Divide, Conquer, and Combine: a New Inference Strategy for Probabilistic Programs with Stochastic Support

Yuan Zhou [1] **Hongseok Yang** [2] **Yee Whye Teh** [3] **Tom Rainforth** [3]

Abstract

Universal probabilistic programming systems (PPSs) provide a powerful framework for specifying rich probabilistic models. They further attempt to automate the process of drawing inferences from these models, but doing this successfully is severely hampered by the wide range of non–standard models they can express. As a result, although one can specify complex models in a universal PPS, the provided inference engines often fall far short of what is required. In particular, we show that they produce surprisingly unsatisfactory performance for models where the support varies between executions, often doing no better than importance sampling from the prior. To address this, we introduce a new inference framework: Divide, Conquer, and Combine, which remains efficient for such models, and show how it can be implemented as an automated and generic PPS inference engine. We empirically demonstrate substantial performance improvements over existing approaches on three examples.

1. Introduction

Probabilistic programming systems (PPSs) provide a flexible platform where probabilistic models are specified as programs and inference procedures are performed in an automated manner. Some systems, such as BUGS (Tesauro et al., 2012) and Stan (Carpenter et al., 2017), are primarily designed around the efficient automation of a small number of inference strategies and the convenient expression of models for which these inference strategies are suitable.

Universal PPSs, such as Church (Goodman et al., 2008), Venture (Mansinghka et al., 2014), Anglican (Wood et al., 2014), and Pyro (Bingham et al., 2018), on the other hand, are set up to try and support the widest possible range of models a user might wish to write. Though this means that such systems can be used to write models which would be otherwise difficult to encode, this expressiveness comes at the cost of significantly complicating the automation of inference. In particular, models may contain random variables with mixed types or have varying, or even unbounded, dimensionalities; characteristics which cause significant challenges at the inference stage.

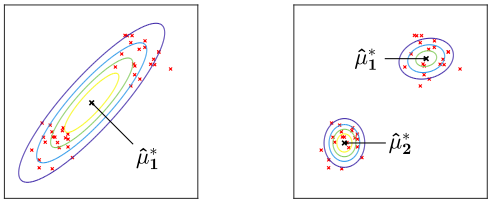


Figure 1. MAP estimate of the means and covariances of a Gaussian mixture model in the cases of $K = 1$ and $K = 2$ clusters. If K is itself random, the model has stochastic support as the parameters of the second cluster, e.g. its mean, only exist when $K = 2$.

In this paper, we aim to address one of the most challenging model characteristics: variables whose very existence is stochastic, often, though not always, leading to the overall dimensionality of the model varying between realizations. Many practical models posses this characteristic. For example, many models contain a variable controlling an allowed number of states, such as the number of clusters in a mixture model (see Figure 1, Richardson & Green 1997; Nobile & Fearnside 2007), or the number of states in a HMM or change point model (Fox et al., 2008). More generally, many inference problems involve some sort of Bayesian model averaging where the constituent models do not share the exact same set of parameters. Other models are inherently defined on spaces with non-static support, such as probabilistic context free grammars (PCFGs) (Manning et al., 1999), program induction models (Perov & Wood, 2014), kernel or function induction models (Schaechtle et al., 2016; Janz et al., 2016), many Bayesian non-parametric models (Roy et al., 2008; Teh, 2010), and a wide range of simulator–based models (Le et al., 2016; Baydin et al., 2019).

[1]Department of Computer Science, University of Oxford, United Kingdom [2]School of Computer Science, KAIST, South Korea [3]Department of Statistics, University of Oxford, United Kingdom. Correspondence to: Yuan Zhou <yuan.zhou@cs.ox.ac.uk>.

Proceedings of the 37^{th} International Conference on Machine Learning, Online, PMLR 119, 2020.

These models can be easily expressed in universal PPSs via branching statements, stochastic loops, or higher-order functions (see e.g. Figure 2). However, performing inference in them is extremely challenging, with the desire of PPSs to automate this inference complicating the problem further.

A number of automated inference engines have been proposed to provide consistent estimates in such settings (Wingate et al., 2011; Wood et al., 2014; Tolpin & Wood, 2015; Ge et al., 2018; Bingham et al., 2018). However, they usually only remain effective for particular subclasses of problems. In particular, we will show that they can severely struggle on even ostensibly simple problems, often doing no better, and potentially even worse, than importance sampling from the prior.

To address these shortfalls, we introduce a completely new framework—***Divide, Conquer, and Combine*** (DCC)—for performing inference in such models. DCC works by ***dividing*** the program into separate straight-line sub-programs with fixed support, ***conquering*** these separate sub-programs using an inference strategy that exploits the fixed support to remain efficient, and then ***combining*** the resulting sub-estimators to an overall approximation of the posterior. The main motivation behind this approach is that the difficulty in applying Markov chain Monte Carlo (MCMC) strategies to such programs lies in the transitioning between variable configurations; within a given configuration efficient inference might still be challenging, but will be far easier than tackling the whole problem directly. Furthermore, this approach also allows us to introduce meta-strategies for allocating resources between sub-programs, thereby explicitly controlling the exploration-exploitation trade-off for the inference process in a manner akin to Rainforth et al. (2018); Lu et al. (2018). To demonstrate its potential utility, we implement a specific realization of our DCC framework as an automated and general-purpose inference engine in the PPS Anglican (Wood et al., 2014), finding that it is able to achieve substantial performance improvements and tackle more challenging models than existing approaches.

To summarize, our key contributions are as follows:

- We highlight shortcomings of existing PPS inference engines in the presence of stochastic support;
- We introduce a new framework, DCC, for performing inference in such problems;
- We implement a specific realization of this DCC framework in the PPS Anglican.

2. Probabilistic Programming

In universal PPSs (Goodman et al., 2008; Mansinghka et al., 2014; Wood et al., 2014; Bingham et al., 2018; Ge et al., 2018), a program may denote a model with varying support: different realizations of a program may lead to different sets of variables being sampled. For example, in Figure 2,

```
1  (let [z0 (sample (normal 0 2))
2        y1 9]
3    (if (< z0 0)
4      (let [z1 (sample (normal -5 2))]
5        (observe (normal z1 2) y1)
6         [z0 z1])
7      (let [z2 (sample (normal 5 2))
8            z3 (sample (normal z2 2))]
9        (observe (normal z3 2) y1))
10       [z0 z2 z3])))
```

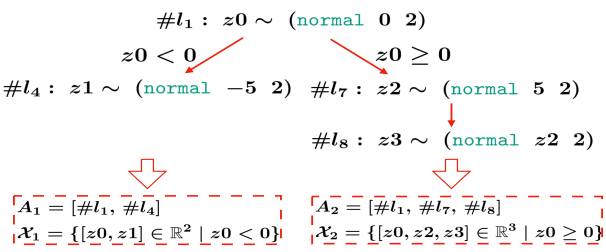


Figure 2. Example program with stochastic support (top) and its possible execution traces (bottom). The two branches of the **if** each produce a different sample path, denoted as A_1 and A_2, and each have a different supports denoted as $\mathcal{X}_1$ and $\mathcal{X}_2$ respectively.

there are two possible variable configurations, $[\texttt{z0}, \texttt{z1}]$ and $[\texttt{z0}, \texttt{z2}, \texttt{z3}]$, due to the stochastic control-flow.

To aid exposition and formalize these programs, we will use the formalization of the particular universal PPS Anglican (Wood et al., 2014; Tolpin et al., 2016), but note that our ideas are applicable to other universal PPSs. Anglican extends the syntax of Clojure with two special forms: **sample** and **observe**, between which the distribution of the program is defined. **sample** statements are used to draw random variables, while **observe** statements are used to condition on data. Informally, they can be thought of as prior and likelihood terms, respectively.

The density of an Anglican program is derived by executing it in a forward manner, drawing from **sample** statements when encountered, and keeping track of density components that originate from the **sample** and **observe** terms. Specifically, let $\{x_i\}_{i=1}^{n_x} = (x_1, \ldots, x_{n_x})$ represent the random variables generated from the encountered **sample** statements, where the i^{th} statement among them has lexical program address a_i, input η_i, and density $f_{a_i}(x_i|\eta_i)$. Analogously, let $\{y_j\}_{j=1}^{n_y} = (y_1, \ldots, y_{n_y})$ represent the observed values of the n_y **observe** statements encountered during execution, which have lexical addresses b_j and corresponding densities $g_{b_j}(y_j|\phi_j)$, where ϕ_j is analogous to η_i. The density is now given by $\pi(x) = \gamma(x)/Z$ where

$$\gamma(x) := \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta_i) \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi_j), \tag{1}$$

$$Z := \int \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta_i) \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi_j) dx_{1:n_x}, \tag{2}$$

and the associated reference measure is implicitly defined through the executed `sample` statements. Note that everything here (i.e. n_x, n_y, $x_{1:n_x}$, $y_{1:n_y}$, $a_{1:n_x}$, $b_{1:n_y}$, $\eta_{1:n_x}$, and $\phi_{1:n_y}$) is a random variable, but each is deterministically calculable given $x_{1:n_x}$ (see §4.3.2 of Rainforth (2017)).

From this, we see that it is sufficient to denote an ***execution trace*** (i.e. realization) of an Anglican program by the sequence of the addresses of the encountered `sample` statements and the corresponding sampled values, namely $[a_i, x_i]_{i=1}^{n_x}$.[1] For clarity, we refer to the sequence $a_{1:n_x}$ as the ***path*** of an execution trace and $x_{1:n_x}$ as the ***draws***. A program with ***stochastic support*** can now be more formally defined as one for which the path $a_{1:n_x}$ varies between realizations: different values of the path correspond to different ***configurations*** of variables being sampled.

3. Shortcomings of Existing Inference Engines

In general, existing inference engines that can be used for (at least some) problems with stochastic support can be grouped into five categories: importance/rejection sampling, particle based inference algorithms (e.g. SMC, PG, PIMH, PGAS, IPMCMC, RM-SMC, PMMH, SMC^2), MCMC approaches with automated proposals (e.g. LMH, RMH), MCMC approaches with user–customized proposals (e.g. RJMCMC), and variational approaches (VI, BBVI). More details on each are provided in Appendix A.

Importance/rejection sampling approaches can straightforwardly be applied in stochastic support settings by using the prior as the proposal, but their performance deteriorates rapidly as the dimensionality increases. Particle–based approaches offer improvements for models with sequential structure (Wood et al., 2014), but become equivalent to importance sampling when this is not the case.

Some variational inference (VI) approaches can, at least in theory, be used in the presence of stochastic support (Bingham et al., 2018; Cusumano-Towner et al., 2019), but this can require substantial problem–specific knowledge to construct a good guide function, requires a host of practical issues to be overcome (Paige, 2016, Chapter 6), and is at odds with PPSs desire for automation. Furthermore, these approaches produce inconsistent estimates and current engines often give highly unsatisfactory practical performance as we will show later. When using such methods, it is common practice to side–step the complications originating from stochastic support by approximating the model with a simpler one with fixed support. For example, though Pyro can ostensibly support running VI in models with stochastic support, their example implementation of a Dirichlet process mixture model (`https://pyro.ai/examples/dirichlet_process_mixture.html`) uses fixed support approximations by assuming a bounded number of mixture components before stripping clusters away.

[1] Strictly speaking, the addresses a_i can be deterministically derived from the sampled values for a given program. However, for our purposes it will be convenient to think about first sampling the path $a_{1:n_x}$ and then sampling $x_{1:n_x}$ conditioned on this path.

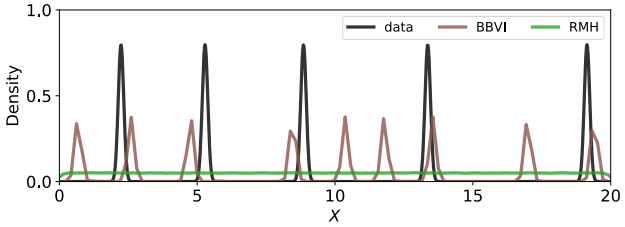


Figure 3. Kernel density estimation of the synthetic data (black) of the univariate mixture model and the posterior predictive distribution from BBVI (brown) and RMH (green) in Anglican.

Because of these issues, arguably the current go–to approaches for programs with stochastic support are specialized MCMC approaches (Wingate et al., 2011; Roberts et al., 2019). However, these are themselves far from perfect and, as we show next, they often given performance far worse than one might expect.

A demonstrative example Consider the following simple mixture model with an unknown number of clusters K

$$K \sim \text{Poisson}(9)+1,\ \mu_k \sim \text{Uniform}\left(\frac{20(k-1)}{K}, \frac{20k}{K}\right),$$
$$z_n \sim \text{Cat}(\{1/K, ..., 1/K\}),\ y_n \sim \mathcal{N}(\mu_{z_n}, 0.1^2).$$

Here $\mu_{1:K}$ are the cluster centers, $z_{1:N}$ are the cluster assignments, and $y_{1:N}$ is the observed data. When conducting inference, we can analytically marginalize out the cluster assignments $z_{1:N}$ and perform inference on K and $\mu_{1:K}$ only. However, as the prior on K is a Poisson distribution, the number of parameters in the model is unbounded. Using the model itself, we generated a synthetic dataset of $y_{1:150}$ for an one-dimensional mixture of five clusters (i.e. $K = 5$).

We now wish to perform inference over both the number of clusters K and the cluster means $\mu_{1:K}$, so that we can make predictions from the posterior predictive distribution. Two approaches we might try are VI, for which we use Anglican's black-box variational inference (BBVI) implementation (Ranganath et al., 2014; Paige, 2016), and MCMC, for which we take its RMH algorithm (Le, 2015), a derivative of the single-site MH algorithm of (Wingate et al., 2011) (see below). Unfortunately, as we see in Figure 3, both approaches fail spectacularly and produce posterior predictive distributions that bare little resemblance to the data. In particular, they fail to properly encapsulate the number of clusters. As we will show in §7.1, importance sampling and particle–based inference engines fare no better for this problem. In fact, we are not aware of any fully *automated* inference engine that is able to give satisfactory performance, despite the apparent simplicity of the problem.

3.1. Why is MCMC so Hard with Stochastic Support?

To run MCMC on a program with stochastic support, one needs to be able to construct a transition kernel that is able to switch between the configurations; many popular MCMC methods, like Hamiltonian Monte Carlo (HMC), cannot be applied. One can either look to construct this transition kernel manually through a carefully chosen user–specified trans–dimensional proposal and then using a reversible jump MCMC (RJMCMC) scheme (Green, 1995; 2003; Roberts et al., 2019; Cusumano-Towner et al., 2019), or use a general–purpose kernel that works on all models (Wingate et al., 2011; Goodman & Stuhlmüller, 2014).

The predominant approaches for the latter are the single-site MH (a.k.a. LMH) algorithm (Wingate et al., 2011) and its extensions (Yang et al., 2014; Le, 2015; Tolpin et al., 2015; Ritchie et al., 2016b). LMH is based around a Metropolis-within-Gibbs (MwG) approach on the program traces (Brooks et al., 2011), whereby one first samples a variable in the execution trace, $i \in 1 : n_x$, uniformly at random and then proposes a MwG transition to this variable, $x_i \to x_i'$, followed by an accept/reject step. Anglican's RMH is a particular case of this LMH approach where the proposal is a mixture of resampling x_i from the prior $f_{a_i}(x_i|\eta_i)$ and a local random walk proposal $p(x_i'|x_i)$.

The problem with LMH approaches is that if the transition of x_i influences the downstream control flow of the program, the downstream draws no longer produce a valid execution trace and so must be redrawn, typically using the prior. This can cause the mixing of the sampler over configurations to become extremely slow; the need to transit between configurations bottlenecks the system.

This problem is also far from specific to the exact transition kernel used by LMH samplers: it is also extremely challenging to hand–craft RJMCMC proposals to be effective. Namely, proposing changes in the configuration introduces new variables that might not be present in the current configuration, such that our proposal for them effectively becomes an importance sampling proposal. Furthermore, the posterior on the other variables may shift substantially when the configurations changes.

In short, one loses a notion of locality: having a sample in a high density region of one configuration typically provides little information about which regions have a high density for another configuration. For example, in a mixture model shown in Figure 1, having a good characterization of $\mu_1|K = 1$ provides little information about the distribution of $\mu_1|K = 2$, as shown by the substantial change in their mode, μ_1^*. It is thus extremely difficult to design proposals which maintain a high acceptance rate when proposing a new configuration: once in a high density region of one configuration, it becomes difficult to switch to another configuration. This problem is further compounded by the fact that RJMCMC only estimates the relative mass of each configuration through the relative frequency of transitions, giving a very slow convergence for the overall sampler.

4. Divide, Conquer, and Combine

The challenges for running MCMC methods on programs with stochastic support stem from the difficultly in transitioning *between* configurations properly. To address this, we now introduce a completely new inference framework for these programs: ***Divide, Conquer, and Combine*** (DCC). Unlike most existing inference approaches which directly target the full program density (i.e. (1)), DCC breaks the problem into individual sub-problems with ***fixed*** support and tackles them separately. Specifically, it ***divides*** the overall program into separate straight-line sub-programs according to their execution paths, ***conquers*** each sub-program by running inference locally, and ***combines*** the results together to form an overall estimate in a principled manner.

In doing this, DCC transfers the problem of designing an MCMC proposal which both efficiently transitions between paths (i.e. varying configurations) and mixes effectively over the draws on that path, to that of a) performing inference locally over the draws of each given path, and b) learning the relative marginal posterior mass of these paths. This separation brings the benefit that the inference for a given path can typically be performed much more efficiently than when using a global sampler, as it can exploit the fixed support and does not need to deal with changes in the variable configuration. Furthermore, it allows the relative posterior mass to be estimated more reliably than with global MCMC schemes, for which this is estimated implicitly through the relative frequency of the, typically infrequent, transitions.

We now explain the general setup for each component of DCC. Specific strategies for each will be introduced in §6, while an overview of the approach is given in Algorithm 1.

4.1. Divide

The aim of DCC's divide step is to split the given probabilistic program into its constituent straight-line programs (SLPs), where each SLP is a partition of the overall program corresponding to a particular sequence of sample addresses encountered during execution, i.e. a particular path $a_{1:n_x}$. Each SLP has a fixed support as the set of variables it draws are fixed by the path, i.e. the program draws from the same fixed set of `sample` statements in the same order.

Introducing some arbitrary indexing for the set of SLPs, we use A_k to denote the path for the k^{th} SLP (i.e. $a_{1:n_{x,k}} = A_k$ for every possible realization of this SLP). The set of of all possible execution paths is now given by $A = \{A_k\}_{k=1}^K$, where K must be countable (but need not be finite). For the

example in Figure 2, this set consists of two paths $A_1 = [\#l_1, \#l_4]$ and $A_2 = [\#l_1, \#l_7, \#l_8]$, where we use $\#l_j$ to denote the lexical address of the `sample` statement is on the j^{th} line. Note that, for a given program, each SLP is uniquely defined by its corresponding path A_k; we will sometimes use A_k to denote an SLP.

Dividing a program into its constituent SLPs implicitly partitions the overall target density into disjoint regions, with each part defining a sub-model on the corresponding subspace. The unnormalized density $\gamma_k(x)$ of the SLP A_k is defined with respect to the variables $\{x_i\}_{i=1}^{n_{x,k}}$ that are paired with the addresses $\{a_i\}_{i=1}^{n_{x,k}}$ of A_k (where we have used the notation $n_{x,k}$ to emphasize that this is now fixed). We use $\mathcal{X}_k$ to denote its corresponding support. Note that the union of all the $\mathcal{X}_k$ is the support of the original program, $\mathcal{X} = \bigcup_{k=1}^{K} \mathcal{X}_k$. Analogously to (1), we now have that the density of SLP k is $\pi_k(x) = \gamma_k(x)/Z_k$ where

$$\begin{aligned}\gamma_k(x) :=& \gamma(x)\mathbb{I}[x \in \mathcal{X}_k] \\ =& \mathbb{I}[x \in \mathcal{X}_k] \prod_{i=1}^{n_{x,k}} f_{A_k[i]}(x_i|\eta_i) \prod_{j=1}^{n_y} g_{b_j}(y_j|\phi_j), \quad (3)\end{aligned}$$

$$Z_k := \int_{x \in \mathcal{X}_k} \gamma_k(x) dx. \quad (4)$$

Unlike for (1), $n_{x,k}$ and A_k are now, critically, deterministic variables so that the support of the problem is fixed. Though b_j and n_y may still be stochastic, they do not effect the reference measure of the program and so this does not cause a problem when trying to perform MCMC sampling.

Following our example in Figure 2, for A_1 we have $x_{1:2} = [z0, z1]$, $\mathcal{X}_1 = \{[x_1, x_2] \in \mathbb{R}^2 \mid x_1 < 0\}$, and $\gamma_1(x) = \mathcal{N}(x_1; 0, 2)\mathcal{N}(x_2; -5, 2)\mathcal{N}(y_1; x_2, 2)\mathbb{I}[x_1 < 0]$. For A_2, we instead have $x_{1:3} = [z0, z2, z3]$, $\mathcal{X}_2 = \{[x_1, x_2, x_3] \in \mathbb{R}^3 \mid x_1 \geq 0\}$ and $\gamma_2(x) = \mathcal{N}(x_1; 0, 2)\mathcal{N}(x_2; 5, 2)\mathcal{N}(x_3; x_2, 2)\mathcal{N}(y_1; x_3, 2)\mathbb{I}[x_1 \geq 0]$.

To implement this divide step, we now need a mechanism for establishing the SLPs. This can either be done by trying to extract them all upfront, or by dynamically discovering them as the inference runs, see §6.2.

4.2. Conquer

Given access to the different SLPs produced by the divide step, DCC's conquer step looks to carry out the local inference for each. Namely, it aims to produce a set of estimates for the individual SLP densities $\pi_k(x)$ and the corresponding marginal likelihoods Z_k. As each SLP has a fixed support, this can be achieved with conventional inference approaches, with a large variety of methods potentially suitable. Note that $\pi_k(x)$ and Z_k need not be estimated using the same approach, e.g. we may use an MCMC scheme to estimate $\pi_k(x)$ and then introduce a separate estimator for Z_k. One possible estimation strategy is given in §6.1.

Algorithm 1 Divide, Conquer, and Combine (DCC)

Input: Program ***prog***, number of iterations T
Output: Posterior approx $\hat{\pi}$, ML estimate $\hat{Z}$

1: Obtain initial set of discovered SLPs $\hat{A}$ ▷ §4.1, §6.2
2: Compute initial estimates $\forall A_k \in \hat{A}$ ▷ §4.2, §6.1
3: **for** $t = 1, \ldots, T$ **do**
4: Choose an SLP $A_k \in \hat{A}$ to update ▷ §6.3
5: Update local estimates $\hat{\pi}_k$ and $\hat{Z}_k$ ▷ §4.2, §6.1
6: [Optional] Look for undiscovered SLPs (e.g. using a global proposal), add any found to $\hat{A}$ ▷ §6.2
7: **end for**
8: Combine local approximations as per (5) ▷ §4.3

An important component in carrying out this conquer step effectively is to note that it is not usually necessary to obtain estimates of equally-high fidelity for all SLPs. Specifically, SLPs with small marginal likelihoods Z_k only make a small contribution to the overall density and thus do not require as accurate estimation as SLPs with large Z_k. As such, it will typically be beneficial to carry out ***resource allocation*** as part of the conquer step, that is, to generate our estimates in an online manner where at each iteration we use information from previous samples to decide the best SLP(s) to update our estimates for. See §6.3 for one possible such strategy.

4.3. Combine

The role of DCC's combine step is to amalgamate the local estimates from the individual SLPs to an overall estimate of the distribution for the original program. For this, we can simply note that, because the supports of the individual SLPs are disjoint and their union is the complete program, we have $\gamma(x) = \sum_{k=1}^{K} \gamma_k(x)$ and $Z = \sum_{k=1}^{K} Z_k$, such that the unnormalized density and marginal likelihoods are both additive. Consequently, we have

$$\begin{aligned}\pi(x) = \frac{\sum_{k=1}^{K} \gamma_k(x)}{\sum_{k=1}^{K} Z_k} &= \frac{\sum_{k=1}^{K} Z_k \pi_k(x)}{\sum_{k=1}^{K} Z_k} \\ &\approx \frac{\sum_{k=1}^{K} \hat{Z}_k \hat{\pi}_k(x)}{\sum_{k=1}^{K} \hat{Z}_k} =: \hat{\pi}(x) \quad (5)\end{aligned}$$

where $\hat{\pi}_k(x)$ and $\hat{Z}_k$ are the SLP estimates generated during the conquer step. Note that, by proxy, this also produces the overall marginal likelihood estimate $\hat{Z} := \sum_{k=1}^{K} \hat{Z}_k$.

When using an MCMC sampler for $\pi_k(x)$, $\hat{\pi}_k(x)$ will take the form of an empirical measure comprising of a set of samples, i.e. $\hat{\pi}_k(x) = \frac{1}{N_k} \sum_{m=1}^{N_k} \delta_{\hat{x}_{k,m}}(x)$. If we use an importance sampling or particle filtering based approach instead, our empirical measure will compose of weighted samples. We note that in this case, the $\hat{Z}_k$ term in the numerator of (5) will cancel with any potential self-normalization term used in $\hat{\pi}_k(x)$, such that we can think of using the estimate $\pi(x) \approx (\sum_{k=1}^{K} \hat{\gamma}_k(x))/(\sum_{k=1}^{K} \hat{Z}_k)$.

5. Theoretical Correctness

We now demonstrate that the outlined general DCC approach is consistent (as $T \to \infty$ where T is the number of iterations) given some simple assumptions about the individual component strategies. At a high level, these assumptions are that the estimators used for each SLP, $\hat{\pi}_k$ and $\hat{Z}_k$, are themselves consistent, we use an SLP extraction strategy that will eventually uncover all of the SLPs with finite probability mass, and our resource allocation strategy selects each SLP infinitely often given an infinite number of iterations.

More formally we have the following result

Theorem 1. *If Assumptions 1-5 in Appendix C hold, then the empirical measure, $\hat{\pi}(\cdot)$, produced by DCC converges weakly to the conditional distribution of the program in the limit of large number of iterations T :*

$$\hat{\pi}(\cdot) \xrightarrow{d} \pi(\cdot) \quad as \quad T \to \infty.$$

The proof is provided in Appendix C. We note that it is typically straightforward to ensure that these assumptions hold; the specific approaches we outline next satisfy them.

6. DCC in Anglican

We now outline a particular realization of our DCC framework. It is implemented in Anglican and can be used to run inference automatically for any valid Anglican program. As part of this, we suggest particular strategies for the individual components left unspecified in the last section, but emphasize that these are far from the only possible choices; DCC should be viewed more as a general framework. Additional details including a complete algorithm block are given in the appendices.

6.1. Local Estimators

Recall that the goal for the local inference is to estimate the local target density $\pi_k(x)$ and the local marginal likelihood Z_k. Straightforward choices include (self-normalized) importance sampling and SMC as both return a marginal likelihood estimate $\hat{Z}_k$. However, knowing good proposals for these a priori is challenging and, as we discussed in §3, naïve choices are unlikely to perform well.

Thankfully, each SLP has a fixed support, which means many of the complications that make inference challenging for universal PPSs no longer apply. In particular, we can use conventional MCMC samplers—such as MH, HMC, or MwG—to approximate $\pi_k(x)$. Due to the fact that individual variable types may be unknown or not even fixed, we have elected to use MwG in our implementation, but note that more powerful inference approaches like HMC may be preferable when they can be safely applied. To encourage sample diversity and assist in estimating Z_k (see below), we further run N independent MwG samplers for each SLP.

As MCMC samplers do not directly provide an estimate for Z_k, we must introduce a further estimator that uses these samples to estimate it. For this, we use PI-MAIS (Martino et al., 2017). Details are given in Appendix D.

6.2. Discovering SLPs

To divide a given model into its constituent sub-models expressed by SLPs, we need a mechanism for discovering them automatically. One possible approach would be to analyze the source code of the program using static analysis techniques (Chaganty et al., 2013; Nori et al., 2014), thereby extracting the set of possible execution paths of the program at compilation time. Though potentially a viable choice in some scenarios, this can be difficult to achieve for all possible programs in a universal PPS. For example, the number of possible paths may be unbounded. We therefore take an alternative approach that discovers SLPs dynamically at run–time as part of the inference. In general, it maintains a set of SLPs encountered so far, and *remembers* any new SLP discovered by the MCMC proposals at each iteration.

Our approach starts by executing the program forward for T_0 iterations to generate some sample execution traces from the prior. The paths traversed by these sampled traces are recorded, and our set of SLPs is initialized as that of these recorded paths. At subsequent iterations, after each local inference iteration, we then perform one *global* MCMC step based on our current sub-model and trace, producing a new trace with path $A_{k'}$ that may or may not have changed. If $A_{k'}$ corresponds to an existing SLP, this sample is discarded (other than keeping count of the number of proposed transitions into each SLP). However, if it corresponds to an unseen path, it is added to our set of SLPs as a new sub-model, followed by T_w MCMC steps restricted to that path to burn-in. The sample of the final step will be stored as initiation for the future local inference on that path.

The key difference between our strategy and running a single global MCMC sampler (e.g. RMH), is that we do not need this new sample to be *accepted* for the new SLP to be "discovered". Because, as we explained in § 3, making effective proposals into a high–density region of a new configuration is very challenging, it is unlikely that the new trace sample we propose has high density: even if it corresponds to a path with large Z_k, we are unlikely to immediately sample a good set of draws to accompany it. Therefore, the global MCMC sampler is very likely to miss or ***forget*** new SLPs since it accepts/rejects movements based on one sample.

DCC, on the other hand, overcomes this problem by first ***remembering*** the path information of any newly proposed SLP, and then carrying out a few local warm-up iterations, before deciding whether an SLP is a promising sub-model or not in later exploration. As a result, DCC does not suffer from the reliance on forward sampling as per RMH to

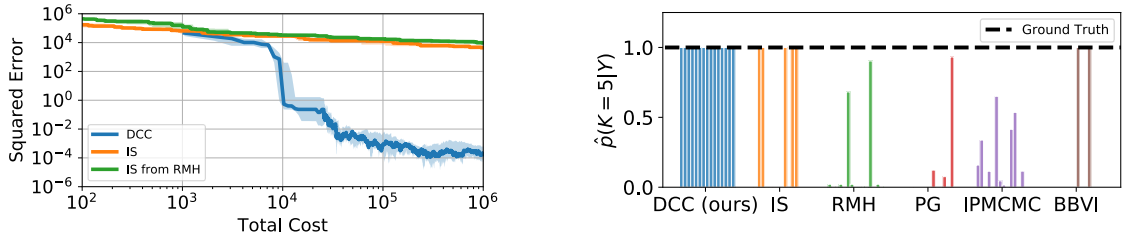


Figure 4. Results for DCC and baselines for the GMM example outlined in §3. [Left] Convergence in squared error in log marginal likelihood estimate $\|\log \hat{Z} - \log Z_{true}\|^2$. The solid line corresponds to the median across 15 runs and the shading region $25\% - 75\%$ quantiles. Note that none of IS, RMH, PG, IPMCMC, and BBVI provide such at estimate, hence their omission; an additional baseline of drawing importance samples from a proposal centered on and RMH chain was considered instead. [Right] Final estimates for $p(K = 5 \mid y_{1:N_y})$ for each of the 15 runs, for the ground truth is roughly 0.9998. In both cases, the ground truth was estimated using a very large number of importance samples with a manually adapted proposal. We see that DCC substantially outperforms the baselines.

discover new SLPs. Moreover, our scheme inherits the hill-climbing behavior of MCMC to discover SLP in a more efficient way: it can find the sub-models of high posterior mass even under an extremely small prior probability, as will be shown in §7.2. Note that there are some subtleties to maintain the efficiency as the number of possible SLPs grow large, see Appendix E for further details.

6.3. Allocating Resources Between SLPs

At each iteration we must choose an SLP from those discovered to perform local inference on. Though valid, it is not wise to split our computational resources evenly among all SLPs; it is more important to ensure we have accurate estimates for SLPs with large Z_k. Essentially, we have a multi-armed bandit problem where we wish to develop a strategy of choosing SLPs that will lead to the lowest error in our *overall* final estimate $\hat{\pi}$. Though it might seem that this is a problem that DCC has introduced, it is actually an inherent underlying problem that must always be solved for models with stochastic support; DCC is simply making the problem explicit. Namely, we do not know upfront which SLPs have significant mass and so any inference method must deal with the computational trade–off involved in figuring this out. Conventional approaches do this in an implicit, and typically highly inefficient, manner. For example, MCMC relies on the relatively frequency of individual transitions between SLPs to allocate resources, which will generally be extremely inefficient for finite budgets.

To address this, we introduce a resource allocation scheme based on an upper confidence bounding (UCB) approach developed in Rainforth et al. (2018). Specifically, we use the existing SLPs estimates to construct a utility function that conveys the relative merit of refining the estimates for each SLP, balancing the need for ***exploitation***, that is improving the estimates for SLPs currently believed to have large relative Z_k, and ***exploration***, that is improving our estimates for SLPs where our uncertainty in Z_k is large. At each iteration, we then update the estimate for the SLP which has the largest utility, defined as

$$U_k := \frac{1}{S_k}\left(\frac{(1-\delta)\hat{\tau}_k}{\max_k\{\hat{\tau}_k\}} + \frac{\delta\hat{p}_k}{\max_k\{\hat{p}_k\}} + \frac{\beta\log\sum_k S_k}{\sqrt{S_k}}\right)$$

where S_k is the number of times we have previously performed local inference on A_k; $\hat{\tau}_k$ is the current estimate of the "reward" of A_k, incorporating both how much mass the SLP contains and how efficient our estimates are for it; $\hat{p}_k$ is a targeted exploration term that helps identify promising SLPs that we are yet to establish good estimates for; $0 \leq \delta \leq 1$ is a hyperparameter controlling the trade–off between these terms; and $\beta > 0$ is the standard optimism boost hyper-parameter. For more details see Appendix F.

7. Experiments

7.1. Gaussian Mixture Model (GMM)

We now further investigate the GMM example with an unknown number of clusters introduced in §3. Its program code written in Anglican is provided in Appendix G.1. We compare the performance of DCC against five baselines: importance sampling (from prior) (IS), RMH (Le, 2015), Particle Gibbs (PG) (Andrieu et al., 2010), interacting Particle MCMC (IPMCMC) (Rainforth et al., 2016), and Black-box Variational Inference (BBVI) (Paige, 2016), taking the same computational budget of 10^6 total samples for each.

We first examine the convergence of the overall marginal likelihood estimate $\hat{Z}$. Here IS is the only baseline which can be used directly, but we also consider drawing importance samples centered around the RMH chain in a manner akin to PI-MAIS. Figure 4 [Left] shows that DCC outperforms both by many orders of magnitude. The sudden drop in the error for DCC is because the dominant sub-model with $K = 5$ is typically discovered after taking around 10^4 samples. Further investigation of that SLP allows DCC to improve the accuracy of the estimate. DCC has visited 23 to 27 sub-models (out of *infinitely* many) among all 15 runs.

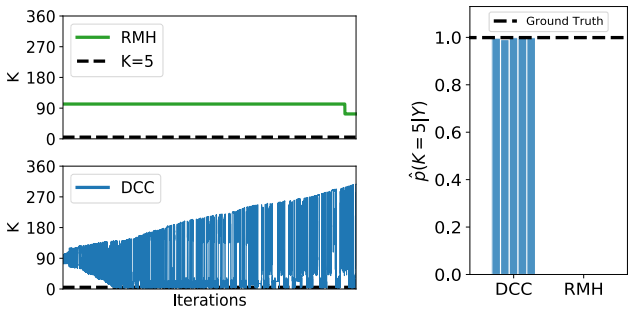
We next examine the posterior distribution of K and report the estimates of $p(K = 5 \mid y_{1:N_y})$ in Figure 4 [Right]. We see that all methods other than DCC struggle. Here , the accuracy of the posterior of K reflects the accuracy in estimating the relative masses of the different SLPs, i.e. Z_k, explicitly or implicitly. The dimension of this model varies between one and infinity and the posterior mass is concentrated in a small sub-region ($K = 5$) with small prior mass. It is therefore challenging for the baselines to either to learn each marginal likelihood simultaneously (eg. in IS) or to estimate the relative masses implicitly through transitions between configurations using an MCMC sampler. By breaking down the model into sub-problems, DCC is able to overcome these challenges and provide superior posterior estimator for the overall model.

7.2. GMM with Misspecified Prior

To further test the capability of each method to discover SLPs—and to examine the MCMC-esque behavior for DCC in SLP space in particular—we adjust the GMM example above slightly so that K now has a, high misspecified, prior of $\text{Poisson}(90)+1$, keeping everything else the same. The dominant SLP is still $K = 5$ (with around 0.9976 posterior mass), but his now has an extremely low prior probability (around 10^{-14}). Consequently, finding this dominant SLP is only practically possible if the algorithm exhibits an effective hill climbing behavior in SLP space. We only now compare DCC to RMH on the basis that: a) none of the baselines could deal with simpler case before, such that they will inevitably not be able to deal with this harder problem; and b) RMH is the only baseline where one might expect to see some hill climbing behavior in SLP space.

Figure 5(a) shows the trace plot for the SLP visit history (i.e. sampled K at each iteration) of each method. As we can see in the bottom plot, DCC starts from the SLPs of K around 90, influenced by the prior, and gradually discovers smaller Ks with higher posterior mass, also exploring large values of K as well. This implies a MCMC-esque hill-climbing behavior guided by our SLP discovery scheme. Moreover, the trace plot also demonstrates the resource allocation within DCC where it gradually spends more computation for SLPs with lower Ks while still maintaining a degree of exploration. Both factors are essential for the resulting accurate posterior approximation shown in Figure 5(b).

By comparison, RMH gets stuck in its initialized SLP (Figure 5(a) top): for the one run shown it only makes one successful transition to another SLP and never gets anywhere close to region of SLPs with significant mass. Equivalent behavior was experience in all the other runs (not shown). As a result, it does not produce a reasonable posterior estimate as shown Figure 5(b); in fact, it always returns an estimate of exactly 0 as it never discovers this SLP. It is worth noting that the local mixing of RMH between SLPs here is even



(a) SLP visit order (b) $\hat{p}(K = 5 \mid y_{1:N_y})$

Figure 5. Comparison of DCC to RMH on GMM with misspecified prior. [Left] visit order of SLPs (i.e. sampled K at each iteration) for single run. [Right] final posterior estimates for 5 different runs.

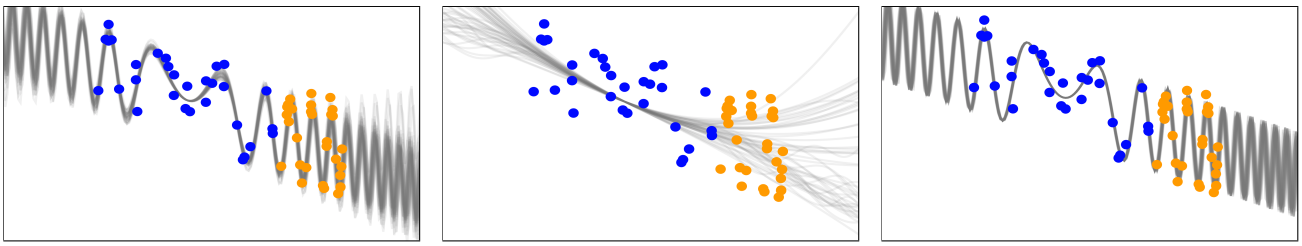
worse than in the previous example. This is because the larger K at which the sampler is initialized induces a higher dimensional space on program draws, i.e. $\mu_{1:K}$. This is catastrophic for RMH because it is effectively importance sampling when transitioning between SLPs and thus suffers acutely from the curse of dimensionality. DCC, meanwhile, gracefully deals with this because of its ability to remember SLPs that are proposed but not accepted and then subsequently perform effective localized inference that exploits hill–climbing effects in the space of the draws of that SLP.

7.3. Function Induction

Function induction is an important task for automated machine learning (Duvenaud et al., 2013; Kusner et al., 2017). In PPSs, it is typically tackled using a probabilistic context free grammar (PCFG) (Manning et al., 1999). Here we consider such a model where we specify the structure of a candidate function using a PCFG and a distribution over the function parameters, and estimate the posterior of both for given data. Our PCFG consists of four production rules with fixed probabilities: $e \to \{x \mid x^2 \mid \sin(a{*}e) \mid a{*}e{+}b{*}e\}$, where x and x^2 are terminal symbols, a and b are unknown coefficient parameters, and e is a non-terminal symbol. The model also has prior distributions over each coefficient parameters. See Appendix G.2 for details.

To generate a function from this model, we must sample both a PCFG rollout and the corresponding parameters. Let Θ be the collection of all the latent variables used in this generative process. That is, Θ consists of the sequence of the discrete variables recording the choices of the grammar rules and all coefficients in the sampled structure. Conditioned on the training data D, we want to infer the posterior distribution $p(\Theta|D)$, and calculate the posterior predictive distribution $p(D'|D)$ for test data $D' = \{x_n, y_n\}_{n=1}^N$.

In our experiment, we control the number of sub-models by requiring that the model use the PCFG in a restricted way: a sampled function structure should have depth at most 3



(a) DCC (ours) (b) IS (c) RMH

Figure 6. Posterior distribution $p(\Theta|D)$ estimated by DCC, IS, and RMH under the same computation. Blue points represent the observed data D and orange ones the test data D'. Grey lines are the posterior samples of the functions from the run with the highest LPPD among 15 independent runs of the three algorithms.

Table 1. Mean and one standard derivation of the LPPD over 15 independent runs.

	DCC (ours)	IS	RMH	PG	IPMCMC
LPPD	**-28.560 ± 0.41**	-73.180 ± 1.08	-32.693 ± 8.51	-200.824 ± 126.63	-70.580 ± 4.71

and cannot use the plus rule consecutively. We generate a synthetic dataset of 30 training data points from the function $f(x) = -x + 2\sin(5x^2)$ and compare the performance of DCC to our baselines on estimating the posterior distribution and the posterior predictive under the same computational budget of 10^6 samples and 15 independent runs. BBVI is omitted from this experiment due to it failing to run at all.

Figure 6 shows the posterior samples generated by DCC, IS, and RMH for one run, with the training data D marked blue and the test data D' in orange. The DCC samples capture the periodicity of the data and provides accurate extrapolation, while retaining an appropriate degree of uncertainty. This indicates good inference results on both the structure of a function and the coefficients. Though RMH does find some good functions, it becomes stuck in a particular mode and does not fully capture the uncertainty in the model, leading to poor predictive performance.

Table 1 shows the test log posterior predictive density (LPPD), $\sum_{n=1}^{N} \log \int_\Theta p(y_n|x_n, \Theta)p(\Theta|D)d\Theta$, of all approaches. DCC substantially outperforms all the baselines both in terms of predictive accuracy and stability. IS, PG, and IPMCMC all produced very poor posterior approximations leading to very low LPPDs. RMH had an LPPD that is closer to DCC, but which is still substantially inferior.

A further issue with RMH was its high variance of the LPPD. This is caused by this model being multi-modal and RMH struggling to move: it gets stuck in a single SLP and fails to capture the uncertainty. Explicitly, 4 sub-models (out of 26) contain most of the probability mass. Two of them are functions of the form used to generate the data, $f(x) = a_1 x + a_2 \sin(a_3 x^2)$, modulo symmetry of the $+$ operator. The other two have the form $f(x) = a_1 \sin(a_2 x) + a_3 \sin(a_4 x^2)$, which can also match the training data well in the region $(-1.5, 1.5)$ as $a_1 \sin(a_2 x) \approx a_1 a_2 x$ for small values of $a_2 x$. Note that the local distributions are also multi-modal due to various symmetries, for example $a_1 \sin(a_2 x^2)$ and $-a_1 \sin(-a_2 x^2)$, meaning the local inference task is non-trivial even in low dimensions.

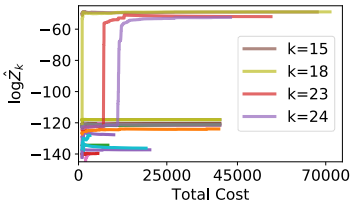


Figure 7. Convergence of DCC's $\log Z_k$ estimate for each SLP and corresponding total amount of resources spent.

To test the effectiveness of the resource allocation strategy, we further investigate the computational resources spent for each SLP by looking at the convergence of the local marginal likelihood estimates $\hat{Z}_k$. In Figure 7, the sub-models 15 and 18 correspond to the form $f(x) = a_1 x + a_2 \sin(a_3 x^2)$ and its mirror, which contain the most posterior mass. The sub-models 23 and 24 correspond to $f(x) = a_1 \sin(a_2 x) + a_3 \sin(a_4 x^2)$ and are the second largest modes. Figure 7 implies that DCC indeed spends more computational resource on sub-models with high probability mass (as signified by the higher final total cost), while also exploring the other sub-models occasionally.

8. Conclusions

In this paper, we have proposed *Divide, Conquer, and Combine (DCC)*, a new inference strategy for probabilistic programs with stochastic support. We have shown that, by breaking down the overall inference problem into a number of separate inferences of sub-programs with fixed support, the DCC framework can provide substantial performance improvements over existing approaches which directly target the full program. To realize this potential, we have shown how to implement a particular instance of DCC as an automated engine in the PPS Anglican and shown that this outperforms existing baselines on three example problems.

Acknowledgements

YZ is sponsored by China Scholarship Council (CSC). HY was supported by the Engineering Research Center Program through the National Research Foundation of Korea (NRF) funded by the Korean Government MSIT (NRF-2018R1A5A1059921), and also by Next-Generation Information Computing Development Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT (2017M3C4A7068177). YWT's and TR's research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP7/2007-2013)/ ERC grant agreement no. 617071. TR was also supported in part by Junior Research Fellowship from Christ Church, University of Oxford and and in part by EPSRC funding under grant EP/P026753/1.

References

Andrieu, C., Doucet, A., and Holenstein, R. Particle Markov chain Monte Carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3): 269–342, 2010.

Baydin, A. G., Shao, L., Bhimji, W., Heinrich, L., Naderiparizi, S., Munk, A., Liu, J., Gram-Hansen, B., Louppe, G., Meadows, L., et al. Efficient probabilistic inference in the quest for physics beyond the standard model. In *Advances in Neural Information Processing Systems*, pp. 5460–5473, 2019.

Bingham, E., Chen, J. P., Jankowiak, M., Obermeyer, F., Pradhan, N., Karaletsos, T., Singh, R., Szerlip, P., Horsfall, P., and Goodman, N. D. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 2018.

Brooks, S., Gelman, A., Jones, G., and Meng, X.-L. *Handbook of markov chain monte carlo*. CRC press, 2011.

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., Brubaker, M., Guo, J., Li, P., and Riddell, A. Stan: A probabilistic programming language. *Journal of statistical software*, 76(1), 2017.

Carpentier, A., Munos, R., and Antos, A. Adaptive strategy for stratified monte carlo sampling. *Journal of Machine Learning Research*, 16:2231–2271, 2015.

Chaganty, A., Nori, A., and Rajamani, S. Efficiently sampling probabilistic programs via program analysis. In *Artificial Intelligence and Statistics*, pp. 153–160, 2013.

Cusumano-Towner, M. F., Saad, F. A., Lew, A. K., and Mansinghka, V. K. Gen: a general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 221–236, 2019.

Doucet, A., De Freitas, N., and Gordon, N. An introduction to sequential monte carlo methods. In *Sequential Monte Carlo methods in practice*, pp. 3–14. Springer, 2001.

Duvenaud, D., Lloyd, J. R., Grosse, R., Tenenbaum, J. B., and Ghahramani, Z. Structure discovery in nonparametric regression through compositional kernel search. *arXiv preprint arXiv:1302.4922*, 2013.

Fox, E. B., Sudderth, E. B., Jordan, M. I., and Willsky, A. S. An hdp-hmm for systems with state persistence. In *Proceedings of the 25th international conference on Machine learning*, pp. 312–319, 2008.

Ge, H., Xu, K., and Ghahramani, Z. Turing: A language for flexible probabilistic inference. In *International Conference on Artificial Intelligence and Statistics*, pp. 1682–1690, 2018.

Goodman, N. D. and Stuhlmüller, A. The Design and Implementation of Probabilistic Programming Languages. http://dippl.org, 2014. Accessed: 2020-2-6.

Goodman, N. D. and Stuhlmüller, A. The design and implementation of probabilistic programming languages, 2014.

Goodman, N. D., Mansinghka, V. K., Roy, D. M., Bonawitz, K., and Tenenbaum, J. B. Church: A Language for Generative Models. In *In UAI*, pp. 220–229, 2008.

Green, P. J. Reversible jump markov chain monte carlo computation and bayesian model determination. *Biometrika*, 82(4):711–732, 1995.

Green, P. J. Trans-dimensional markov chain monte carlo. *Oxford Statistical Science Series*, pp. 179–198, 2003.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. Stochastic variational inference. *The Journal of Machine Learning Research*, 14(1):1303–1347, 2013.

Janz, D., Paige, B., Rainforth, T., van de Meent, J.-W., and Wood, F. Probabilistic structure discovery in time series data. *arXiv preprint arXiv:1611.06863*, 2016.

Kucukelbir, A., Tran, D., Ranganath, R., Gelman, A., and Blei, D. M. Automatic differentiation variational inference. *The Journal of Machine Learning Research*, 18(1): 430–474, 2017.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1945–1954. JMLR. org, 2017.

Le, T. A. Inference for higher order probabilistic programs. *Masters thesis, University of Oxford*, 2015.

Le, T. A., Baydin, A. G., and Wood, F. Inference compilation and universal probabilistic programming. *arXiv preprint arXiv:1610.09900*, 2016.

Lu, X., Rainforth, T., Zhou, Y., van de Meent, J.-W., and Teh, Y. W. On exploration, exploitation and learning in adaptive importance sampling. *arXiv preprint arXiv:1810.13296*, 2018.

Manning, C. D., Manning, C. D., and Schütze, H. *Foundations of statistical natural language processing*. MIT press, 1999.

Mansinghka, V., Selsam, D., and Perov, Y. Venture: a higher-order probabilistic programming platform with programmable inference. *arXiv preprint arXiv:1404.0099*, 2014.

Martino, L., Elvira, V., Luengo, D., and Corander, J. Layered adaptive importance sampling. *Statistics and Computing*, 27(3):599–623, 2017.

Metropolis, N. and Ulam, S. The Monte Carlo method. *Journal of the American statistical association*, 44(247): 335–341, 1949.

Narayanan, P., Carette, J., Romano, W., Shan, C.-c., and Zinkov, R. Probabilistic inference by program transformation in hakaru (system description). In *International Symposium on Functional and Logic Programming*, pp. 62–79. Springer, 2016.

Nobile, A. and Fearnside, A. T. Bayesian finite mixtures with an unknown number of components: The allocation sampler. *Statistics and Computing*, 17(2):147–162, 2007.

Nori, A., Hur, C.-K., Rajamani, S., and Samuel, S. R2: An efficient mcmc sampler for probabilistic programs. In *Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2014.

Paige, T. B. *Automatic inference for higher-order probabilistic programs*. PhD thesis, University of Oxford, 2016.

Perov, Y. N. and Wood, F. D. Learning probabilistic programs. *arXiv preprint arXiv:1407.2646*, 2014.

Rainforth, T. *Automating Inference, Learning, and Design using Probabilistic Programming*. PhD thesis, University of Oxford, 2017.

Rainforth, T., Naesseth, C., Lindsten, F., Paige, B., Vandemeent, J.-W., Doucet, A., and Wood, F. Interacting particle markov chain monte carlo. In *International Conference on Machine Learning*, pp. 2616–2625, 2016.

Rainforth, T., Zhou, Y., Lu, X., Teh, Y. W., Wood, F., Yang, H., and van de Meent, J.-W. Inference trees: Adaptive inference with exploration. *arXiv preprint arXiv:1806.09550*, 2018.

Ranganath, R., Gerrish, S., and Blei, D. Black box variational inference. In *Artificial Intelligence and Statistics*, pp. 814–822, 2014.

Richardson, S. and Green, P. J. On bayesian analysis of mixtures with an unknown number of components (with discussion). *Journal of the Royal Statistical Society: series B (statistical methodology)*, 59(4):731–792, 1997.

Ritchie, D., Horsfall, P., and Goodman, N. D. Deep amortized inference for probabilistic programs. *arXiv preprint arXiv:1610.05735*, 2016a.

Ritchie, D., Stuhlmüller, A., and Goodman, N. D. C3: lightweight incrementalized MCMC for probabilistic programs using continuations and callsite caching. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, AISTATS 2016, Cadiz, Spain, May 9-11, 2016*, pp. 28–37, 2016b. URL http://proceedings.mlr.press/v51/ritchie16.html.

Roberts, D., Gallagher, M., and Taimre, T. Reversible jump probabilistic programming. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 634–643, 2019.

Roy, D. M., Teh, Y. W., et al. The mondrian process. In *NeurIPS*, pp. 1377–1384, 2008.

Schaechtle, U., Saad, F., Radul, A., and Mansinghka, V. Time series structure discovery via probabilistic program synthesis. *arXiv preprint arXiv:1611.07051*, 2016.

Ścibior, A., Kammar, O., and Ghahramani, Z. Functional programming for modular bayesian inference. *Proceedings of the ACM on Programming Languages*, 2(ICFP): 1–29, 2018.

Teh, Y. W. Dirichlet process., 2010.

Tesauro, G., Rajan, V., and Segal, R. Bayesian inference in Monte-Carlo tree search. *arXiv preprint arXiv:1203.3519*, 2012.

Tolpin, D. and Wood, F. Maximum a Posteriori Estimation by Search in Probabilistic Programs. In *Eighth Annual Symposium on Combinatorial Search*, 2015.

Tolpin, D., van de Meent, J., Paige, B., and Wood, F. D. Output-sensitive adaptive metropolis-hastings for probabilistic programs. In *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML*

PKDD 2015, Porto, Portugal, September 7-11, 2015, Proceedings, Part II, pp. 311–326, 2015. doi: 10.1007/978-3-319-23525-7_19. URL https://doi.org/10.1007/978-3-319-23525-7_19.

Tolpin, D., van de Meent, J.-W., Yang, H., and Wood, F. Design and implementation of probabilistic programming language anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional programming Languages*, pp. 1–12, 2016.

Wingate, D. and Weber, T. Automated variational inference in probabilistic programming. *arXiv preprint arXiv:1301.1299*, 2013.

Wingate, D., Stuhlmüller, A., and Goodman, N. Lightweight implementations of probabilistic programming languages via transformational compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pp. 770–778, 2011.

Wood, F., van de Meent, J. W., and Mansinghka, V. A New Approach to Probabilistic Programming Inference. In *Artificial Intelligence and Statistics*, pp. 1024–1032, 2014.

Yang, L., Hanrahan, P., and Goodman, N. Generating efficient mcmc kernels from probabilistic programs. In *Artificial Intelligence and Statistics*, pp. 1068–1076, 2014.