag Kolobov. BLOG: Probabilistic models with unknown objects. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 1352–1359. Morgan Kaufmann Publishers Inc., 2005. (Cited on pages 66, 74, 120, 137, and 141)

Alejandro Molina, Antonio Vergari, Karl Stelzner, Robert Peharz, Pranav Subramani, Nicola Di Mauro, Pascal Poupart, and Kristian Kersting. SPFlow: An easy and extensible library for deep probabilistic learning using sum-product networks. *arXiv*, 1901.03704, 2020. doi:10.48550/arXiv.1901.03704. (Cited on pages 130, 142, and 159)

Wenlong Mou, Nhat Ho, Martin J. Wainwright, Peter L. Bartlett, and Michael I. Jordan. Sampling for Bayesian mixture models: MCMC with polynomial-time mixing. *arXiv*, 1912.05153, 2019. doi:10.48550/arXiv.1912.05153. (Cited on page 214)

Peter Mueller and Fernando Quintana. Random partition models with regression on covariates. *Journal of Statistical Planning and Inference*, 140(10):2801–2808, 2010. doi:10.1016/j.jspi.2010.03.002. (Cited on page 96)

Peter Mueller, Fernando Quintana, and Gary L. Rosner. A product partition model with regression on covariates. *Journal of Computational and Graphical Statistics*, 20(1):260–278, 2011. doi:10.1198/jcgs.2011.09066. (Cited on page 96)

Stephen Muggleton and Luc de Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19–20:629–679, 1994. doi:10.1016/0743-1066(94)90035-3. (Cited on page 114)

Kevin Murphy and Mark A. Paskin. Linear-time inference in hierarchical HMMs. In *Proceedings of the 15th Conference on Neural Information Processing Systems*, volume 14 of *Advances in Neural Information Processing Systems*, pages 833–840. MIT Press, 2001. (Cited on page 122)

Kevin P. Murphy. Conjugate Bayesian analysis of the Gaussian distribution. Technical report, University of British Columbia, 2007. (Cited on page 85)

Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive Computation and Machine Learning Series. MIT Press, Cambridge, MA, 2012. (Cited on page 18)

Masahiro Nakano, Katsuhiko Ishiguro, Akisato Kimura, Takeshi Yamada, and Naonori Ueda. Rectangular tiling process. In *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 361–369. PMLR, 2014. (Cited on page 99)

Praveen Narayanan and Chung-chieh Shan. Symbolic disintegration with a variety of base measures. *ACM Transactions on Programming Languages and Systems*, 42(2):9.1–9.60, 2020. doi:10.1145/3374208. (Cited on pages 120, 142, and 178)

Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic inference by program transformation in Hakaru (system description). In *Proceedings of the 13th International Symposium on Functional and Logic Programming*, volume 9613 of *Lecture Notes in Computer Science*, pages 62–79. Springer, 2016. doi:10.1007/978-3-319-29604-3_5. (Cited on pages 117, 142, and 178)

Radford Neal. The harmonic mean of the likelihood: Worst Monte Carlo method ever | Radford Neal's Blog, 2008. URL https://radfordneal.wordpress.com/2008/08/17/the-harmonic-mean-of-the-likelihood-worst-monte-carlo-method-ever/. (Cited on page 167)

Radford M. Neal. Markov chain sampling methods for Dirichlet process mixture models. *Journal of Computational and Graphical Statistics*, 9(2):249–265, 2000. doi:10.1080/10618600.2000.10474879. (Cited on pages 72, 88, 90, 91, 105, 106, and 207)

Radford M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001. doi:10.1023/A:1008923215028. (Cited on pages 167 and 169)

Joseph Near and David Darais. Differentially private synthetic data. *Cybsecuirty Insights: A NIST Blog*, 2021. URL https://www.nist.gov/blogs/cybersecurity-insights/differentially-private-synthetic-data. (Cited on page 77)

Flemming Nielson, Hanne Riis Nielson, and Chris Hanking. *Principles of Program Analysis*. Springer, Berlin, 1999. doi:10.1007/978-3-662-03811-6. (Cited on page 15)

Luis E. Nieto-Barajas and Alberto Contreras-Cristan. A Bayesian nonparametric approach for time series clustering. *Bayesian Analysis*, 9(1):147–170, 2014. doi:10.1214/13-BA852. (Cited on page 97)

Luis E. Nieto-Barajas and Fernando A. Quintana. A Bayesian non-parametric dynamic AR model for multiple time series analysis. *Journal of Time Series Analysis*, 37(5):675–689, 2016. doi:10.1111/jtsa.12182. (Cited on page 97)

Davide Nitti, Tinne De Laet, and Luc De Raedt. Probabilistic logic programing for hybrid relational domains. *Machine Learning*, 103:407–449, 2016. doi:10.1007/s10994-016-5558-8. (Cited on page 120)

Aditya Nori, Chung-Kil Hur, Sriram Rajamani, and Selva Samuel. R2: An efficient MCMC sampler for probabilistic programs. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 2476–2482. AAAI Press, 2014. (Cited on pages 119 and 136)

Aditya V. Nori, Sherjil Ozair, Sriram K. Rajamani, and Deepak Vijaykeerthy. Efficient synthesis of probabilistic programs. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 208–217. Association for Computing Machinery, 2015. doi:10.1145/2737924.2737982. (Cited on pages 64 and 65)

Iku Ohama, Hiromi Iida, Takuya Kida, and Hiroki Arimura. An extension of the infinite relational model incorporating interaction between objects. In *Proceedings of the 17th Pacific-Asia Conference on Knowledge Discovery and Data Mining*, volume 79819 of *Lecture Notes in Artificial Intelligence*, pages 147–159. Springer, 2013. doi:10.1007/978-3-642-37456-2_13. (Cited on page 114)

Eduardo A. Olaberria. *The Macroeconomics of Rare Events*. PhD thesis, University of Maryland, College Park, 2010. (Cited on page 36)

Agnieszka Oniśko. *Probabilistic Causal Models in Medicine: Application to Diagnosis of Liver Disorders*. PhD thesis, Institute of Biocybernetics and Biomedical Engineering, Polish Academy of Science, 2003. (Cited on pages 172 and 173)

Peter Orbanz and Daniel M. Roy. Bayesian models of graphs, arrays and other exchangeable random structures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):437–461, 2015. doi:10.1109/TPAMI.2014.2334607. (Cited on page 115)

Long Ouyang, Michael H. Tessler, Daniel Ly, and Noah D. Goodman. webppl-oed: A practical optimal experiment design system. In *Proceedings of the 40th Annual Cognitive Science Society Meeting*, pages 2192–2197. Cognitive Science Society, 2018. (Cited on page 178)

Brooks Paige and Frank Wood. Inference networks for sequential Monte Carlo in graphical models. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 3040–3049. PMLR, 2016. (Cited on page 178)

Liam Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15(6):1191–1253, 2003. doi:10.1162/089976603321780272. (Cited on page 177)

Ju-Hyun Park and David B. Dunson. Bayesian generalized product partition model. *Statistica Sinica*, 20(3):1203–1226, 2010. (Cited on page 96)

Neha Patki, Roy Wedge, and Kalyan Veeramachaneni. The synthetic data vault. In *Proceedings of the 2016 IEEE International Conference on Data Science and Advanced Analytics*, pages 399–410. IEEE Press, 2016. doi:10.1109/DSAA.2016.49. (Cited on page 80)

J. A. Peacock. Two-dimensional goodness-of-fit testing in astronomy. *Monthly Notices of the Royal Astronomical Society*, 202(3):615–627, 1983. doi:10.1093/mnras/202.3.615. (Cited on page 183)

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, San Mateo, 1988. (Cited on pages 18, 135, and 161)

Judea Pearl. *Causality.* Cambridge University Press, Cambridge, MA, 2nd edition, 2009. doi:10.1017/CBO9780511803161. (Cited on page 214)

Karl Pearson. 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. *Philosophical Magazine*, 5:157–175, 1900. doi:10.1080/14786440009463897. (Cited on page 185)

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(85):2825–2830, 2011. Visual guide at https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html. (Cited on page 16)

Robert Peharz, Antonio Vergari, Karl Stelzner, Alejandro Molina, Xiaoting Shao, Martin Trapp, Kristian Kersting, and Zoubin Ghahramani. Random sum-product networks: A simple and effective approach to probabilistic deep learning. In *Proceedings of the 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 334–344. PMLR, 2019. (Cited on page 108)

Fernando Pérez-Cruz. Estimation of information theoretic measures for continuous random variables. In *Proceedings of the 22nd Conference on Neural Information Processing Systems*, volume 21 of *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2008. (Cited on page 177)

Yura N. Perov and Frank D. Wood. Learning probabilistic programs. *arXiv*, 1407.2646, 2014. doi:10.48550/arXiv.1407.2646. (Cited on page 65)

Joshua C. Peterson, David D. Bourgin, Mayank Agrawal, Daniel Reichman, and Griffiths Thomas L. Using large-scale experiments and machine learning to discover theories of human decision-making. *Science*, 372(6547):1209–1214–, 2021. doi:10.1126/science.abe2629. (Cited on page 216)

Anthony N. Pettitt and Michael A. Stephens. The Kolmogorov–Smirnov goodness-of-fit statistic with discrete and grouped data. *Technometrics*, 19(2):205–210, 1977. doi:10.1080/00401706.1977.10489529. (Cited on page 185)

Ben Poole, Sherjil Ozair, Aaron Van Den Oord, Alex Alemi, and George Tucker. On variational bounds of mutual information. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5171–5180. PMLR, 2019. (Cited on page 177)

Hoifung Poon and Pedro Domingos. Sum-product networks: A new deep architecture. In *Proceedings of the 27th Annual Conference Uncertainty in Artificial Intelligence*, pages 337–346. AUAI Press, 2011. doi:10.48550/arXiv.1202.3732. (Cited on pages 69, 71, 118, 119, and 129)

Frederic Portoraro. Automated reasoning. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021. (Cited on page 15)

James G. Propp and David B. Wilson. Exact sampling with coupled Markov chains and applications to statistical mechanics. *Random Structures & Algorithms*, 9(1–2):223–252, 1996. doi:10.1002/(SICI)1098-2418(199608/09)9:1/2<223::AID-RSA14>3.0.CO;2-O. (Cited on page 207)

Michael A. Proschan and Brett Presnell. Expect the unexpected from conditional expectation. *The American Statistician*, 52(3):248–252, 1998. doi:10.1080/00031305.1998.10480576. (Cited on page 135)

Thierry Pun. A new method for grey-level picture thresholding using the entropy of the histogram. *Signal Processing*, 2(3):223–237, 1980. doi:10.1016/0165-1684(80)90020-1. (Cited on page 161)

Meng Qu, Yoshua Bengio, and Jian Tang. GMNN: Graph Markov neural networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5241–5250. PMLR, 2019. (Cited on page 114)

Joaquin Quiñonero-Candela and Carl E. Rasmussen. A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959, 2005. (Cited on page 64)

Jeff Racine and Qi Li. Nonparametric estimation of regression functions with both categorical and continuous data. *Journal of Econometrics*, 119(1):99–130, 2004. doi:10.1016/S0304-4076(03)00157-X. (Cited on page 74)

Tom Rainforth, Rob Cornish, Hongseok Yang, Andrew Warrington, and Frank Wood. On nesting Monte Carlo estimators. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4267–4276. PMLR, 2018. (Cited on page 177)

Carl E. Rasmussen. The infinite Gaussian mixture model. In *Proceedings of the 13th Conference on Neural Information Processing Systems*, volume 12 of *Advances in Neural Information Processing Systems*, pages 554–560. MIT Press, 1999. (Cited on page 115)

Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning.* The MIT Press, Cambridge, MA, 2006. (Cited on page 63)

Timothy R. C. Read and Noel A. C. Cressie. *Goodness-of-Fit Statistics for Discrete Multivariate Data.* Springer Series in Statistics. Springer-Verlag, New York, 1988. doi:10.1007/978-1-4612-4578-0. (Cited on page 185)

Scott Reed and Nando de Freitas. Neural programmer-interpreters. *arXiv*, 1511.06279, 2016. doi:10.48550/arXiv.1511.06279. (Cited on page 67)

Matthew Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006. doi:10.1007/s10994-006-5833-1. (Cited on page 114)

Fred Rieke, David Warland, Robert De Ruyter van Steveninck, and William Bialek. *Spikes: Exploring the Neural Code.* Computational Neuroscience Series. MIT Press, Cambridge, MA, 1997. (Cited on page 161)

Fabrizio Riguzzi. *Foundations of Probabilistic Logic Programming: Languages, Semantics, Inference and Learning.* River Publishers Series in Software Engineering. River Publishers, Delft, 2018. (Cited on page 120)

Daniel Ritchie, Paul Horsfall, and Noah D. Goodman. Deep amortized inference for probabilistic programs. *arXiv*, 1610.05735, 2016. doi:10.48550/arXiv.1610.05735. (Cited on page 178)

Christian Ritter and Martin A. Tanner. Facilitating the Gibbs sampler: The Gibbs stopper and the griddy-Gibbs sampler. *Journal of the American Statistical Association*, 87(419):861–868, 1992. doi:10.1080/01621459.1992.10475289. (Cited on page 88)

Herbert Robbins. The empirical Bayes approach to statistical decision problems. *The Annals of Mathematical Statistics*, 35(1):1–20, 1964. doi:10.1214/aoms/1177703729. (Cited on page 88)

Robert W. Robinson. Counting unlabeled acyclic digraphs. In *Combinatorial Mathematics V*, volume 622 of *Lecture Notes in Mathematics*, pages 28–43. Springer, 1977. doi:10.1007/BFb0069178. (Cited on page 114)

Abel Rodriguez and Enrique ter Horst. Bayesian dynamic density estimation. *Bayesian Analysis*, 3(2): 339–365, 2008. doi:10.1214/08-BA313. (Cited on page 97)

Daniel M. Roy and Yee Whye Teh. The Mondrian process. In *Proceedings of the 22nd Conference on Neural Information Processing Systems*, volume 21 of *Advances in Neural Information Processing Systems*, pages 833–840. Curran Associates, Inc., 2008. (Cited on pages 99 and 115)

Jean-François Rual et al. Towards a proteome-scale map of the human protein-protein interaction network. *Nature*, 437:1173–1178, 2005. doi:10.1038/nature04209. (Cited on page 99)

Walter Rudin. *Principles of Mathematical Analysis.* International Series in Pure and Applied Mathematics. McGraw-Hill, New York, 3rd edition, 1976. (Cited on page 189)

Rudolph J. Rummel. Dimensionality of nations project: Attributes of nations and behavior of nation dyads, 1950-1965. Technical Report ICPSR 5409, Inter-university Consortium for Political and Social Research, 1999. (Cited on pages 100 and 108)

Stuart J. Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach.* Pearson, Hoboken, 4th edition, 2021. (Cited on page 18)

Feras Saad and Vikash Mansinghka. A probabilistic programming approach to probabilistic data analysis. In *Proceedings of the 30th Conference on Neural Information Processing Systems*, volume 29 of *Advances in Neural Information Processing Systems*, pages 2011–2019. Curran Associates, Inc., 2016a. (Cited on pages 71, 130, and 178)

Feras Saad and Vikash Mansinghka. Probabilistic data analysis with probabilistic programming. *arXiv*, 1608.05347, 2016b. doi:10.48550/arXiv.1608.05347. (Cited on page 71)

Feras Saad and Vikash Mansinghka. Detecting dependencies in sparse, multivariate databases using probabilistic programming and non-parametric Bayes. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 632–641. PMLR, 2017. (Cited on pages 71 and 178)

Feras Saad, L. Casarsa, and Vikash Mansinghka. Probabilistic search for structured data via probabilistic programming and nonparametric Bayes. *arXiv*, 1704.01087, 2017. doi:10.48550/arXiv.1704.01087. (Cited on pages 71 and 178)

Feras A. Saad and Vikash K. Mansinghka. Temporally-reweighted Chinese restaurant process mixtures for clustering, imputing, and forecasting multivariate time series. In *Proceedings of the 21st International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 755–764. PMLR, 2018. (Cited on pages 25 and 67)

Feras A. Saad and Vikash K. Mansinghka. Hierarchical infinite relational model. In *Proceedings of the 37th Conference on Uncertainty in Artificial Intelligence*, volume 161 of *Proceedings of Machine Learning Research*, pages 1067–1077. PMLR, 2021. (Cited on pages 25 and 67)

Feras A. Saad, Marco F. Cusumano-Towner, Ulrich Schaechtle, Martin C. Rinard, and Vikash K. Mansinghka. Bayesian synthesis of probabilistic programs for automatic data modeling. *Proceedings of the ACM on Programming Languages*, 3(POPL):37.1–37.32, 2019a. doi:10.1145/3290350. (Cited on pages 20, 25, and 64)

Feras A. Saad, Cameron E. Freer, Nathanael L. Ackerman, and Vikash K. Mansinghka. A family of exact goodness-of-fit tests for high-dimensional discrete distributions. In *Proceedings of the 22nd International Conference on Artificial Intelligence and Statistics*, volume 89 of *Proceedings of Machine Learning Research*, pages 1640–1649. PMLR, 2019b. (Cited on pages 25 and 197)

Feras A. Saad, Martin C. Rinard, and Vikash K. Mansinghka. SPPL: Probabilistic programming with fast exact symbolic inference. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Design and Implementation*, pages 804–819. Association for Computing Machinery, 2021. doi:10.1145/3453483.3454078. (Cited on pages 25 and 178)

Feras A. Saad, Marco Cusumano-Towner, and Vikash K. Mansinghka. Estimators of entropy and information via inference in probabilistic models. In *Proceedings of the 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 5604–5621. PMLR, 2022. (Cited on page 25)

Ruslan Salakhutdinov, Joshua B. Tenenbaum, and Antonio Torralba. Learning with hierarchical-deep models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1958–1971, 2013. doi:10.1109/TPAMI.2012.269. (Cited on page 115)

Tim Salimans, Diederik Kingma, and Max Welling. Markov chain Monte Carlo and variational inference: Bridging the gap. In *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1218–1226. PMLR, 2015. (Cited on page 167)

Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. In *Proceedings of the 34th ACM SIGPLAN Conference Programming Language Design and Implementation*, pages 447–458. Association for Computing Machinery, 2013. doi:10.1145/2491956.2462179. (Cited on pages 117 and 142)

Leonard J. Savage. *The Foundations of Statistics.* John Wiley & Sons, Inc., New York, 1954. doi:10.1002/nav.3800010316. (Cited on page 18)

Ulrich Schaechtle, Feras Saad, Alexey Radul, and Vikash Mansinghka. Time series structure discovery via probabilistic program synthesis. *arXiv*, 1611.07051, 2017. doi:10.48550/arXiv.1611.07051. (Cited on page 42)

Ulrich Schaechtle, Cameron Freer, Zane Shelby, Feras Saad, and Vikash Mansinghka. Bayesian AutoML for databases via the InferenceQL probabilistic programming system. In *Proceedings of the 1st International Conference on Automated Machine Learning (Late-Breaking Workshop Track)*, 2022. (Cited on page 71)

Mark J. Schervish. *Theory of Statistics.* Springer Series in Statistics. Springer-Verlag, New York, 1995. doi:10.1007/978-1-4612-4250-5. (Cited on page 167)

Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. In *Proceedings of the 18th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 305–316. Association for Computing Machinery, 2013. doi:10.1145/2451116.2451150. (Cited on page 66)

Moses Schönfinkel. Über die Bausteine der mathematischen Logik. *Mathematische Annalen*, 92:305–316, 1924. doi:10.1007/BF01448013. (Cited on page 66)

Patrick Shafto, Charles Kemp, Vikash Mansinghka, and Joshua B. Tenenbaum. A probabilistic model of cross-categorization. *Cognition*, 120(1):1–25, 2011. doi:10.1016/j.cognition.2011.02.010. (Cited on page 71)

Babak Shahbaba and Radford Neal. Nonlinear models using Dirichlet process mixtures. *Journal of Machine Learning Research*, 10(63):1829–1850, 2009. (Cited on page 96)

Michael Shalev. Limits and alternatives to multiple regression in comparative research. In Lars Mjoset and Tommy H. Clausen, editors, *Capitalism Compared*, number 24 in Comparative Social Research, chapter 7, pages 261–308. Emerald Group Publishing, Ltd., 2007. doi:10.1016/S0195-6310(06)24006-7. (Cited on page 216)

Claude E. Shannon. A mathematical theory of communication. *The Bell System Technical Journal*, 27 (3):379–423, 1948. doi:10.1002/j.1538-7305.1948.tb01338.x. (Cited on pages 161 and 164)

Bernard W. Silverman. *Density Estimation for Statistics and Data Analysis*, volume 26 of *Monographs on Statistics and Applied Probability.* Chapman and Hall, London, 1986. (Cited on page 74)

Edward E. Smith. Concepts and induction. In Michael I. Posner, editor, *Foundations of Cognitive Science*, pages 501–526. MIT Press, 1989. (Cited on page 15)

Armando Solar-Lezama, Liviu Tancau, Rastislav Bodik, Sanjit Seshia, and Vijay Saraswat. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 404–415. Association for Computing Machinery, 2006. doi:10.1145/1168857.1168907. (Cited on page 67)

Ghanem Soltana, Mehrdad Sabetzadeh, and Lionel C. Briand. Synthetic data generation for statistical testing. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, pages 872–882. IEEE Press, 2017. doi:10.1109/ASE.2017.8115698. (Cited on page 77)

David J. Spiegelhalter, A. Philip Dawid, Steffen L. Lauritzen, and Robert G. Cowell. Bayesian analysis in expert systems. *Statistical Science*, 8(3):219–247, 1993. doi:10.1214/ss/1177010888. (Cited on page 136)

Siddharth Srivastava, Nicholas Hay, Yi Wu, and Stuart Russell. The extended semantics for probabilistic programming languages. In *Workshop on Probabilistic Programming Semantics*, 2017. (Cited on page 120)

Michael Steele and Janet Chaseling. Powers of discrete goodness-of-fit test statistics for a uniform null against a selection of alternative distributions. *Communications in Statistics—Simulation and Computation*, 35(4):1067–1075, 2006. doi:10.1080/03610910600880666. (Cited on pages 201 and 209)

Brett Stevens and Aaron Williams. The coolest order of binary strings. In *Proceedings of the 6th International Conference on Fun with Algorithms*, volume 7288 of *Lecture Notes in Computer Science*, pages 322–333. Springer, 2012. doi:10.1007/978-3-642-30347-0_32. (Cited on page 203)

Andreas Stuhlmüller and Noah Goodman. A dynamic programming algorithm for inference in recursive probabilistic programs. *arXiv*, 1206.3555, 2012. doi:10.48550/arXiv.1206.3555. (Cited on page 143)

Xiaoyuan Su and Taghi M. Khoshgoftaar. A survey of collaborative filtering techniques. *Advances in Artificial Intelligence*, 2009(421425), 2009. doi:10.1155/2009/421425. (Cited on page 99)

Ilya Sutskever, Russ R. Salakhutdinov, and Joshua B. Tenenbaum. Modelling relational data using Bayesian clustered tensor factorization. In *Proceedings of the 23rd Conference on Neural Information Processing Systems*, volume 22 of *Advances in Neural Information Processing Systems*, pages 1821–1828. Curran Associates, Inc., 2009. (Cited on page 99)

Synthetic Data Vault. sdv-dev/SDV: Synthetic Data Generation for Tabular, Relational, and Time Series Data, 2022. URL https://github.com/sdv-dev/SDV. (Cited on page 80)

Sean Talts, Michael Betancourt, Daniel Simpson, Aki Vehtari, and Andrew Gelman. Validating Bayesian inference algorithms with simulation-based calibration. *arXiv*, 1804.06788, 2018. doi:10.48550/arXiv.1804.06788. (Cited on pages 183 and 207)

Ben Taskar, Peter Abbeel, and Daphne Koller. Discriminative probabilistic models for relational data. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence*, pages 485–492. AUAI Press, 2002. doi:10.48550/arXiv.1301.0604. (Cited on page 114)

Sean J. Taylor and Benjamin Letham. Forecasting at scale. *The American Statistician*, 72(1):37–45, 2018. doi:10.1080/00031305.2017.1380080. (Cited on pages 36, 40, and 96)

Joshua B. Tenenbaum, Charles Kemp, Thomas L. Griffiths, and Noah D. Goodman. How to grow a mind: Statistics, structure, and abstraction. *Science*, 331(6022):1279–1285, 2011. doi:10.1126/science.1192788. (Cited on pages 18 and 67)

Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics*. Intelligent Robotics and Autonomous Agents Series. MIT Press, Cambridge, MA, 2005. (Cited on page 18)

Luke Tierney. Markov chains for exploring posterior distributions. *The Annals of Statistics*, 22(4): 1701–1728, 1994. doi:10.1214/aos/1176325750. (Cited on pages 48 and 49)

Hu Kuo Ting. On the amount of information. *Theory of Probability & Its Applications*, 7(4):439–447, 1962. doi:10.1137/1107041. (Cited on page 164)

Anh Tong and Jaesik Choi. Automatic generation of probabilistic programming from time series data. *arXiv*, 1607.00710, 2016. doi:10.48550/arXiv.1607.00710. (Cited on page 66)

JunYong Tong and Nick Torenvliet. Temporally-reweighted dirichlet process mixture anomaly detector. In *Proceedings of the 2020 International Conference on Data Mining Workshops*, pages 267–274. IEEE Press, 2020. doi:10.1109/ICDMW51313.2020.00045. (Cited on page 98)

Neil Toronto, Jay McCarthy, and David Van Horn. Running probabilistic programs backwards. In *Proceedings of the 24th European Symposium on Programming*, volume 12075 of *Lecture Notes in Computer Science*, pages 53–79. Springer, 2015. doi:10.1007/978-3-662-46669-8_3. (Cited on page 142)

Martin Trapp, Robert Peharz, Hong Ge, Franz Pernkopf, and Zoubin Ghahramani. Bayesian learning of sum-product networks. In *Proceedings of the 33rd Conference in Neural Information Processing Systems*, volume 32 of *Advances in Neural Information Processing Systems*, pages 6347–6358. Curran Associates, Inc., 2019. (Cited on pages 69 and 158)

Oskar Triebe, Hansika Hewamalage, Polina Pilyugina, Laptev Nikolay, Christoph Bergmeir, and Ram Rajagopal. NeuralProphet: Explainable forecasting at scale. *arXiv*, 2111.15397, 2021. doi:10.48550/arXiv.2111.15397. (Cited on page 36)

Franklyn Turbak and David Gifford. *Design Concepts in Programming Languages*. MIT Press, Cambridge, MA, 2008. (Cited on page 54)

Amos Tversky and Daniel Kahneman. Advances in prospect theory: Cumulative representation of uncertainty. *Journal of Risk and Uncertainty*, 5:297–323, 1992. doi:10.1007/BF00122574. (Cited on page 216)

Tomer D. Ullman and Joshua B. Tenenbaum. Bayesian models of conceptual development: Learning as building models of the world. *Annual Review of Developmental Psychology*, 2(1):533–558, 2020. doi:10.1146/annurev-devpsych-121318-084833. (Cited on pages 18 and 65)

Tomer D. Ullman, Noah D. Goodman, and Joshua B. Tenenbaum. Theory learning as stochastic search in the language of thought. *Cognitive Development*, 27(4):455–480, 2012. doi:10.1016/j.cogdev.2012.07.005. (Cited on page 65)

Tomer D. Ullman, Andreas Stuhlmüller, Noah D. Goodman, and Joshua B. Tenenbaum. Learning physical parameters from dynamic scenes. *Cognitive Psychology*, 104:57–82, 2018. doi:10.1016/j.cogpsych.2017.05.006. (Cited on page 65)

Union of Concerned Scientists. UCS Satellite Database, 2016. URL https://www.ucsusa.org/resources/satellite-database. (Cited on page 77)

U.S. Bureau of Transportation Statistics. Passengers: All Carriers - All Airports, 2022. URL https://transtats.bts.gov/Data_Elements.aspx. (Cited on page 27)

Gregory Valiant and Paul Valiant. Estimating the unseen: An n/log(n)-sample estimator for entropy and support size, shown optimal via new CLTs. In *Proceedings of the 43rd Annual ACM Symposium on Theory of Computing*, pages 685–694. Association for Computing Machinery, 2011. doi:10.1145/1993636.1993727. (Cited on page 186)

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv*, 1809.10756, 2021. doi:10.48550/arxiv.1809.10756. (Cited on page 18)

Antonio Vergari, Alejandro Molina, Robert Peharz, Zoubin Ghahramani, Kristian Kersting, and Isabel Valera. Automatic bayesian density analysis. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 5207–5215. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33015207. (Cited on pages 69 and 158)

Jonas Vlasselaer, Guy Van den Broeck, Angelika Kimmig, Wannes Meert, and Luc De Raedt. Anytime inference in probabilistic logic programs with Tp-compilation. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence*, pages 1852–1858. International Joint Conferences on Artificial Intelligence, 2015. (Cited on page 143)

Satosi Watanabe. Information theoretical analysis of multivariate correlation. *IBM Journal of Research and Development*, 4(1):66–82, 1960. doi:10.1147/rd.41.0066. (Cited on page 164)

David A. Williams. Improved likelihood ratio tests for complete contingency tables. *Biometrika*, 63(1): 33–37, 1976. doi:10.1093/biomet/63.1.33. (Cited on page 185)

Michael Williams. How good are your fits? Unbinned multivariate goodness-of-fit tests in high energy physics. *Journal of Instrumentation*, 5(09):P09004, 2010. doi:10.1088/1748-0221/5/09/P09004. (Cited on page 183)

Andrew G. Wilson and Ryan P. Adams. Gaussian process kernels for pattern discovery and extrapolation. In *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1067–1075. PMLR, 2013. (Cited on page 67)

David Wingate and Theophane Weber. Automated variational inference in probabilistic programming. *arXiv*, 1301.1299, 2013. doi:10.48550/arXiv.1301.1299. (Cited on page 117)

Yi Wu, Siddharth Srivastava, Nicholas Hay, Simon Du, and Stuart Russell. Discrete-continuous mixtures in probabilistic programming: Generalized semantics and inference algorithms. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5343–5352. PMLR, 2018. (Cited on pages 117, 119, 120, and 130)

Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models. *arXiv*, 1611.04273, 2016. doi:10.48550/arXiv.1611.04273. (Cited on page 177)

Lei Xu, Maria Skoularidou, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Modeling tabular data using Conditional GAN. In *Proceedings of the 33rd Conference in Neural Information Processing Systems*, volume 32 of *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019. (Cited on page 80)

Zhao Xu, Volker Tresp, Kai Yu, and Hans-Peter Kriegel. Infinite hidden relational models. In *Proceedings of the 22nd Conference on Uncertainty in Artificial Intelligence*, pages 544–551. AUAI Press, 2006. doi:10.48550/arXiv.1206.6864. (Cited on pages 99, 101, 108, and 115)

Zhao Xu, Volker Tresp, Shipeng Yu, Kai Yu, and Hans-Peter Kriegel. Fast inference in infinite hidden relational models. In *Proceedings of the 5th International Workshop on Mining and Learning with Graphs*, 2007. (Cited on page 106)

Junyu Xuan, Jie Lu, Guangquan Zhang, Richard Y. D. Xu, and Xiangfeng Luo. Bayesian nonparametric relational topic model through dependent gamma processes. *IEEE Transactions on Knowledge and Data Engineering*, 40(7), 2017. doi:10.1109/TKDE.2016.2636182. (Cited on page 99)

Jiasen Yang, Qiang Liu, Vinayak Rao, and Jennifer Neville. Goodness-of-fit testing for discrete distributions via Stein discrepancy. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 5561–5570. PMLR, 2018. (Cited on page 186)

Byung-Jun Yoon. Hidden Markov models and their applications in biological sequence analysis. *Current Genomics*, 10(6):402–415, 2009. doi:10.2174/138920209789177575. (Cited on page 216)

Jieyuan Zhang and Jingling Xue. Incremental precision-preserving symbolic inference for probabilistic programs. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 237–252. Association for Computing Machinery, 2019. doi:10.1145/3314221.3314623. (Cited on pages 117 and 142)

Lina Zhou, Shimei Pan, Jianwu Wang, and Athanasios V. Vasilakos. Machine learning on big data: Opportunities and challenges. *Neurocomputing*, 237(10):350–361, 2017. doi:10.1016/j.neucom.2017.01.026. (Cited on page 16)

Rongxi Zhou, Ru Cai, and Guanqun Tong. Applications of entropy in finance: A review. *Entropy*, 15 (11):4909–4931, 2013. doi:10.3390/e15114909. (Cited on page 161)

Xiaojin Zhu, Zoubin Ghahramani, and John Lafferty. Time-sensitive Dirichlet process mixture models. Technical Report CMU-CALD-05-104, Carnegie-Mellon University, 2005. (Cited on page 97)

Pedro Zuidberg Dos Martires, Anton Dries, and Luc De Raedt. Exact and approximate weighted model integration with probability density functions using knowledge compilation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, pages 7825–7833. AAAI Press, 2019. doi:10.1609/aaai.v33i01.33017825. (Cited on page 143)