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

A. Existing PPSs and Inference Engines for Probabilistic Programs with Stochastic Support

Table 2. List of popular universal PPSs and their supported inference algorithms for models with stochastic support.[3]

	Rejection	IS	SMC	PMCMC	VI	Customized proposal MCMC	Automated proposal MCMC
Venture (Mansinghka et al., 2014)	✓		✓	✓	✓		✓
WebPPL (Goodman & Stuhlmüller, 2014)	✓		✓	✓			✓
MonadBayes (Ścibior et al., 2018)			✓	✓			
Anglican (Wood et al., 2014)		✓	✓	✓	✓		✓
Turing.jI (Ge et al., 2018)		✓	✓	✓		✓	✓
Pyro (Bingham et al., 2018)		✓	✓		✓	✓	
Gen (Cusumano-Towner et al., 2019)		✓	✓		✓	✓	✓
Hakaru (Narayanan et al., 2016)		✓				✓	
Stochaskell (Roberts et al., 2019)						✓	

In Table 2 above, we have listed which inference engines from the five categories in §3 are supported by each of the most popular universal PPSs. The fundamental difficulty in performing inference on probabilistic models with stochastic support is that the posterior mass of such models is usually concentrated in one or many separated sub-regions which are non-trivial to be fully discovered, especially in a high dimensional space. Moreover, even if a variable exists in different variable configurations, the posterior might shift substantially like the mean μ_1 for the GMM shown in the Figure 1, which complicates the design for a "proper" proposal distribution, let alone automate this procedure in PPS. We now have a more detailed look at each category with the related PPSs and uncover the reasons why these engines are not suitable for probabilistic programs with stochastic support.

Importance/rejection sampling Basic inference schemes, such as importance sampling (IS) and rejection sampling (RS), are commonly supported by many PPSs due to their generality and simplicity. See Table 2. They can be directly applied to a model with stochastic support, but their performance deteriorates rapidly (typically exponentially) as the dimension of the model increases; they suffer acutely from the curse of dimensionality. Furthermore, their validity relies on access to a valid proposal. This can often be easily ensured, e.g. by sampling from the prior, but doing this while also ensuring the proposal is efficient can be very challenging; the prior is only a practically viable proposal for very simple problems. Though schemes for developing more powerful proposals in amortized inference settings have been developed (Le et al., 2016; Ritchie et al., 2016a), these are not appropriate for conventional inference problems.

[3] All information are taken from the published paper or the online manual.

Particle based methods Particle based inference methods, such as Sequential Monte Carlo (SMC) (Doucet et al., 2001), can offer improvements for models with natural sequential structure (Wood et al., 2014; Rainforth et al., 2016). More explicitly, SMC improves IS when there exist interleaved observe statements in the model program. However, it similarly rapidly succumbs to the curse of dimensionality in the more general case where this does not occur. Such methods also cannot be used at all if the number of observe statements is not fixed, which can be common for stochastic support problems.

One might then tend to more sophisticated methods such as particle MCMC (PMCMC) methods (Andrieu et al., 2010) but unfortunately these suffer from the same underlying issues. The basic setups of PMCMC include the Particle Independent Metropolis Hasting (PIMH) (in Anglican and Turing.jI) and Particle Gibbs (PG) (in Anglican and Turing.jI (Ge et al., 2018)), where one uses, respectively, *independent* and *conditional* SMC sweeps as the proposal within the MCMC sampler. These building-block sweeps suffer from exactly the same issue as conventional SMC, and thus offer no advantage over simple importance sampling without interleaved observe statements.

These advanced variants do though allow one to treat global parameters separately: they update the global parameters using a Metropolis–within–Gibbs (MwG) step and then update the rest latent variables using a (conditional) SMC sweep with those global parameter values. When combined with PIMH, this leads to Particle Marginal Metropolis Hasting (PMMH) algorithm (available in WebPPL (Goodman & Stuhlmüller, 2014), Turing.jI and MonadBayes (Ścibior et al., 2018)). It already constitutes a valid step for PG. Unfortunately, in both cases this MwG suffers from exactly the same issues as those already discussed for LMH. As such, these approaches again offer little advantage over importance sampling without sequential problem structure.

Variational inference Following the discussion in § 3, three universal PPSs under our survey that allow Variational Inference (VI) in stochastic support settings are Pyro (Bingham et al., 2018), Gen (Cusumano-Towner et al., 2019) and Anglican (Wood et al., 2014) (note many others allow VI for statistic support, but cannot be used when the support varies). Pyro supports Stochastic Variational Inference (SVI) (Hoffman et al., 2013; Wingate & Weber, 2013; Kucukelbir et al., 2017) and allows an auto-generated guide function (AutoGuide), i.e. the variational proposal, or a user-specified guide. However, AutoGuide only works for some basic probabilistic programs, which we cannot directly apply for the GMM example in § 3. With the customized guide, one cannot deal with the exact same model because of the need to upper-bound the number of the variational parameters according to the tutorial[4].

Both Gen and Anglican support the Black box Variational Inference (BBVI) (Ranganath et al., 2014). We have tested the Anglican's BBVI with the GMM example in § 3, but it does not provide very accurate estimates as can be seen in the Figure 3.

One key challenge for implementing VI dealing with models with stochastic support is that one might still have never-before-seen variables after finite number of training steps. Moreover, it is usually very challenging in stochastic support settings to ensure that the variational family is defined in a manner that ensures the KL is well defined. For example, if using $\mathrm{KL}(q||p)$ (for proposal q and target p), then the KL will be infinite if q places support anywhere p does not. Controlling this with static support is usually not especially problematic, but in stochastic support settings it can become far more challenging.

Overcoming these complications is beyond the scope of our paper but could be a potential interesting direction for future work.

MCMC with customized proposal To perform Markov chain Monte Carlo (MCMC) methods (Metropolis & Ulam, 1949) on the models with stochastic support, one needs to construct the transitional kernel such that the sampler can switch between configurations. A number of PPSs such as Hakaru, Turing.jI, Pyro and Gen allow the user to customize the kernel for Metropolis Hastings (a.k.a programmable kernel) whereas Stohaskell explicitly supports reversible jump Markov chain Monte Carlo (RJMCMC) (Green, 1995; 2003) methods. However, their application is fundamentally challenging as we have discussed in § 3 due to the difficulty in designing proposals which can transition efficiently as well as the posterior shift on the variable under different configuration.

MCMC with automated proposal One MCMC method that can be fully automated for PPSs is the Single-site Metropolis Hastings or the Lightweight Metropolis Hastings algorithm (LMH) of (Wingate et al., 2011) and its extensions (Yang et al., 2014; Tolpin et al., 2015; Le, 2015; Ritchie et al., 2016b), for which implementations are provided in a number of systems such as Venture (Mansinghka et al., 2014), WebPPL and Anglican. In particular, Anglican supports LMH and its variants,

[4] https://pyro.ai/examples/dirichlet_process_mixture.html

Random-walk lightweight Metropolis Hastings (RMH) (Le, 2015), which uses a mixture of prior and local proposal to update the selected entry variable x_i along the trace $x_{1:n_x}$.

Many shortcomings of LMH and RMH have been discussed in §3 and we reiterate a few points here. Though widely applicable, LMH relies on proposing from the prior whenever the configuration changes for the downstream variables. This inevitably forms a highly inefficient proposal (akin to importance sampling from the prior), such that although samples from different configurations might be proposed frequently, these samples might not be "good" enough to be accepted. This usually causes LMH get stuck in one sub-mode and struggles to switch to other configurations. For example, we have observed this behavior in the GMM example in Figure 4. As a result, LMH typically performs very poorly for programs with stochastic support, particularly in high dimensions.

Note that LMH/RMH could have good mixing rates for many models with fixed support in general akin to standard Metropolis-within-Gibbs methods. This is because when x_i is updated, the rest of the random variables can still be re-used, which ensures a high acceptance rate. That is also why we can still use LMH/RMH as the local inference algorithm within the DCC framework for a fixed SLP to establish substantial empirical improvements.

B. Detailed Algorithm Block for DCC in Anglican

Algorithm 2 An Implementation of DCC in Anglican

Input: Program ***prog***, number of iterations T, inference hyper-parameters Φ (e.g. number of initial iterations T_0, times proposed threshold C_0),

Output: Posterior approximation $\hat{\pi}$ and marginal likelihood estimate $\hat{Z}$

Execute ***prog*** forward multiple times (i.e. ignore observes) to obtain an initial set of discovered SLPs A^{total}
for $t = 1, \ldots, T$ **do**
  Select the model(s) k' in A^{total} whose proposed times $C_{k'} \geq C_0$ and add them into A^{active}
  if exist any new models **then**
    Initialize the inference with N parallel MCMC chains
    Perform T_{init} optimization step for each chain by running a "greedy" RMH locally and only accepting samples with higher $\hat{\gamma}_k$ to boost burning-in; store only the last MCMC samples as the initialization for each chain
    Draw M importance samples using each previous MCMC samples as proposal to generate initial estimate for $\hat{Z}_k$
  end if
  Step 1: select a model $\mathbf{A}_{\mathbf{k}^*}$
  Choose a sub-model with index k^* with the maximum utility value as per Equation 14
  Step 2: perform local inference on $\mathbf{A}_{\mathbf{k}^*}$
  Perform one or more RMH step locally for all N chains of model K^* to update $\hat{\pi}_{k^*}$
  Draw M importance samples using each previous MCMC samples as proposal to update $\hat{Z}_{k^*}$
  Step 3: explore new models (optional)
  Perform one RMH step using a global proposal for all N chains to discover more SLPs $A_{k''}$
  Add $A_{k''}$ to A^{total} if $A_{k''} \notin A^{total}$; increment the $C_{k''}$
end for
Combine local approximations into overall posterior estimate $\hat{\pi}$ and overall marginal likelihood estimate $\hat{Z}$ as per (5)

C. Details for Proof of Theoretical Correctness

In this section, we provide a more formal demonstration of the consistency of the DCC algorithm. We start by explaining the required assumptions, before going on to the main theoretical result.

More formally, our first assumption, which may initially seem highly restrictive but turns out to be innocuous, is that we only split our program into a finite number of sub-programs:

Assumption 1. *The total number of sub-programs K is finite.*

We note that this assumption is not itself used as part of the consistency proof, but is a precursor to Assumptions 4 and 5 being satisfiable. We can always ensure the assumption is satisfied even if the number of SLPs is infinite; we just need to be careful about exactly how we specify a sub-program. Namely, we can introduce a single sub-program that combines all

the SLPs whose path is longer than n_{thresh}, i.e. those which invoke $n_x > n_{\text{thresh}}$ sample statements, and then ensure that the local inference run for this specific sub-program is suitable for problems with stochastic support (e.g. we could ensure we always use importance sampling from the prior for this particular sub-program). If n_{thresh} is then set to an extremely large value such that we can safely assume that the combined marginal probability of all these SLPs is negligible, this can now be done without making any notable adjustments to the practical behavior of the algorithm. In fact, we do not even envisage this being necessary for actual implementations of DCC: it is simply a practically inconsequential but theoretically useful adjustment of the DCC algorithm to simplify its proof of correctness. Moreover, the fact that we will only ever have finite memory to store the SLPs means that practical implementations will generally have this property implicitly anyway.

For better notational consistency with the rest of the paper, we will use the slightly inexact convention of referring to each of these sub-programs as an SLP from now on, such that the K^{th} "SLP" may actually correspond to a collection of SLPs whose path length is above the threshold if the true number of SLPs is infinite.

Our second and third assumptions simply state that our local estimators are consistent given sufficient computational budget:

Assumption 2. *For every SLP $k \in \{1, \ldots, K\}$, we have a local density estimate $\hat{\pi}_k$ (taking the form on an empirical measure) which converges weakly to the corresponding conditional distribution of that SLP π_k in limit of large allocated budget S_k, where $\pi_k(x) \propto \gamma(x)\mathbb{I}[x \in \mathcal{X}_k]$, $\gamma(x)$ is the unnormalized distribution of the program, and $\mathcal{X}_k$ is the support corresponding to the SLP k.*

Assumption 3. *For every SLP, we have a local marginal probability estimate $\hat{Z}_k$ which converges in probability to the corresponding to true marginal probability $Z_k = \int \gamma(x)\mathbb{I}[x \in \mathcal{X}_k]dx$ in limit of large allocated budget S_k. We further assume that if $Z_k = 0$, then $\hat{Z}_k$ also equals 0 with probability 1 (i.e. we never predict non-zero marginal probability for SLPs that contain no mass).*

The final part of this latter assumption, though slightly unusual, will be satisfied by all conventional estimators: it effectively states that we do not assign finite mass in our estimate to any SLP for which we are unable to find any valid traces with non-zero probability.

Our next assumption is that the SLP extraction strategy will uncover all K SLPs in finite time.

Assumption 4. *Let T_{found} denote the number of iterations the DCC approach takes to uncover all SLPs with $Z_k > 0$. We assume that T_{found} is almost surely finite, i.e. $P(T_{found} < \infty) = 1$.*

A sufficient, but not necessary, condition for this assumption to hold is to use a method for proposing SLPs that has a non-zero probability of proposing a new trace from the prior. This condition is satisfied by LMH style proposals like the RMH proposal we adopt in practice. We note that as with Assumption 1, this assumption is not itself used as part of the consistency proof, but is a precursor to Assumption 5 (below) being satisfiable.

Our final assumption is that our resource allocation strategy asymptotically allocates a finite proportion of each of its resources to each SLP with non-zero marginal probability:

Assumption 5. *Let T denote the number of DCC iterations and $S_k(T)$ the number of times that we have chosen to allocate resources to an SLP k after T iterations. We assume that there exists some $\epsilon > 0$ such that*

$$\forall k \in \{1, \ldots, K\}.\ Z_k > 0 \implies \frac{S_k(T)}{T} > \epsilon. \tag{6}$$

Given these assumptions, we are now ready to demonstrate the consistency of the DCC algorithm as follows:

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 :*

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

Proof. By Assumption 5, we have that for all k with $Z_k > 0$, $S_k \to \infty$ as $T \to \infty$. Using this along with Assumptions 2 and 3 gives us that, in the limit $T \to \infty$,

$$\hat{Z}_k \xrightarrow{p} Z_k \quad \forall k \in \{1, \ldots, K\} \tag{7}$$

$$\hat{\pi}_k(\cdot) \xrightarrow{d} \pi_k(\cdot) \quad \forall k \in \{1, \ldots, K\} \text{ with } Z_k > 0 \tag{8}$$

so that all our local estimates converge.

The result now follows through a combination of Equation 5, linearity, and Slutsky's theorem. Namely, let us consider an arbitrary bounded continuous function $f : \mathcal{X} \to \mathbb{R}$, for which we have

$$\begin{aligned}\int f(x)\hat{\pi}(dx) &= \frac{\int f(x)\sum_{k=1}^{K}\hat{Z}_k\hat{\pi}_k(dx)}{\sum_{k=1}^{K}\hat{Z}_k} \\ &= \frac{\sum_{k=1}^{K}\int f(x)\hat{Z}_k\hat{\pi}_k(dx)}{\sum_{k=1}^{K}\hat{Z}_k}.\end{aligned}$$

Given Equations (7) and (8), using Slutsky's theorem, we can conclude that as $T \to \infty$, the above integral converges to

$$\begin{aligned}\frac{\sum_{k=1}^{K}\int f(x)Z_k\pi_k(dx)}{\sum_{k=1}^{K}Z_k} &= \frac{\int f(x)\sum_{k=1}^{K}Z_k\pi_k(dx)}{Z} \\ &= \frac{\int f(x)\gamma(dx)}{Z} \\ &= \mathbb{E}_{\pi(x)}[f(x)].\end{aligned}$$

We thus see that the estimate for the expectation of $f(x)$, which is calculated using our empirical measure $\hat{\pi}(\cdot)$, converges to its true expectation under $\pi(\cdot)$. As this holds for an arbitrary integrable $f(x)$, this ensures, by definition, that $\hat{\pi}(\cdot)$ converges in distribution to $\pi(\cdot)$, thereby giving the desired result. □

We finish by noting that our choices for the particular DCC implementation in Anglican straightforwardly ensure that these assumptions are satisfied (provided we take the aforementioned care around Assumption 1). Namely:

- Using RMH for the local inferences will provide $\hat{\pi}_k$ that satisfies Assumption 2.
- Using PI-MAIS will provide $\hat{Z}_k$ that satisfies Assumption 3 provided we construct this with a valid proposal.
- The method for SLP extraction has a non-zero probability of discovering any of the SLPs with $Z_k > 0$ at each iteration because it has a non-zero probability of proposing a new trace from prior, which then itself has a non-zero probability of proposing each possible SLP.
- The resource allocation strategy will eventually choose each of its possible actions with non-zero rewards (which in our case are all SLPs with $Z_k > 0$) infinitely often, as was proven in (Rainforth et al., 2018).

D. Additional Details on Local Estimators

Recall that the goal for the local inference is to estimate the local target density $\pi_k(x)$ (where we only have access to $\gamma_k(x)$), and the local marginal likelihood Z_k for a given SLP A_k. Each SLP has a fixed support, i.e. a fixed configuration of the random variables, now where many of the complicated factors from varying support no longer apply.

To estimate the local target density $\pi_k(x)$ for a given SLP A_k, DCC establishes a multiple-chain MCMC sampler in order to ensure a good performance in the setting with high dimensional and potentially multi-modal local densities. Explicitly, we perform one RMH step in each chain for N independent chains in total at each iteration. Suppose the total iteration to run local inference in A_k is T_k. With all the MCMC samples $(\hat{x}^{(k)}_{1:N,1:T_k})$ within A_k, we then have the estimator

$$\hat{\pi}_k(x) := \frac{1}{NT_k}\sum_{n=1}^{N}\sum_{t=1}^{T_k}\delta_{\hat{x}^{(k)}_{n,t}}(\cdot). \tag{9}$$

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). Though ostensibly an adaptive importance sampling algorithm, PI-MAIS (Martino et al., 2017) is based around using the set of N proposals each centered on the outputs of an MCMC chain. It can be used to generates marginal likelihood estimates from a set of MCMC chains, as we require.

More precisely, given the series of previous generated MCMC samples, $\hat{x}^{(k)}_{1:N,1:T_k}$, PI-MAIS introduces a mixture proposal distribution for each iteration of the chains by using the combination of separate proposals (e.g. a Gaussian) centered on

each of those chains:

$$q_t^{(k)}(\cdot|\hat{x}_{1:N,t}^{(k)}) := \frac{1}{N}\sum_{n=1}^{N} q_{n,t}^{(k)}(\cdot|\hat{x}_{n,t}^{(k)}) \quad \text{for } t \in \{1, 2, \ldots, T_k\}. \tag{10}$$

This can then be used to produce an importance sampling estimate for the target, with Rao-Blackwellization typically applied across the mixture components, such that M samples, $(\tilde{x}_{n,t,m}^{(k)})_{m=1}^{M}$, are drawn separately from each $q_{n,t}^{(k)}$ with weights

$$\tilde{w}_{n,t,m}^{(k)} := \frac{\gamma_k(\tilde{x}_{n,t,m}^{(k)})}{q_t^{(k)}(\tilde{x}_{n,t,m}^{(k)}|\hat{x}_{n,t}^{(k)})} \text{ with } \tilde{x}_{n,t,m}^{(k)} \sim q_{n,t}^{(k)}(\cdot|\hat{x}_{n,t}^{(k)}), \qquad \text{for } m \in \{1, \ldots, M\} \text{ and } n \in \{1, \ldots, N\}. \tag{11}$$

We then have the marginal likelihood estimate $\hat{Z}_k$ as

$$\hat{Z}_k := \frac{1}{NT_kM}\sum_{n=1}^{N}\sum_{t=1}^{T_k}\sum_{m=1}^{M} \tilde{w}_{n,t,m}^{(k)}. \tag{12}$$

An important difference for obtaining $\hat{Z}_k$ using an adaptive IS scheme with multiple MCMC chain as the proposal, compared to vanilla importance sampling (from the prior), is that MCMC chains form a much more efficient proposal than the prior as they gradually converge to the local target distribution $\pi_k(x)$. These chains are running locally, i.e. restricted to the SLP A_k, which means that the issues of the LMH transitioning between SLPs as discussed in §3 no longer apply and local LMH could maintain a much higher mixing rate where it becomes the standard MwG sampler on a fixed set of variables. Furthermore, the benefit of having multiple chains is that they will approximate a multi-modal local density better. With N chains, we no longer require one chain to discover all the modes but instead only need each mode being discovered by at least one chain. As a result, Equation 12 provides a much more accurate estimator than basic methods.

An interesting point of note is that one can also use the importance samples generated by the PI-MAIS for the estimate $\hat{\pi}_k(x)$, where $\hat{\pi}_k(x)$ from Equation 9 will be

$$\hat{\pi}_k(x) := \sum_{n=1}^{N}\sum_{t=1}^{T_k}\sum_{m=1}^{M} \bar{w}_{n,t,m}^{(k)}\delta_{\tilde{x}_{n,t,m}^{(k)}}(\cdot), \quad \text{where } \bar{w}_{n,t,m}^{(k)} := \tilde{w}_{n,t,m}^{(k)} \Big/ \sum_{n=1}^{N}\sum_{t=1}^{T_k}\sum_{m=1}^{M} \tilde{w}_{n,t,m}^{(k)}. \tag{13}$$

The relative merits of these approaches depend on the exact problem. For problems where the PI-MAIS forms an efficient adaptive importance sampler, the estimate it produces will be preferable. However, in some cases, particularly high-dimensional problems, this sampler may struggle, so that it is more effective to take the original MCMC samples. Though it might seem that we are doomed to fail anyway in such situations, as the struggling of the PI-MAIS estimator is likely to indicate our Z_k estimates are poor, this is certainly not always the case. In particular, for many problems, one SLP will dominate, i.e. $Z_{k^*} \gg Z_{k\neq k^*}$ for some k^*. In that case, we do not necessarily need accurate estimates of the Z_k's to achieve an overall good approximation of the posterior. We just need to identify the dominant Z_k.

E. Additional Details on SLP Extraction

To better understand our SLP extraction procedure, one can imagine that we maintain two stacks of information of SLPs: one `total` stack and one `active` stack (A^{total} and A^{active} respectively in Algorithm 2). The `total` stack records all the information of all discovered SLPs and the `active` stack keeps the SLPs that are believed to be promising so far.

Let's now have a detailed look at Algorithm 2 together. To prevent the rate of models being generated from outstripping our ability to perform inference on current models, we probably only want to perform inference on a subset of all possible sub-models given finite computational budget, which is A^{active}. However, to determine which SLP might be good, i.e. have high posterior mass, is somewhat part of the job of the inference. Therefore, in DCC, we propose that if a model in $\{A^{total} \setminus A^{active}\}$ is "close" enough to the promising models discovered so far as in A^{active}, it would be regarded as being *potentially good* and added to A^{active}. To quantify the closeness, we count how many times a discovered SLP not in A^{active} gets proposed by a model in A^{active} during the global exploration step (at line 13, Algorithm 2). We will add a newly discovered SLP into A^{active} for the resource allocation only when its count reaches some threshold C_0 (line 3).

One might worry the number of models in A^{active} might still go beyond the capacity of the inference engine. We have considered the following design choices to avoid this situation in the DCC in Anglican. The first one is to increase C_0 accordingly as the iteration grows. Intuitively, an SLP needs to be proposed more often to demonstrate that it might be a good one when more computational resources are provided. Another design choice is to control the total number of sub-models in A^{active}. For instance, before one can add a new model in A^{active}, one needs to take an SLP out of A^{active},

e.g. the one with the least $\hat{\gamma}_k$, if the upper bound of the total "active" number has reached. Our DCC also randomly chooses one "non-active" SLPs to perform local inference to ensure the overall correctness.

These design choices, though, are not always necessary if the number of possible sub-models does not explode naturally. For example, in the GMM example, this number is controlled by the value of a Poisson random variable which diminishes quickly, in which case we do not need to further bound the number of active sub-models. But when it comes to the GMM with the misspecified prior where the possible number of active models can easily grow quickly, these design choices become essential. They prevent too much computation resources from being waste on keeping discovering new sub-models rather than being used to perform inference in the discovered ones.

From the practical perspective, one might also want to "split" on discrete variables as their values are likely to affect the downstream program path. This means that for specific discrete variable(s), not only their addresses but also their values are included in defining a program path. It equivalently transforms sampling a discrete variable to observing the variable being a fixed value. The benefit of doing so is when performing inference locally, we no longer need to propose changes for that discrete variable but instead evaluate its conditional probability. Therefore, we can "avoid" proposing samples out of current SLP too often to waste the computation. The posterior of that discrete variable can be obtained from the local marginal likelihood estimates. In our implementation of DCC in Anglican, we require the user to specify which discrete variables that they want to "split" on. Automatically distinguishing which discrete variable will or will not affect program paths is beyond the scope of this paper, and we shall leave it for future work.

F. Additional Details on Resource Allocation

Once a new candidate of SLP has been selected to be added into A_{active}, DCC firstly performs T_{init} optimization steps to boost initialization of inference. Informally, we want to burn in the MCMC chains quickly such that the initialization of each chain would be close to the local mode of the target distribution. By doing so, DCC applies a "greedy" RMH where it only accepts the MCMC samples with the larger $\hat{\gamma}_k$, which enforces the hill-climbing behavior in MCMC to discover modes. Note that only the MCMC samples of the last step in the optimization will be stored as the initialization of each chain and therefore this optimization strategy will not affect the correctness of the local inference.

As introduced in § 6.3, the resource allocation scheme is based on an Upper Confidence Bound (UCB) scheme (Carpentier et al., 2015) developed by Rainforth et al. (2018). We recall the utility function for each SLP being

$$U_k := \frac{1}{S_k}\left(\frac{(1-\delta)\hat{\tau}_k}{\max_k\{\hat{\tau}_k\}} + \frac{\delta\hat{p}_k}{\max_k\{\hat{p}_k\}} + \frac{\beta\log\sum_k S_k}{\sqrt{S_k}}\right) \tag{14}$$

where S_k is the number of times that A_k has been chosen to perform local inference so far, $\hat{\tau}_k$ is the "exploitation target" of A_k, $\hat{p}_k$ is a target exploration term, and δ and β are hyper-parameters.

As proved by Rainforth et al. (2018, §5.1), the optimal asymptotic allocation strategy is to choose each A_k in proportion to $\hat{\tau}_k = \sqrt{Z_k^2 + (1+\kappa)\sigma_k^2}$ where κ is a smoothness hyper-parameter, Z_k is the local marginal likelihood, and σ_k^2 is the variance of the weights of the individual samples used to generate Z_k. Intuitively, this allocates resources not only to the SLPs with high marginal probability mass, but also to the ones having high variance on our estimate of it. We normalize each $\hat{\tau}_k$ by the maximum of $\hat{\tau}_{1:K}$ as the reward function in UCB is usually in $[0, 1]$.

The target exploration term $\hat{p}_k$ is a subjective tail-probability estimate on how much the local inference *could improve* in estimating the local marginal likelihood if given more computations. This is motivated by the fact that estimating Z_k accurately is difficult, especially at the early stage of inference. One might miss substantial modes if only relying on optimism boost to undertake exploration. As per (Rainforth et al., 2018), we realize this insight by extracting additional information from the log weights. Namely, we define $\hat{p}_k := P(\hat{w}_k(T_a) > w_{th}) \approx 1 - \Psi_k(\log w_{th})^{T_a}$, which means the probability of obtaining at least one sample with weight w that exceeds some threshold weight w_{th} if provided with T_a "look-ahead" samples. Here $\Psi_k(\cdot)$ is a cumulative density estimator of the log local weights (eg. the cumulative density function for the normal distribution), T_a is a hyperparameter, and w_{th} can be set to the maximum weight so far among all SLPs. If $\hat{p}_k$ is high, it implies that there is a high chance that one can produce higher estimates of Z_k given more budget.

```
(defdist lik-dist
    [mus std-scalar]
    []   ; auxiliary bindings
    (sample* [this] nil)  ;; not used
    (observe* [this y]  ;; customize likelihood
        (reduce log-sum-exp
            (map #(- (observe* (normal %1 std-scalar) y) (log (count mus)))
                 mus))))

(with-primitive-procedures [lik-dist]
  (defquery gmm-open [data]
     (let [poi-rate 9
           ;; sample the number of total clusters
           K (+ 1 (sample (poisson poi-rate)))
           lo 0.
           up 20.
           ;; sample the mean for each k-th cluster
           mus (loop [k 0
                      mus []]
                 (if (= k K)
                    mus  ;; return mus
                    (let [mu-k (sample (uniform-continuous
                                        (+ lo (* (/ k K) (- up lo)))
                                        (+ lo (* (/ (+ k 1) K) (- up lo)))))]
                      (recur (inc k) (conj mus mu-k)))))
           obs-std 0.1]
       ;; evaluate the log likelihood
       (map (fn [y] (observe (lik-dist mus obs-std) y)) data)
       ;; output
       (cons K mus))))
```

Figure 8. Code example of GMM in Anglican

G. Details on Experiments

G.1. Gaussian Mixture Model

The Gaussian Mixture Model defined in §3 can be written in Anglican as in Figure 8. The kernel density estimation of the data is shown Figure 3 (black line) with the raw data file provided in the code folder.

G.2. Function Induction

The Anglican program for the function induction model in in §7.3 is shown in Figure 9. The prior distribution for applying each rule $R = \{e \to x \mid x^2 \mid \sin(a*e) \mid a*e+b*e\}$ is $P_R = [0.3, 0.3, 0.2, 0.2]$. To control the exploding of the number of sub-models, we set the maximum depth of the function structure being three and prohibit consecutive plus. Both our training data (blue points) and test data (orange points) displayed in Figure 6 are generated from $f(x) = -x + 2\sin(5x^2)$ with observation noise being 0.5 and the raw data files are provided in the code folder.

```
(defm gen-prog [curr-depth max-depth prev-type]
    (let [expr-type (if (< curr-depth max-depth)
                      (if (= prev-type 3)
                         (sample (discrete [0.35  0.35  0.3]))
                         (sample (discrete [0.3  0.3  0.2 0.2])))
                      (sample (discrete [0.5 0.5])))]
        (cond
            (= expr-type 0)
            (if (nil? prev-type)
                (let [_ (sample (normal 0 1))]
                   'x)
                'x)

            (= expr-type 1)
            (let [_ (sample (normal 0 1))]
               (list '* 'x 'x))

            (= expr-type 2)
            (let [a (sample (normal 0 1))
                  curr-depth (+ curr-depth 1)
                  expr-sin (list 'Math/sin
                                 (list '* a (gen-prog curr-depth max-depth expr-type)))]
               expr-sin)

            (= expr-type 3)
            (let [a (sample (normal 0 1))
                  b (sample (normal 0 1))
                  curr-depth (+ curr-depth 1)
                  expr-plus (list '+
                                  (list '* a (gen-prog curr-depth max-depth expr-type))
                                  (list '* b (gen-prog curr-depth max-depth expr-type)))]
               expr-plus))))

(defm gen-prog-fn [max-depth]
    (list 'fn ['x] (gen-prog 1 max-depth nil)))

(defquery pcfg-fn-new [ins outs]
    (let [obs-std 0.5
          max-depth 3
          f (gen-prog-fn max-depth)
          f-x (mapv (eval f) ins)]
        (map #(observe (normal %1 obs-std) %2) f-x outs)
        f))
```

Figure 9. Code example of the Function Induction model in Anglican