Coarse-to-Fine Abstract Interpretation for Probabilistic Programs

Anonymous Author(s)
Affiliation
Address
email

Abstract

Many practical techniques for probabilistic inference require a sequence of distributions that interpolate between a tractable distribution and an intractable distribution of interest. Usually, the sequences used are simple, e.g., based on geometric averages between distributions. When models are expressed as probabilistic programs, the models themselves are highly structured objects that can be used to derive annealing sequences that are more sensitive to domain structure. We build on the concept of a Galois connection from traditional abstract interpretation to derive coarse-to-fine sequences of programs. In a sequential Monte Carlo framework, these allow for substantial improvements over existing generic inference algorithms when abstractions match domain structure.

1 Introduction

When faced with a complex reasoning task, what should you do? Often, it helps to take a step back, try to understand the big picture, and then focus on what seems most promising. Working from coarse to fine levels of detail is intuitively appealing; the big picture tends to have fewer moving parts, and its parts tend to be easier to understand. In the context of probabilistic reasoning, approaches following similar principles are known as lifted and coarse-to-fine inference. Such approaches are typically limited to specific applications [e.g., 2, 11, 18]. In this paper, we present a *generic* method for deriving coarse-to-fine sequences of models for *probabilistic programs*.

Probabilistic programs are models expressed in Turing-complete languages that supply primitives for random sampling and probabilistic conditioning [e.g., 7, 12, 17]. For our purposes, the most important property of probabilistic programs is that they expose semantically meaningful model structure in an explicit way. We will describe a method that uses such structure to automatically derive a coarsened version of a program.

We build on tools provided by the field of abstract interpretation [3, 4]. The concept of a *Galois Connection* in particular will allows us to find a unique best abstraction relative to our requirements. In our setting, what it means to *abstract* a distribution is to replace this distribution with a mixture of simpler distributions, possibly incurring some loss— extra entropy—along the way. We can stack such coarsenings and build sequences of more and more abstract models.

We use these sequences in a sequential Monte Carlo (SMC) setting. Sampling strategies that use sequences of distributions from a tractable distribution to a intractable distribution of interest are broadly applied for probabilistic inference [e.g., 15, 16, 19]. For example, annealed importance sampling is one of the best estimators for the marginal likelihood across a wide range of models [9]. Usually, simple annealing paths such as geometric averages between distributions are used [10]. We expect that one can do much better by taking domain structure into account. In particular, we expect that good coarse-to-fine sequences lead to better coverage of regions with high posterior probability, and that they enable more efficient pruning of low-probability regions. A finite set of fine-grained

particles may not cover the entire region, which can lead to a situation where all particles assign low probability to the next filtering step (particle decay). Abstract particles correspond to distributions on fine-grained states, thus each abstract particle can cover a bigger region [18]. Good coarse-to-fine sequences can allow us to prune entire parts of the state space in one go, only considering refinements of abstract states that have sufficiently high posterior probability [11]. We test these intuitions empirically on two examples—a factorial Hidden Markov model and an image clustering model—and sketch abstractions for additional applications.

Our main contributions are as follows: We define abstractions for probability distributions, and show how to optimally extend these abstractions to entire probabilistic programs. We then describe a coarse-to-fine sequential Monte Carlo algorithm that makes use of abstract programs for more efficient particle filtering and MCMC, and experimentally demonstrate the value of this algorithm.

2 Abstract Interpretation for Probabilistic Programs

Abstract interpretation is a tool for gaining information about the execution paths of a program without running each execution individually. We are going to use abstract interpretation to produce a coarse-grained picture of the distribution induced by a model expressed as a probabilistic program. This will support efficient inference by allowing a more informed exploration of the program's state space.

In the following, we first introduce *Galois connections*, one of the major formal approaches to abstract interpretation [3, 4]. We then define abstractions for distributions, show under which conditions a Galois connection holds, and use this fact to optimally extend abstractions to entire programs.

2.1 Background: Galois Connections

Let $\langle L, \leq_L \rangle$ be a lattice representing *concrete values* and $\langle M, \sqsubseteq_M \rangle$ a lattice representing *abstract values*. A pair of maps $\alpha : L \to M$ (*abstraction*) and $\gamma : M \to L$ (*concretization*) is a *Galois connection*, also written as

$$\langle L, \leq_L \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle M, \sqsubseteq_M \rangle$$

if and only if

$$\forall_{l \in L} \forall_{m \in M} \;\; \alpha(l) \sqsubseteq_M m \iff l \leq_L \gamma(m).$$

This is equivalent to the following set of requirements:

1. α and γ are monotonic,
2. $\gamma \circ \alpha$ is extensive: $\forall_{l \in L} \;\; l \leq_L \gamma \circ \alpha \, (l)$
3. $\alpha \circ \gamma$ is reductive: $\forall_{m \in M} \, \alpha \circ \gamma \, (m) \sqsubseteq_M m$, and

If the lattices in question are *complete lattices*, then, for any given concrete function $f : L \to L$, the corresponding *most precise* sound approximation $f' : M \to M$ is uniquely given by $f' = \alpha \circ f \circ \gamma$. We will use this fact to lift functions in a probabilistic program to the abstract domain.

Galois connections compose: if $\langle \alpha, \gamma \rangle$ is a Galois connection between L and M, and $\langle \alpha', \gamma' \rangle$ is a Galois connection between M and N, then $\langle \alpha' \circ \alpha, \gamma \circ \gamma' \rangle$ is a Galois connection between L and N. This will allow us to define a *ladder* of abstractions.

2.2 Abstractions for Distributions

Let $\mathcal{V}$ be a set of values, and $\texttt{Dist}\ \mathcal{V}$ the set of distributions on $\mathcal{V}$. We are looking to coarsen elements in $\texttt{Dist}\ \mathcal{V}$. We will define abstractions for distributions guided by two intuitions: First, abstraction for distributions corresponds to the introduction of hierarchical structure. Second, a good abstraction is one that leads to a minimal increase in entropy with respect to the original state space.

Consider an abstraction function $\alpha : \texttt{Dist}\ \mathcal{V} \to \texttt{Dist}\ \mathcal{V}'$, with $\mathcal{V}' \subseteq \texttt{Dist}\ \mathcal{V}$. This kind of abstraction takes a flat distribution and clusters it by turning it into a distribution on distributions. For concretization, we use the function $\gamma : \texttt{Dist}\ \mathcal{V}' \to \texttt{Dist}\ \mathcal{V}$ that mixes a distribution on distributions into a flat distribution on concrete values: $\gamma(d') = \sum_{(d_i, p_i) \in d'} p_i d_i$.

Listing (1) Base program	Listing (2) Lifted to distributions	Listing (3) Abstract program
`D = [(1,2,3),(8/10,1/10,1/10)]`	`D = [(1,2,3),(8/10,1/10,1/10)]`	$A = [(1,2),(8/9,1/9)], B = \delta(3)$
`x = sample(D)`	`x = δ(sample(D))`	`x = δ(sample([(A,B),(9/10,1/10)]))`
`y = 3`	`y = δ(3)`	`y = δ(B)`
$x + y$	$x +_{\texttt{Dist}} y$	$\alpha(\gamma(x) +_{\texttt{Dist}} \gamma(y))$

Figure 1: Deriving abstract programs from concrete programs. We lift to distributions, then apply the abstraction operator α to all distributions, and lift primitives using $f' = \alpha \circ f \circ \gamma$.

Let $\leq_H$ be the ordering by entropy, *i.e.,* $P \leq_H Q \iff H(P) \leq H(Q)$. Let $\leq_{H'}$ be the corresponding ordering of flattened distributions, *i.e.,* $P \leq_{H'} Q \iff H(\gamma(P)) \leq H(\gamma(Q))$.

With these definitions out of the way, we propose the following Galois connection between concrete and abstract distributions:

$$\langle \texttt{Dist}\ \mathcal{V}, \leq_H \rangle \underset{\alpha}{\overset{\gamma}{\leftrightarrows}} \langle \texttt{Dist}\ \mathcal{V}', \leq_{H'} \rangle \tag{1}$$

For this to be a Galois connection, requirements 1-3 in Section 2.1 need to be satisfied. γ is trivially monotone since $\leq_{H'}$ is defined in terms of γ. This leaves the following requirements for α: (1) α is monotone in H, (2) $\forall d \in \texttt{Dist}\ \mathcal{V}\ H(d) \leq H'(\alpha(d))$, *i.e.,* α cannot reduce entropy, and (3) $\forall d' \in \texttt{Dist}\ \mathcal{V}'\ H(\alpha(\gamma(d'))) \leq H'(d')$, *i.e.,* α cannot increase entropy for distributions in the range of γ (but can otherwise).

The lattices $\langle \texttt{Dist}\ \mathcal{V}, \leq_H \rangle$ and $\langle \texttt{Dist}\ \mathcal{V}', \leq_{H'} \rangle$ are complete: for any subset, the supremum (infimum) is given by the highest-entropy (lowest-entropy) distribution in the subset. Hence, for functions f that are monotone in entropy, $f' = \alpha \circ f \circ \gamma$ is the abstraction that least increases entropy.

We will distinguish two types of abstractions: locally lossless abstractions and lossy abstractions. A locally lossless abstraction is an abstraction such that $\forall d \in \texttt{Dist}\ \mathcal{V} : \gamma(\alpha(d)) = d$, *i.e.,* α and γ are inverses of each other. For example, consider a distribution $D = [(1, 8/10), (2, 1/10), (3, 1/10)]$. An abstraction function that groups values as $\{\{1,2\},\{3\}\}$ without loss would map d onto $[(A, 9/10), (B, 1/10)]$ with $A = [(1, 8/9), (2, 1/9)]$ and $B = [(3, 1/1)]$. By contrast, a lossy *maximum entropy* abstraction function would require that $A = [(1, 1/2), (2, 1/2)]$.

2.3 Abstractions for Probabilistic Programs

A probabilistic program is a program in a Turing-complete language that may include `sample` and `condition` expressions. A `sample` expression is of the form `sample(`d`)` with $d \in \texttt{Dist}\ \mathcal{V}$. The value of `sample(`$d$`)` is a sample from distribution d. A `condition` expression is of the form `condition(`e`)` where e is an expression that evaluates to a Boolean. `condition` rules out all program executions where e does not evaluate to `true`.

Probabilistic programs operate on values $\mathcal{V}$, whereas abstractions are defined on $\texttt{Dist}\ \mathcal{V}$. To apply abstractions to programs, we therefore first lift programs from values to distributions. This initial lifting Θ does not affect the essential behavior of the program; it simply makes the operators and relations sensitive to probabilities by transforming them to act over distributions, and is performed by *a*) lifting the results of `sample` expressions to δ-distributions, *b*) lifting all remaining constants to δ-distributions, and *c*) lifting all primitive operators to operate on distributions.

For example, consider the program shown in Listing 1 and its lifted version in Listing 2. Lifting addition to operate on distributions D_x and D_y results in the function $D_x +_{\texttt{Dist}} D_y = [(x + y, p_x p_y) \mid (x, p_x) \in D_x, (y, p_y) \in D_y]$. Θ does *not* replace sampling with deterministic distribution values—in the lifted program, a `sample` expression returns a δ-distribution on a randomly chosen value.

Once lifted, we derive an abstract program by applying α to all distributions, and by lifting all primitive operators f using $f' = \alpha \circ f \circ \gamma$. The value domain of the abstract program is $\texttt{Dist}\ \mathcal{V}'$.

The preceding abstraction mechanism defines the semantics of the abstract program. For efficient implementation, it is critical that abstract primitives are *not* computed at run-time using α and γ whenever they are encountered. We use caching and approximations to keep the computation in the abstract domain. This, in turn, motivates lossy abstractions—if abstractions are lossless, the number

of elements in the abstract domain is generally too large for efficient caching or approximation to be feasible.

3 Coarse-to-Fine Sequential Monte Carlo

In the previous section, we have seen how to generate abstract programs P_1, P_2, ..., P_N from a given concrete probabilistic program P_0. We now describe how to use this sequence of programs for efficient inference.

As usual in Bayesian inference, the program P_0 defines a joint distribution $p(x, y) = p(x)p(y|x)$ on latent variables x and observed variables y. Typical problems of interest are sampling $p(x|y) \propto p(x)p(y|x) = p^*(x)$ and estimating expectations, in particular the partition function $p(y) = \sum_x p(x)p(y|x)$.

We will use importance sampling and its multi-stage sibling, sequential Monte Carlo. Suppose our target distribution is X with probability mass function p. Importance sampling generates samples from an approximating distribution Y (with probability mass function q) and re-weights the samples to account for the difference between true and approximating using $w(x) = p(x)/q(x)$. To compute estimates of $\psi = \mathbb{E}_{x\sim X}[f(x)]$ given samples $y_1, \ldots, y_n$ we use $\hat{\psi} = \sum_{i=1}^n w(y_i)f(y_i)/\sum_{i=1}^n w(y_i)$. To generate approximate samples from $p(x)$, we resample from the set of samples in proportion to the importance weights.

3.1 Coarse-to-Fine importance sampling

Suppose we can sample exactly from a coarsened program P_1. Let $\phi : \texttt{Dist}\ \mathcal{V} \to \mathcal{V}$ be the function that randomly samples from a given distribution. Since abstract states correspond to distributions on concrete states, we can stochastically *instantiate* them—*i.e.,* push them to the next-lower level—using ϕ. In particular, when $p(x) = \prod_{x^j} p(x^j)$, we can simply instantiate all latent variables independently[1].

Our importance distribution is given by the two-stage process $x_1 \sim P_1, x_0 \sim \Phi(x_1)$. We refer to the probability of sampling x_1 from X_1 as $q_{X_1}(x_1)$, and to the instantiation probability as $q_\Phi(x_0|x_1)$.

The importance weight for a sample from this process is

$$w(x_0) = \frac{p^*(x_1)}{q_{X_1}(x_1)q_\Phi(x_0|x_1)}. \tag{2}$$

For lossless abstractions, $p(x_0) = q(x_1)q_\Phi(x_0|x_1)$. In this case, the weight simplifies to $w(x_0) = p(y_0|x_0)/p(y_1|x_1)$.

Note that, to compute $p(y|x)$ and $p(y'|x')$, we need to run the abstract and concrete probabilistic programs.

3.2 Coarse-to-Fine Sequential Monte Carlo

The above process naturally extends to the case of multiple levels of abstractions and can be augmented by applying MCMC transition operators on each level. Let $w_i(x)$ refer to the weight between the programs P_i and P_{i+1}. Listing **??** shows the Sequential Monte Carlo algorithm. We expect this to work well when abstractions are chosen such that the KL divergence between initial program P_N and target distribution P_0 is broken into many similarly spaced steps, and when abstractions match the domain structure such that they provide informative summaries that allow early pruning of parts of the state space.

Abstractions tend to reduce the dependence between the inputs and outputs of deterministic primitives. Moving from a coarse to a fine program incrementally re-introduces these dependencies, which can help inference in programs with strong determinism.

[1] In the general case of dependent random choices, instantiation and program execution need to be interleaved.

Algorithm 1: Coarse-to-Fine Sequential Monte Carlo

Require: Number of particles N, number of abstraction levels K

for $i = 1$ **to** N **do**
  $x_K^{(i)} \leftarrow$ run program p_K
end for
for $k = K$ **to** 1 **do**
  for $i = 1$ **to** N **do**
    Apply MCMC rejuvenation steps to $x_k^{(i)}$
    $x_{k-1}^{(i)} \leftarrow$ sample from $\Phi(x_k^{(i)})$
    $w^{(i)} \leftarrow \frac{p(x_{k-1}^{(i)})}{p(x_k^{(i)})p(\Phi(x_k^{(i)})=x_{k-1}^{(i)})}$
  end for
  Resample $x_{k-1}^{(1:N)}$ in proportion to $w^{(1:N)}$
end for
return $x_0^{(1:N)}$

4 Examples

In this section, we give examples of applications where abstractions are useful. We apply the coarse-to-fine SMC algorithm to two models, a factorial HMM and an image clustering model. Both examples were chosen such that their state space can easily be scaled up, such that a deep hierarchy of abstractions is possible, and such that model dynamics suggest a natural coarsening. In Section 4.3, we discuss other types of coarsening and to what extent they can and cannot be seen as members of the class of abstract interpretations discussed in this document.

4.1 Factorial HMM: Coarsening States and Observations

In our first example, we test the hypothesis that abstractions are useful as a means to avoid particle collapse in large state spaces. For this reason, we chose the factorial HMM, a model with a large effective state size even within a single particle filter step. The Factorial HMM is a HMM where the state factors into multiple variables [6]. If there are M possible values for each latent state, and k state variables per time step, then the effective state size is M^k. If only few of these have high probability, then even for moderate M and k it is possible that there are not sufficiently many fine-grained particles to cover all regions of high posterior probability.

We coarsen the factorial HMM by merging some state and observation symbols. To test the hypothesis that coarse-to-fine inference will work best when abstractions match the dynamics of the model, we generate transition and observation matrices with approximately hierarchical structure as follows. Enumerate state 1 to N. For states i and j, we let the transition probability be approximately proportional to $2^{-|i-j|}$. Similarly, for state i, the probability of generating observation k is proportional to $2^{-|i-k|}$.

In our first experiment, we use a factorial HMM with 3 variables per step, 32 possible state values per variable, and 5 observed time steps. We run both coarse-to-fine filtering (with 5 abstraction levels) and flat filtering for 60 seconds, each with 100 particles per run. We estimate marginals from particles, and compute the error in marginals by comparing against ground truth calculated using the Forward-Backward algorithm. Figure 2 shows that the error consistently declines more quickly for coarse-to-fine filtering than for standard filtering.

In our second experiment (Figure 3), we increase the number of possible state values per variable to 128. We vary the number of abstraction levels used in the coarse-to-fine algorithm from 0 (making it equivalent to the detailed particle filter) to $\log_2 128 = 7$. As the number of abstraction levels increases, the error in estimated marginals after a fixed inference time of 60 seconds decreases.

In our third experiment (Figure 4), we compare the behavior of flat and coarse-to-fine filtering as the number of HMM states per variable increases from 2 to 128. To make the overall task more difficult, we use 5 variables per time step in this experiment. For low numbers of states, flat filtering is more

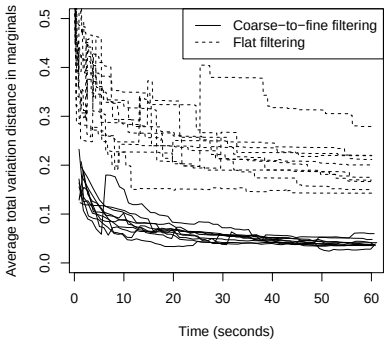

Figure 2: For a factorial HMM with 3 variables per step, 32 values per variable, the error in estimated marginals decreases more quickly for coarse-to-fine filtering than for flat filtering.

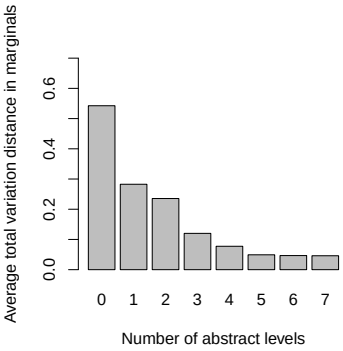

Figure 3: If we keep inference time fixed while we increase the number of abstraction levels, we observe a lower error when using more levels of abstractions.

efficient than coarse-to-fine filtering, but as the number of states goes beyond 8, the error for flat filtering increases quickly.

To make computation at the abstract level efficient, we cache all operations on abstract distributions such that we never touch the concrete domain (through concretization) when evaluating a program on a abstract level. This strategy is not obviously tractable—the number of abstract distributions could be too large. In our final HMM experiment (Figure 5), we compare locally lossless and maximum entropy coarsening with respect to how many distinct abstract distributions are encountered (and cached). We expected to find that this number increases substantially more quickly for lossless coarsening, but in fact we find that the two numbers are almost equal, and both tractable for the model sizes explored above.

Another technique required to make abstractions useful in this domain is *incremental instantiation*: when we instantiate an abstract distribution at the next-lower level, we do not instantiate all time steps at once. Instead, we instantiate each level observation by observation, just as traditional particle filtering does for the most concrete level.

4.2 Image Clustering: Coarsening Matrices

In the previous example, we exclusively used particle filtering without MCMC steps, *i.e.,* not SMC. The near-deterministic structure of the underlying model would have made mixing difficult for MCMC. We now consider an example with a large, relatively smooth state space where MCMC can be applied, but where single-site proposals on the original program are inefficient due to the large number of changes necessary to move between modes. Two common moves in such scenarios are to (1) change the underlying model and (2) to hand-engineer MCMC proposals. We explore test whether abstract interpretation can expose domain structure such that we can semi-automatically derive proposals that allow moves between previously distant program states.

Listing 4 shows the generative model. We draw a $N \times N$ prototype image for each category, a category for each observation, and then condition on each observation being a noisy observation of the category's prototype. We generate a dataset of 5 vertical gradients and 5 horizontal gradients. We condition the generative process on these 10 images and infer (a) the prototype for each category and (b) the category assignment for each image.

For this example, the process of abstraction involves merging four matrix pixels into a single "superpixel". To make computations in the abstract level more efficient, we avoid the concretization-abstraction step and approximate this step by directly applying primitives in the abstract domain.

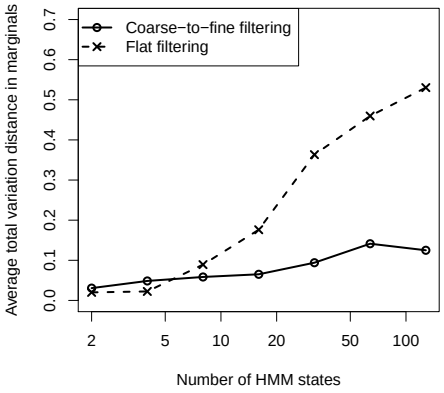

Figure 4: As the number of states grows, the error for flat filtering increases more quickly than for coarse-to-fine filtering.

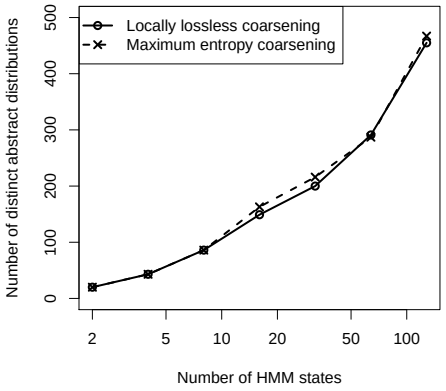

Figure 5: Perhaps surprisingly, the number of distinct abstract distributions is almost identical for locally lossless and maximum-entropy coarsening.

```
(query
 (define num-examples (length observed-data))
 (define num-categories 2)
 (define prototypes (replicate (sample images-prior) num-categories))
 (define categories (replicate (sample categories-prior) num-examples))
 (list prototypes categories)
 (for-each
  (lambda (category data)
   (condition (multivariate-normal (list-ref prototypes category) sigma)
       data))
  categories observed-data))
```

Listing 4: A probabilistic program for a two-category mixture model for images

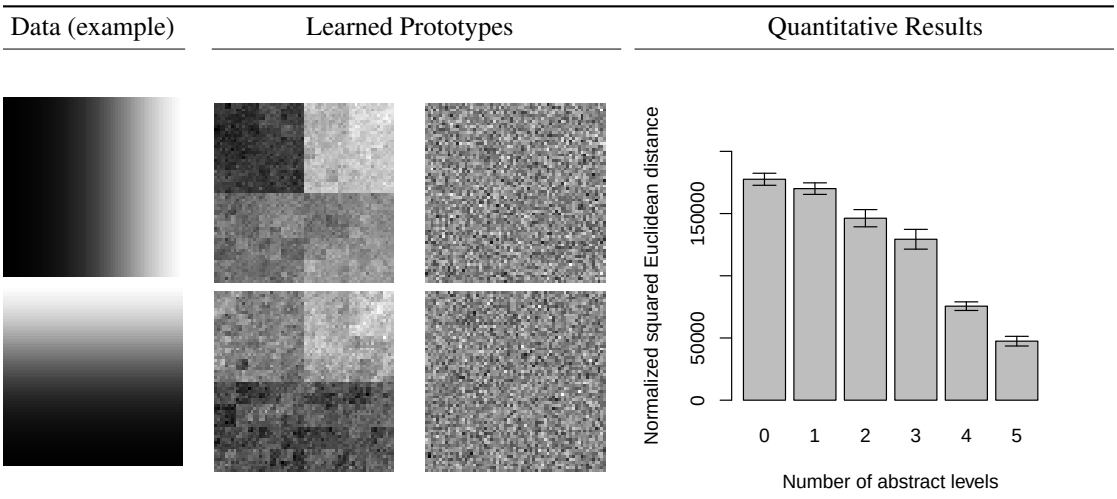

Figure 6: **Left:** Examples of elements from the dataset together with prototypes inferred from the dataset by coarse-to-fine MCMC (grid artefacts) and flat MCMC (noise). **Right:** The more abstraction levels we use, the more accurately we recover ground truth prototypes.

Fix inference time T (todo: what time?). Figure 6 (right) shows that the overall error between true and inferred prorotype is smaller when more abstraction levels are used. This supports our hypothesis that abstractions allow moves in program state space that can help MCMC mix.

4.3 Other Examples and Non-Examples

We now discuss a few additional examples to build intuition for which kinds of abstractions are a good fit for the framework described in this paper.

Model truncation Consider a probabilistic program with stochastic recursion, *e.g.,* a recursive definition of the geometric distribution. In this case, we might want to simplify the model by truncating its depth, *i.e.,* stopping recursion and sampling from a default distribution when a certain depth is reached. This can be accomplished using a type $R(v, i)$ with two fields, a value field v and a recursion depth indicator i. The abstraction mapping can then map elements of this type to uninformative distributions when $i > i_{\max}$.

Coarsening physical dynamics Given a physics simulator that simulates the collision for many pairs of objects (chosen from a potentially large and nonhomogeneous set S), how can we coarsen this process? One option is to map all options onto an uninformative distribution on S, and to use the resulting "average" collision dynamics.

Data truncation: Can we view standard particle filtering where observations are introduced one-by-one as an instance of coarse-to-fine abstract interpretation? It seems that this is not a natural instance of our framework, since easy ways to accomplish this would locally reduce entropy (*e.g.,* by deterministically mapping a and b in `condition(`a`==`b`)` to the same object).

5 Discussion and Related Work

Relation to hierarchical models The coarse-to-fine algorithm presented in this paper is at the same time orthogonal and interestingly related to hierarchical Bayesian models. It is orthogonal from the perspective of computational efficiency: while hierarchical models can improve statistical efficiency by sharing information, they leave computational efficiency untouched—at each inference step, it is still necessary to execute the model computation in all detail. By contrast, our algorithm can reduce this cost by executing some of the computation purely on more abstract levels. From another point of view, the algorithm does interact with hierarchical models: such models provide one source for an abstraction hierarchy. To see this, imagine that each value generated by a hierarchical model is

tagged with the values of all of the random choices in the provenance of the value, *i.e.,* all choices that played a causal role in generating this value. We can then derive a coarsening by "forgetting" the more concrete parts of a value and using an equivalence class that only considers a prefix of the more abstract parts of the dependencies.

Related work The work in this paper is related to and inspired by a broad background of work on coarse-to-fine and lifted inference, including (but not limited to) work by Charniak et al. [2] on coarse-to-fine inferene in PCFGS, work on coarse-to-fine inference for first-order probabilistic models by Kiddon and Domingos [11], and attempts to "fatten" particles for filtering with broader coverage [13, 18]. We propose an approach to the problem of selecting annealing paths that has been posed and partially addressed in (e.g.) Grosse et al. [10], Neal [16]. Outside of machine learning, we take inspiration from *Approximate Bayesian Computation* (conditioning on summary statistics) and *renormalization group* approaches to inference in Ising models (cite). In future work, it may turn out fruitful to understand the relation to probabilistic abstract interpretation Cousot and Monerau [5], Monniaux [14].

Future work. In general, there are many possible abstraction operators, and thus many abstract programs for any given concrete program P. It may not be clear a priori which is best. We can conceptualize this as a directed graph of annealing paths that all end at P (*Graph SMC*). Every single path in this graph corresponds to a valid SMC algorithm. Given such a graph, can we learn online how to focus computation on high-quality paths, *e.g.,* by updating path weights? One way to approach this is to look at the importance weights that resulted from all samples that passed through a node V and to treat the (log of the) average importance weight—a stochastic lower bound on $\log(Z)$ [8]—as a reward signal, updating using a multiplicative-weights style algorithm [1].

Finally, we expect that the most interesting applications of abstract interpretation for efficient inference are yet to come. This approach has most to say for models with rich deterministic structure. Can we use abstract interpretation to provide partial credit in the setting of learning symbolic rules and programs from data—a problem that is known to be difficult to approximate?

References

[1] S. Arora, E. Hazan, and S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory of Computing*, 8(1):121–164, 2012.

[2] E. Charniak, M. Johnson, M. Elsner, J. Austerweil, D. Ellis, I. Haxton, C. Hill, R. Shrivaths, J. Moore, and M. Pozar. Multilevel coarse-to-fine PCFG parsing. *Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 168–175, 2006.

[3] P. Cousot and R. Cousot. Static determination of dynamic properties of programs. In *Proceedings of the second International Symposium on Programming*, pages 106–130. Paris, 1976.

[4] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

[5] P. Cousot and M. Monerau. Probabilistic abstract interpretation. In *Programming Languages and Systems*, volume 7211 of *Lecture Notes in Computer Science*, pages 169–193. Springer, 2012.

[6] Z. Ghahramani and M. I. Jordan. Factorial Hidden Markov Models. *Machine Learning*, 29(2-3):245–273, Nov. 1997.

[7] N. D. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence*, page 220–229, 2008.

[8] R. Grosse. Unbiased estimators of partition functions are basically lower bounds, Jan. 2013. URL https://hips.seas.harvard.edu/blog/2013/01/14/unbiased-estimators-of-partition-functions-are-basically-lower-bounds/.

[9] R. B. Grosse. *Model Selection in Compositional Spaces*. PhD thesis, Massachusetts Institute of Technology, 2014.

[10] R. B. Grosse, C. J. Maddison, and R. Salakhutdinov. Annealing between distributions by averaging moments. In *Advances in Neural Information Processing Systems*, pages 2769–2777, 2013.

[11] C. Kiddon and P. Domingos. Coarse-to-fine inference and learning for first-order probabilistic models. In *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, pages 1049–1056, Aug. 2011.

[12] D. Koller, D. McAllester, and A. Pfeffer. Effective bayesian inference for stochastic programs. In *AAAI/IAAI*, pages 740–747, 1997.

[13] T. D. Kulkarni, A. Saeedi, and S. Gershman. Variational particle approximations. *CoRR*, abs/1402.5715, 2014.

[14] D. Monniaux. Abstract interpretation of probabilistic semantics. In *Static Analysis*, volume 1824 of *Lecture Notes in Computer Science*, pages 322–339. Springer, 2000.

[15] R. M. Neal. Sampling from multimodal distributions using tempered transitions. *Statistics and computing*, 6(4):353–366, 1996.

[16] R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

[17] A. Pfeffer. 14 the design and implementation of ibal: A general-purpose probabilistic language. *Introduction to statistical relational learning*, page 399, 2007.

[18] J. Steinhardt and P. Liang. Filtering with abstract particles. In *Proceedings of The Thirty-First International Conference on Machine Learning*, pages 727–735, June 2014.

[19] R. H. Swendsen and J.-S. Wang. Replica monte carlo simulation of spin glasses. *Physical Review Letters*, 57(21):2607–2609, 1986.