TECHNISCHE UNIVERSITEIT DELFT

MASTER OF SCIENCE THESIS IN COMPUTER SCIENCE

SMT-guided Metropolis-Hastings inference

Stefania-Mara COMAN

Supervisors:
Dr. Sebastijan DUMANČI
Dr. Neil YORKE-SMITH

9th July 2025

Delft University of Technology

SMT-guided Metropolis-Hastings inference

Master's Thesis in Computer Science

Algorithmics group
Faculty of Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology

Stefania-Mara Coman

9th July 2025

Author
Stefania-Mara Coman

Title
SMT-guided Metropolis-Hastings inference

MSc presentation
July 16, 2025

Graduation Committee

Chair: Dr. N. Yorke-Smith	Delft University of Technology
Main supervisor: Prof. S Dumančić	Delft University of Technology
Committee member: Prof. A. Costea	Delft University of Technology

Abstract

Probabilistic programming offers an intuitive and expressive way to define statistical models, rendering it particularly effective in modeling problems where uncertainty plays a crucial role. As adoption increases and models become more expressive, the challenge of effective inference becomes increasingly pronounced. Effective inference often requires tailoring algorithms to the structure of the underlying model. While many probabilistic programming systems allow users to implement custom inference strategies via programmable inference, this process remains largely manual and heavily reliant on domain-specific expertise, particularly for sampling-based methods. This paper investigates the use of Satisfiability Modulo Theories (SMT) to automate the generation of tailored, observation-aware proposals for guiding inference within Metropolis-Hastings in Gen, a probabilistic programming system. By reformulating the search for high-likelihood traces as a constraint optimization problem, this work explores whether SMT-based solutions can improve proposal quality and convergence. Empirical results indicate that SMT-derived traces offer a promising starting point for inference but are less effective as an active search heuristic. These findings suggest a new direction for automated, structure-aware proposal generation in probabilistic programming.

Preface

This thesis concludes my academic journey at TU Delft. Reflecting on the past years, I am grateful for the challenges that shaped me, both as a student and as a person. In these years, I found new depths of friendship and family, and uncovered more than a few versions of myself. Within the darker seasons, I found light in new passions and a restless drive for life. The road, or rather the roller coaster, has carried me here, and I would like to thank all those who rode alongside me on the bumpy tracks, making both the joy and the fright unforgettable.

First, I would like to thank my supervisor, Sebastijan Dumančić. Your guidance, patience, and encouragement made this journey not only possible but truly enjoyable. Thank you for your insights and genuine kindness that have made this collaboration memorable and rewarding.

I would also like to extend my appreciation to the PONY group for the vibrant exchange of ideas, the collective curiosity, and the inspiring depths of our discussions. Being part of this collective has been a true highlight. Thank you for sharing the joy of search and discovery. Special thanks to Tilman, Rueben, and Issa: your perspectives and conversations were deeply motivating.

To my colleagues and friends, who made late nights and endless debugging sessions not only bearable but incredibly enjoyable (in hindsight). I am grateful for the friendships and memories built here and for the ways they'll continue to shape the person I am. To Shaghy, thank you for the kind of friendship that feels like home, and for the laughter that rose above the chaos of our twenties. To Smara, thank you for being an inspirational pillar in my life. Your strength and presence have been a guiding light. And to Louise, thank you for a lifetime of shared chapters and for always bringing a little sunshine with you, no matter the weather.

A special thanks to my family for their unwavering support and love, always present no matter the distance. Thank you for inspiring courage, especially in times of uncertainty. I am deeply grateful for your wisdom and patience, always trusting me to find my own path.

Stefania-Mara Coman

Delft, The Netherlands
9th July 2025

Contents

Chapter 1

Introduction

Probabilistic programming (PP) offers an intuitive and expressive way to define statistical models. By turning probabilistic models into executable programs, PP offers a powerful abstraction for reasoning under uncertainty. This representation enables direct simulation of the model, allowing for synthetic data generation. This paradigm has found broad application across diverse domains, including protein interaction networks in computational biology [Merrell and Gitter, 2020], scene perception in car driving [Geiger et al., 2011], disease transmission modeling in epidemiology [Smedemark-Margulies et al., 2022], and audio signal processing [Villescas et al., 2020].

As adoption increases and models become more expressive, the challenge of effective inference becomes increasingly pronounced. In this context, learning is typically framed as an inference problem, involving estimating unknown parameters of a model given observed data. Efficient inference strategies often require algorithms that are carefully tailored to the structure of the model, which is typically a manual, time-intensive, and expertise-heavy process.

Probabilistic programming systems (PPS) allow users to experiment with a range of inference strategies and adapt them to the specific structure of the model. These systems provide a structured framework for expressing and solving probabilistic models by decoupling model specification from inference. Modern PPSs offer a suite of standard inference algorithms, such as Gibbs sampling [Geman and Geman, 1984], Metropolis-Hastings [Hastings, 1970], and Variational inference [Hoffman et al., 2013], alongside varying degrees of customizability.

This work employs Gen, a probability programming system, which offers a powerful and flexible infrastructure for defining probabilistic models and implementing tailored inference routines [Cusumano-Towner et al., 2019]. Through high-level abstractions, such as trace manipulation, density computation, and custom proposals, Gen enables users to craft model-specific inferences without focusing on low-level mathematical implementation. Custom proposals specifically allow the construction of sophisticated heuristics that improve inference performance. However, the ability to define custom proposals shifts the responsibility

of designing effective heuristic strategies onto the user. This remains a manual, time-consuming endeavor that demands considerable experimentation and domain expertise. This poses a major bottleneck in the process of leveraging Gen's full inference power and advancing towards automated inference.

This motivates the need for automated, model-aware mechanisms that assist or replace manual proposal design. Automating custom inference, in particular, the generation of high-quality proposal traces, could dramatically reduce development effort and make inference tailoring more accessible to non-experts.

While various inference strategies exist within PPS frameworks, this work focuses on Metropolis-Hastings (MH) [Hastings, 1970], a widely used Markov Chain Monte Carlo (MCMC) method. MH is especially well-suited for general-purpose inference, offering the ability to sample from an unknown target distribution, due to its simplicity and theoretical guarantees. However, the generality of the method comes at a cost: it can suffer from slow convergence on highly complex models, especially ones involving a mix of discrete-continuous samples and branching control flow. In these cases, practical efficiency heavily depends on the quality of the proposal mechanism, which, if poorly designed, leads to a low acceptance rate, delaying convergence.

Previous works have explored modifying the proposal distribution to better guide the sampling algorithm. However, these approaches still rely on sampling randomness, with convergence remaining an asymptotic guarantee. Additionally, custom proposal distributions often depend on the programmer's understanding of the underlying model. This paper proposes moving away from resampling traces and towards deterministic programming techniques to determine valid and higher-probability traces. For this purpose, this work investigates employing constraint programming as a method of identifying suitable traces to guide the MH algorithm.

A compelling case for constraint programming as a promising alternative lies in the use of discrete random choice domains in probabilistic programs. While Metropolis-Hastings is capable of handling both discrete and continuous samples, it often struggles to identify effective transitions in the discrete case. In discrete spaces, small changes can lead to drastic likelihood shifts, resulting in a higher rejection rate and slower convergence. This issue is further compounded by existing branching structures, which increase the difficulty of sampling high-probability traces. Additionally, the requirement to satisfy observations through random proposals introduces non-trivial constraints on the sampling space. Together, these challenges suggest a natural reformulation of the inference task as a constraint satisfaction problem, specifically for Satisfiability Modulo Theories (SMT) solvers, which are designed to efficiently handle structured constraints over mixed domains.

Contribution Building on this motivation, this paper investigates whether SMT solvers, a class of constraint programming techniques, can be effectively integrated within Metropolis-Hastings inference to improve practical performance. The central premise is leveraging SMT-derived traces as structured, high-quality pro-

posals to guide the sampling process toward a high-probability region more efficiently. To this end, the potential of SMT-solver-generated traces to accelerate convergence, improve acceptance rates, and enhance the automated development of model-specific inference procedures is explored. Accordingly, the following primary research question is posed:

Q: Can SMT guidance improve the effectiveness of MH-based inference in probabilistic programs?

To investigate this question, the following sub-questions are considered:

- **Q1:** Does using an SMT-derived trace as the starting point accelerate convergence in MH inference?
- **Q2:** Can SMT-derived traces, combined with a Gaussian drift mechanism, effectively guide local MH proposals during early inference?
- **Q3:** Can SMT-derived traces be used on-demand to dynamically guide MH inference when the sampler stagnates or enters low-probability regions?

The first two sub-questions focus on whether SMT-based trace initialization enhances both the starting point and early progression of MH inference. The third explores the use of SMT traces as a more general guiding mechanism deployed when necessary during inference.

To investigate these research questions, this paper proposes a framework that translates probabilistic programs into constraint models. The reformulation allows trace probabilities to be encoded as objective functions, to be maximized through SMT solving. To support this, this work presents several trace scoring approximations to facilitate the optimization task's tractability. Finally, the resulting SMT-provided trace solutions are integrated into the inference process, either as initial proposals or dynamic, mid-run interventions, to guide the sampling process more effectively.

Experimental results demonstrate that high-quality SMT-based trace solutions are capable of improving early-stage convergence but are less effective as active heuristic mid-inference.

The contribution of this work lies in introducing a model-specific yet automated and structured method for generating custom proposals, aimed at improving inference performance across diverse probabilistic models. By sidestepping the need for labor-intensive, handcrafted heuristics, this approach takes a step toward the broader goal of general-purpose automated inference. It lays the groundwork for future systems that combine symbolic reasoning and probabilistic inference to deliver both flexibility and efficiency, offering a promising direction for scaling inference in increasingly complex models.

Chapter 2

Background

This chapter outlines the foundational concepts necessary to understand the methods presented in this thesis. It first introduces core ideas from probabilistic programming, followed by an overview of solver technologies relevant to the proposed approach.

2.1 Probabilistic programming

Probabilistic programs (PP) provide an intuitive way to represent statistical models. They differ from normal programs through the addition of two types of statements: sampling statements, where variables can take random values from various distributions, and observation statements, where variables can be conditioned on observed data. Here, two types of sampled variables are distinguished: those with observations and those to be inferred later; the latter ones are called latent variables.

In probabilistic programs, the random choices dictate the computation and possible outcomes of the model. As a result, the program induces a probability distribution over its execution traces. A trace represents the sequence of random choices made along an execution path. As probabilistic programs are executable, sampling from this distribution can be achieved by running the program. Effectively, probabilistic programs implement a simulator for a distribution.

Inference in probabilistic programs refers to the process of estimating latent variables, unobserved random choices, that explain observed data. This involves conditioning the model on observed values, thereby restricting the distribution over execution traces to those that align with the observations, resulting in the posterior distribution.

To tackle the inference task, there are two main approaches: exact methods and approximate methods. Exact inference aims to compute the posterior distribution precisely, either by solving it analytically or by exhaustively evaluating all possible variable configurations. This method is only applicable for models with limited complexity and structure, such as small-scale graphical models, where summing or integrating over all possible configurations is tractable. For most real-world

models, however, exact inference becomes computationally infeasible, making approximate inference methods essential. These methods trade precision for scalability. Approximate inference encompasses a broad range of algorithms, including sampling-based methods such as those from the Markov Chain Monte Carlo (MCMC) family, for example, Metropolis-Hastings, Gibbs sampling [Geman and Geman, 1984], and Hamiltonian Monte Carlo [Neal and Neal, 2011], as well as Sequential Monte Carlo (SMC) methods and Variational Inference (VI) techniques [Hoffman et al., 2013]. Each inference technique is best suited to specific classes of problems, where efficient inference requires tailoring the algorithm to the structure and properties of the model.

2.1.1 Probabilistic programming systems

Probabilistic programming systems (PPS) aim to decouple model specification from inference implementation, enabling users to experiment with different inference strategies on the same model. These systems formalize probabilistic modeling and inference by providing a modeling language in which users can express models, along with high-level constructs that automate, abstract, or expose parts of the inference process. This separation opens up new possibilities for tailoring and composing inference techniques to better suit specific modeling needs.

While many PPSs provide standard inference algorithms out of the box, some systems additionally support inference tailoring, allowing procedures to be adapted or customized to better align with the structure of the model. A wide range of probabilistic programming systems exists, each offering varying degrees of support for model expressivity and inference control. Among systems that provide customizable and composable inference, there are: Turing [Ge et al., 2018], Venture [Mansinghka et al., 2014], Pyro [Bingham et al., 2019], NumPyro [Phan et al., 2019], Gen [Cusumano-Towner et al., 2019], WebPPL [Goodman and Stuhlmüller, 2014], and Birch [Murray and Schön, 2020]. To implement SMT-guided Metropolis-Hastings, this work leverages Gen, which offers high-level abstractions and programmable support for constructing custom inference algorithms.

2.1.2 Gen

Gen [Cusumano-Towner et al., 2019] is a general-purpose probabilistic programming system designed for representing and reasoning about probabilistic models. It combines flexible model specification and an inference library for building efficient, custom algorithms using high-level abstractions.

To specify a generative model (i.e., a probabilistic program), Gen augments Julia's syntax with the construct `{address} ~ distribution` to represent random choices at a given address. A generative function example that models points on a line is given in Listing 2.1. A trace, a record of all the random choices sampled within an execution path, is represented in Gen as a dictionary mapping unique address names to their sampled value. Observations in Gen are simply such a choice

map, containing the observational values. For the given line model, observations would map the (:y, i) addresses to observed data points, corresponding to the observed outcomes of a linear regression problem.

```
@gen function line_model(xs::Vector{Float64})
    # Sample a slope and intercept from prior distributions
    slope = ({:slope} ~ normal(0, 1))
    intercept = ({:intercept} ~ normal(0, 2))

    # Sample noisy y-values for each x in the input
    for (i, x) in enumerate(xs)
        ({(:y, i)} ~ normal(slope * x + intercept, 0.1))
    end

    return y
end
```

Listing 2.1: Line model with random slope and intercept in Gen.jl

Posterior inference in this setting involves inferring the slope and intercept values that best explain the observed (x, y) points, that is, finding a trace that is consistent with the data and likely under the model. More generally, inference can be viewed as a function that takes the generative model, a set of observational constraints, and a limited amount of computational resources, and returns a trace that best explains the data.

To support this process, Gen offers powerful abstractions for developing custom inference algorithms. Traces can be inspected, modified, and scored, with automatic log-density computations. For any given trace, whether sampled through generative program execution or manually provided, Gen computes the log-density of the trace automatically by summing the contributions from individual random choices. This capability is particularly useful when working with externally provided traces, such as those returned by the SMT solver. For each random choice, the log-density of its value is evaluated under the specified prior distribution.

Traces can be sampled using the 'generate' method, which executes a generative function and returns the resulting trace. Additionally, Gen provides a constraint-based variant of 'generate' to produce traces that are consistent with a given set of constraints, such as observations.

A powerful strategy in Gen involves writing proposals using probabilistic programs that incorporate posterior knowledge via heuristic algorithms to target potential high-posterior regions [Cusumano-Towner and Mansinghka, 2018]. These proposals can be integrated into Metropolis-Hastings updates, as discussed in the following subsection. One example of such a heuristic, used in the line-fitting model, is Random Sample Consensus (RANSAC) [Fischler and Bolles, 1981], which identifies a promising latent configuration from a small random subset of observation points. While effective, heuristic algorithms are typically handcrafted and model-specific. This work builds on this concept and investigates whether

SMT-based solutions can serve as an effective automated heuristic to guide MH inference. Importantly, such proposal mechanisms preserve the asymptotic guarantees of the inference process.

2.1.3 Metropolis-Hastings

Metropolis-Hastings (MH) [Hastings, 1970] is a fundamental algorithm within the MCMC [Robert et al., 1999] family, widely used to sample from an unknown target distribution. The core idea is to construct a Markov chain whose stationary distribution is the desired posterior. The algorithm proceeds in two key steps: a proposal step, where a candidate state is sampled from a proposal distribution conditioned on the current state, and an acceptance step, where the candidate is either accepted or rejected, in which case the chain remains at the current state. The acceptance criteria is computed as the ratio of the target density and proposal probabilities. This reflects both how favorable the proposed state is under the target distribution and how likely the transition is compared to its reverse.

In the context of probabilistic programming, a state corresponds to an execution trace. MH operates over the space of traces, proposing modifications and accepting them according to their consistency with the model and observations. The steps of the algorithm are outlined in Box 2.1.3.

The proposal mechanism can alter any part of the execution trace. A standard method is single-component Metropolis [Metropolis et al., 1953], which updates only one latent variable at a time in sequence. Random Walk Metropolis, on the other hand, utilizes a Gaussian distribution centered at the current trace to propose local moves. The proposal distribution plays a crucial role in the effectiveness of MH in exploring the target distribution. A substantial body of work has focused on customizing the proposals to improve convergence, often by incorporating prior, gradient, or trace history knowledge. Other proposal strategies are discussed in Chapter 3.

By repeatedly applying the two main steps, the algorithm converges in the limit to the target distribution. A crucial factor for this guarantee is the design of the proposal distribution and satisfaction detailed balance (DB) in the acceptance step. Detailed balance requires that, for any two states, the probability of transitioning from one to the other is equal to the probability of the reverse transition, ensuring the process models the target distribution. While DB is sufficient to ensure convergence to the stationary distribution, it is not necessary, opening the door to more advanced, non-reversible samplers that relax this constraint.

Algorithm: Metropolis-Hastings Sampling

1. **Initialization:** Start with an initial arbitrary value x_0.
2. **Propose a Candidate:** Propose a new sample x_{proposed} from a proposal distribution $Q(x_{\text{current}} \to x_{\text{proposed}})$.
3. **Compute the Acceptance Probability:** Calculate the acceptance probability α as:

$$\alpha = \min\left(1, \frac{P(x_{\text{proposed}})Q(x_{\text{proposed}} \to x_{\text{current}})}{P(x_{\text{current}})Q(x_{\text{current}} \to x_{\text{proposed}})}\right).$$

   If the proposal distribution Q is symmetric, e.g., $Q(x_a \to x_b) = Q(x_b \to x_a)$, this simplifies to:

$$\alpha = \min\left(1, \frac{P(x_{\text{proposed}})}{P(x_{\text{current}})}\right).$$

4. **Accept or Reject:**
   - Accept x_{proposed} with probability α, setting $x_{t+1} = x_{\text{proposed}}$.
   - Otherwise, reject and set $x_{t+1} = x_{\text{current}}$.
5. **Iterate:** Repeat steps 2–4 for a fixed number of iterations or until convergence.

The simplicity of the MH procedure renders it widely applicable for sampling from a broad range of target distributions. However, its efficiency deteriorates dramatically as the dimensionality and structural complexity of the target distribution increases. Importantly, each proposed candidate must not only be plausible under the model but also consistent with the observed data, or it risks immediate rejection. Additionally, a change to a latent variable that alters the program's control flow requires resampling of execution trace portions to maintain consistency. These factors further slow down inference and form part of the core motivation of our work.

Metropolis-Hastings in Gen Gen provides three method abstractions for invoking Metropolis-Hastings updates, presented in Listing 2.2. Each Metropolis-Hastings function takes the current trace along the required input for the update strategy, and returns a new trace along with a boolean indicating whether the proposal was accepted. Additionally, observational constraints can be provided to ensure the proposed traces remain consistent with the observed data. The first variant resamples a user-specified selection of random variables using ancestral sampling. The selection may include one or more latent variables, for example, proposing new values to one or all of the mean components in a Gaussian Mixture model. The second leverages a custom generative proposal to sample new values, allowing model-specific tailored proposal [Cusumano-Towner and Mansinghka, 2018]. This allows users to incorporate posterior domain knowledge to craft more effective proposals. For instance, one may implement a Gaussian drift proposal that perturbs the mean components with additive noise. Listing 2.3 exemplifies this for the first mean component. The third variant uses a more general involutive transformation, enabling generalized and reversible-jump style proposals [Cusumano-Towner et al., 2020].

```
# 1. MH update for selected addresses
(new_trace, accepted) = metropolis_hastings(
    trace, selection::Selection;
    check=false, observations=EmptyChoiceMap())

# 2. MH update with a generative proposal function
(new_trace, accepted) = metropolis_hastings(
    trace, proposal::GenerativeFunction, proposal_args::
   Tuple;
    check=false, observations=EmptyChoiceMap())

# 3. MH updated based on an involution
(new_trace, accepted) = metropolis_hastings(
    trace, proposal::GenerativeFunction, proposal_args::
   Tuple,
    involution::Union{TraceTransformDSLProgram,Function};
    check=false, observations=EmptyChoiceMap())
```

Listing 2.2: Three variants of `metropolis_hastings` in Gen.jl

```
@gen function drift_proposal(current_trace)
    mean1 ~ normal(trace[:current_trace], 0.1)
end
```

Listing 2.3: Gaussian drift proposal over a selected GMM component mean.

The score computation and acceptance decision are handled internally by Gen. This allows users to focus on specifying proposals without manually managing the acceptance ratios and trace scoring.

2.2 SMT solving

Satisfiability Modulo Theories (SMT) refers to the problem of determining whether a formula expressed in first-order logic is satisfiable with respect to a given background theory [Barrett et al., 2018]. It generalizes the classical Boolean Satisfiability Problem (SAT) by incorporating additional theories such as reals, integers, arrays, and quantifiers. SMT has become a foundational tool across various areas of computer science, where many problems can be reduced to checking the satisfiability of logical formulas under specific theories. Its power lies in combining propositional reasoning with theory-specific decision procedures, enabling the analysis of rich, structured constraints.

Z3 [De Moura and Bjørner, 2008] is a state-of-the-art SMT (Satisfiability Modulo Theories) solver developed by Microsoft Research. It provides a powerful framework for modeling and solving formulas over arithmetic and logical constraints. Z3 supports a wide range of theories relevant to probabilistic modeling, including linear and nonlinear arithmetic, uninterpreted functions, arrays, and more.

At its core, Z3 is based on the CDCL(T) architecture, short for Conflict-Driven Clause Learning modulo Theories. This architecture extends classical SAT-style conflict learning with specialized solvers for individual theories, enabling Z3 to reason efficiently over both Boolean structure and arithmetic constraints. In practice, Z3 maintains a suite of core solving engines optimized for different problem types, and automatically invokes alternative engines when they are better suited than the default. For nonlinear arithmetic, Z3 employs a waterfall-based strategy to decompose and simplify constraints before applying specialized solving tactics. These theory-aware strategies allow Z3 to dynamically adapt its solving approach, enabling robust hybrid reasoning. By balancing completeness and performance, Z3 offers a scalable and expressive framework. Its broad theory support makes it particularly promising for constraint-based reasoning in probabilistic program inference.

For the purpose of this thesis, non-linear arithmetic and transcendental functions are essential, as they appear in likelihood formulations for various distributions. However, both non-linear integer arithmetic (NIA) and transcendental-augmented non-linear real arithmetic (NTA) are known to be undecidable [Matiyasevich, 1993, Richardson, 1968]. Current implementations in Z3 support these theories, but without completeness guarantees. As a result, when dealing with non-linear constraints, the solver may return unknown or loop indefinitely. Other solvers that support nonlinear theories include CVC5 [Kremer et al., 2022], MathSAT5 [Cimatti et al., 2013], and Yices2 [Hader et al., 2024]. However, Z3 was chosen for its extensive implementation support, particularly its well-developed API.

Z3 relies on several methods for dealing with non-linear constraints, including Cylindrical Algebraic Decomposition (CAD) [Collins, 1974] and Incremental linearization [Cimatti et al., 2018]. CAD [Collins, 1974] addresses multivariate polynomial inequalities by projecting and decomposing the solution space. Incremental linearization handles non-linear and transcendental functions by pro-

gressively bounding them via piecewise-linear approximations. These methods are complemented by additional techniques such as Gröbner basis simplification and bounds propagation [Bjørner and Nachmanson, 2024], all of which contribute to Z3's ability to reason over non-linear arithmetic search spaces.

Z3Py, the Z3 API in Python, is used in this work to enable native Python integration, supporting modular class structures and streamlined constraint generation. To circumvent non-linear optimization, which is not natively supported in Z3, a custom model optimization loop is implemented in Python to iteratively identify satisfiable models.

Chapter 3

Related work

This chapter discusses related work, examining various improvements to the base Metropolis-Hastings algorithm and additions to Markov-Chain Monte Carlo methods. To date, no published work has investigated the use of constraint programming to guide inference.

Improving mixing and target distribution convergence A crucial factor influencing mixing and convergence in MCMC methods algorithms is the design of the proposal distribution [Gelman* et al., 1996] [Tierney, 1998]. The transition kernel influences how the target distribution is explored. Many variants have been proposed to improve effectiveness through more informed or structured proposals.

An early variant is the Random Walk Metropolis, where all components are updated simultaneously using a symmetric noise proposal, typically Gaussian. Similar to the original Metropolis Algorithm, it suffered in high-dimensional spaces.

To address the sensitivity of MH algorithms to fixed proposal distributions, a line of work focuses on adaptive MCMC methods, where the proposal distribution is altered during the sampling process. The Adaptive Proposal (AP) algorithm [Haario et al., 1999] uses Gaussian proposals centered at the current state, with a covariance estimated from a fixed-size recent sample history window. In contrast, the Adaptive Metropolis (AM) algorithm [Haario et al., 2001] utilizes the full trace history to update the Gaussian proposal. To do so, AM monitors acceptance rates and adapts accordingly. Both methods maintain ergodicity while adaptively tuning the proposal kernel for more effective exploration. More generally, adaptive MCMC [Gilks et al., 1998] formalizes the idea by introducing auxiliary chains that guide the adaptation, typically using regeneration points where previous states are forgotten. These methods aim to reduce manual tuning and improve target distribution exploration.

Gradient-based MCMC methods, such as Metropolis-adjusted Langevin algorithms (MALA) [Roberts et al., 1996] and Hamiltonian Monte Carlo (HMC) [Neal and Neal, 2011], leverage local gradient information to construct informed proposals that improve convergence in smooth, continuous spaces. However, their reliance on random sampling limits their applicability in settings with discrete variables.

It has been shown that non-reversible MCMC algorithms can lead to improved mixing by removing backtracking specific to reversible-jumps. These algorithms do not satisfy detailed balance but still satisfy stationarity. Lifted Metropolis-Hastings [Turitsyn et al., 2011] augments the state space with a discrete direction variable, enabling non-reversible transitions that reduce random-walk behavior and improve mixing. By preserving global balance while breaking detailed balance, this approach achieves more persistent exploration with minimal modification to standard MH.

Recent advances in MCMC have explored the use of determinism in the sampling process, particularly through non-reversible dynamics where states evolve according to deterministic flows. This idea follows the core process of Piecewise-Deterministic Markov Processes (PDMPs) [Davis, 1984], where smooth deterministic motions are combined with stochastic jumps. Piecewise-Deterministic MCMC (PD-MCMC) leverages PDMPs to construct samplers that sample from a target distribution. Notable algorithms in this set domain include Bouncy Particle Sampler [Bouchard-Côté et al., 2017], the Zig-Zag sampler [Bierkens et al., 2019], and Coordinate Sampler [Wu and Robert, 2019], later extended to incorporate Hamiltonian dynamics [Vanetti et al., 2018]. Although these samplers are often described as rejection-free, this only relates to the use of proposals without an accept-reject step, meaning that steps do not always improve sample quality, but effectively explore and converge to the target distribution. The sampler moves through the search space, mimicking physical dynamics, still entering low-probability regions.

Another line of recent research focuses on crafting effective proposals by rewriting models and analyzing the program structure before inference [Nori et al., 2014]. This work prevents rejections caused by executions that would otherwise violate observed data, leading to a modified Metropolis-Hastings that samples only from feasible regions of latent space, through truncated proposals.

All of the above methods rely on stochastic proposals, which, even when carefully designed, may still result in rejections due to unsatisfied observations or low target density. The approach presented here seeks to find a structured and constraint-aware way to propose better candidates. Similar in spirit to R2 [Nori et al., 2014], the goal is to eliminate rejections caused by unsatisfied constraints.

Inference automation

As probabilistic programming systems mature, a growing body of works aims to reduce the burden of manual inference design by automating the construction of proposals. Several works provide automation for tailoring base inference algorithms to be model specific. For example, automatic guide generation in variational inference [Ritchie et al., 2016], implemented in Pyro [Bingham et al., 2019]. Another technique uses trained neural networks as proposal distributions in sequential importance sampling inference [Le et al., 2017].

While inference automation focuses on reducing manual effort in designing proposal distributions, another important line of work addresses the structural complexity of probabilistic programs with stochastic support. In such programs, traces may involve a varying number of random variables. This requires transitioning

between different state spaces during sampling. Two main approaches address this challenge.

The first approach relies on a carefully constructed proposal mechanism that accounts for dimensionality-changing transitions through appropriate volume corrections to ensure valid acceptance probability. The involutive MCMC framework in GEN automates much of this process [Cusumano-Towner et al., 2020], provided the user supplies a valid involution. This framework generalizes classical Reversible Jump MCMC (RJ MCMC) [Green, 1995] [Neklyudov et al., 2020]. The first incorporates careful proposal designs and volume transformations between the trace transitions in order to compute correct acceptance and preserve detailed balance. Gen involutive framework provides automated computations provided the user can provide an involution [Cusumano-Towner et al., 2020]. The involutive framework covers the reversible jump MCMC proposals [Green, 1995], and general involutions expanded in [Neklyudov et al., 2020].

An alternative strategy, Divide Conquer and Combine (DCC) [Zhou et al., 2020], avoids inference over a variable program structures by decomposing the program into straight-line subprograms. Inference is then performed within each subprogram, using conventional inference strategies, exploiting the fixed number of variables.

Chapter 4

Approach

A primary objective when employing Metropolis-Hastings (MH) methods is identifying traces with higher posterior probability. At its core, the joint work of random sampling and the acceptance step is responsible for steering the inference towards traces that are more probable given the model, those that explain the observed data. In this context, inference involves a search over latent variables, whose values are not directly observed but are constrained by the observations. These observations limit the space of plausible traces by reducing the range of probable latent variable configurations. Beyond satisfying observational constraints, this search must also prioritize traces with higher likelihood, favoring latent configurations that not only align with the data but are also more probable under the model.

This suggests a possible reformulation of the high-likelihood trace identification as a constraint optimization problem. Here, the decision variables take the place of the latent variables, the observations act as constraints, while the objective of improvement is the trace probability, equal to the joint likelihood of both observed and inferred values.

Thus, the main contribution of this paper is a framework that reframes the probabilistic inference task as an optimization problem and leverages its solutions to guide the MH sampling process. The framework consists of two main steps: (1) constraint model construction and optimization of trace assignments to maximize likelihood, and (2) integration of these solutions within the MH inference. The goal is to eliminate reliance on handcrafted proposals by exploiting SMT solving to directly identify high-probability traces and enhance convergence. The remainder of this chapter expands on the motivations behind the reformulation and provides details on the translation process as well as the integration of solver outputs into the MH sampling procedure.

4.1 Removing the blindfold in search

To illustrate the inference process in a standard Metropolis-Hastings (MH) approach, consider the following Gaussian Mixture model with K clusters as a guid-

ing example.

$$\mu_k \sim \text{Uniform}\left(\frac{20(k-1)}{K}, \frac{20k}{K}\right)$$
$$z_n \sim \text{Categorical}\left(\left\{\frac{1}{K}, \ldots, \frac{1}{K}\right\}\right)$$
$$y_n \sim \text{Normal}(\mu_{z_n}, 0.1)$$

In this model, the variables $\mu_{1:K}$ represent the cluster centers, while $z_{1:N}$ denote the cluster assignments for each observation. The observed data is given by $y_{1:N}$. The generative process assumes that each data point y_i is generated around one of the K cluster means. In this setup, the cluster centers $\mu_{1:K}$ and the assignment variables $z_{1:N}$ constitute the latent variables, while the observed variables are $y_{1:N}$. Inference in the standard Metropolis-Hastings (MH) algorithm proceeds by sampling modifications to individual latent variables, such as proposing new cluster assignments z_i or applying perturbations to cluster means μ_k. Each proposal results in a new configuration of latent variables, which is evaluated by computing the probability of the corresponding trace. A trace τ refers to a complete instantiation of all latent variables in the model, and its probability is given by the joint distribution over both latent and observed variables. Specifically, in the case of the Gaussian mixture model, this joint probability takes the form:

$$P(\tau) = P(\mu_{1:K}) \cdot P(z_{1:N}) \cdot P(y_{1:N} \mid z_{1:N}, \mu_{1:K}).$$

MH explores the space of such traces by accepting or rejecting proposed modifications based on their relative posterior probabilities, using the standard acceptance criterion. In this stochastic search process, the goal is to identify high-probability traces, those that explain the observations well under the generative model.

A subtle complexity arises from the structure of the model itself. The line $y_n \sim \text{Normal}(\mu_{z_n}, 0.1)$ implicitly defines a branching structure over the latent variable `zi`: depending on its value, the model selects a corresponding Gaussian component. Operationally, this is analogous to an explicit case distinction, where the cluster assignment `zi` conditions on a different cluster mean of origin μ_k (here noted as $mean_k$) for the observed data point, such as presented in Listing 4.1.

This structure creates conditional dependencies in the trace that are highly sensitive to discrete choices. Part of the difficulty in identifying effective trace moves lies in the model composition, particularly those involving a mixture of continuous and discrete random choices. Discrete-valued variables can introduce sharp discontinuities in the probability landscape; small changes in such variables may lead to dramatic shifts in the overall trace probability. This sensitivity makes naive or unguided, random proposals especially fragile in such settings, often leading to inefficient exploration of the posterior space.

```
    if zi == 1
        yi ~ normal(mean1, 0.1)
    elseif z1 == 2
        yi ~ normal(mean2, 0.1)
    else
        yi ~ normal(mean3, 0.1)
    end
```

Listing 4.1: Implicit branching structure defined by the cluster mean selection μ_{z_n} in $y_n \sim \text{Normal}(\mu_{z_n}, 0.1)$

Constraint programming, however, is well-suited for navigating search spaces dominated by discrete decisions and reasoning about constraints, such as observations. Its ability to prune infeasible domains and focus the search on viable regions makes it a compelling tool for guiding inference in such hybrid probabilistic models.

In the remainder of this chapter, a framework that leverages constraint programming to address these challenges is presented.

4.2 Abstract framework

This section introduces the framework's core components at a conceptual level. The approach is structured around two main components: (1) a translation procedure that maps a probabilistic program and its observations into a constraint model, and (2) an integration mechanism that incorporates solver output into the Metropolis-Hastings inference loop.

4.2.1 From probabilistic formulation to constraint formulation

This section outlines how the core concepts of probabilistic programming: latent samples, observed data, and likelihoods, are reinterpreted in the context of constraint optimization. Intuitively, latent and observed variables must be translated into decision variables and constraints, while likelihoods must inform what constitutes a good solution. The following subsections detail how these notions transfer from the probabilistic domain into a form that can be handled by constraint solvers.

Latent variables For each latent variable, such as z_i or μ_k, a corresponding decision variable is introduced into the constraint model. The type of decision variable, discrete or continuous, is determined by the distribution from which the original variable is drawn. For example, categorical variables such as z_i give rise to integer decision variables, while continuous distributions such as uniform yield real-valued decision variables.

One crucial aspect that must be made explicit in the constraint formulation is the domain support of each variable. While this is implicitly enforced by the sampling distribution in the probabilistic program, it must be explicitly encoded

in the constraint model. For example, in the case of a Gaussian Mixture Model with $K = 3$ clusters, the variables z_1, z_2, z_3 must be constrained to take values in the set $\{1, 2, 3\}$, while the means μ_k must lie within the intervals defined by their Uniform priors—e.g., $\mu_1 \in [0, \frac{20}{3}]$, $\mu_2 \in [\frac{20}{3}, \frac{40}{3}]$, and so on. While some distributions, such as Categorical or Uniform, naturally induce finite support that can be directly translated into hard domain constraints, others, such as the commonly encountered Normal distribution, have infinite support. In such cases, it is still possible to introduce soft bounds that guide the search toward more probable regions of the space. For instance, in the case of a normal distribution, values closer to the mean are typically more likely than those in the tails, and constraints can be formulated to reflect this preference. These strategies are discussed in more detail in Section 4.4.2.

Observed random choices In the constraint model, observations are incorporated as fixed values that impose constraints on the latent variables. Each observed random choice is transformed into a corresponding decision variable. A constraint is then added to enforce equality between this variable and the observed value. For example, each observed value v_{y_i} of a variable y_i is translated into a decision variable with an associated constraint $y_i = v_{y_i}$, that fixes the observed values. This process is illustrated in Table 4.1.

While the probabilistic program treats observations as conditioning events during inference, the constraint formulation encodes them as hard constraints that must be satisfied by any valid solution. For each observed variable y_i, its value restricts the feasible assignments of the corresponding latent variables: the cluster assignment z_i and the relevant mean μ_{z_i}. The latent variable configurations, thus, must reflect those that are consistent with the generative relationship.

Observation (v_{y_i})	**Decision Variable** (y_i)	**Constraint** ($y_i = v_{y_i}$)
3.5	y_1	$y_1 = 3.5$
-1.2	y_2	$y_2 = -1.2$
0.0	y_3	$y_3 = 0.0$

Table 4.1: Mapping from observed values to decision variables and constraints.

The nature of this restriction can vary. In some cases, the observation may directly reduce the domain of latent variables. For example, in a discrete model where $y_i \sim$ Categorical(v_{z_i}), and let $v_1 = [0.8, 0.2]$ and $v_2 = [1, 0]$. Observing $y_i = 2$ may immediately eliminate all z_i assignments for which the second outcome has zero probability, such as $z_i = 2$, which yields v_2. In other cases, the observation does not rule out configurations entirely, but instead influences the search toward more plausible regions of the latent space through the likelihood structure; for instance, by preferring latent values that would result in higher likelihoods under a continuous distribution. This dual role of observations, as both hard filters and soft guides, is critical in shaping the feasible and high-probability regions of the

constraint model.

Variable naming So far, the discussion has not addressed the naming and identification of variables in the constraint formulation. Each latent and observed variable in the probabilistic program corresponds to a decision variable in the constraint model. To preserve the semantic correspondence between the two representations, these decision variables inherit their names from the original probabilistic program.

In practice, Gen's trace-based addressing mechanism provides a convenient way to assign unique identifiers to random choices, making this mapping straightforward in most cases. However, special care is required when dealing with random choices that share the same name but occur in distinct execution paths, such as the case of branching structures. In such cases, multiple decision variables are introduced, each associated with a unique path-specific identifier. These are linked to their respective likelihood computations via their path conditions, ensuring that only the relevant choices contribute to the trace probability under a given execution path.

Consider again the implicit branching in the Gaussian mixture model (Listing 4.1), where each observation y_i is sampled from a distribution that depends on the cluster assignment z_i. To represent this structure in the constraint model, multiple decision variables are introduced: y_{i_1}, y_{i_2}, y_{i_3}, each corresponding to the distribution associated with a specific cluster. The model also includes path conditions: the boolean constraints reflecting the control flow of the original program. These conditions determine which branch-specific decision variable is active. For instance, when the path condition $z_i = 2$ holds, only $y_{i,2}$ contributes to the objective function.

In this way, the constraint model explicitly captures the branching semantics of the probabilistic program by unfolding conditional dependencies into parallel decision structures, each gated by its corresponding path condition. This design ensures that the likelihood contributions remain consistent with the original execution flow, while enabling the solver to reason over multiple potential execution paths within a unified constraint framework. Further details on how multiple execution paths are handled can be found in Section 4.4.3.

The constraint formulation of a GMM with one observation is schematically illustrated in Box 4.2.1.

Constraint model for a GMM with $K = 3$ Clusters and one observation

Given: One observation $y_1 = 5.3$

Decision Variables:

- $z_1 \in \{1, 2, 3\}$ (discrete cluster assignment)
- $\mu_1 \in \left[0, \frac{20}{3}\right]$
- $\mu_2 \in \left[\frac{20}{3}, \frac{40}{3}\right]$
- $\mu_3 \in \left[\frac{40}{3}, 20\right]$
- y_1

Constraints:

- $y_1 = 7.2$ (observation fixed by hard constraint)
- Path conditions: For each possible value of z_1, define a branch:
  - If $z_1 = 1$: enforce $y_1 \sim \mathcal{N}(\mu_1, 0.1)$
  - If $z_1 = 2$: enforce $y_1 \sim \mathcal{N}(\mu_2, 0.1)$
  - If $z_1 = 3$: enforce $y_1 \sim \mathcal{N}(\mu_3, 0.1)$

Trace probability The final component needed to complete the optimization formulation is the objective function. In this context, the objective corresponds to the trace probability, more precisely, the log-probability of a complete trace under the original probabilistic model. A direct translation of this quantity would require summing the log-likelihoods of all selected random choices, including both latent and observed variables. However, many solvers struggle to handle non-linear and transcendental functions, such as logarithms or exponential functions. Probability density functions typically contain multiple compositions of such functions. In practice, incorporating these expressions directly into the constraint model renders the optimization problem intractable with current non-linear solving engines. To address this, the approach taken in this work is to approximate the objective by partially linearizing the contribution of certain distributions. By reformulating the likelihoods in a piecewise-linear or truncated linear form, the optimization process can leverage linear or mixed-integer programming techniques while still favoring higher-probability traces. This trade-off enables tractable inference while maintaining alignment with the original probabilistic semantics. The trace probability encoding is further detailed in Section 4.5.

4.2.2 Integrating solver output into Metropolis-Hastings

Once the constraint model has been constructed, its solution can be used to inform and enhance the proposal mechanism within the Metropolis-Hastings (MH) loop. Rather than replacing the MH procedure entirely, the optimization-based solution serves as a heuristic proposal, guiding the sampler toward more promising regions of the trace search space.

It is important to note that optimization alone cannot fulfill the task of MH inference. While the solvers are designed to identify high-scoring trace configurations, the goal of MH is to explore the posterior distribution, rendering a probability over traces, not solely revealing the most probable one. As such, the integration must preserve the stochastic nature of the MH algorithm, using the solver to accelerate convergence without compromising the diversity of sampled traces. This integration pertains to the asymptotic guarantees of the original MH procedure, provided the acceptance step of the solver's proposed trace is handled properly.

To this end, two principal modes of integration are identified: *warm-start* and *warm-jump*. In the **warm-start** scenario, the solver solution is used to initialize the MH trace, effectively starting the sampler in a high-probability region of the posterior. In the **warm-jump** setting, the solver is called during inference to propose moves that side-step standard proposals to enable the sampler to escape stagnation. In this latter case, strategies for ensuring solution diversity, expanded upon in 4.6.2, need to be added to the solver in order to vary the possible candidate solutions. These solver-based jumps are evaluated using a custom acceptance criteria, ensuring only meaningful jumps are accepted, while improving proposal quality.

Both strategies aim to reduce the inefficiencies of traditional MH, namely, slow mixing and poor handling of discrete structures, by injecting observation-aware solutions. This approach also provides an automated alternative to quality heuristic proposals, reducing reliance on manually crafted strategies by a domain expert. The framework is designed to ease adoption regardless of the probabilistic model.

4.3 Framework

This section expands on the framework introduced earlier, detailing its components and how they interact within the system. While the described system presents one concrete realization of the framework, the broader contribution lies in integrating constrained reasoning into probabilistic inference.

4.3.1 Framework pipeline

The general SMT-guided inference framework is illustrated in Figure 4.1. This subsection details the core components and their interactions.

The process begins within the inference procedure when an SMT-based trace injection is triggered, either at initialization as a warm-start or during inference as a warm-jump. The inference engine passes the generative function to a parser module, which extracts structural and relational properties of the probabilistic model.

The parser produces a model specification containing information about the random samples, specifically their prior distributions and their path conditions. The extraction procedure is detailed in Subsection4.3.2. Combined with the observed data, this specification forms the basis for building the probabilistic constraint model.

Collecting observational data is straightforward. Compared to other probabilistic programming languages, Gen specifies observations as constraints rather than modeling the observation in the generative model. This distinction simplifies the extraction of observations: observed values can be directly mapped to variable names and passed along to the constraint model as known quantities.

In the warm-jump setting, additional inputs such as diversity constraints and the current trace are required to guide the solver towards state-aware, diverse solutions. These are passed to the Python module, along with specifications and observations. The influence of these additional inputs on the resulting trace configurations is discussed in Subsection4.6.2.

Within the Python module, the provided input is used to construct a constraint representation of the probabilistic program. The resulting model is then solved to yield a satisfiable latent variable assignment, which is further refined through objective-driven optimization. The construction of the constraint model, including decision variable definitions and associated constraints, is presented in Section 4.4, while the objective formulation is specified in Section 4.5.

Since Z3 does not natively support non-linear optimization, trace optimization is achieved through a manual refinement loop that progressively increases the lower bound and checks the satisfiability of the resulting model. Each iteration proposes several candidate improvements, and the total number of iterations is configurable.

The final trace solution is returned to the inference procedure, either to initialize the MH sampler or to propose an alternative trace mid-inference. The integration of the SMT-based traces in the inference procedure is detailed in Section4.6.

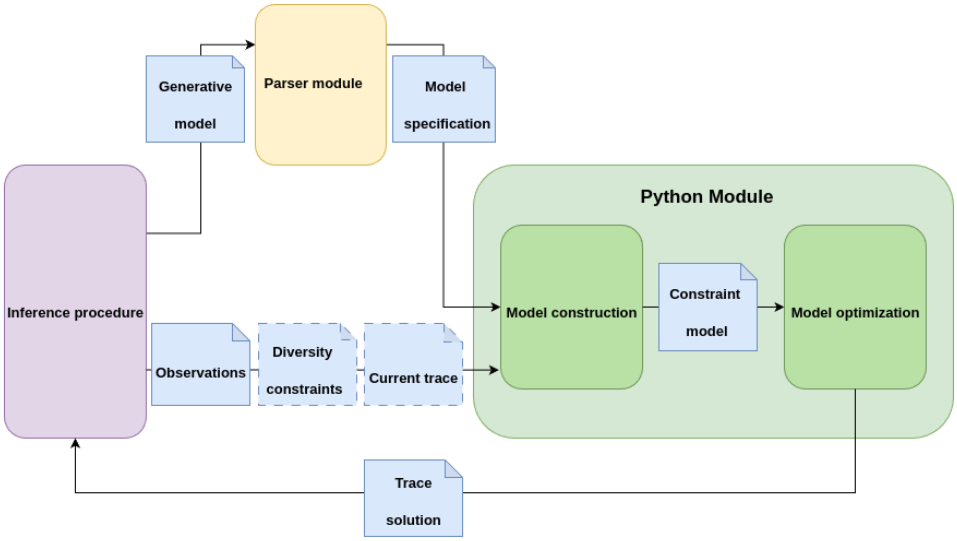


Figure 4.1: Solver-guided inference pipeline.

4.3.2 Probabilistic Program Parser

The parser module captures the relational structure of the probabilistic program by extracting sample statements from the generative function. This section proceeds by describing the parsing process and the construction of the model specification based on the recovered elements.

Julia provides built-in support for inspecting the expression parse tree, facilitating model traversal. The parse tree is traversed to extract sample statements and deterministic assignments. For each sample statement, the parser records the variable name, the distribution and its parameters, as well as the path condition. Table 4.2 presents the extracted information for each sample statement in the GMM example with one observation. This information is then passed to the Python module for constraint model construction.

Variable	Distribution	Parameters	Path Condition
`mean1`	Uniform	$0,\ \frac{20}{3}$	True
`mean2`	Uniform	$\frac{20}{3},\ \frac{40}{3}$	True
`mean3`	Uniform	$\frac{40}{3},\ 20$	True
`z1`	Categorical	$[1/3, 1/3, 1/3]$	True
`y1`	Unkown	-	-
$\texttt{y1}_1$	Normal	μ_1, σ	`z1` == 1
$\texttt{y1}_2$	Normal	μ_2, σ	`z1` != 1 && `z1` == 2
$\texttt{y1}_3$	Normal	μ_3, σ	`z1` != 1 && `z1` != 2

Table 4.2: Latent and observed variable specifications extracted by the parser for the GMM problem with one observation y_1.

To avoid naming conflicts, repeated sample statements with the same variable name, typically originating from different execution paths, are automatically renamed with unique identifiers (e.g., $\texttt{y1}_1$, $\texttt{y1}_2$, as shown in Table 4.2). To indicate that these variables correspond to alternatives for the same address, the original name is retained in the specification with an 'Unknown' distribution type. These are later aggregated and handled within the Python model (see Section 4.4.3 for more details).

The path condition encodes the boolean constraints that must hold for the statement to be reachable in the program. For variables outside any branching structure, the corresponding path condition is set to `true`. To clarify the formulation, consider again the GMM example. As just mentioned, variables shared across all execution paths, such as z_i and the cluster means μ_1, μ_2, μ_3, are always present in the trace, reflected by the True value in their path condition. Path condition becomes relevant in the presence of branching structures, as is the case of the $y1$ alternatives determined by the choice of cluster assignment $z1$. During parsing, the context of each sample statement is preserved, capturing the logical conditions required to reach it; these conditions define the corresponding path condition.

The same procedure applies to non-stochastic assignments, allowing the use of intermediate variables and providing support for wider program expressivity. For example, instead of hardcoding a noise value into a distribution, one might write noise = 0.1 and later use Normal(μ, noise). Similarly, logical compositions such as alarm = earthquake `||` burglar are captured to preserve the program's internal dependencies. For such assignments, the variable name, assigned expression, and path condition are maintained.

Stochastic support Stochastic support is not addressed in the current framework, in order to maintain the focus on integrating constraint solving into probabilistic inference. Omitting this feature permits streamlining probabilistic structure extraction and constraint model construction. Traditional constraint programming frameworks are not inherently designed to handle an unknown number of variables at runtime. As such, this feature cannot be encoded directly within the standard constraint model. Supporting dynamic, data-dependent iteration would require alternative design choices and more advanced symbolic reasoning capabilities. This exploration is left for future work.

Assuming a fixed and known depth, loops and arrays are manually unrolled prior to parsing. Rather than maintaining array-based representations, values are extrapolated into individual decision variables. For example, array accesses such as `z[n]` and `y[n]` are unrolled into collections of variables, represented as $\{z_i, y_i \mid i = 1, \ldots, n\}$.

This naive unrolling strategy avoids the complexity of symbolic loop handling. Instead, the repeated pattern is expanded before parsing, preserving the semantics of the original program. While this approach is limited to cases where loop bounds are statically resolvable, it provides a straightforward and tractable encoding for many practical examples, including the GMM setting discussed earlier. The GMM model with unrolled loops and arrays is presented in Listing 4.2.

```
1
2  mean1 = {:mean1} ~ uniform(0, 20/3)
3  mean2 = {:mean2} ~ uniform(20/3, 40/3)
4  mean3 = {:mean3} ~ uniform(40/3, 20)
5
6  z1 = {:z1} ~ categorical([1/3, 1/3, 1/3])
7
8  if z1 == 1
9      y1 = {:y1} ~ normal(mean1, 0.1)
10 elseif z1 == 2
11     y1 = {:y1} ~ normal(mean2, 0.1)
12 else
13     y1 = {:y1} ~ normal(mean3, 0.1)
14 end
```

Listing 4.2: Unrolled generative functing for the Gaussian-Mixture model

4.4 Constraint-based representation of the probabilistic model

This section outlines how the constraint model is constructed based on the model specification provided by the parser module and the observations. The model defines both latent and observed variables from the specification as decision variables. Each receives bounding constraints derived from its prior distribution support. Additionally, model consistency constraints are introduced to ensure the resulting solutions form valid latent configurations. Both bounding and consistency constraints are defined in Subsection4.4.2. Further, constraints that model control-flow are described in Subsection4.4.3

4.4.1 Random choices as decision variables

Before a random choice can be defined as a decision variable, its type must first be determined. This is trivially achieved by mapping the distribution name to value type: *int* for discrete distributions, and *real* for continuous ones. The full distribution to type correspondence is shown in Table 4.3.

Distributions	**Type**
Normal, Beta, Cauchy, Exponential, Gamma, Laplace, Multivariate Normal, Uniform	real
Bernoulli, Binomial, Categorical, Geometric, Poisson, Uniform Discrete	int

Table 4.3: Grouped distribution types by variable type.

Observations can then be incorporated into the model through constraints that fix the observed variable to its corresponding value. Typically, this is done by directly constraining the variable by name. However, in models with conditional structure, such as the GMM example, additional control-flow constraints must ensure that one of the alternatives ($y1_1$, $y1_2$, or $y1_3$) is equal to this observation. More on this in Subsection 4.4.3.

The next step is to infer the appropriate sample bounds from the prior distributions and their parameters. For the uniformly distributed means, the lower and upper bounds are directly determined by the distribution's support. Categorical variables, such as the cluster assignments $z1$, are constrained through equality conditions. In contrast, normally distributed variables like $y1$ have unbounded support, and no direct constraints can be inferred.

To make this concrete, consider the Gaussian Mixture Model (GMM) example. The decision variable types and constraints, which are directly inferred from the model specification, are shown in Table 4.4.

Variable	Type	Bounding Constraints	Observation
mean1	real	$\texttt{mean1} \geq 0$ && $\texttt{mean1} \leq \frac{20}{3}$	—
mean2	real	$\texttt{mean2} \geq \frac{20}{3}$ && $\texttt{mean2} \leq \frac{40}{3}$	—
mean3	real	$\texttt{mean3} \geq \frac{40}{3}$ && $\texttt{mean3} \leq 20$	—
z1	int	$\texttt{z1} == 1 \mid\mid \texttt{z1} == 2 \mid\mid \texttt{z1} == 3$	—
$y1$	real	—	$y1 = 7.2$
$y1_1$	real	—	—
$y1_2$	real	—	—
$y1_3$	real	—	—

Table 4.4: Constraint model variable definitions and related constraints for a GMM with three mixture components.

4.4.2 Constraints for distributions support

Support constraints directly shape the solver's search space of latent variables. Adjusting or approximating unbounded supports can significantly influence the solution's feasibility. This subsection explores how such bounds are applied to both latent and observed decision variables and how they can be leveraged to guide the search more effectively.

Bounding latent variables Bounding latent variables ensures the search produces valid latent configurations in line with the distribution support, as well as reducing the search space. For distributions with limited support, such as the uniform distribution, this task is straightforward. For distributions with support over the real domain, we found it practical to approximate them by truncating the range of possible values. This not only reduces the search space for the solver, but also guides the search towards choosing more probable values. For example, in the case of the normal distribution, the 68–95–99.7 rule tells that 99.7% of values lie within three standard deviations of the mean. Thus, a valuable approximation restricts normally distributed random choices to this interval.

Bounding observed variables As observed variables already have a fixed value assigned, it may seem counterintuitive to bound their respective decision variable. This is, however, not the case. Firstly, from the observed value, one can infer information about the distribution parameters. This distinctive case is illustrated by the uniform distribution. Since a uniformly distributed sample must lie within its parameter-defined interval, an observation on its value imposes lower and upper bounds on those parameters. This is exemplified in the following short example (Listing 4.3), where X has an observed value of 5, through constraints on the support domain of X ($A \leq X \leq B$), the domain of A is reduced to $[1, 5]$ and B to $[5, 100]$.

Unfortunately, the uniform distribution is unique in providing true domain reductions via its explicit bounded support. Most distributions, such as normal or exponential, have infinite or semi-infinite support, making it impractical to derive

```
A ~ uniform(1, 100)
B ~ uniform(1, 100)

X ~ uniform(A, B) # X = 5 is observed
```

Listing 4.3: Simple model where the observation imposes constraints on the latent values

meaningful bounding constraints directly from their definitions. In these cases, applying strict domain bounds may be ineffective, possibly resulting in conflicts with the observed values.

Secondly, bounding observations originating from branching structures can help inform or constrain branch selection. To exemplify this, consider a case where variable A is sampled from one of three normal distributions, selected conditionally, as seen in Listing 4.4. Suppose A is observed with a value of 98. Intuitively, the third branch appears most likely to have produced the observation. Applying restrictive bounds for the normal distribution, $A_i \in (\mu_i \pm 3\sigma_i)$, yielss the following intervals:
$-14 <= A_1 <= 16; \quad -5 <= A_2 <= 25; \quad 75 <= A_3 <= 115$

Supplementing that the observed value of A matches one of the three branch-specific variables (A_1 or A_2 or A_3), the constraint system can infer that only A_3 is feasible.

```
if condition1
    A ~ normal(1, 5) #A1
elseif condition2
    A ~ normal(10, 5) #A2
else
    A ~ normal(100, 5) #A3
end

# A = 98 is observed
```

Listing 4.4: The observation A = 98 strongly favors the third branch due to higher likelihood under the normal distribution.

Soft versus hard constraints for restrictive bounds Ideally, the restrictive bounds should be modeled as soft constraints, representing the preferred range of values for random choice variables. This applies to both latent and observed variables. However, in practice, soft constraints introduce an additional decision layer, requiring the solver to determine which of the constraints to satisfy. The added complexity significantly impacts the performance of the solver. Additionally, when soft constraints cannot be met, the solver loses valuable domain reduction opportunities, further impacting performance. To balance expressiveness with tractability, a hybrid strategy is adopted: soft constraints are used for observed variables to avoid conflicts with fixed values, while hard constraints are applied to latent variables.

Approximated support and consistency constraints Having discussed the impact of bounding both latent and observed variables, the specific constraints introduced for each distribution type are now outlined. The domain bounds are summarized in Table 4.5.

For the **Bernoulli** distribution, the support remains unchanged. However, additional constraints are required to ensure valid probability sampling when the probability parameter takes on boundary values. Specifically, for $x \sim \mathrm{Ber}(p)$, if $p = 0$, then $x = 0$; also if $p = 1$, then $x = 1$.

Similarly, in the case of both continuous and discrete **uniform** distributions, the true support domain is fully preserved. However, consistency conditions have to be enforced on the distribution parameters: the lower and upper bounds must appear in increasing order. Specifically, $a < b$ for continuous distributions and $a \leq b$ for discrete ones.

Categorical-distributed variables must ensure that the allowed values lie within the valid index range, between 1 and the size of the category set.

Variables drawn from **normal** distributions are restricted to an approximated support of $[\mu - 3\sigma,\ \mu + 3\sigma]$ following the 68–95–99.7% rule. To avoid division by zero in the Gaussian likelihood computation, σ must be different from zero.

Variables from **Beta**, **Gamma**, and **Poisson** distributions use a hybrid approach: combining their true domain with an approximated Gaussian support derived from the mixture-of-Gaussians (MoG) rewriting [MAZ'YA and SCHMIDT, 1996]. This choice supports a more tractable approximation of likelihoods in the objective function, as detailed in the next section. The corresponding rewritings are written below:

$$
\begin{aligned}
\text{Normal}(\mu, \sigma^2) &= \text{MoG}\left([\mathcal{N}(\mu,\ \sigma)]\right) \\
\text{Poisson}(\lambda) &\sim \text{MoG}\left([\mathcal{N}(\lambda,\ \sqrt{\lambda})]\right) \\
\text{Gamma}(\alpha_1, \alpha_2) &\sim \text{MoG}\left([\mathcal{N}(\alpha_1\alpha_2,\ \sqrt{\alpha_1\alpha_2})]\right) \\
\text{Beta}(\alpha_1, \alpha_2) &\sim \text{MoG}\left(\left[\mathcal{N}\left(\frac{\alpha_1}{\alpha_1+\alpha_2},\ \sqrt{\frac{\alpha_1\alpha_2}{(\alpha_1+\alpha_2)^2(\alpha_1+\alpha_2+1)}}\right)\right]\right)
\end{aligned}
$$

Figure 4.2: Gaussian mixture approximations of standard distributions using a single-component mixture model. Each approximation expresses the target distribution as a one-element vector of Gaussian components with corresponding means and standard deviations.

The model bounds incorporate the original domain restrictions, introducing zero as a lower bound for all three distributions, and setting an upper bound of one for the Beta distribution. The second pair of bounding constraints is applied on the Gaussian approximation: values are restricted to the interval $[\mu - 3 \cdot \sigma, \mu + 3 \cdot \sigma]$, where μ and σ are derived from the MoG rewriting. Consistent with the handling

of normal distribution, the rewritten variance must remain strictly positive.

For both **Geometric** and **Exponential** distributions, the original support is maintained, with the lower bound of zero explicitly enforced.

The **Binomial**, **Laplace**, **Dirichlet**, **Multivariate Normal**, and **Cauchy** are not covered in this work.

Distribution	**Notation**	**Support**	**Approximated Support**
Bernoulli	$\mathrm{Ber}(p)$	$\{0, 1\}$	Same
Beta	$\mathrm{Beta}(\alpha, \beta)$	$(0, 1)$	(0,1) and $[\mu_b - 3\sigma_b, \mu_b + 3\sigma_b]$
Binomial	$\mathrm{Bin}(n, p)$	$\{0, 1, \ldots, n\}$	-
Categorical	$\mathrm{Cat}(p_1, \ldots, p_K)$	$\{1, \ldots, K\}$	Same
Cauchy	$\mathrm{Cauchy}(\mu, \gamma)$	$\mathbb{R}$	-
Dirichlet	$\mathrm{Dir}(\boldsymbol{\alpha})$	$\{\boldsymbol{x} \in \mathbb{R}^K \mid x_i > 0, \sum x_i = 1\}$	-
Exponential	$\mathrm{Exp}(\lambda)$	$(0, \infty)$	Same
Gamma	$\mathrm{Gamma}(\alpha, \beta)$	$(0, \infty)$	> 0 and $[\mu_g - 3\sigma_g, \mu_g + 3\sigma_g]$
Geometric	$\mathrm{Geom}(p)$	$\{1, 2, \ldots\}$	Same
Laplace	$\mathrm{Laplace}(\mu, b)$	$\mathbb{R}$	-
Multivariate Normal	$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	$\mathbb{R}^d$	-
Normal	$\mathcal{N}(\mu, \sigma^2)$	$\mathbb{R}$	$[\mu - 3\sigma, \mu + 3\sigma]$
Poisson	$\mathrm{Pois}(\lambda)$	$\{0, 1, 2, \ldots\}$	≥ 0 and $[\mu_p - 3\sigma_p, \mu_p + 3\sigma_p]$
Uniform	$\mathcal{U}(a, b)$	$[a, b]$	Same
Uniform Discrete	$\mathcal{U}_d(a, b)$	$\{a, a + 1, \ldots, b\}$	Same

Table 4.5: Overview of supported distributions and support bounding approximations

4.4.3 Dealing with execution control flow

Reasoning over possible execution paths presents a core challenge in probabilistic inference, due to the discontinuity present in the trace probability landscape. These discontinuities are often governed by discrete choices, where small random perturbations can lead to abrupt changes in the overall probability. A common strategy to address this is to remove the handling of branching behavior during inference by focusing on straight-line programs [Zhou et al., 2020]. Possible traces are usually explored in a greedy manner, dynamically balancing the discovery of new paths with the improvement of known ones. While this is a viable solution in the constraint model context, modeling solely straight-line programs and resolving branch handling prior to solving, this paper explores an encoding that unifies reasoning over possible branches within a single constraint model. This decision was motivated by the premise that constraint solvers excel at reasoning with discrete

choices, rendering them a candidate for untapped potential.

To correctly encode conditional execution paths, such as those induced by `if` statements, a set of auxiliary decision variables is introduced to indicate whether a given sample statement is *alive*, that is, relevant in the current execution trace.

For each random choice variable a_k listed in the specification, a corresponding boolean decision variable $alive_{a_k}$ is introduced to represent the *path condition* under which that sample is active. If a path condition is provided in the specification for a_k, the model includes a constraint that binds this condition to the corresponding $alive_{a_k}$ variable.

Variables without an explicit path condition specified must aggregate the path conditions of their corresponding branching alternatives. This is trivially done in the Python module by collecting variables with the same name and distinct identifiers. Consider the $y1$ variable in the GMM specification; this variable models the choice between the three alternatives. Accordingly, the associated alive variable ($alive_{y1}$) is constrained to be the disjunction of the path conditions of the alternatives to capture the presence of y_1 across all paths:

$$alive_{y1} = alive_{y1_1} \vee alive_{y1_2} \vee alive_{y1_3}.$$

Furthermore, the value of y_i, along with its associated likelihood, is conditioned on which alternative is active. This relationship is expressed as:

$$y1 = \begin{cases} y1_1 & \text{if } alive_{y1_1} \\ y1_2 & \text{if } alive_{y1_2} \\ y1_3 & \text{if } alive_{y1_3} \end{cases} \qquad likelihood_{y_i} = \begin{cases} \ell(y1_1, \mu_1, \theta) & \text{if } alive_{y1_1} \\ \ell(y1_2, \mu_2, \theta) & \text{if } alive_{y1_2} \\ \ell(y1_3, \mu_3, \theta) & \text{if } alive_{y1_3} \end{cases}$$

It is important to note that due to the aggregation of alternatives associated with a unique address under a single variable (e.g., y_i), additional type consistency restrictions must be imposed to ensure aggregation compatibility. Specifically, all alternatives must share the same type (`Int` or `Real`). This assumption restricts the set of allowed distributions for alternatives of a unique address to ones that share a common type. While limiting such a restriction is not uncommon in probabilistic programming systems.

By capturing the branching structure in the encoding, the solver is able to reason over possible execution traces and identify assignments governed by discrete choices. These alive components are later incorporated into the objective function to control which likelihood terms contribute to the trace probability.

4.5 Encoding the objective function

In the previous section, decision variables and constraints were introduced to capture the structure of the probabilistic model. The decision variables represent latent random choices, while the *alive* variables define the active execution path. To define the search over latent configurations, the next step is to encode the objective function for optimization. The natural candidate is the trace probability, as MH favors transitions towards higher-probability traces.

However, directly optimizing the full product of likelihood terms often fails in practice due to the substantial occurrence of non-linear terms and transcendental functions present in such computation. The compounding complexity of these functions likely contributes to solver failure. To reduce the computational cost and improve solver tractability, the objective function is redefined using approximations of the likelihood equations. The remainder of this section identifies viable encodings that balance the accuracy and solver performance.

4.5.1 Gaussians and Bernoullis: The expressivity core

To capture the core expressive capacity of probabilistic models, focus is placed on two fundamental distributions: Bernoulli and Gaussian. Resolving the likelihood encoding for these two enables, in principle, support for a wide range of models, since many complex distributions can be expressed as compositions or mixtures of them. In particular, Mixtures of Gaussians (MoG) are known to serve as universal encoders due to their broad approximation capabilities [MAZ'YA and SCHMIDT, 1996].

Encoding the likelihood of a **Bernoulli**-distributed sample $x \sim \text{Ber}(p)$ is straightforward. Since the sample can take one of two values, the likelihood is deterministically defined by the sampled value:

$$\text{Likelihood}(x) = \begin{cases} p & \text{if } x = 1 \\ 1 - p & \text{if } x = 0 \end{cases}$$

This form allows the likelihood to be directly expressed as logical constraints, avoiding the need for reformulation or approximation.

The encoding becomes more nuanced in the case of samples drawn from a **Gaussian** distribution. Gaussian likelihood introduces terms that depend on the sampled value, mean, and variance parameters, each involving non-trivial operations. The expression includes non-linear and transcendental functions, impeding solver tractability: non-linear division and multiplication, and exponentiation.

$$\text{Likelihood}(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

The effect is further amplified when multiple such likelihoods are compounded, as is the case in the GMM example, where several prior components factor in the trace

probability computation. Thus, approximation strategies are necessary to render the likelihood computation practical within SMT solving and to enable support for Gaussian distributions.

4.5.2 Gaussian likelihood encoding trials

This subsection focuses on identifying a viable encoding for the Gaussian likelihood that is both computationally tractable for the Z3 solver and preserves the essential probabilistic information. The goal is to replace or simplify the non-trivial components of the likelihood equation, those involving non-linear or transcendental functions. Preserving the impression of the likelihood will remain an essential goal in these encodings, as they ensure that the objective preserves guidance toward high-posterior regions. The remainder of this subsection describes several considered implementation strategies, including unsuccessful attempts and the final adopted solution. Notably, across these trials, the variance parameter had to be fixed prior to solving, as Z3's non-linear capabilities are limited otherwise.

Approximating exponential component

A first approximation attempt targeted the exponential function, due to its central role in the Gaussian likelihood. Instead of relying on incremental linearization, a non-linear approach that progressively tightens the lower and upper bounds of the equation, the exponential is directly approximated with a truncated Taylor series:

$$\exp(x) \approx \sum_{k=0}^{n} \frac{x^k}{k!}$$

Substituting the exponent in the Gaussian likelihood, the approximation becomes:

$$\exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right) \approx \sum_{k=0}^{n} \frac{1}{k!}\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)^k = exp_{Tay}$$

This truncated series, denoted as $\exp_{Tay}$, is then used in the final likelihood encoding:

$$\mathrm{E1}(x) = \frac{1}{\sqrt{2\pi\sigma^2}}\, exp_{Tay}$$

Despite removing the transcendental component, the resulting expression remains highly non-linear, due to nested multiplications and divisions. In practice, truncating the Taylor series at n = 100 did not yield meaningful solver performance. Lower truncation values led to substantial precision loss, degrading the quality of the likelihood approximation. Conversely, with larger series expansions, the problem remained intractable at large.

Linear piecewise approximations

The more drastic strategy involved replacing the Gaussian likelihood with linear piecewise approximations, effectively trading precision for the use of linear functions. As mentioned in the previous section, the distribution will be truncated,

spanning only over the $[\mu - 3 \cdot \sigma, \mu + 3 \cdot \sigma]$ interval, capturing the majority of its mass. Within this interval, the likelihood curve is approximated using 2k linear segments, k on each side of the mean. Each segment endpoint within the range $[\mu, \mu + 3 \cdot \sigma]$ is defined as:

$$\left(i, \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)\right) \quad \text{where} \quad i \in \{1, 2, \ldots, k\}$$

The left half is mirrored symmetrically around the mean. Together with the central point at the mean, define the linear segments that approximate the likelihood. Figure 4.3 illustrates the approximation for k = 3, highlighting the constructed piecewise segments.

This encoding minimizes the number of points where the true likelihood is evaluated to *2k + 1*. To compute the likelihood of a random choice decision variable, the corresponding interval segment is identified, and the likelihood is obtained through linear interpolation between the segment endpoints:

$$\text{E2}(x) = a + \frac{(x - x_i)}{(x_{i+1} - x_i)} \cdot (b - a), \quad \text{where } x \in [x_i, x_{i+1}]$$

Here, a and b are the segment likelihood values at x_i and x_{i+1}. Outside the $[\mu - 3 \cdot \sigma, \mu + 3 \cdot \sigma]$ interval, the likelihood evaluates to zero. The resulting encoding preserves essential likelihood information while reducing the number of expensive evaluations.

It is important to note that the computation of the endpoints for each Gaussian piecewise involves creating new decision variables for those points if the parameters of the distribution (σ, μ) are decision variables; otherwise, their probability values will be fixed constants.

In practice, for k = 3, the remaining evaluations of the true likelihood continue to impede tractability, due to the combined effect of the non-linear terms and the exponential transcendental function.

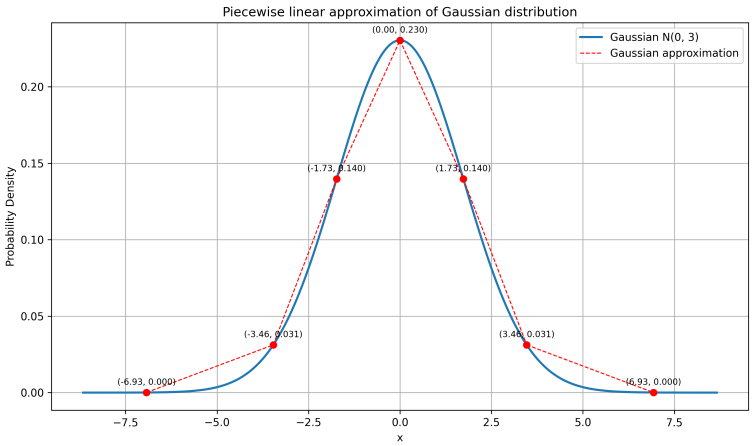


Figure 4.3: Piecewise linear approximation of Gaussian distribution

A third approximation strategy combines the two previous encodings *E1* and *E2*, resulting in a hybrid piecewise approach. Rather than evaluating the true likelihood at segment endpoints (as in *E2*), this method substitutes those values with the approximations derived from the truncated Taylor expansion in *E1*. The resulting segments retain the linear interpolation scheme but rely on cheaper endpoint estimates, removing the transcendental computation.

$$\text{E3}(x) = a_{Tay} + \frac{(x - x_i)}{(x_{i+1} - x_i)} \cdot (b_{Tay} - a_{Tay}), \quad \text{where } x \in [x_i, x_{i+1}]$$

Here, a_{Tay} and b_{Tay} represent the segment likelihood values, where each endpoint is evaluated using the truncated Taylor approximation from *E1*. For k = 3, this encoding renders tractable solutions for small-scale models, containing up to 20 data points. It was successfully tested on the Gaussian mixture model and linear regression with and without outliers (see Appendix A for models). However, it fails to identify solutions within a reasonable time for larger models due to the cumulative complexity of the expressions.

A related strategy builds on *E3*, but further simplifies the likelihood expression by modifying the term inside the exponent. Specifically, the term is replaced with $-l^2 i^i/8$, effectively decoupling the interaction between μ and σ. This removes the dependency on the mean and reduces the exponential to a function of the variance only. The endpoints to the right of the mean have the following formula:

$$\left(i, \frac{1}{\sigma\sqrt{2\pi}} \cdot \exp\left(-\frac{\ell^2 \cdot i^2}{8}\right)\right) \quad \text{where} \quad \ell = \frac{3\sigma}{k}$$

This approximation incurs a further loss of accuracy, however, in practice, it improves solver tractability and remains effective in identifying high-quality solutions for small- and large-scale models.

Non-probabilistic scoring effects

It is important to note that the approximations described above do not produce valid probability density values. This may limit interpretability and comparability between distributions. Nevertheless, the approximations preserve meaningful relative values that reflect the underlying likelihood.

Crucially, this limitation does not impede the integration of SMT solutions with the Metropolis-Hastings procedure, as the true probability of the SMT-derived trace can be evaluated within the inference algorithm.

4.5.3 Encodings for other distributions

With functional encodings established for the two primary distribution types, attention now turns to the remaining distributions supported in this work. This subsection outlines the encoding strategies applied to these additional distributions.

The **categorical** distribution is a basic discrete distribution used to model outcomes over a finite set of categories. Its likelihood function is straightforward and requires no approximation. For an observation $x \in \{1, \ldots, K\}$, the likelihood is given by $P(x) = p_x$, where $\mathbf{p} = [p_1, \ldots, p_K]$ is the vector of outcome probabilities. The implementation can be represented as the exhaustive conditional assignments:

$$\text{Likelihood}(x) = p_i \quad \text{if } x = i$$

This corresponds to a sequence of conditional checks returning the appropriate probability based on the observed category.

The likelihoods of both the **continuous** and **discrete uniform** distributions preserve the original likelihood. For $x \in [a, b]$, the continuous uniform distribution assigns likelihood $\frac{1}{b-a}$, and zero otherwise. Similarly, the discrete uniform distribution over $\{a, \ldots, b\}$ assigns likelihood $\frac{1}{b-a+1}$ within the support. Despite the division term being non-linear, it is preserved in the implementation, as it does not pose significant challenges to the solver, and the available nonlinear theories can be applied to this format.

The **exponential** distribution is estimated via the truncated Taylor series expansion with limited effectiveness. The likelihood of the **geometrical** distribution is preserved in the original format.

While more complex continuous distributions are inherently harder to approximate, this work follows the approach of using Gaussian mixtures as a practical solution. The approximations proposed for the **Beta**, **Gamma**, and **Poisson** distributions have demonstrated notable success in improving the efficiency of likelihood computations [Nori et al., 2015]. It is important to emphasize that these approximations do not provide exact representations, as the approximations use only one Gaussian component. However, they offer a sufficiently accurate resemblance that can be effectively exploited. This work adopts the same approach for these three distributions, rewriting them via Gaussian mixtures and applying piecewise linearization over the rewriting.

Other distributions, including Laplace, Binomial, Cauchy, and Dirichlet, are not addressed in the current work and are left for future investigation.

4.5.4 Combining likelihood contributions

Having established the individual likelihood encodings for the components of a trace, namely, latent and observed variables, the next step is to determine how these components are aggregated into a global objective. This objective corresponds to the overall trace probability, which reflects how consistent a given assignment is with the observations under the model.

The direct approach, in line with the original probabilistic formulation, is to compute the product of the individual likelihoods associated with each address in the trace. In the constraint model, this would consist of the combined value of both the probabilistic variable and the associated alive variable. Multiplying these two

decision variables ensures that the considered trace is consistent with the program's execution path. Formally, the likelihood of a trace can be expressed as the product over all unique addresses:

$$\mathcal{L}_{\text{trace}} = \prod_{a \in \mathcal{A}_{\text{unique}} \,:\, \text{alive}_a} P_e(x_a)$$

where $\mathcal{A}_{unique}$ denotes the set of unique addresses in the trace, alive_a is a binary indicator for whether the component is active, and $Pe(x_a)$ is the approximated likelihood encoding of the value at address a given its parameters. The prior specific parameters are left out for simplicity.

However, this multiplicative structure introduces significant non-linearity and variable coupling into the resulting objective function. The presence of products between variables greatly increases the function's complexity. As a result, this form of aggregation becomes intractable for current solvers, which struggle to handle the resulting high-degree polynomial or non-convex terms.

A natural alternative is to apply a logarithmic transformation to the objective, converting the product of likelihoods into a sum of log-likelihoods. This is standard in probabilistic inference and would yield:

$$\log \mathcal{L}_{\text{trace}} = \sum_{a \in \mathcal{A}} \text{alive}_a \cdot \log P_e(x_a)$$

The $alive_a$ term can be included inside the sum. While this formulation mitigates the issues related to multiplication, it introduces new non-linearities, the logarithmic transcendental function applied to every likelihood term. The formulation remains problematic for solvers operating in the constraint satisfaction domain.

Since directly applying the logarithm function is not recommended, given the current limitations of non-linear theory support in solvers, an alternative direction would be to explore explicit encodings of the log-likelihood terms themselves. No immediate or generalizable solutions were identified in the scope of this work, but the direction remains open for future research.

As a result, a pragmatic, though coarse, substitution is to aggregate the likelihood components via direct summation, directly adding the likelihoods. This simplifies the objective considerably and enables the solver to reason about the whole through the individual components. Although this approach breaks the probabilistic semantics of the original trace, it offers a tractable proxy objective. Importantly, it encourages the solver to optimize individual component likelihoods, which can still lead to globally meaningful solutions. While the summation departs from a fully probabilistic interpretation, its computational simplicity justifies the compromise to achieve solver tractability. The final form of the objective is:

$$\mathcal{L}_{\text{trace}} = \sum_{a \in \mathcal{A}_{unique}} \text{alive}_a \cdot P_e(x_a)$$

4.6 Integration of SMT-derived traces in Metropolis Hastings inference

This stage is responsible for incorporating the trace returned by the solver to guide the Metropolis-Hastings procedure. This work identifies two applications: warm-start and warm-jump. The first replaces the initial random proposal in MH with the SMT-derived trace, offering a more informed starting point. In contrast, the second injects SMT trace solutions dynamically during inference, conditioned on trace improvement stagnation.

The above strategies enable distinct modes of solver-guided control during inference, either as initial impulses or on-demand transitions. Additionally, a Gaussian drift phase is introduced as a mechanism for local refinement following the SMT injection, aimed to guide the sampler's subsequent proposals. The remainder of this section describes each of these strategies in further detail.

4.6.1 Warm-start: Replacing initial random guess

The warm-start integration mode replaces the standard random initialization with the SMT solver output. In this setting, the solver optimizes the full probabilistic problem from scratch, including constraints that encode prior bounds, observed evidence, model structure, and coherence conditions.

The solver first identifies an initial satisfiable latent variable assignment. It further attempts to refine the initial trace by maximizing the estimated probability over a fixed number of optimization steps. The final variable assignment is then passed to the inference procedure for integration.

To integrate the solver output, *Gen.generate* function is used to construct a trace consistent with the solver's latent variable assignments and problem observations. Since the solver produces latent assignments consistent with the model, the generate function can produce a complete and valid trace. The true trace probability can then be computed, allowing the inference procedure to continue unchanged and ensuring correctness is maintained.

By starting inference from a high-probability trace, this strategy bypasses the slow exploration during early inference. It provides an initial impulse, potentially accelerating convergence, without altering the core sampling mechanics.

4.6.2 Warm-jump: On-demand SMT trace proposals

The second strategy, the warm-jump, entails intermittently performing a jump to an SMT trace throughout the sampling process. By monitoring the inference progress, such as scores or low acceptance rates, it is possible to detect when the sampler stagnates or is exploring low-probability regions of the posterior. In these cases, an on-demand SMT call can provide a trace with qualitatively different latent configurations and potentially a higher-probability alternative. These directed interventions act as informed jumps, with the potential to drive the inference process

in more promising posterior regions. Warm-jumps are designed to complement the sampling process by providing more effective candidate traces.

To identify potential moments to perform a warm-jump, active tracking of inference diagnostics is required. One indicator is the acceptance rate, which can be monitored by inspecting the boolean value returned by the *Gen.mh* proposal function. A warm jump can be triggered when the acceptance rate drops below a certain threshold, defined relative to the inference history. Another signal is stagnation in the trace score progression, which can be analyzed through repeated calls to *Gen.get_score(trace)*, a function that returns the score of a current trace. These signals help detect when inference could benefit from an external intervention. The jump condition, can be computed for example, based on the acceptance rate of a single iteration of MH standard inference.

$$\texttt{jumpCondition} = \frac{\#\text{Accepted Proposals}}{\#\text{Total Proposals}} < \frac{1}{2}$$

In contrast to using an SMT-derived trace for initialization, performing a warm-jump mid-inference raises additional concerns relating to acceptance criteria and detailed balance. In this work, the jump functions as a reset of the latent state rather than as a conventional proposal, since standard proposal mechanisms do not permit deterministic assignment of variable values. Developing a valid proposal distribution that samples around SMT-derived values remains an open problem, particularly for discrete variables, and is left for future work. Although the SMT move itself does not follow a symmetric proposal distribution, violating detailed balance, it does not compromise the overall guarantee. This is valid provided the jump is treated as a complementary intervention to the inference procedure. To maintain this validity, the number of warm-jumps should remain limited.

Nevertheless, it is wise to include an acceptance step when injecting an SMT-derived trace to ensure that the move can provide meaningful information, such as improved likelihood. If the solver fails to find a satisfiable solution or if the likelihood is not improved, the procedure should revert to standard MH proposals. However, an exception may be warranted when the aim of the proposal is to switch posterior modes or explore structurally different configurations.

A final concern is ensuring diversity among the SMT-proposed solutions. This becomes particularly important when performing multiple warm-jumps or when exploring multimodal posterior distributions. Since the Z3 solver is deterministic, additional information is required to guide it towards distinct traces. Several constraints, designed to complement the base model with information from the current trace or previous solver solutions, are presented in Subsection 4.6.2.

Combining the above components, the warm-jump setup triggers an SMT intervention whenever a jump condition is met, up to a limited number of times. When triggered, the SMT solver is invoked with the base probabilistic problem, augmented with additional diversity constraints specific to the current inference stage. If the returned trace satisfies the acceptance criteria, the jump is executed; otherwise, standard inference is continued. The above procedure is exemplified

in Algorithm 1. This strategy enables on-demand interventions during inference, helping the sampler escape local stagnation and explore new posterior regions more effectively.

Algorithm 1: MH Inference with Warm Jump Proposal

```
if warmJumpsCount > 0 and jumpCondition then
    candidateTrace ←
     smtTrace(pProgram, observations, trace, diversity);

    if candidateTrace ≠ None and
     score(candidateTrace) > score(trace) then
        trace ← candidateTrace;
    warmJumpsCount ← warmJumpsCount − 1;
trace ← standardInference(trace);
```

Diversity in solutions

To support meaningful multiple jumps, a mechanism is required that ensures diversity between trace solutions. Without such mechanisms, the deterministic solver would return identical solutions that are not suitable for on-demand trace proposals. These additional constraints take into account the current state of the inference process or past solver solutions to produce diverse solutions. Moreover, diversity can guide the process of exploring a multi-modal posterior structure.

There are two general strategies for introducing diversity between solutions: complementing the objective function with a diversity term or enforcing it through constraints. In the first strategy, the diversity component is modeled in a problem-specific manner or relies on predefining a distance measure between possible solutions [Ingmar et al., 2020]. In the current framework, such a term is not suitable due to the goal of maintaining applicability across a wide range of probabilistic models, as well as a considerable discrepancy in likelihood estimates between traces. Instead, this paper proposes the use of explicit constraints to achieve variation between solutions. The following constraint-based strategies target either latent values or the final trace probability.

Fixing latent variables

In a similar fashion to block resimulation in Metropolis-Hastings inference, where related latent variables are modified in groups, partial trace optimization can be performed by fixing a subset of latent variables and solving for the remaining ones. The diversity in the resulting traces is provided by the choice of variable selection and their stochastic values in the current trace within the sampling process. This approach assumes that variable groupings, referred to as blocks, are chosen on the basis of structural coherence, meaning that variables within a block are justifiable to vary together.

In the Gaussian mixture model, partial trace optimization can be performed by

fixing a selection of latent variables, such as the cluster means, to their current trace value. These fixed values are passed down to the solver as constraints, which then optimizes the remaining variables, the cluster assignments. This strategy allows on-demand exploration of alternative traces while preserving assignments of the current trace. The process is illustrated in Figure 4.4.

In partial trace optimization, the SMT model is complemented with a set of hard constraints assigning the selected variables to their current values. In addition, restrictive bounding constraints associated with these values are removed. Since this subset of latent variables is no longer the subject of search, they can be excluded from the optimization objective. Specifically, the contribution of their likelihood estimation is omitted, reducing the objective function. These modifications ensure that the solver focuses the search for the partial trace while preserving the fixed subset.

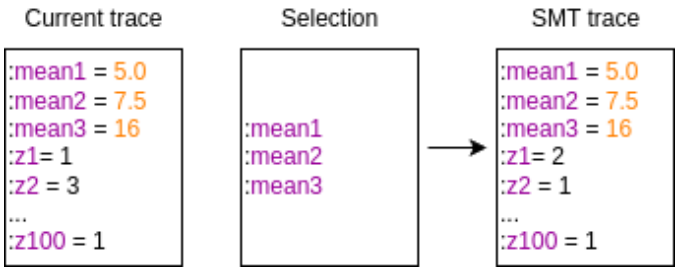


Figure 4.4: Illustration of partial trace optimization, where a selection of variables from the current trace is held fixed and modeled as constraints, while the remaining variables are optimized via SMT

Restricting latent values To avoid repeatedly identifying the same solutions through the deterministic optimization process of the solver, constraints can be introduced to restrict latent variables from taking the same values as in the previous solution. The choice of variables to restrict directly influences the resulting diversity. For example, in the Gaussian mixture model, constraints can be applied for the cluster means, cluster assignments, or a combination thereof.

Depending on the dependency structure of the variables, restricting the value of a single variable can trigger cascading changes in the model, potentially leading the solver toward alternative high-probability regions of the posterior. Such constraints are useful for exploring distinct traces. This effect is illustrated in Figure 4.5 where a single restriction on z_1 leads to changes in both the assignment variables and the inferred cluster means. These types of constraints require moderate model structural awareness and are harder to generalize.

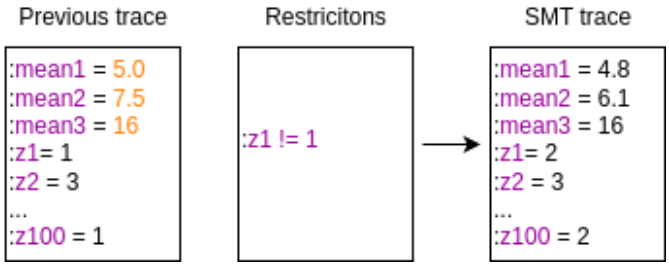


Figure 4.5: Illustration

Improving current trace probability A third strategy focuses on directly improving the probability of the current trace by using it as a target within the SMT-based solver. Instead of generating entirely new traces or enforcing diversity through restrictions, this approach evaluates the estimated likelihood of the current trace under the SMT model and sets it as a baseline for optimization. The goal is to guide the solver toward finding a trace with strictly higher probability, effectively using the current trace as a lower bound. To achieve this, the current trace is fed into the solver, which evaluates its estimated probability under the model's objective function. An additional constraint is then introduced, requiring that any new solution must exceed this baseline, expressed as $\mathcal{L}_{\text{trace}} > \mathcal{L}_{\text{current trace}}$.

This is particularly useful for on-demand interventions, where the goal is not diversification but local improvement. Normally, the SMT solver begins optimization from an unconstrained initial state, producing an initial feasible solution and refining it over several iterations. In contrast, this method skips the search from scratch and challenges the solver to surpass a concrete, achievable trace score, potentially leading to faster and more targeted improvements in inference quality.

4.6.3 Gaussian drift as guide

To further guide inference following SMT trace injection, this paper proposes a limited Gaussian drift phase aimed at keeping the subsequent proposals in proximity to the provided trace. Unlike standard random perturbations, Gaussian drift leverages local information to propose more informed candidates, increasing the likelihood of acceptance.

Within the fixed drift window, Gaussian drifts are applied over continuous latent variables to encourage gradual improvement. Noise is added to the latent values of the current trace, either individually or over predefined groupings. To preserve trace locality, the step size (i.e., the variance of the drift) for a latent variable should be carefully selected to reflect the scale of the prior probability landscape.

In the case of the GMM example, Gaussian drifts are applied to the three cluster means μ_1, μ_2, μ_3 individually using a standard deviation of 0.01. Each proposal is defined by sampling from a normal distribution centered at the current trace value, exemplified for the first mean in Listing 4.5.

```
@gen function mean1_drift_proposal(current_trace)
    mean1 ~ normal(current_trace[:mean1], 0.01)
end
```

Listing 4.5: Gaussian drift proposal for `mean1`

The custom proposal function is passed to *Gen.MH*, directing the sampler to generate new candidates using the generative function, producing locally informed updates to the trace. The previous inference algorithm is enhanced with a drift phase as presented in Algorithm 2. The phase is defined by a fixed number of MH moves. While not optimized, the length of the drift phase influences the trade-off between early trace exploitation and broader exploration. Alternatively, the Gaussian drift can be interleaved with standard proposals for discrete random samples.

Algorithm 2: MH Inference for GMM with Drift Phase

if *within drift phase* **then**
  for $i = 1$ *to* K **do**
    MH update using `mean`$_i$`_drift_proposal`
else
  Perform standard MH updates

Once the phase concludes, standard inference resumes, thus detailed balance and convergence guarantees are preserved. This local refinement phase enhances the effectiveness of the SMT initialization or jumps by maintaining the sampler in high-probability regions before broader exploration resumes.

Chapter 5

Results

This chapter presents the experimental results assessing the potential benefits of incorporating SMT guidance into Metropolis-Hastings inference for probabilistic programs. The evaluation is organized around the three sub-questions. The first two sub-questions (**Q1** and **Q2**) are addressed in the warm-start setup, while the third (**Q3**) is analyzed in the warm-jump setting. The general experimental setup is described first.

5.1 General experimental setup

The model is built and optimized using PyZ3, with Z3 chosen for its broad support for SMT theories, including non-linear arithmetic. Python's interface enables dynamic constraint construction and flexible manipulation of likelihood computation. In addition, the object-oriented paradigm enables modular representation of distributions and their interactions, facilitating experimentation with alternative constraint encodings. The implementation and models are available for reproducibility and further exploration [Coman, 2025]. It is important to note that while the specific solver technology and implementation choices are a singular solution, the paper proposes a broader focus, namely, the incorporation of constrained reasoning into probabilistic inference frameworks.

For all SMT-based trace constructions, the initial satisfiability check is unconstrained, while subsequent optimization attempts are each limited to 2000 seconds, with a maximum of 10 optimization calls.

Experimental model suite To evaluate the generality of the warm-start and warm-jump strategies, experiments were conducted across five representative probabilistic models. The models were selected and adapted to avoid unsupported distributions, substituting them with Gaussian or uniform distributions where necessary. Full probabilistic specifications are provided in Appendix A. Synthetic observational data were generated to match each model's probabilistic structure. The probabilistic models are described below:

- **GMM (Gaussian Mixture Model):**A standard Gaussian Mixture model,

where each data point is drawn from one of several Gaussian components, each centered around a latent mean. The number of clusters is fixed.

- **LR (Linear Regression):** A classic linear regression model with latent normally-distributed intercept and slope.
- **OLR (Outlier Linear Regression):** Linear regression model with outliers. In contrast to the previous model, OLR includes discrete random choices that indicate whether a given observation is an outlier.
- **Gaussians:** A collection of observed Gaussian variables, all drawn from the same unknown latent mean and variance.
- **N-Schools:** A hierarchical model capturing treatment effects across multiple schools. Treatment effects are nested within school-level, district-level, and state-level structures, each with its own latent hyperparameters.

5.2 Warm-start

The first SMT-guided MH setup considered is the warm-start scenario, in which the standard random initialization is replaced by the SMT-derived solution to the full probabilistic problem. This setup aims to accelerate the identification of high-probability traces and improve convergence rates during MH inference.

An additional experiment follows the effect of introducing a brief Gaussian drift phase after the warm-start to allow the neighbourhood exploration of the solver-provided trace to improve early-stage inference performance.

Experimental Setup Each inference variant logs trace score values at every MH update step, referred to as *score progression*. Evaluation focuses on the average score progression across 100 independent runs, using identical inference configurations for both random and SMT-based initializations. To ensure fairness and reproducibility, identical fixed random seeds are used across runs for each variant. As trace scores represent the trace log-density, less negative values are preferred as they indicate a better fit. Each inference run is capped at a maximum of 2000 MH update steps.

The SMT-based initialization is obtained by solving a constraint model that encodes the full probabilistic structure of the model, along with all observational constraints. Since the underlying probabilistic problem remains unchanged, the same SMT-trace solution is reused in all runs.

5.2.1 Experiment 1: Comparison between warm-start and random-start

The first experiment addresses **Q1** by comparing inference initialized with SMT-derived values (referred to as warm-start) against the standard random initialization (referred to as random-start).

Focused results: Gaussian Mixture Model

In the GMM example, the chosen baseline inference procedure performs Metropolis-Hastings (MH) updates sequentially, targeting each latent variable individually. Specifically, updates are applied first to the means μ_i, followed by updates to the cluster assignments z_i. To illustrate the inference strategy, the pseudocode in Algorithm 3 summarizes the update loop.

Algorithm 3: MH baseline inference for GMM, updating first the means, then the cluster assignments individually.

for $i = 1$ *to* K **do**
  MH update (trace, μ_i)
for $j = 1$ *to* N **do**
  MH update (trace, z_i)

Figure 5.1 illustrates the average score progression for the GMM inference runs under two different initialization strategies. As the figure shows, the warm-start variant consistently achieves higher trace scores across iterations, demonstrating a significant advantage in early-stage inference. This advantage is further quantified in Table 5.1, which shows that it takes over 2000 Metropolis-Hastings (MH) proposals for the randomly initialized inference to reach or surpass the score quality of the SMT-based warm-start.

A closer inspection of the early trace evolution (visible in Figure 5.2) reveals that most improvements occur during updates to the component means μ_i, while the cluster assignments z_i remain largely stable. This suggests that the SMT initialization already provides highly accurate assignments, and that inference could benefit from focusing MH updates primarily on the mean variables in such cases.

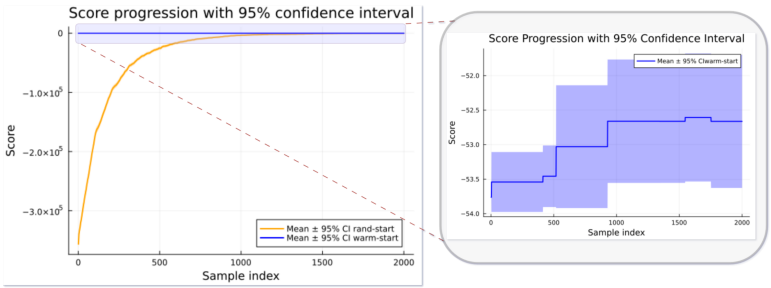


Figure 5.1: Score progression of Metropolis-Hastings procedure with different starting points. Random guess in yellow and SMT-derived trace in blue.

Results

To assess the generality of this finding, similar comparisons were conducted across the remaining models. Table 5.1 demonstrates that, on average, SMT-based initialization yields higher initial scores compared to random initialization. This supports the use of solver-derived traces as effective warm-starts for MH inference.

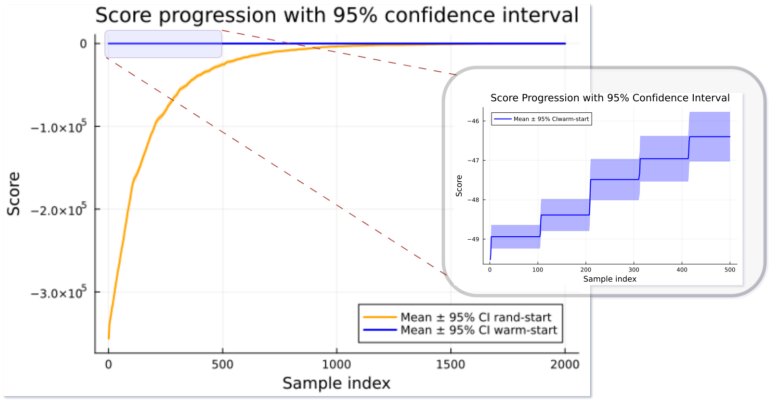


Figure 5.2: Early score progression of Metropolis-Hastings procedure with SMT-derived initialization.

An additional metric, surpass time, quantifies how many MH update steps the random-start variant requires to reach or exceed the trace score achieved by the SMT-derived initialization. This metric serves as a proxy for the amount of early-stage exploration that can be bypassed with this strategy. The surpass time metric reveals that random-start requires additional steps exploring low-probability regions before catching up with the SMT-based initialization. This early advantage leads to improved convergence overall, although the magnitude of improvement varies depending on model complexity and structure.

The utility of such warm-starts is influenced by model complexity and problem size. Notably, models like GMM and the n-schools exhibit longer average surpass times before the warm-start advantage is overtaken. This indicates that solver-informed initialization is particularly valuable in more structurally complex settings.

Table 5.1: Performance comparison of warm-start strategies across different models.

Model	**Average Initialization**		**Surpassing Time (MH proposals)**
	Baseline	**Ours**	
GMM (100 pts, 3 clusters)	-389573.3656	-53.7621	$>$ 2000
OLR (20 pts)	-2675	-71	$>$2000
OLR (100 pts)	-14599	-350	1511
LR (20 pts)	-2548	-76	33
LR (100 pts)	-11780	-2196	4
Gaussians (20pts)	-867	-59	49
Gaussians (100pts)	-9144	-701	5
N-schools (100pts)	-175022	-22127	$>$2000

5.2.2 Experiment 2: Gaussian drift as local guide

This experiment addresses **Q2** and sets out to investigate the use of Gaussian drift as a guiding strategy to enhance early-stage convergence, by maintaining the exploration in more promising regions. The experiments compare war-start initialization with one augmented by an additional Gaussian drift phase.

While the warm-start initialization has a higher initial score on average, it can prove difficult for standard MH proposals to progress from a locally optimized point through random perturbations. To address this challenge, a short "grace period" of Gaussian drift moves is introduced immediately after the initialization. This mechanism aims to maintain proximity to high-quality traces during early inference by leveraging information from the SMT-derived trace. The hypothesis is that controlled local perturbations around the SMT trace can help exploit its structure more effectively before allowing broader exploration.

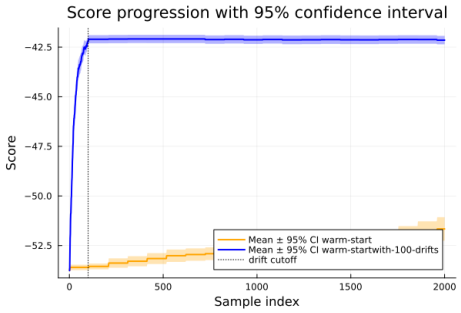
Gaussian drift proposals are applied to latent variables originating from continuous distributions. In this work, these are manually implemented to serve as a starting point to tailor such proposals to the structure of each probabilistic model. The drift variance is chosen to reflect the model's underlying structure and scale. Each experiment uses a drift window cutoff of 100 MH update moves before transitioning to the baseline inference.

Focused results: Gaussian Mixture Model

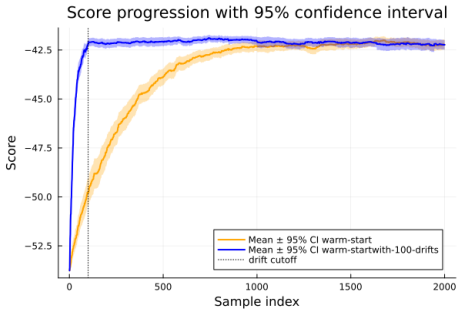
Figure 5.3a shows that introducing a Gaussian drift phase over the component means significantly improves inference performance compared to the baseline. Specifically, it enables the model to uncover higher-probability traces following warm-start initialization. Within the 100-step drift window, the enhanced variant captures more informative moves and consistently proposes candidates with greater posterior relevance.

To isolate the effect of μ_i perturbations, given that the trace already contains good z_i assignments, we compare inference that proposes updates only to μ_i, excluding latent cluster assignments. As illustrated in Figure 5.3b, both inference modes improve more rapidly under this restricted setting. However, the Gaussian drift variant advances faster toward high-probability regions. This is also reflected in the proposal acceptance rates: 78.03% with Gaussian drifts versus 1.63% without.

Notably, in this GMM example, a window of 100 Gaussian drift proposals is sufficient to reach a high-probability plateau, suggesting that a short, targeted drift phase can meaningfully accelerate convergence while preserving information from the SMT-derived initialization.



(a) Score progression comparison with a 100 Gaussian drift window interleaved with standard MH updates for cluster assignments.



(b) Score progression comparison with a 100 Gaussian drift window, excluding cluster assignments resampling.

Figure 5.3: Score progression comparison illustrating the impact of Gaussian drift phase following and SMT-derived warm-start.

Results

To complement our findings on warm-start inference with Gaussian drift proposals, the evaluation was extended to other model classes to explore adaptability and performance in different structural settings. An additional metric, window gap, quantifies the average score improvement a the end of the drift phase. Results are summarized in Table 5.2.

In the **OLR model**, the SMT-derived trace does not produce a perfect outlier assignment. In this instance, interleaving Gaussian drift line proposals with outlier resampling improved convergence and posterior scores. The result highlights the importance of adapting the drift mechanism to the structure of the model, especially when operating without the assumption of perfect discrete assignment.

For the **N-Schools model**, Gaussian drifts were applied to the beta latent variables representing state-, district-, and type- effect, encountering higher acceptance rates with the drift variant. Notably, selectively applying drift proposals to specific subsets of these latent dimensions (e.g., state-level only) led to improved performance, suggesting that drift effectiveness is not unique but highly dependent on the structure of the posterior and the informativeness of the SMT-derived trace.

In the **Gaussian model**, the variance was fixed to a random value prior to optimization. Consequently, the drift proposal was applied only to the mean parameter, interleaved with standard perturbations for the variance. The modest score improvement can be explained by the substantial contribution of the variance proposals.

Table 5.2: Drift-phase performance comparison between warm-start and drift-augmented warm-start using acceptance percentage and drift window size of 100 MH updates.

Model	**Acceptance % (Drift Phase)**		Window gap
	Baseline (no drifts)	**Ours (100 drifts)**	
GMM (100 pts, 3 clusters)	1.63 %	78.03 %	-7
OLR (20 pts)	59.74 %	66.51 %	-9
OLR (100 pts)	89.%	89 %	-0.5
LR (20 pts)	1.73 %	34.71 %	-19
LR (100 pts)	3.69 %	31.75 %	-104
Gaussians (20 pts)	47.21 %	5.1 %	-0.2
Gaussians (100 pts)	6.0 %	67.55 %	-1
N-schools	72.6 %	76.9 %	-215

Collectively, these results indicate that Gaussian drift proposals provide a viable mechanism for early-stage trace improvement across a range of models. These custom proposals function as guided moves, enabling more informed and higher-quality transitions during inference. Crucially, the combination of SMT-derived initialization with Gaussian drift proposals allows the sampler to remain within, and effectively explore, high-probability regions of the posterior. These findings directly address Q2, confirming that SMT-derived structure can be successfully leveraged to guide the Metropolis-Hastings search following initialization. While this experiment focuses on drift immediately after initialization, the results motivate further investigation into the potential of Gaussian drift as a general-purpose refinement mechanism following any SMT-trace injection. Naturally, the effectiveness of this guidance is conditional on the informativeness of the SMT-derived trace, an issue further examined in Chapter 6.

5.3 Warm-jumps

The warm-start strategy incorporates SMT-derived traces only once, at initialization. In contrast, warm-jump strategies introduce the possibility of dynamically injecting SMT-derived traces during inference to course-correct when the sampler stagnates. This section addresses **Q3** by investigating whether such mid-run interventions can steer inference toward high-probability regions, improving overall convergence. Since frequent SMT invocations are computationally prohibitive and undesirable, this paper considers a limited number of jump opportunities triggered by inference diagnostics, such as score plateaus or low acceptance rates.

Experimental setup During inference, two warm-jump opportunities are identified. The warm-jump trigger is defined based on the acceptance rate of a standard

procedure, measuring whether more than half of the MH update proposals were rejected. An SMT-derived trace is accepted if it yields a higher score than the current one. Two state-aware trace construction strategies are considered:

- **Block-based jump strategy:** A subset of latent variables is fixed, and SMT solves for the remaining structure. This aims to generate a trace that is diverse yet consistent with the current partial structure.
- **Score improvement jump strategy:** SMT is used to refine the current trace, explicitly aiming to improve its estimated score.

For the block-based jump strategy, specific latent variables were fixed prior to the SMT call. While the choice of variables to fix is, in principle, flexible, the fixed values in these experiments were manually selected to align with the logic of block resimulation and to capitalize on the solver's strength. The selection for each model is summarized in Table 5.3. For the GMM model, means were fixed rather than cluster assignments, allowing the SMT solver to resolve discrete choices. Similarly, in the OLR model, non-discrete variables were constrained to focus the SMT search on the most challenging aspects of inference.

Table 5.3: Fixed latent variables at each warm-jump opportunity for different models

Model	**Jump Opportunity 1**	**Jump Opportunity 2**
GMM	mean1	mean2
LR	slope	intercept
OLR	intercept	intercept
Gaussians	sigma	sigma

Due to the substantial computational cost of SMT-based warm-jump interventions, each run incurs significant overhead, often several hours per instance, which necessarily limits the number of independent trials. Additionally, because probability plateaus and stagnation points do not align across different runs, averaging results over multiple trials is avoided. Instead, both diversification strategies were evaluated under identical, reproducible conditions using fixed random seeds, ensuring that each was tested on the same jump opportunities. Effectiveness is assessed based on the number of successful interventions, where a satisfiable SMT-derived trace was identified and resulted in an improved score. The tests are performed for the GMM model and small-scale LR and OLR models. For cases where the intervention was successful, improvement and convergence were examined for individual instances.

Results

This experiment compares the number of successful interventions between the two state-aware trace construction strategies. The added diversity constraints in

these two strategies may fail to yield a satisfiable solution. As the strategies aim for a trace score improvement, instead of multimodal exploration, this paper measures the amount of informative jumps, ones that increase current likelihood. The results are presented in Table 5.4.

Model	**Block-based jump strategy** (unsat / unused / used)	**Score improvement jump strategy** (unsat / unused / used)
GMM (100pts)	0 / 17 / 3	0 / 4 / 16
LR (20pts)	0 / 3 / 17	0 / 11 / 9
OLR (20pts)	0 / 11 / 9	0 / 10 / 10
Gaussian(20pts)	0 / 18 / 2	1 / 18 / 1

Table 5.4: SMT-guided jump outcomes per model and diversification strategy. Each cell shows the number of SMT solutions that resulted in unsatisfiable, rejected (by the acceptance condition), or accepted. The reported results are from 10 runs with two jump opportunities.

As presented in the table, both strategies frequently encounter unsatisfiability. For the block-based jump strategy, this is largely attributed to the combination of restrictive bounds and additional equality constraints that fix part of the trace. In the score improvement jump strategy, the optimizer often fails to identify solutions that yield a higher estimated score. Even when feasible solutions are found, they do not consistently improve the current trace, and neither strategy emerges as categorically superior. These outcomes underscore the potential of SMT-based warm-jump strategies as an active guidance heuristic.

For the block-based strategy, in particular, these findings suggest that solving for a partially fixed trace mid-inference is insufficient for achieving reliable improvements. Although setting a subset of latent variables reduces the number of free decision variables for the SMT solver, the lack of satisfied soft constraints for domain reduction ultimately limits the solver's effectiveness.

When a satisfiable SMT-derived trace was identified and resulted in a score improvement, warm-jumps led to a measurable gains in trace probability improvement and convergence. In several such instances, the injected trace helped steer the sampler toward higher-probability regions of the posterior. For example, Figure 5.4 illustrates a representative run for the GMM model, where the injected trace effectively redirected inference towards a more probable solution and improved convergence.

However, as previously discussed, such improvements were not consistent across trials. This variability limits the strength of any general conclusion: while warm-jumps can have a significant positive impact in certain cases, their effectiveness remains sensitive to both model structure and solver performance.

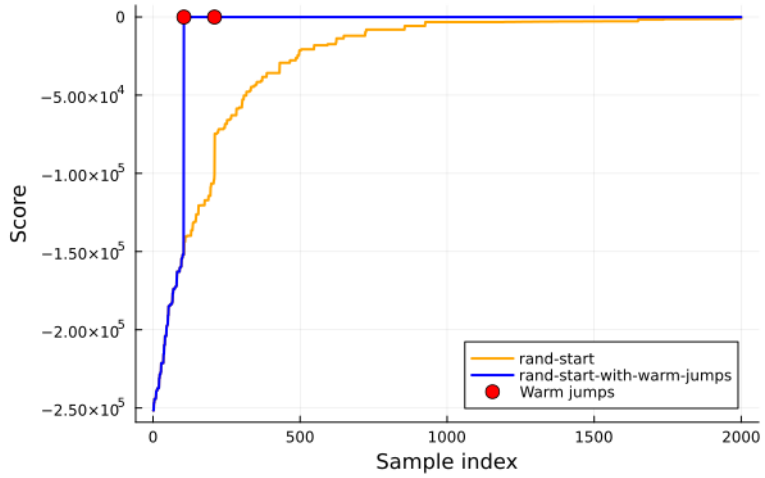


Figure 5.4: Comparison between a representative run with two successful jumps and the baseline inference for the GMM model. The jumps rely on the score improvement trace construction strategy.

5.4 Solver overhead

The practical utility of SMT-driven Metropolis-Hastings also depends on the time required to transfer and solve the inference problem via constraint programming. Both the complexity of the problem and the type of constraints play a critical role in the optimization process. Table 5.5 reports the time spent solely on the SMT optimization task without diversification constraints.

Table 5.5: SMT solving time for each model.

Model	**SMT Time (s)**
GMM (100 pts, 3 clusters)	201.98
OLR (100 pts)	6685.7551
LR (100 pts)	143.2019
Gaussians (100 pts)	346.7974
N-schools	49.5601

Chapter 6

Discussion

The complexity of modern probabilistic models and the increasing demand for scalable, tailored inference have driven the development of innovative methodologies to improve and automate proposal distributions in Metropolis-Hastings (MH) algorithms. This discussion explores a structured approach for generating high-quality, observation-aware traces, using Satisfiability Modulo Theories (SMT), designed to improve inference and enhance sampling efficiency in constrained latent space.

In this chapter, we analyze the empirical results obtained from introducing SMT-trace injection in inference for various probabilistic models. We investigate what these results reveal about the effectiveness and generalizability of deterministic proposals, in particular to conventional MH updates.

Following the result discussion, we examine the practical limitations of the current implementation, including challenges of constraint-based encoding of probabilistic models and the computational overhead of SMT solving. By addressing both the strengths and weaknesses of our approach, we aim to provide a balanced perspective on its applicability and potential integration into the broader landscape of inference automation.

6.0.1 Results and implications

SMT-derived MH initialization As a first step in exploring the use of SMT guidance within Metropolis-Hastings inference, we examine the feasibility of constructing an informed trace using only the probabilistic program and its observational data. The resulting trace is used as warm-start initialization and compared to random initialization to quantify the impact. These results directly address **Q1**. As shown in Table 5.1, using an SMT-based initialization demonstrates a clear advantage, producing high-quality trace estimates that satisfy the observations and are likely under the model. The extent of this benefit, however, varies across models, with varying levels of early-stage efficiency gains.

Notably, SMT-based initialization provides stronger early advantages in terms of bypassed MH updates in smaller-scale models, where the interplay of likelihood

terms in the optimization objective is easier to resolve and more directly guides trace construction. The non-trivial interactions of these terms are most likely the factor that limits solver effectiveness in larger models, rendering the optimization landscape harder to navigate. For such larger-scale cases, a potential line of future work would involve partial trace solving, removing a fraction of the likelihood estimation components from the objective to reduce complexity during solving. Methods of relaxing large-scale probabilistic models exist, such as removing independent groups of variables, and could be integrated within our framework.

Overall, our findings show that SMT-based trace solutions can help bypass a portion of the burn-in period typical to MCMC algorithms, accelerating convergence during the early stages of inference.

Post initialization guidance To address **Q2**, we examined the potential of SMT-derived traces to guide the inference process beyond the injection moment. The results, presented in Table 5.2, show that Gaussian drift proposals can successfully retain information from the SMT trace and guide early-stage inference. However, these results should be interpreted with caution.

The effectiveness of the Gaussian drift proposal is closely tied to the quality of the SMT-derived initialization. If the trace poorly aligns with the true posterior, then the inference is initialized from a fundamentally uninformed point, no better than random initialization. In such cases, Gaussian drift may only delay convergence. Conversely, if the injected trace is situated in a local optimum, the Gaussian drift mechanism, while more informed than random proposals, may lead to slow progress and delayed escape. In both cases, the guidance provided by the drift becomes limited, underscoring the importance of both trace construction and placement within the posterior landscape.

A small drift window can facilitate local exploration immediately after injection, enabling quick recovery if the trace is poor while still exploiting informative ones. While this window was fixed in our implementation, in practice, it could be dynamically adapted by monitoring the acceptance rate, allowing an informed transition between SMT-guided to general inference.

SMT-derived traces as dynamic interventions Beyond initialization, we addressed **Q3** by evaluating SMT-derived traces as dynamic interventions during inference, provided the sampler encounters a stagnation period with a low acceptance rate. However, our results show that the effectiveness of these jumps is inconsistent. Notably, our experiments focused on state-aware improvements, without testing jumps designed to explore the broader structure of the posterior. Under these settings, the use of soft constraints employed to satisfy partial trace fixation and loss of likelihood precision proved insufficient to reliably guide the solver toward informative trace solutions. The resulting traces often technically satisfied the constraints but failed to contribute meaningful improvements to the sampling trajectory. Additionally, the large overhead of solver invocation presents a practical limitation. Each SMT-guided jump requires solving a new constraint problem with additional diversity constraints.

Automating informed proposals A central question guiding this work was whether SMT-based guidance can improve the effectiveness of MH-based inference in probabilistic programs. Specifically, we explored whether solver-derived traces could serve as automated informed proposals, reducing the need for manual design efforts and serving as structured initialization or heuristic intervention.

While our results compare the SMT-guided approach to a basic Metropolis-Hastings baseline, the methods can be integrated with any MH variant. Since it operates as a warm-start initialization or as a limited jump mechanism, it preserves the theoretical guarantees of the original algorithm. In this sense, the SMT component acts as an enhancement layer and can be flexibly combined with existing MH alternatives.

However, our findings suggest that, while useful as an initialization method, the current formulation is unsuitable as a recurrent proposal strategy beyond initialization. The computational overhead of SMT solving and diminishing returns of constraint-derived jumps limit its practicality in active sampling.

6.1 Limitation

SMT-solving overhead One key limitation of our approach lies in the overhead of SMT solving. Even solving the base probabilistic problem as a constraint model requires a non-negligible amount of time, which becomes more pronounced as the model scales in size and complexity. The inclusion of diversity constraints, used to encourage varied on-demand proposals, further increases the solver's burden. The use of soft constraints increases the search space, making the solver less effective at finding satisfying solutions. As a result, the method is currently best suited for pre-inference initialization rather than being used repeatedly within an active sampling loop.

Modeling challenges Modeling a probabilistic program as a constraint problem involves further challenges. Our approach is currently restricted to models with fixed support and non-nested generative structure. In particular, it does not support nested generative calls or dynamically sized arrays. To enable SMT-based reasoning, all loops and arrays were pre-unrolled, yielding a static constraint representation. While this enables precise trace construction for a subset of models, it limits applicability to more expressive or recursive probabilistic programs. Additionally, not all distributions used in generative models are supported in the constraint encoding. Continuous distributions with complex or non-linear likelihoods require careful approximation to strike a balance between optimization tractability and score fidelity. Identifying qualitative likelihood estimates, along with advances in non-linear constraint solving, would further extend the applicability and effectiveness of this approach.

Another limitation arises from the optimization objective using an additive formulation, summing individual score components, rather than operating in log space or multiplying likelihoods. This design choice avoids non-linearity, but it dimin-

ishes the penalizing effect of low-likelihood components that would, under a proper probabilistic objective, reduce the overall trace probability. As a result, the optimization may occasionally favor structurally consistent traces that are statistically unlikely under the true generative model.

We further simplified the model by making a specific choice for the variances of the normal distributions. To support tractable solving, variances were fixed to a random value prior to solving. This introduces the risk that a poorly chosen variance may bias the solver towards low-quality trace assignments. A more flexible, higher-level greedy strategy for selecting the variance values could improve trace quality and solution robustness.

A final limitation of the constraint modeling approach involves decision variables corresponding to random variables with infinite support. While hard constraints to restrict the support can guide the search through latent space, they also restrict the solution space and risk excluding plausible traces. Alternatively, implementing domain restrictions as soft constraints introduces an additional layer of optimization objectives that increases the solver burden and may fail to yield informative or meaningful trace proposals.

Chapter 7

Conclusions and Future Work

This chapter presents the main contribution of this work, summarizes key results, and states concluding insights. Lastly, it outlines promising lines for future work to address current limitations and expand the applicability of the proposed framework.

7.1 Conclusions

This thesis introduced a novel inference framework: SMT-guided Metropolis-Hastings (MH). By leveraging the structured search capabilities and deterministic reasoning power of Satisfiability Modulo Theories (SMT), the approach identifies valid traces that satisfy observational constraints and are probable under the model. The framework reformulates the probabilistic inference problem into a constrained optimization problem to propose informative trace configurations to guide the sampling process. Its modular design allows flexible experimentation with likelihood estimators, ensuring adaptability and extensibility.

This work was motivated by the need for a model-specific proposal design that reduces manual effort and reliance on domain expertise typically required for handcrafted inference strategies. Traditional sampling methods often rely on randomized proposals that are uninformed by observational data, leading to low acceptance rates and inefficient exploration through low-probability regions. The aim of this thesis was not only to improve convergence by steering the search towards traces that are both valid and likely under the model, but also to contribute toward automating tailored inference, supporting the growing body of probabilistic programming applications across diverse and increasingly complex domains.

With the proposed framework, this study aimed to investigate whether SMT guidance could effectively improve convergence in MH inference. To this end, two modes of integrating SMT-derived traces into the sampling process were explored: warm-start initialization, which provides informed starting points for inference, and a jumping mechanism, which injects SMT solutions dynamically during inference. In addition, a post-injection strategy based on Gaussian drift was introduced to further refine proposals and boost acceptance rates.

Experimental results demonstrate that warm-starting MH with SMT-derived trace successfully reduces early-stage exploration by providing high-quality initial samples. This leads to a faster convergence and an overall improvement in sampling efficiency. However, when applied as a form of dynamic intervention during inference, on-demand SMT-guided jumps showed limited effectiveness. To preserve MH's convergence guarantees and avoid solver overhead, their frequency had to be restricted. While individual jumps may be impactful, this restriction prevents the method from serving as an active or frequent proposal mechanism throughout inference. Moreover, the implementation struggled to find qualitative traces under added diversity constraints, conditioned on the current trace, resulting in no consistent performance gains. Nonetheless, the post-injection Gaussian drift strategy proved beneficial, assuming a strong initial posterior likelihood. This highlights the importance of trace quality at the time of integration.

Overall, these results suggest that SMT-guidance is effective as an initialization strategy and may also help inform subsequent steps. However, within the current implementation, it remains insufficient as a repeated proposal mechanism.

In summary, this research contributes a new perspective on integrating SMT reasoning methods within probabilistic inference. While dynamic SMT guidance remains a challenge, the demonstrated advantages of warm-start initialization and guided local proposals mark a step forward in automated, structure-aware inference design.

7.2 Future Work

This work contributes to the broader effort toward integrating automatic structure-aware proposal mechanisms into probabilistic inference. By combining reasoning and sampling-based methods, it opens possibilities for future research on hybrid approaches that move beyond purely random proposals and begin to exploit model structure in a principled way. Several promising directions remain for extending the SMT-guided MH framework and addressing the challenges encountered in this study:

Firstly, improving scalability is essential for real-world applications. One direction involves generating partial trace solutions using user-defined heuristics, reducing the number of data points, for example, enabling the solver to focus only on fragments of the original problem. Similarly, removing variables that are conditionally independent given observations can help shrink the search space.

Secondly, alleviating the objective complexity of the constrained inference task is a key priority. This includes developing more efficient likelihood estimators and identifying opportunities to eliminate likelihood components, both of which could reduce the computational burden on the solver.

Thirdly, SMT solving can be made more effective by incorporating additional constraints that direct the solver towards feasible regions of the posterior. For instance, integrating methods such as R2 [Nori et al., 2014] to truncate priors before

search could increase search effectiveness and trace solutions.

Lastly, extending support to a broader range of distributions or providing stochastic support would significantly enhance the generality of the framework. Thanks to its modular design, the framework allows for flexible experimentation with alternative likelihood estimators and the incorporation of prior-specific support restrictions to handle distributions not currently covered. One promising direction is the use of mixture-of-Gaussians approximations to model complex prior distributions. Regarding stochastic support, integrating Divide Conquer and Combine (DCC) [Zhou et al., 2020] framework presents a compelling direction. Since DCC enables specialized inference for single-line probabilistic programs, SMT-guided MH could be introduced within these simplified contexts, where the number of decision points is significantly reduced, potentially rendering solver-guided proposals more tractable and effective.

Bibliography

C. Barrett, C. Barrett, C. Tinelli, and C. Tinelli. Satisfiability modulo theories. *Handbook of Model Checking*, 2018. doi: 10.1007/978-3-319-10575-8_11.

J. Bierkens, P. Fearnhead, and G. Roberts. The zig-zag process and super-efficient sampling for bayesian analysis of big data. *The Annals of Statistics*, 47(3), June 2019. ISSN 0090-5364. doi: 10.1214/18-aos1715. URL http://dx.doi.org/10.1214/18-AOS1715.

E. Bingham, J. P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P. Szerlip, P. Horsfall, and N. D. Goodman. Pyro: deep universal probabilistic programming. *J. Mach. Learn. Res.*, 20(1):973–978, Jan. 2019. ISSN 1532-4435.

N. S. Bjørner and L. Nachmanson. Arithmetic solving in z3. *International Conference on Computer Aided Verification*, 2024. doi: 10.1007/978-3-031-65627-9_2.

A. Bouchard-Côté, S. J. Vollmer, and A. Doucet. The bouncy particle sampler: A non-reversible rejection-free markov chain monte carlo method, 2017. URL https://arxiv.org/abs/1510.02451.

A. Cimatti, A. Cimatti, A. Griggio, A. Griggio, B. J. Schaafsma, B. J. Schaafsma, R. Sebastiani, and R. Sebastiani. The mathsat5 smt solver. *International Conference on Tools and Algorithms for Construction and Analysis of Systems*, 2013. doi: 10.1007/978-3-642-36742-7_7.

A. Cimatti, A. Griggio, A. Irfan, M. Roveri, and R. Sebastiani. Incremental linearization for satisfiability and verification modulo nonlinear arithmetic and transcendental functions. *ACM Trans. Comput. Logic*, 19(3), Aug. 2018. ISSN 1529-3785. doi: 10.1145/3230639. URL https://doi.org/10.1145/3230639.

G. E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition–preliminary report. *SIGSAM Bull.*, 8(3):80–90, Aug. 1974. ISSN 0163-5824. doi: 10.1145/1086837.1086852. URL https://doi.org/10.1145/1086837.1086852.

M. Coman. Smt-guided metropolis-hastings, 2025. URL https://doi.org/10.5281/zenodo.15832694.

M. Cusumano-Towner, A. K. Lew, and V. K. Mansinghka. Automating involutive mcmc using probabilistic and differentiable programming, 2020. URL https://arxiv.org/abs/2007.09871.

M. F. Cusumano-Towner and V. K. Mansinghka. Using probabilistic programs as proposals, 2018. URL https://arxiv.org/abs/1801.03612.

M. F. Cusumano-Towner, F. A. Saad, A. K. Lew, and V. K. Mansinghka. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 221–236, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6712-7. doi: 10.1145/3314221.3314642. URL http://doi.acm.org/10.1145/3314221.3314642.

M. H. A. Davis. Piecewise-deterministic markov processes: A general class of non-diffusion stochastic models. *Journal of the Royal Statistical Society. Series B (Methodological)*, 46(3):353–388, 1984. ISSN 00359246. URL http://www.jstor.org/stable/2345677.

L. De Moura and N. Bjørner. Z3: an efficient smt solver. In *Proceedings of the Theory and Practice of Software, 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, page 337–340, Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3540787992.

M. A. Fischler and R. C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. ISSN 0001-0782. doi: 10.1145/358669.358692. URL https://doi.org/10.1145/358669.358692.

H. Ge, K. Xu, and Z. Ghahramani. Turing: A language for flexible probabilistic inference. In A. Storkey and F. Perez-Cruz, editors, *Proceedings of the Twenty-First International Conference on Artificial Intelligence and Statistics*, volume 84 of *Proceedings of Machine Learning Research*, pages 1682–1690. PMLR, 09–11 Apr 2018. URL https://proceedings.mlr.press/v84/ge18b.html.

A. Geiger, M. Lauer, and R. Urtasun. A generative model for 3d urban scene understanding from movable platforms. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, page 1945–1952, USA, 2011. IEEE Computer Society. ISBN 9781457703942. doi: 10.1109/CVPR.2011.5995641. URL https://doi.org/10.1109/CVPR.2011.5995641.

A. Gelman*, G. O. Roberts**, and W. R. Gilks***. Efficient metropolis jumping rules. In *Bayesian Statistics 5: Proceedings of the Fifth Valencia International Meeting*. Oxford University Press, 05 1996. ISBN 9780198523567. doi: 10.1093/oso/9780198523567.003.0038. URL https://doi.org/10.1093/oso/9780198523567.003.0038.

S. Geman and D. Geman. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(6):721–741, 1984. doi: 10.1109/TPAMI.1984.4767596.

W. R. Gilks, G. O. Roberts, and S. K. Sahu. Adaptive markov chain monte carlo through regeneration. *Journal of the American Statistical Association*, 93(443):1045–1054, 1998. ISSN 01621459, 1537274X. URL http://www.jstor.org/stable/2669848.

N. D. Goodman and A. Stuhlmüller. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014. Accessed: 2025-6-26.

P. J. Green. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995. ISSN 00063444, 14643510. URL http://www.jstor.org/stable/2337340.

H. Haario, E. Saksman, and J. Tamminen. Adaptive proposal distribution for random walk metropolis algorithm. *Comput. Stat.*, 14(3):375–395, Sept. 1999. ISSN 0943-4062. doi: 10.1007/s001800050022. URL https://doi.org/10.1007/s001800050022.

H. Haario, E. Saksman, and J. Tamminen. An adaptive Metropolis algorithm. *Bernoulli*, 7(2):223 – 242, 2001.

T. Hader, D. Kaufmann, A. Irfan, S. Graham-Lengrand, and L. Kovács. Mcsat-based finite field reasoning innbsp;thenbsp;yices2 smt solver (short paper). In *Automated Reasoning: 12th International Joint Conference, IJCAR 2024, Nancy, France, July 3–6, 2024, Proceedings, Part I*, page 386–395, Berlin, Heidelberg, 2024. Springer-Verlag. ISBN 978-3-031-63497-0. doi: 10.1007/978-3-031-63498-7_23. URL https://doi.org/10.1007/978-3-031-63498-7_23.

W. K. Hastings. Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57(1):97–109, 04 1970. ISSN 0006-3444. doi: 10.1093/biomet/57.1.97. URL https://doi.org/10.1093/biomet/57.1.97.

M. Hoffman, D. M. Blei, C. Wang, and J. Paisley. Stochastic variational inference, 2013. URL https://arxiv.org/abs/1206.7051.

L. Ingmar, M. Garcia de la Banda, P. J. Stuckey, and G. Tack. Modelling diversity of solutions. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34 (02):1528–1535, Apr. 2020. doi: 10.1609/aaai.v34i02.5512. URL https://ojs.aaai.org/index.php/AAAI/article/view/5512.

G. Kremer, A. Reynolds, C. Barrett, and C. Tinelli. Cooperating techniques for solving nonlinear real arithmetic in the cvc5 smt solver (system description). In *Automated Reasoning: 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8–10, 2022, Proceedings*, page 95–105, Berlin, Heidelberg, 2022. Springer-Verlag. ISBN 978-3-031-10768-9. doi: 10.1007/978-3-031-10769-6_7. URL https://doi.org/10.1007/978-3-031-10769-6_7.

T. A. Le, A. G. Baydin, and F. Wood. Inference compilation and universal probabilistic programming, 2017. URL https://arxiv.org/abs/1610.09900.

V. Mansinghka, D. Selsam, and Y. Perov. Venture: a higher-order probabilistic programming platform with programmable inference, 2014. URL https://arxiv.org/abs/1404.0099.

Y. V. Matiyasevich. *Hilbert's tenth problem*. MIT Press, Cambridge, MA, USA, 1993. ISBN 0262132958.

V. MAZ'YA and G. SCHMIDT. On approximate approximations using gaussian kernels. *IMA Journal of Numerical Analysis*, 16(1):13–29, 1996. doi: 10.1093/imanum/16.1.13.

D. Merrell and A. Gitter. Inferring signaling pathways with probabilistic programming. *Bioinformatics*, 36(Supplement$_2$) : $i822$–$i830, Dec.2020. ISSN1367 - 4811. doi : . URL http://dx.doi.org/10.1093/bioinformatics/btaa861.

N. Metropolis, N. Metropolis, A. W. Rosenbluth, A. W. Rosenbluth, M. N. Rosenbluth, M. N. Rosenbluth, A. H. Teller, A. H. Teller, E. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 1953. 10.1063/1.1699114.

L. M. Murray and T. B. Schön. Automated learning with a probabilistic programming language: Birch, 2020. URL https://arxiv.org/abs/1810.01539.

R. M. Neal and R. M. Neal. Mcmc using hamiltonian dynamics. *arXiv: Computation*, 2011. 10.1201/b10905-7.

K. Neklyudov, M. Welling, E. Egorov, and D. Vetrov. Involutive mcmc: a unifying framework, 2020. URL https://arxiv.org/abs/2006.16653.

A. V. Nori, A. V. Nori, C.-K. Hur, C.-K. Hur, S. K. Rajamani, S. K. Rajamani, S. Samuel, and S. Samuel. R2: an efficient mcmc sampler for probabilistic programs. *AAAI Conference on Artificial Intelligence*, 2014. 10.1609/aaai.v28i1.9060.

A. V. Nori, A. V. Nori, S. Ozair, S. Ozair, S. K. Rajamani, S. K. Rajamani, D. Vijaykeerthy, and D. Vijaykeerthy. Efficient synthesis of probabilistic programs. *ACM-SIGPLAN Symposium on Programming Language Design and Implementation*, 2015. 10.1145/2737924.2737982.

D. Phan, N. Pradhan, and M. Jankowiak. Composable effects for flexible and accelerated probabilistic programming in numpyro, 2019. URL https://arxiv.org/abs/1912.11554.

D. Richardson. Some undecidable problems involving elementary functions of a real variable. *The Journal of Symbolic Logic*, 33(4):514–520, 1968. ISSN 00224812. URL http://www.jstor.org/stable/2271358.

D. Ritchie, P. Horsfall, and N. D. Goodman. Deep amortized inference for probabilistic programs, 2016. URL https://arxiv.org/abs/1610.05735.

C. P. Robert, C. P. Robert, G. Casella, and G. Casella. Monte carlo statistical methods. *Springer Texts in Statistics*, 1999. 10.1198/tech.2005.s272.

G. O. Roberts, G. O. Roberts, R. L. Tweedie, and R. L. Tweedie. Exponential convergence of langevin distributions and their discrete approximations. *Bernoulli*, 1996. 10.2307/3318418.

N. Smedemark-Margulies, N. Smedemark-Margulies, R. Walters, R. Walters, H. Zimmermann, H. Zimmermann, L. Laird, L. Laird, C. van der Loo, C. van der Loo, N. Kaushik, N. Kaushik, R. S. Caceres, R. S. Caceres, J.-W. van de Meent, and J.-W. van de Meent. Probabilistic program inference in network-based epidemiological simulations. *PLOS Computational Biology*, 2022. 10.1371/journal.pcbi.1010591.

L. Tierney. A note on metropolis-hastings kernels for general state spaces. *The Annals of Applied Probability*, 8(1):1–9, 1998. ISSN 10505164. URL http://www.jstor.org/stable/2667233.

K. S. Turitsyn, M. Chertkov, and M. Vucelja. Irreversible monte carlo algorithms for efficient sampling. *Physica D: Nonlinear Phenomena*, 240(4–5):410–414, Feb. 2011. ISSN 0167-2789. 10.1016/j.physd.2010.10.003. URL http://dx.doi.org/10.1016/j.physd.2010.10.003.

P. Vanetti, A. Bouchard-Côté, G. Deligiannidis, and A. Doucet. Piecewise-deterministic markov chain monte carlo, 2018. URL https://arxiv.org/abs/1707.05296.

M. R. Villescas, B. de Vries, S. Stuijk, and H. Corporaal. Real-time audio processing for hearing aids using a model-based bayesian inference framework. In *Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*, SCOPES '20, page 82–85, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450371315. 10.1145/3378678.3397528. URL https://doi.org/10.1145/3378678.3397528.

C. Wu and C. P. Robert. The coordinate sampler: A non-reversible gibbs-like mcmc sampler, 2019. URL https://arxiv.org/abs/1809.03388.

Y. Zhou, H. Yang, Y. W. Teh, and T. Rainforth. Divide, conquer, and combine: a new inference strategy for probabilistic programs with stochastic support, 2020. URL https://arxiv.org/abs/1910.13324.

Appendix A

Probabilistic model evaluation suite

A.1 Gaussian mixture model

$$
\begin{aligned}
\text{Cluster means:} \quad & \mu_k \sim \text{Uniform}\left(\frac{20(k-1)}{K}, \frac{20k}{K}\right) \\
\text{Cluster assignment:} \quad & z_n \sim \text{Categorical}\left(\left\{\frac{1}{K}, \ldots, \frac{1}{K}\right\}\right) \\
\text{Observations:} \quad & y_n \sim \text{Normal}(\mu_{z_n}, 0.1)
\end{aligned}
$$

A.2 Linear regression (LR)

$$
\begin{aligned}
\text{Slope:} \quad & \text{a} \sim \mathcal{N}(0, 2) \\
\text{Intercept:} \quad & \text{b} \sim \mathcal{N}(0, 2) \\
\text{Observations:} \quad & y_i \sim \mathcal{N}(\text{a} + \text{b}x_n, 0.5)
\end{aligned}
$$

A.3 Linear regression with outliers (OLR)

$$
\begin{aligned}
\text{Slope:} \quad & \text{a} \sim \mathcal{N}(0, 2) \\
\text{Intercept:} \quad & \text{b} \sim \mathcal{N}(0, 2) \\
& \sigma \sim \text{Uniform}(0.1, 1) \\
\text{Outlier probability:} \quad & p_{\text{outlier}} \sim \text{Uniform}(0, 1) \\
\text{Outlier choice:} \quad & o_i \sim \text{Bernoulli}(p_{\text{outlier}}) \\
& \mu_i = x_i \cdot \text{a} + \text{b} \\
\text{Observation:} \quad & y_i \sim \begin{cases} \mathcal{N}(\mu_i, \sigma), & \text{if } o_i = 1 \\ \mathcal{N}(0, 10), & \text{if } o_i = 0 \end{cases}
\end{aligned}
$$

A.4 Gaussians

$$
\begin{aligned}
\text{Mean:} \quad & \mu \sim \mathcal{N}(100, 30) \\
\text{Variance:} \quad & \sigma \sim \text{Uniform}(1, 50) \\
\text{Observations:} \quad & y_n \sim \mathcal{N}(\mu, \sigma)
\end{aligned}
$$

A.5 N-Schools Probabilistic Model

$$
\begin{aligned}
\text{Global baseline:} \quad & \beta_{\text{baseline}} \sim \mathcal{N}(1,\ 50) \\
\text{Hyperparameters:} \quad & \sigma_{\text{state}} \sim \text{Uniform}(0.1,\ 1.0) \\
& \sigma_{\text{district}} \sim \text{Uniform}(0.1,\ 1.0) \\
& \sigma_{\text{type}} \sim \text{Uniform}(0.1,\ 1.0) \\
\text{State effects:} \quad & \beta_s^{\text{state}} \sim \mathcal{N}(0,\ \sigma_{\text{state}}) \quad \text{for } s = 1, \ldots, S \\
\text{District effects:} \quad & \beta_{s,d}^{\text{district}} \sim \mathcal{N}(0,\ \sigma_{\text{district}}) \quad \text{for } s = 1, \ldots, S,\ d = 1, \ldots, D \\
\text{Type effects:} \quad & \beta_t^{\text{type}} \sim \mathcal{N}(0,\ \sigma_{\text{type}}) \quad \text{for } t = 1, \ldots, T \\
\text{Observation model:} \quad & \sigma_i \sim \text{Uniform}(0.5,\ 1.5) \\
& \mu_i = \beta_{\text{baseline}} + \beta_{\text{state}[i]}^{\text{state}} + \beta_{\text{state}[i],\ \text{district}[i]}^{\text{district}} + \beta_{\text{type}[i]}^{\text{type}} \\
& y_i \sim \mathcal{N}(\mu_i,\ \sigma_i) \quad \text{for } i = 1, \ldots, N
\end{aligned}
$$