Rethinking Variational Inference for Probabilistic Programs with Stochastic Support

Tim Reichelt[1] **Luke Ong**[1,2] **Tom Rainforth**[1]
[1] University of Oxford
[2] Nanyang Technological University, Singapore
{tim.reichelt,lo}@cs.ox.ac.uk rainforth@stats.ox.ac.uk

Abstract

We introduce *Support Decomposition Variational Inference* (SDVI), a new variational inference (VI) approach for probabilistic programs with stochastic support. Existing approaches to this problem rely on designing a single global variational guide on a variable-by-variable basis, while maintaining the stochastic control flow of the original program. SDVI instead breaks the program down into sub-programs with static support, before automatically building separate sub-guides for each. This decomposition significantly aids in the construction of suitable variational families, enabling, in turn, substantial improvements in inference performance.

1 Introduction

Probabilistic programming systems (PPSs) enable users to express probabilistic models with computer programs and provide tools for inference. Many PPS, such as Stan [1] or PyMC3 [2], limit the expressiveness of their language to ensure that the programs in their language always correspond to models with static support—i.e. the number of variables and their support do not vary between program executions. In contrast, *universal PPSs* [3–11] can encode programs where the sequence of variables itself—not just the variable values—changes between executions, leading to models with stochastic support. These models have applications in numerous fields, such as natural language processing [12], Bayesian Nonparametrics [13], and statistical phylogenetics [14]. A wide range of simulator-based models similarly require such stochastic control flow [15–17].

The effectiveness of PPSs is heavily reliant on the underlying inference schemes they support. Variational inference (VI) is one of the most popular such schemes, both in PPSs and more generally [18–20]. This popularity is due to its ability to use derivatives to scale to large datasets and high-dimensional models [21–24], often providing much faster and more scalable inferences compared to Monte Carlo approaches [25]. To provide the required derivatives, a number of modern universal PPSs—such as Pyro [5], ProbTorch [26], PyProb [15], Gen [7], and Turing [6]—have introduced automatic differentiation [27] capabilities for programs with stochastic control flow. One of the core aims behind these developments was to support VI schemes in such settings [5].

However, constructing appropriate variational families, typically known as guides in PPSs, can be very challenging for problems with stochastic support, even for expert users. This is because the stochasticity of the control flow induces discontinuities and complex dependency structures that are difficult to remain faithful to and design parameterized approximations for. Furthermore, while there are a plethora of different automatic guide construction schemes for static support problems [18–20], there is a lack of suitable schemes applicable to models with stochastic support. Consequently, existing methods tend to give unreliable results in such settings, as we demonstrate in Figure 1.

We argue that a significant factor of this shortfall is that standard practice—for both manual and automated methods—is to construct the guide on a variable-by-variable basis [28–31]. Namely,

36th Conference on Neural Information Processing Systems (NeurIPS 2022).

```
def model():
  x = sample("x", Normal(0, 1))
  if x < 0:
    z = sample("z1", Normal(-3, 1))
  else:
    z = sample("z2", Normal(3, 1))
  sample("y", Normal(z, 2), obs=2.0)
```

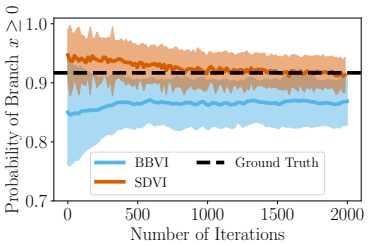


Figure 1: Pyro program with stochastic control flow [Left]. Existing procedures for automatically constructing the guide mirror the control flow of the input program [BBVI, Blue]. However, this produces an inherently limited variational family, leading to unsatisfactory performance despite the problem's simplicity. By breaking down the guide over paths, SDVI [Red] is able to provide accurate inference. Results computed over 10^2 replications, plotted are mean and standard deviation.

existing approaches generally use a single global guide that mirrors the control flow of the input program, then introduce a variational approximation for each unique variable. This is problematic because control flows inherently introduce discontinuities into the program's density function, such that the conditional distribution of each variable will typically change significantly whenever the program path—that is the sequence of random variables—changes. Thus it is extremely challenging to learn a single approximation for each variable that is appropriate across all paths. Further, as the set of variables that exist can itself be stochastic, it is difficult for such guides to appropriately condition on previously sampled variables. Existing automated approaches, therefore, typically rely on mean-field assumptions [29], thereby forgoing any conditioning on the program path itself, consequently leading to poor approximations for most problems.

To overcome these difficulties, we propose *Support Decomposition Variational Inference* (SDVI), a new VI approach based around a novel way of constructing the variational guide. SDVI "rethinks" the guide construction by breaking it down over *paths*, instead of building it on a variable-by-variable basis. Specifically, by utilizing the fact that any program can be reformulated as a mixture of *straight-line programs* (SLPs) [32–34]—each defined by a unique realization of the path—SDVI constructs the guide as a mixture of sub-guides with static support. We show that optimizing the variational objective with this guide structure leads to a natural decomposition of the overall optimization problem into independent sub-problems, each taking the form of a VI with static support. The sub-guides can thus be effectively constructed and trained using more standard VI techniques, before being recombined to form our overall variational approximation. To make SDVI accessible to a wide audience, we have implemented it in Pyro [5]. We evaluate it on a set of example problems with synthetic and real-world data, finding that it provides substantial improvements over existing techniques.

2 Background

2.1 Probabilistic Programs in Universal PPSs

PPSs allow users to express probabilistic models and condition on observed data [31, 35]. A common mechanism to achieve this is to extend standard programming languages with two new primitives: `sample` and `observe`.[1] `sample(id, dist)` draws samples from the distribution object `dist`, where `id` is a unique lexical identifier. `observe(id, data, dist)` enables conditioning on an observed outcome `data`, where `dist` and `id` are as before. For problems admitting a Bayesian formulation, the `sample` and `observe` terms can informally be thought of as prior and likelihood factors respectively.

Universal PPSs allow users to write complex models whose support can vary from one execution to the next, e.g. stochastic branching can mean certain variables only sometimes exist. This can substantially complicate the process of performing inference.

A probabilistic program in a universal PPS defines an unnormalized *density function* $\gamma(x_{1:n_x})$ over the *raw random draws* $x_{1:n_x} \in \mathcal{X}$—defined as the (sequences of) direct outputs of `sample` statements—where $n_x \in \mathbb{N}^+$ is itself potentially random. Though each outcome of $x_{1:n_x}$ uniquely defines a program execution, it is notationally convenient to further associate an *address* a_i to each draw x_i that indicates the position in the program the draw was made. This address can be uniquely defined as the tuple formed by the `id` of the `sample` and the number of times that `sample` has previously been called. For a given execution of the program, the addresses now form an *address path* $A = a_{1:n_x}$.

[1] Pyro instead overloads the `sample` primitive: `sample(id,dist,obs=data)` $\equiv$ `observe(id,data,dist)`.

Each `sample` statement encountered during execution contributes the factor $f_{a_i}(x_i \mid \eta_i)$ to the program density, where a_i is the address of the `sample` statement, f_{a_i} is a parameterized density function, and η_i are its associated parameters. Similarly, each encountered `observe` statement contributes the factor $g_{b_j}(y_j \mid \phi_j)$, with b_j denoting an address, y_j the observed value, g_{b_j} a parameterized density function, and ϕ_j its parameters. Following [36, §4.3.2], we write the program density function as

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

All of $n_x, n_y, a_{1:n_x}$, $\eta_{1:n_x}$, $y_{1:n_y}$, $b_{1:n_y}$, and $\phi_{1:n_y}$ are potentially random variables. The goal of *inference* is to approximate the conditional distribution of the program, which has normalized density $\pi(x_{1:n}) = \gamma(x_{1:n})/Z$ with marginal likelihood $Z = \int_{\mathcal{X}} \gamma(x_{1:n_x}) dx_{1:n_x}$ and the integral is computed with respect to a reference measure that is implicitly defined by the `sample` statements in the program.

2.2 Variational Inference

Variational Inference (VI) [18, 37] solves the inference problem by transforming it to an optimization problem. Specifically, given an unnormalized joint distribution $\gamma(x)$ and a parameterized distribution $q(x;\phi)$, VI computes the variational parameters ϕ such that $q(x;\phi)$ most closely approximates $\pi(x) = \gamma(x)/Z$. This is most commonly done by maximizing the *Evidence Lower Bound* (ELBO) $\mathcal{L}(\phi) := \mathbb{E}_{q(x;\phi)}\left[\log {}^{\gamma(x)}/_{q(x;\phi)}\right]$ via stochastic gradient ascent using Monte Carlo estimates of $\nabla_\phi \mathcal{L}(\phi)$ [38]. Two popular estimators are the score function estimator [39, 40], and the reparameterized gradient estimator [24, 41, 42]. The latter provides lower variance gradient estimates but requires that the distribution $q(x;\phi)$ can be reparameterized and that $\gamma(x)$ is differentiable everywhere.

3 Difficulties for Variational Inference in Universal PPSs

The starting point for any VI scheme is to construct an appropriate variational family, also known as a guide. To automate inference, we desire to (at least partially) automate the process of constructing this guide. Existing methods for this all generate the guide on a variable-by-variable basis [28, 29, 31]: they introduce a single variational distribution $q_{a_i}(x;\phi_{a_i})$ for each unique sampling address a_i, then form the guide by replacing all the original random draws, $x_i \sim f_{a_i}(\cdot \mid \eta_i)$, with draws from the corresponding variational distribution instead. This forms a global guide that maintains the stochastic dependency structure of the original program, such that the guide itself has stochastic support.

Our motivating insight is that this high-level approach has some fundamental limitations. Consider the simple example from Fig. 1. Here the variable `x` influences the program's control flow. This causes discontinuities that mean it is difficult to approximate its conditional density with a single variational approximation, especially if that variational approximation is restricted to a simple distribution class. Here the different possible paths are essentially working against each other, as what is helpful for the approximation of `x` on one path, is generally detrimental for the other.

A further complication occurs when the stochastic control flow of a program influences whether a variable exists at all. Here it can become extremely challenging to set up guides which are faithful to the dependency structure of previously sampled variables [43], as the set of variables we condition on is itself stochastic. Because of this, existing approaches typically rely on mean-field assumptions across paths [28–30]. However, this assumption is rarely reasonable given that the path typically strongly influences the distribution of individual variables.

Finally, creating a single unique variational approximation for each address also leads to challenging optimization problems: the same address can be present in multiple program paths, and the number of variables and their dependencies can vary between different paths.

4 Support Decomposition Variational Inference

We now introduce a novel VI approach to overcome the challenges mentioned in Sec. 3. We call our method *Support Decomposition Variational Inference* (SDVI), because the key design decision is the choice (and automatic construction) of a guide that takes the form of a mixture distribution over the set of possible paths a program can take. That is, rather than constructing a single guide with the same stochastic control flow as the original program and a separate variational approximation for

each *unique address*, we instead construct separate sub-guides with deterministic control flows for each *unique path*. These are then combined into an overall guide using the mixture distribution which maximizes the overall ELBO. As we will see, this alternative approach substantially simplifies the process of constructing effective guides and allows the full weight of the well-developed techniques for VI in the static support setting to be brought to bear on problems with stochastic support.

4.1 Decomposing Probabilistic Programs into Straight-Line Programs

As noted by, e.g., [32–34], all probabilistic programs can be reformulated as mixture distributions over *straight-line programs* (SLPs), which are sub-programs without any stochastic control flow. Building on our earlier notation, the constituent SLPs of a program correspond directly to the unique instances of the program path, A. Given the path, the set of variables making up the raw random draws in the program is fixed, along with their form and reference distribution; that is each SLP represents a probabilistic model with fixed variable typing and support.

Following the notation of Zhou et al. [34], we can apply an arbitrary fixed ordering on the set of SLPs in a program, such that we can uniquely define and index them using the set of possible addresses A_k for $k \in \mathcal{K}$, where $\mathcal{K}$ is a countable (but potentially infinite) indexing set. Each SLP A_k now corresponds to a particular sub-region, $\mathcal{X}_k$, of the raw random draw sample space, $\mathcal{X}$. These sub-regions are disjoint and their union is the full sample space. Unlike $\mathcal{X}$, each element in any given $\mathcal{X}_k$ has the same length n_k and is measurable with respect to the same reference measure. The unnormalized density for the kth SLP is now given by

$$\gamma_k(x_{1:n_k}) = \mathbb{I}[x_{1:n_k} \in \mathcal{X}_k]\, \gamma(x_{1:n_k}) = \mathbb{I}[x_{1:n_k} \in \mathcal{X}_k] \prod_{i=1}^{n_k} f_{A_k[i]}(x_i \mid \eta_i) \prod_{j=1}^{n_y} g_{b_j}(y_j \mid \phi_j), \quad (2)$$

and the unnormalized density function for the original program can be written as a simple sum of the individual SLP densities: $\gamma(x_{1:n_x}) = \sum_{k \in \mathcal{K}} \gamma_k(x_{1:n_x})$. The corresponding normalized conditional density can then be written as mixture distribution over the conditional distributions of the individual SLPs, with mixture weights given by their (normalized) local partition functions:

$$\pi(x) = \sum_{k \in \mathcal{K}} \pi(x \mid k)\, \pi(k) \;\text{ where }\; \pi(x \mid k) = \frac{\gamma_k(x)}{Z_k},\; \pi(k) = \frac{Z_k}{\sum_{\ell \in \mathcal{K}} Z_\ell},\; Z_k = \int_{\mathcal{X}_k} \gamma_k(x) dx. \quad (3)$$

4.2 Decomposing the Variational Family into Straight-Line Programs

The key idea behind SDVI is now to construct the guide using a factorization that is analogous to that of the SLP decomposition above. Precisely, we aim to learn variational approximations of the form

$$q(x; \phi, \lambda) = \sum_{k=1}^{K} q_k(x; \phi_k) q(k; \lambda) \quad (4)$$

where $q(k; \lambda)$ defines a categorical distribution over the indices of the SLPs, with support $k \in \{1, \ldots, K\}$; and $q_k(x; \phi_k)$ is the local guide of the kth SLP, with support $x \in \mathcal{X}_k$. Critically, as each $\mathcal{X}_k$ represents a fixed support, the local variational families q_k can be automatically constructed using standard techniques for static problems, as we discuss in Sec. 4.5. Note that it is valid for the guide $q(x; \phi, \lambda)$ to not cover all SLPs, i.e. it is possible that $K < |\mathcal{K}|$.

Writing $\phi = \{\phi_k\}_{k=1}^{K}$, the KL divergence we wish to minimize for standard VI is now

$$\mathrm{KL}(q(x; \phi, \lambda) \parallel \pi(x)) = \mathbb{E}_{q(x;\phi,\lambda)} \left[\log q(x; \phi, \lambda) - \log \pi(x)\right], \quad (5)$$

which we call the *global KL divergence*. By standard reasoning, minimizing this is equivalent to maximizing the *global ELBO*

$$\mathcal{L}(\phi, \lambda) = \mathbb{E}_{q(x;\phi,\lambda)} \left[\log \gamma(x) - \log q(x; \phi, \lambda)\right] \quad (6)$$

which, as we show in App. A, can be rewritten as

$$\mathcal{L}(\phi, \lambda) = \mathbb{E}_{q(k;\lambda)} \left[\mathcal{L}_k(\phi_k) - \log q(k; \lambda)\right], \;\text{ where }\; \mathcal{L}_k(\phi_k) := \mathbb{E}_{q_k(x;\phi_k)} \left[\log \frac{\gamma_k(x)}{q_k(x; \phi_k)}\right] \quad (7)$$

is the term we refer to as the *local ELBO* for the kth SLP. Notice that each $\mathcal{L}_k(\phi_k)$ depends only on the parameter ϕ_k and the local SLP density γ_k; it is completely independent of $q(k; \lambda)$, the other SLPs, and $\phi_{k'}$ for $k' \neq k$. Thus, it follows from (7) that the inference problem for the whole program can be decomposed into independent 'local' inference problems for the component SLPs, along with establishing the mixture probabilities $q(k; \lambda)$. Furthermore, it turns out that the optimal $q(k; \lambda)$ is simply the softmax of $\mathcal{L}_1, \ldots, \mathcal{L}_K$, as shown by the following result.

Algorithm 1 Support Decomposition Variational Inference

Require: Target program γ, iteration budget T, minimum no. of SH candidates m

Extract SLPs $\{\gamma_k\}_{k=1}^K$ from γ and set $\mathcal{C} = \{1, \ldots, K\}$ ▷ Sec 4.3
Formulate guide q_k for each SLP and initialize parameters ϕ_k ▷ Sec 4.5
for $l = 1, \ldots, L = \lceil \log_2(K) - \log_2(m) + 1 \rceil$ **do**
  for $k \in \mathcal{C}$ **do**
    Perform $\lfloor T/L|\mathcal{C}| \rfloor$ optimization iterations of ϕ_k targeting $\mathcal{L}_{\text{surr},k}(\phi_k)$ ▷ Sec 4.5
  end for
  Remove $\min(\lfloor |\mathcal{C}|/2 \rfloor, |\mathcal{C}| - m)$ SLPs from $\mathcal{C}$ with the lowest $\mathcal{L}_k(\phi_k)$ ▷ Sec. 4.4
end for
Truncate q_k outside of SLP support, $\mathcal{X}_k$, using Eq. (13)
Estimate each $\mathcal{L}_k(\phi_k)$ using Monte Carlo estimate of Eq. (7)
Calculate $q(k; \lambda)$ according to Eq. (8) and return $q(x; \phi, \lambda)$ as per Eq. (4)

Proposition 1. *Let $L = \{\mathcal{L}_1, \ldots, \mathcal{L}_K\}$ be the set of local ELBOs, defined as per (7), where L is countable but potentially not finite. If $0 < \sum_{k=1}^K \exp(\mathcal{L}_k) < \infty$, then the optimal corresponding $q(k; \lambda)$ in terms of the global ELBO (6) is given by*

$$q(k; \lambda) = \exp(\mathcal{L}_k) \Big/ \sum_{\ell=1}^K \exp(\mathcal{L}_\ell). \tag{8}$$

The proof of this result is given in App. A. Though each of the $\mathcal{L}_k$ terms here is itself intractable, they can be estimated efficiently and accurately by simple Monte Carlo. We can thus straightforwardly construct $q(k; \lambda)$ once we have learned our local variational approximations: noting that these two processes are separable, $q(k; \lambda)$ is not needed until *after* the individual q_k are trained.

4.3 Finding SLPs

We have just shown how we can solve the VI problem of a probabilistic program with stochastic support by reducing it to a set of *independent* and *simpler* VI problems, each concerning an SLP, a program with static support. However, we still need a mechanism to 'discover' the SLPs, i.e. extract the possible address paths from a program.

Here we first note that we only need to consider an SLP if it has a non-zero probability of being identified under forward simulation of the program, while ignoring conditioning statements. This hints at a cheap and simple discovery mechanism whereby we draw samples by forward simulation and take note of the unique paths that have been generated. We can either do this upfront, or in an online manner whereby we seek new SLPs as our budget increases and we have scope to deal with them (see App. B for details). Although this is a stochastic process that is not guaranteed to find all the SLPs for finite budgets, for the problems considered in our experiments, it was always able to reliably identify all SLPs with *non-negligible posterior mass*. Nonetheless, this approach may not be sufficient for all problems, such as when the likelihood concentrates in an area of very low prior mass. Here one should instead look to employ more sophisticated discovery methods instead, such as those based on MCMC sampling [34] or static analysis of the program code [32, 44, 45].

4.4 Allocating Resources

Using the same amount of computational budget on each SLP is potentially wasteful, particularly if there is a large number of SLPs with insignificant marginal likelihoods. Therefore, we seek a scheme that allocates more computational resources to promising SLPs, making sure to exploit the fact that the different inference problems are trivially parallelizable.

To formalize this resource allocation problem, let T represent some fixed resource budget. Further, let t_k be the amount of this budget we spend on optimizing the kth SLP, such that $\sum_k t_k = T$ at the end of our training. Our ultimate aim is produce the maximum possible final global ELBO, which will be a function of $\phi_1(t_1), \ldots, \phi_K(t_K)$, where $\phi_k(t_k)$ denotes the value of ϕ_k achieved after allocating t_k resources to that SLP. By plugging the optimal mixture distribution $q(k; \lambda)$ from (8) into (6), we see that, after some rearranging, our resource allocation can be formulated as trying to maximize

$$\mathcal{L}(\phi, \lambda^*) = \log \sum_{k=1}^K \exp(\mathcal{L}_k(\phi_k(t_k))) \quad \text{s.t.} \sum_{k=1}^K t_k = T. \tag{9}$$

In practice, this is not a suitable objective for controlling our resource allocation directly, as it is still itself a random variable given $t_1, \ldots, t_K$, because the optimization procedure is stochastic. Moreover, we cannot consider its expectation, since the distribution of the $\phi_k(t_k)$ is unknown. However, it does provide insight into how we ideally would like to allocate resources: we want to allocate more resources to SLPs whose exponentiated ELBOs are significant. In particular, we can think of the 'reward' for allocating ϵ more resources to SLP k as $\exp(\mathcal{L}_k(\phi_k(t_k + \epsilon))) - \exp(\mathcal{L}_k(\phi_k(t_k)))$.

One could now, in principle, formulate the problem as a sequential decision making problem [46]. However, the diminishing nature of the rewards and the fact that they are highly unlikely to be sub-Gaussian, along with the need to allow choosing multiple arms at once for parallelization, mean that setting up such an approach which is effective in practice is likely to be quite challenging.

Instead, we propose a simple heuristic, based on the *Successive Halving* algorithm (SH) [47] (see App. B for a description), an approach commonly used for resource allocation in hyperparameter optimization (HO) [48, 49]. In standard SH, the final objective is to identify and train the single best candidate, whereas ours is to maximize the *sum* of all the local ELBOs. Despite this difference, the use of SH can still be justified by the fact that the distribution over $\exp(\mathcal{L}_k(\phi_k(\infty)))$ will typically be heavily concentrated to a small number of SLPs, often only a single one. Nonetheless, we make a small adaptation to the approach to stop over-focusing on a single SLP: we stop the halving process when a chosen number, $1 \leqslant m \leqslant K$, of the candidates are left, with $m = 1$ corresponding to standard SH. The minimum proportion of the budget allocated to any given candidate by this scheme is $1/(K\lceil \log_2 K - \log_2 m + 1 \rceil)$, so we can use m as a hyperparameter to control how evenly resources are allocated, with $m = K$ corresponding to uniform allocation. This approach is also helpful for parallelization, as we can set m equal to the number of available cores.

Putting everything together, a summary of our SDVI algorithm is given in Algo. 1. In App. B, we further show how this can be extended to an online variant of the approach, wherein we repeatedly run SH using the objective $\exp(\alpha \mathcal{L}_k(\phi_k(t_k)))/t_k$, where $0 < \alpha \leqslant 1$ is a hyperparameter, with smaller values of α encouraging more exploration.

4.5 Formulating and Training the Local Guides

In Algo. 1 we assume a mechanism to construct the *local guide* $q_k(x; \phi_k)$ for each SLP specified by path A_k. In many situations—notably when the program path is uniquely determined by the sampled values from discrete distributions—it is possible to construct guides q_k that are guaranteed to place support within the sub-region $\mathcal{X}_k$, which, in turn, allows us to use the reparameterized gradient estimator for the gradients of $\mathcal{L}_k(\phi_k)$. Many models encountered in practice, e.g. mixture models [13], have this property. In this case it is possible to eliminate all the variables which influence the control flow by conditioning, effectively setting them to constants; see App. C for further details.

In situations where we cannot easily construct a q_k which places support only within $\mathcal{X}_k$, we need to take care when training our guide. Recall that for path A_k the number of variables n_k and their type sequence is fixed, which allows us to construct an initial guide $\tilde{q}_k$ with correct dimensionality and variable typing. Let the support of this guide be denoted by $\mathcal{X}'_k = \text{supp}(\tilde{q}_k)$. In general, we will have $\mathcal{X}_k \subset \mathcal{X}'_k$, because the control flow in the program imposes additional constraints on each individual variable. Having constructed a guide with $\text{supp}(\tilde{q}_k) = \mathcal{X}'_k$, one might be tempted to optimize $\text{KL}(\tilde{q}_k(x; \phi_k) \parallel \pi(x \mid k))$, but we cannot guarantee the absolute continuity condition (i.e. $\tilde{q}_k(x; \phi_k) = 0$ if $\pi(x \mid k) = 0$), and so, the KL divergence may not be well-defined, giving an ELBO of $-\infty$. To alleviate this issue we temporarily create a new surrogate target density defined as

$$\tilde{\gamma}_k(x_{1:n_k}) := \gamma_k(x_{1:n_k}) + c\,\mathbb{I}[x_{1:n_k} \notin \mathcal{X}_k], \tag{10}$$

for a small positive, finite constant c. This surrogate density is used solely for optimizing $\tilde{q}_k(x; \phi_k)$. We train ϕ_k to optimize the corresponding surrogate ELBO

$$\mathcal{L}_{\text{surr},k}(\phi_k) := \mathbb{E}_{\tilde{q}_k(x;\phi_k)}\left[\log \tilde{\gamma}_k(x) - \log \tilde{q}_k(x; \phi_k)\right]. \tag{11}$$

We need to be careful to choose an appropriate c that is sufficiently small compared to the values of $\gamma_k(x)$ for $x \in \mathcal{X}_k$, which we ensure by setting c adaptively. During the SLP discovery phase (Line 1 in Algo. 1), we keep track of the smallest density value encountered so far, and call that d_{min}. We then set $c = 0.01 d_{min}$ to ensure that the density values for $\tilde{\gamma}_k(x)$ outside of $\mathcal{X}_k$ are significantly below the values of $\tilde{\gamma}_k(x)$ for $x \in \mathcal{X}_k$. Hence, optimizing (11) faithfully optimizes q_k to be a good approximation to $\gamma_k(x)$ while avoiding the issues of infinite ELBO values. While $\tilde{\gamma}_k$ is not a proper

unnormalized density (it will in general not integrate to a finite value) this is not an issue in practice due to the mode-seeking behaviour of optimizing the ELBO.

Unfortunately, the bounds on the support of the SLP inevitably create a discontinuity in the objective. Thus, for fully unbiased gradients we need to use the score function estimator or some extension thereof. However, in some cases, the bias of the reparameterization gradient estimator may be sufficiently small to warrant its use. Note that $\mathcal{L}_{\text{surr},k}(\phi_k)$ retains the desirable property of the standard ELBO that, if the observations are conditionally independent given the latent variables, we can get unbiased estimates of the ELBO using minibatches of the full dataset [19, 22, 42].

Further we need to be careful when initializing $\tilde{q}_k$ as we require it to place sufficient probability mass within $\mathcal{X}_k$ to provide a suitable training signal. To ensure this, we initialize ϕ_k by minimizing the *forward* KL divergence between the prior density of the kth SLP and $\tilde{q}_k(x;\phi_k)$

$$\mathrm{KL}(\pi_{prior,k}(x) \parallel \tilde{q}_k(x;\phi_k)) \propto \mathbb{E}_{\pi_{prior}(x)}\left[-\mathbb{I}[x \in \mathcal{X}_k] \log \tilde{q}_k(x;\phi_k)\right] \tag{12}$$

where $\pi_{prior}(x_{1:n_x}) := \prod_{i=1}^{n_x} f_{a_i}(x_i \mid \eta_i)$. This objective can be optimized via stochastic gradient descent (cf. App. C). Note that, for the purpose of initialization, we are targeting the prior, and thus we do not have to resort to expensive schemes to estimate the gradients which are necessary if one aims to minimize the forward KL targeting the posterior [50].

So far we have outlined how to train $\tilde{q}_k$ but to evaluate the local ELBOs, $\mathcal{L}_k$, we need to construct a distribution q_k which satisfies the hard constraint $\mathrm{supp}(q_k) = \mathcal{X}_k$. Our solution for this is *truncating* $\tilde{q}_k$ by checking whether specific raw random draws $x'_{1:n_k}$ are valid for the path A_k, i.e. whether $\mathbb{I}\left[x'_{1:n_k} \in \mathcal{X}_k\right]$. We can do this by simply executing the program with fixed draws set to $x'_{1:n_k}$ and then noting that the program terminates and follows the address path A_k if, and only if, $x'_{1:n_k} \in \mathcal{X}_k$. Thus, we truncate $\tilde{q}_k$ using

$$q_k(x;\phi_k) = \frac{\tilde{q}_k(x;\phi_k)\mathbb{I}\left[x \in \mathcal{X}_k\right]}{\tilde{Z}_k(\phi_k)}, \quad \text{where} \quad \tilde{Z}_k(\phi_k) = \int_{\mathcal{X}'_k} \tilde{q}_k(x;\phi_k)\mathbb{I}\left[x \in \mathcal{X}_k\right] dx. \tag{13}$$

Hence, q_k is implicitly defined as the output of a rejection sampler with $\tilde{q}_k$ as a proposal. Note, that as we use the surrogate ELBO in (11) when training ϕ_k, we never need to take gradients through q_k or $\tilde{Z}_k(\phi_k)$, thereby avoiding the significant practical issues this would cause (see App. G). Thus, the local guide q_k (Eq. (13)) is only used for estimating the local ELBOs (Eq. (7)). This is done by first drawing N samples $\{x^{(i)}\}_{i=1}^N$ from $\tilde{q}_k$, then rejecting samples which do not fall into the SLP and estimate $\tilde{Z}_k$ as the acceptance rate of this sampler (i.e. N_A/N where N_A is the number of samples accepted). Using A to denote the set of indices of accepted samples, we form our ELBO estimate as

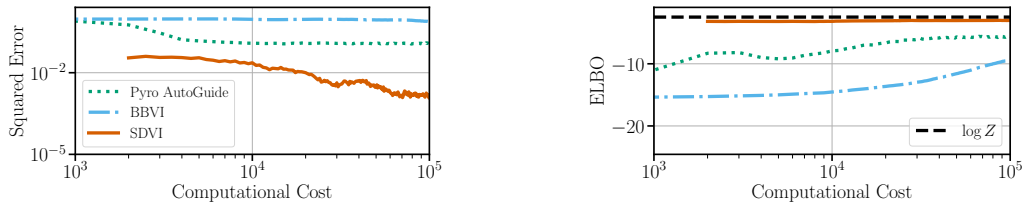
$$\hat{\mathcal{L}}_k := \frac{1}{N_A} \sum\nolimits_{i \in A} \log(N_A\, \gamma_k(x^{(i)})) - \log(N\, \tilde{q}_k(x^{(i)};\phi_k)). \tag{14}$$

Note here that A and N_A are random variables that both implicitly depend on ϕ_k, which is why we can use this for estimation, but not training.

5 Related Work

The vast majority of prior work on deriving automated VI algorithms focuses on the setting of static support [20, 23, 51–55]. Of particular note, [43, 56, 57] also consider using variational families that do not match the dependency structure of the original problem, but they still require static support. More generally, there have been models with stochastic support for which bespoke guides where developed which do not follow the control-flow structure of the input program [58]. However, these custom guides do not leverage the breakdown of the input program into SLPs.

The Divide-Conquer-Combine (DCC) algorithm [34] also exploits the breakdown of the program density into individual SLPs. However, [34] mainly focused on local inference algorithms that are sampling based, especially MCMC. As we showed in Sec. 4 unique challenges and opportunities arise when we consider the breakdown from a variational perspective. Further, our work shows that using a variational family based on SLPs naturally leads to divide-and-conquer style algorithm, due to the resulting separability of the ELBO. One of the most practical differences is that SDVI only requires (exponentiated) ELBOs to be estimated for each SLP, rather than marginal likelihoods. The former can typically be estimated substantially more accurately for a given budget, allowing SDVI to



(a) Squared error.

(b) ELBOs.

Figure 2: Results for the model in § 6.1. Computational cost is measured in the number of likelihood evaluations. For each metric we show the mean and standard deviation over 10 runs. a) Squared error between the true SLP weights and the estimated SLP weights. b) Evidence Lower Bounds (ELBOs) for the variational algorithms, dashed line indicates the analytic log marginal likelihood.

scale better to high dimensional problems (see Sec. 6.2). [32] and [33] both also use the general idea of breaking down programs into SLPs, but both papers consider starkly different problem settings. Neither have any direct link to variational inference.

Our work is situated in the larger context of automated inference for universal PPSs. Other popular approaches include particle-based methods [6, 11, 59–61] and MCMC approaches with automated proposals [1, 62–64]. Some work has looked to perform *amortized* inference over a range of possible datasets [16, 56, 57, 65], often by training a proposal that is similar to a variational approximation.

6 Experiments

To make SDVI easily accessible to practitioners we have implemented it in Pyro with code available at `github.com/treigerm/sdvi_neurips`. The first baseline we consider, **Pyro AutoGuide**, uses the `AutoNormalMessenger` class to automatically generate a guide, and trains it with Pyro's built-in tools for VI (`http://pyro.ai/examples/svi_part_i.html`). As an additional VI baseline, we also implement a custom guide for each model which uses the variable-by-variable scheme outlined in Sec. 3, in combination with the score function gradient estimator; we refer to this baseline as **BBVI**. For SDVI, we run SH until there are 10 active SLPs left (i.e. $m = 10$ in Algo. 1) and parallelize the computation across 10 cores. We further construct each local guide distribution q_k as a mean-field normal. The specific configurations for each method for each experiment are provided in App. D.

6.1 Program with Normal Distributions

We use our first experiment to further clarify the failure modes of existing VI approaches. We consider an extension of the model from Fig. 1 to contain more SLPs. The full model is

$$\begin{aligned} u &\sim \mathcal{N}(0, 5^2), \\ x &\sim \mathcal{N}(z, 1), \\ y &\sim \mathcal{N}(x, 1). \end{aligned} \qquad \text{where} \qquad z = \begin{cases} 0, & \text{if } u \in (-\infty, -4] \\ K, & \text{if } u \in (-5+K, -4+K] \text{ for } K = 1, \dots, 8 \\ 9, & \text{if } u \in (4, \infty) \end{cases} \tag{15}$$

We assume we have observed $y = 2$. The results in Fig. 2 demonstrate that SDVI is able to overcome the limitations of the other variational approaches. BBVI and Pyro AutoGuide both use the same guide in this model; BBVI uses the score function gradient estimator for training, whereas Pyro AutoGuide uses the reparameterized gradient estimator. This difference results in different posterior approximations for the different baselines. The BBVI guide tends to place all its mass on a single SLP and then provides a suitable approximation for only that one SLP, ignoring all the others. This explains the large standard deviations for the ELBO values in Fig. 2b as the ELBOs in different SLPs will converge to drastically different values. For Pyro AutoGuide the biased gradient estimates will train the variational approximation for variable u to be close to the prior $\mathcal{N}(0, 5^2)$. SDVI is able to avoid the shortcomings of the baselines as it provides an overall better posterior approximation leading to larger ELBO values, i.e. lower KL divergences to the true posterior, and a more accurate weighting of the different SLPs (Fig. 2a).

Table 1: Log posterior predictive densitiy (LPPD), ELBO, and maximum a posteriori (MAP) estimate for K for GMM model. Mean and standard deviation for LPPD and ELBO computed over 5 runs.

Method	LPPD ($\uparrow, \times 10^3$)	ELBO ($\uparrow, \times 10^3$)	MAP K
DCC	-9842.90 ± 3904.57	N/A	14, 11, 16, 14, 15
BBVI	-2217.07 ± 146.31	-8770.55 ± 544.95	25, 25, 25, 25, 25
SDVI	$\mathbf{32.84 \pm 0.02}$	$\mathbf{128.76 \pm 0.17}$	5, 5, 6, 6, 5
S-SDVI	$\mathbf{32.80 \pm 0.02}$	$\mathbf{128.63 \pm 0.22}$	5, 5, 6, 5, 6

6.2 Infinite Gaussian Mixture Model

Our next model is a Gaussian Mixture Model (GMM) with a Poisson prior on the number of clusters:

$$K \sim \text{Poisson}(9) + 1; \quad u_k \sim \mathcal{N}(\mathbf{0}, 10\,\mathrm{I}) \text{ for } k = 1, \ldots, K; \quad y \sim \frac{1}{K} \sum_{k=1}^{K} \mathcal{N}(\mu_k, 0.1\,\mathrm{I}),$$

where I is the $D \times D$ identity matrix and $\mathbf{0}$ is a D dimensional vector of zeros (we set $D = 100$). A similar model was considered in Zhou et al. [34] but with $D = 1$ instead of $D = 100$. We generate a dataset of 1250 observations with $K = 5$. To compare and evaluate the different algorithms, we hold out 250 data points as a test dataset to compute the log posterior predictive density (LPPD).

The Pyro AutoGuide baseline from the previous experiment is not applicable here since it assumes all latent variables are continuous. In BBVI, for practical reasons, we had to cap the maximum number of clusters in the guide at 25 (cf. App. D). To provide a further baseline, we have also implemented **DCC** [34] in Pyro with Random-walk lightweight Metropolis-Hastings (RMH) [63] as a local inference algorithm. We chose DCC in particular because it also exploits the same breakdown into SLPs, so comparing against DCC is an opportunity to highlight the benefits of using a VI method.

In this model, the observations are assumed to be conditionally independent given the latent variables, thus enabling SDVI to work on subsets of the whole dataset [19, 22]. Specifically, we run SDVI on a model which samples a random minibatch of size $B = 100$ at each iteration and then scales the likelihood by the factor N/B, where $N = 1000$ is the size of the full dataset; we refer to this setup as Stochastic SDVI (S-SDVI). Furthermore, for this model SDVI is able to directly construct valid local guides q_k (using the mechanism for models branching on discrete variables outlined in Sec. 4.5) and therefore (S-)SDVI can use the reparameterized gradient estimator.

Table 1 shows that SDVI and S-SDVI significantly outperform the baselines, yielding a several orders of magnitude larger posterior predictive density and providing the only reasonable predictions for the numbers of clusters. In the few instances were (S-)SDVI returns a suboptimal MAP estimate of $K = 6$, this was because the local guide for the SLP with 5 components had fallen into a local model that fails to correctly identify all the clusters in the data, in turn returning a suboptimal local ELBO. BBVI and DCC struggle with this model due to the high-dimensional parameter space. DCC's local inference algorithm, RMH, only updates one variable at a time which results in slow mixing times. Note, DCC does not provide any ELBO values; its marginal likelihood estimator PI-MAIS [66] constructs an importance sampling (IS) proposal distribution based on the outputs of MCMC chains which could theoretically be used to estimate an ELBO value. However, as IS requires over-dispersed proposals, the ELBO scores for this approach are trivially $-\infty$, preventing a sensible comparison.

6.3 Inferring Gaussian Process Kernels

For our final experiment, we consider the problem of inferring the kernel structure of a Gaussian Process (GP). Following [67, 68], we place a prior over kernel functions using a probabilistic context-free grammar (PCFG) . We consider the squared exponential (SE), rational quadratic (RQ), periodic (PER), and linear (LIN) base kernels, and use the production rules

$$\mathcal{K} \rightarrow \text{SE} \mid \text{RQ} \mid \text{PER} \mid \text{LIN} \mid \mathcal{K} \times \mathcal{K} \mid \mathcal{K} + \mathcal{K}.$$

Sampling from the PCFG is implemented with a recursive probabilistic program that uses sam-

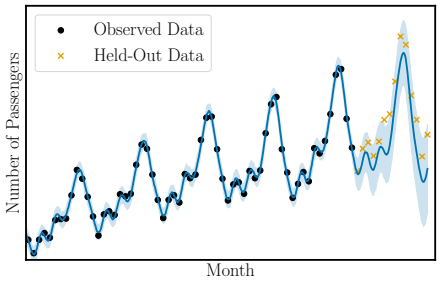


Figure 3: Posterior predictions of the GP for SDVI, shaded regions indicate 2 standard deviations that are computed from 100 posterior samples.

ples from a categorical distribution to decide which production rule in the PCFG should be applied. In addition to the kernel structure, we also perform inference over the kernel hyperparameters for each base kernel and the observation noise; we place an inverse-gamma prior on each base kernel hyperparameter and a half-normal prior on the observation noise. We further assume a normal likelihood function and marginalize out the latent GP. Additional model details are in App. D. We apply this model to a dataset of monthly counts of international airline passengers [69], withholding the last 10 % of all observations as a test dataset.

For SDVI we can construct local valid proposals q_k using the mechanism for models with discrete branching outlined in Sec 4.5. Hence, in each SLP the local guide q_k provides a posterior approximation over the kernel hyperparameters and the observation noise; the posterior distribution over kernel structures is implicitly defined through the mixture distribution over program paths. Table 2 shows that SDVI provides higher LPPD values, and is also able to achieve a higher final ELBO value compared to BBVI. Fig. 3 shows the posterior predictions for the SDVI run with the median LPPD score. SDVI is able to provide qualitatively reasonable predictions, as the predictions follow the periodic trend in the observed data.

Table 2: Final log posterior predictive densitiy (LPPD) and ELBO for GP model. Shown are mean and standard deviation computed over 5 runs.

Method	LPPD (↑)	ELBO (↑)
DCC	-58.92 ± 32.47	N/A
BBVI	-18.82 ± 1.20	-48.48 ± 0.33
SDVI	$\mathbf{2.05 \pm 3.30}$	$\mathbf{34.53 \pm 21.42}$

7 Discussion

We believe that SDVI provides a number of significant contributions towards the goal of effective (automated) inference for probabilistic programs with stochastic support, nonetheless it still naturally has some limitations. Perhaps the most obvious is that it, if there is a very large number of SLPs that cannot be easily discounted from having significant posterior mass, it can be challenging to learn effective variational approximations for all of them, such that SDVI is likely to perform poorly if the number becomes too large. Here, customized conventional VI or reversible jump MCMC approaches might be preferable, as they can be set up to focus on the transitions between SLPs, rather than trying to carefully characterize individual SLPs.

Another limitation is that our current focus on automation means that there are still open questions about how best to construct more customized guides within the SDVI framework. Here the breakdown into individual SLPs and use of resource allocation strategies will still often be useful, but changes to our implementation would be required to allow more user control and customization. For example, the discovery of individual SLPs using the prior is a potential current failure mode, and it would be useful to support the use of more sophisticated program analysis techniques (e.g. [45]).

A more subtle limitation is that the local inferences of each SLP can sometimes still be quite challenging themselves. If the true posterior places a lot of mass near the boundaries of the SLP, there can still be a significant posterior discontinuity, meaning we might need advanced local variational families (e.g. normalizing flows) and/or gradient estimators. Such problems also occur in static support settings and are usually much more manageable than the original stochastic support problem, but further work is needed to fully automate dealing with them.

Finally, variational methods are often used not only for inference, but as a basis for model learning as well. In principle, SDVI could also be used in such settings, but as described in App. F, there are still some hurdles that need to be overcome to do this in practice.

8 Conclusion

We have presented SDVI and shown that it is able to overcome the limitations of existing VI approaches for programs with stochastic support by using a novel guide structure that breaks the program down into SLPs with fixed support, rather than matching the original stochastic control flow. The structure of the variational family separates the ELBO into multiple independent inference problems which naturally motivates a divide-and-conquer style training procedure with explicit resource allocation. Experimentally we found that these innovations meant that SDVI was able to provide significant performance improvements over the previous state-of-the-art approaches.

Acknowledgments and Disclosure of Funding

We would like to thank Yuan Zhou for useful discussions in the early stages of this project. Tim Reichelt is supported by the UK EPSRC CDT in Autonomous Intelligent Machines and Systems with the grant EP/S024050/1. Luke Ong would like to acknowledge funding from EPSRC UK and National Research Foundation Singapore NRF-RSS2022-009.

References

[1] Bob Carpenter, Andrew Gelman, Matthew D. Hoffman, Daniel Lee, Ben Goodrich, Michael Betancourt, Marcus Brubaker, Jiqiang Guo, Peter Li, and Allen Riddell. Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, 76:1–32, January 2017. ISSN 1548-7660. doi: 10.18637/jss.v076.i01. URL https://doi.org/10.18637/jss.v076.i01.

[2] John Salvatier, Thomas V. Wiecki, and Christopher Fonnesbeck. Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55, April 2016. ISSN 2376-5992. doi: 10.7717/peerj-cs.55. URL https://peerj.com/articles/cs-55.

[3] David Tolpin, Jan-Willem van de Meent, Hongseok Yang, and Frank Wood. Design and Implementation of Probabilistic Programming Language Anglican. In *Proceedings of the 28th Symposium on the Implementation and Application of Functional Programming Languages - IFL 2016*, pages 1–12, Leuven, Belgium, 2016. ACM Press. ISBN 978-1-4503-4767-9. doi: 10.1145/3064899.3064910. URL http://dl.acm.org/citation.cfm?doid=3064899.3064910.

[4] Noah D. Goodman, Vikash K. Mansinghka, Daniel Roy, Keith Bonawitz, and Joshua B. Tenenbaum. Church: A language for generative models. In *Proceedings of the Twenty-Fourth Conference on Uncertainty in Artificial Intelligence*, UAI'08, pages 220–229, Arlington, Virginia, USA, July 2008. AUAI Press. ISBN 978-0-9749039-4-1.

[5] Eli Bingham, Jonathan P. Chen, Martin Jankowiak, Fritz Obermeyer, Neeraj Pradhan, Theofanis Karaletsos, Rohit Singh, Paul Szerlip, Paul Horsfall, and Noah D. Goodman. Pyro: Deep Universal Probabilistic Programming. *Journal of Machine Learning Research*, 20(28):1–6, 2019. ISSN 1533-7928. URL http://jmlr.org/papers/v20/18-403.html.

[6] Hong Ge, Kai Xu, and Zoubin Ghahramani. Turing: A Language for Flexible Probabilistic Inference. In *International Conference on Artificial Intelligence and Statistics*, pages 1682–1690. PMLR, March 2018. URL http://proceedings.mlr.press/v84/ge18b.html.

[7] Marco F. Cusumano-Towner, Feras A. Saad, Alexander K. Lew, and Vikash K. Mansinghka. Gen: A general-purpose probabilistic programming system with programmable inference. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2019, pages 221–236, New York, NY, USA, June 2019. Association for Computing Machinery. ISBN 978-1-4503-6712-7. doi: 10.1145/3314221.3314642. URL https://doi.org/10.1145/3314221.3314642.

[8] Vikash Mansinghka, Daniel Selsam, and Yura Perov. Venture: A higher-order probabilistic programming platform with programmable inference. *arXiv:1404.0099 [cs, stat]*, March 2014. URL http://arxiv.org/abs/1404.0099.

[9] Praveen Narayanan, Jacques Carette, Wren Romano, Chung-chieh Shan, and Robert Zinkov. Probabilistic Inference by Program Transformation in Hakaru (System Description). In Oleg Kiselyov and Andy King, editors, *Functional and Logic Programming*, volume 9613, pages 62–79. Springer International Publishing, Cham, 2016. ISBN 978-3-319-29603-6 978-3-319-29604-3. doi: 10.1007/978-3-319-29604-3_5. URL http://link.springer.com/10.1007/978-3-319-29604-3_5.

[10] Noah D Goodman and Andreas Stuhlmüller. WebPPL - probabilistic programming for the web, 2014. URL http://webppl.org/.

[11] Lawrence M. Murray and Thomas B. Schön. Automated learning with a probabilistic programming language: Birch. *Annual Reviews in Control*, 46:29–43, 2018. ISSN 1367-5788. doi: https://doi.org/10.1016/j.arcontrol.2018.10.013. URL https://www.sciencedirect.com/science/article/pii/S1367578818301202.

[12] Christopher Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, Cambridge, MA, USA, May 1999. ISBN 978-0-262-13360-9.

[13] Sylvia. Richardson and Peter J. Green. 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. ISSN 1467-9868. doi: 10.1111/1467-9868.00095. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-9868.00095.

[14] Fredrik Ronquist, Jan Kudlicka, Viktor Senderov, Johannes Borgström, Nicolas Lartillot, Daniel Lundén, Lawrence Murray, Thomas B. Schön, and David Broman. Universal probabilistic programming offers a powerful approach to statistical phylogenetics. *bioRxiv*, page 2020.06.16.154443, December 2020. doi: 10.1101/2020.06.16.154443. URL https://www.biorxiv.org/content/10.1101/2020.06.16.154443v4.

[15] Atilim Güneş Baydin, Lei Shao, Wahid Bhimji, Lukas Heinrich, Lawrence Meadows, Jialin Liu, Andreas Munk, Saeid Naderiparizi, Bradley Gram-Hansen, Gilles Louppe, Mingfei Ma, Xiaohui Zhao, Philip Torr, Victor Lee, Kyle Cranmer, Prabhat, and Frank Wood. Etalumis: Bringing probabilistic programming to scientific simulators at scale. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, pages 1–24, New York, NY, USA, November 2019. Association for Computing Machinery. ISBN 978-1-4503-6229-0. doi: 10.1145/3295500.3356180. URL https://doi.org/10.1145/3295500.3356180.

[16] Tuan Anh Le, Atilim Gunes Baydin, and Frank Wood. Inference Compilation and Universal Probabilistic Programming. In *Artificial Intelligence and Statistics*, pages 1338–1348. PMLR, April 2017. URL http://proceedings.mlr.press/v54/le17a.html.

[17] Bradley Gram-Hansen, Christian Schröder de Witt, Tom Rainforth, Philip H. S. Torr, Yee Whye Teh, and Atılım Güneş Baydin. Hijacking Malaria Simulators with Probabilistic Programming. *arXiv:1905.12432 [cs, stat]*, May 2019. URL http://arxiv.org/abs/1905.12432.

[18] David M. Blei, Alp Kucukelbir, and Jon D. McAuliffe. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, July 2017. ISSN 0162-1459. URL https://www.tandfonline.com/doi/full/10.1080/01621459.2017.1285773.

[19] Alp Kucukelbir, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic Variational Inference in Stan. *arXiv:1506.03431 [stat]*, June 2015. URL http://arxiv.org/abs/1506.03431.

[20] Abhinav Agrawal, Daniel R Sheldon, and Justin Domke. Advances in black-box VI: Normalizing flows, importance weighting, and optimization. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 17358–17369. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/c91e3483cf4f90057d02aa492d2b25b1-Paper.pdf.

[21] Cheng Zhang, Judith Bütepage, Hedvig Kjellström, and Stephan Mandt. Advances in variational inference. *IEEE transactions on pattern analysis and machine intelligence*, 41(8):2008–2026, 2018.

[22] Matthew D Hoffman, David M. Blei, Chong Wang, and John Paisley. Stochastic Variational Inference. *Journal of Machine Learning Research*, 14(5), May 2013.

[23] Danilo Rezende and Shakir Mohamed. Variational Inference with Normalizing Flows. In *Proceedings of the 32nd International Conference on Machine Learning*, pages 1530–1538. PMLR, June 2015. URL https://proceedings.mlr.press/v37/rezende15.html.

[24] Diederik P. Kingma and Max Welling. Auto-Encoding Variational Bayes. *arXiv:1312.6114 [cs, stat]*, May 2014. URL http://arxiv.org/abs/1312.6114.

[25] Steve Brooks, Andrew Gelman, Galin Jones, and Xiao-Li Meng. *Handbook of Markov Chain Monte Carlo*. CRC Press, May 2011. ISBN 978-1-4200-7942-5.

[26] N. Siddharth, Brooks Paige, Jan-Willem van de Meent, Alban Desmaison, Noah D. Goodman, Pushmeet Kohli, Frank Wood, and Philip Torr. Learning disentangled representations with semi-supervised deep generative models. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5927–5937. Curran Associates, Inc., 2017.

[27] Atilim Gunes Baydin, Barak A Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Marchine Learning Research*, 18:1–43, 2018.

[28] David Wingate and Theophane Weber. Automated Variational Inference in Probabilistic Programming. *arXiv:1301.1299 [cs, stat]*, January 2013. URL http://arxiv.org/abs/1301.1299.

[29] Timothy Brooks Paige. *Automatic Inference for Higher-Order Probabilistic Programs*. http://purl.org/dc/dcmitype/Text, University of Oxford, 2016. URL https://ora.ox.ac.uk/objects/uuid:d912c4de-4b08-4729-aa19-766413735e2a.

[30] Jan-Willem Vandemeent, Brooks Paige, David Tolpin, and Frank Wood. Black-Box Policy Search with Probabilistic Programs. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 1195–1204. PMLR, May 2016. URL https://proceedings.mlr.press/v51/vandemeent16.html.

[31] Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An Introduction to Probabilistic Programming. *arXiv:1809.10756 [cs, stat]*, September 2018. URL http://arxiv.org/abs/1809.10756.

[32] Arun Chaganty, Aditya Nori, and Sriram Rajamani. Efficiently Sampling Probabilistic Programs via Program Analysis. In *Proceedings of the Sixteenth International Conference on Artificial Intelligence and Statistics*, pages 153–160. PMLR, April 2013. URL https://proceedings.mlr.press/v31/chaganty13a.html.

[33] Sriram Sankaranarayanan, Aleksandar Chakarov, and Sumit Gulwani. Static analysis for probabilistic programs: Inferring whole program properties from finitely many paths. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, pages 447–458, New York, NY, USA, June 2013. Association for Computing Machinery. ISBN 978-1-4503-2014-6. doi: 10.1145/2491956.2462179. URL https://doi.org/10.1145/2491956.2462179.

[34] Yuan Zhou, Hongseok Yang, Yee Whye Teh, and Tom Rainforth. Divide, Conquer, and Combine: A New Inference Strategy for Probabilistic Programs with Stochastic Support. In *Proceedings of the 37th International Conference on Machine Learning*, pages 11534–11545. PMLR, November 2020. URL https://proceedings.mlr.press/v119/zhou20e.html.

[35] Andrew D. Gordon, Thomas A. Henzinger, Aditya V. Nori, and Sriram K. Rajamani. Probabilistic programming. In *Future of Software Engineering Proceedings*, FOSE 2014, pages 167–181, New York, NY, USA, May 2014. Association for Computing Machinery. ISBN 978-1-4503-2865-4. doi: 10.1145/2593882.2593900. URL https://doi.org/10.1145/2593882.2593900.

[36] Thomas William Gamlen Rainforth. *Automating Inference, Learning, and Design Using Probabilistic Programming*. http://purl.org/dc/dcmitype/Text, University of Oxford, 2017. URL https://ora.ox.ac.uk/objects/uuid:e276f3b4-ff1d-44bf-9d67-013f68ce81f0.

[37] Martin J Wainwright, Michael I Jordan, et al. Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305, 2008.

[38] Shakir Mohamed, Mihaela Rosca, Michael Figurnov, and Andriy Mnih. Monte Carlo Gradient Estimation in Machine Learning. *Journal of Machine Learning Research*, 21(132):1–62, 2020. ISSN 1533-7928. URL http://jmlr.org/papers/v21/19-346.html.

[39] Jack P. C. Kleijnen and Reuven Y. Rubinstein. Optimization and sensitivity analysis of computer simulation models by the score function method. *European Journal of Operational Research*, 88(3):413–427, February 1996. ISSN 0377-2217. doi: 10.1016/0377-2217(95)00107-7. URL https://www.sciencedirect.com/science/article/pii/0377221795001077.

[40] Ronald J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Language*, 8(3-4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696. URL https://doi.org/10.1007/BF00992696.

[41] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. *arXiv:1401.4082 [cs, stat]*, May 2014. URL http://arxiv.org/abs/1401.4082.

[42] Michalis Titsias and Miguel Lázaro-Gredilla. Doubly Stochastic Variational Bayes for non-Conjugate Inference. In *Proceedings of the 31st International Conference on Machine Learning*, pages 1971–1979. PMLR, June 2014.

[43] Stefan Webb, Adam Golinski, Robert Zinkov, N. Siddharth, Tom Rainforth, Yee Whye Teh, and Frank Wood. Faithful Inversion of Generative Models for Effective Amortized Inference. *arXiv:1712.00287 [cs, stat]*, November 2018. URL http://arxiv.org/abs/1712.00287.

[44] Aditya V Nori, Chung-Kil Hur, Sriram K Rajamani, and Selva Samuel. R2: An Efficient MCMC Sampler for Probabilistic Programs. *AAAI Conference on Artificial Intelligence (AAAI)*, page 7, 2015.

[45] Raven Beutner, Luke Ong, and Fabian Zaiser. Guaranteed bounds for posterior inference in universal probabilistic programming. *PLDI 2022: International Conference on Programming Language Design and Implementation*, 2022.

[46] Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020. doi: 10.1017/9781108571401.

[47] Zohar Karnin, Tomer Koren, and Oren Somekh. Almost optimal exploration in multi-armed bandits. In *International Conference on Machine Learning*, pages 1238–1246. PMLR, 2013.

[48] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. Hyperband: A novel bandit-based approach to hyperparameter optimization. *Journal of Machine Learning Research*, 18(185):1–52, 2018. URL http://jmlr.org/papers/v18/16-558.html.

[49] Stefan Falkner, Aaron Klein, and Frank Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1437–1446. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/falkner18a.html.

[50] Christian Naesseth, Fredrik Lindsten, and David Blei. Markovian score climbing: Variational inference with kl(p || q). In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 15499–15510. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/b20706935de35bbe643733f856d9e5d6-Paper.pdf.

[51] Rajesh Ranganath, Sean Gerrish, and David Blei. Black Box Variational Inference. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics*, pages 814–822. PMLR, April 2014. URL https://proceedings.mlr.press/v33/ranganath14.html.

[52] Alp Kucukelbir, Dustin Tran, Rajesh Ranganath, Andrew Gelman, and David M. Blei. Automatic differentiation variational inference. *J. Mach. Learn. Res.*, 18(1), Jan 2017.

[53] Luca Ambrogioni, Kate Lin, Emily Fertig, Sharad Vikram, Max Hinne, Dave Moore, and Marcel van Gerven. Automatic structured variational inference. In *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pages 676–684. PMLR, March 2021. URL https://proceedings.mlr.press/v130/ambrogioni21a.html.

[54] Akash Kumar Dhaka, Alejandro Catalina, Manushi Welandawe, Michael Riis Andersen, Jonathan Huggins, and Aki Vehtari. Challenges and Opportunities in High-dimensional Variational Inference. *arXiv:2103.01085 null*, March 2021. URL http://arxiv.org/abs/2103.01085.

[55] Wonyeol Lee, Hangyeol Yu, and Hongseok Yang. Reparameterization Gradient for Non-differentiable Models. In *Advances in Neural Information Processing Systems*, 2018. URL https://papers.nips.cc/paper/2018/hash/b096577e264d1ebd6b41041f392eec23-Abstract.html.

[56] Brooks Paige and Frank Wood. Inference networks for sequential monte carlo in graphical models. In *International Conference on Machine Learning*, pages 3040–3049. PMLR, 2016.

[57] Andreas Stuhlmüller, Jacob Taylor, and Noah Goodman. Learning stochastic inverses. *Advances in neural information processing systems*, 26, 2013.

[58] S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E. Hinton. Attend, Infer, Repeat: Fast Scene Understanding with Generative Models. *arXiv:1603.08575 [cs]*, March 2016. URL http://arxiv.org/abs/1603.08575.

[59] Brooks Paige and Frank Wood. A Compilation Target for Probabilistic Programming Languages. *arXiv:1403.0504 [cs, stat]*, July 2014. URL http://arxiv.org/abs/1403.0504.

[60] Frank Wood, Jan Willem Meent, and Vikash Mansinghka. A New Approach to Probabilistic Programming Inference. In *Artificial Intelligence and Statistics*, pages 1024–1032. PMLR, April 2014. URL http://proceedings.mlr.press/v33/wood14.html.

[61] Tom Rainforth, Christian Naesseth, Fredrik Lindsten, Brooks Paige, Jan-Willem Vandemeent, Arnaud Doucet, and Frank Wood. Interacting particle markov chain monte carlo. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2616–2625, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/rainforth16.html.

[62] David Wingate, Andreas Stuhlmueller, and Noah Goodman. Lightweight Implementations of Probabilistic Programming Languages Via Transformational Compilation. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 770–778. JMLR Workshop and Conference Proceedings, June 2011. URL https://proceedings.mlr.press/v15/wingate11a.html.

[63] Tuan Anh Le. *Inference for Higher Order Probabilistic Programs*. Master's Thesis, University of Oxford, 2016.

[64] Carol Mak, Fabian Zaiser, and Luke Ong. Nonparametric Hamiltonian Monte Carlo. In *Proceedings of the 38th International Conference on Machine Learning*, pages 7336–7347. PMLR, July 2021. URL https://proceedings.mlr.press/v139/mak21a.html.

[65] William Harvey, Andreas Munk, Atılım Güneş Baydin, Alexander Bergholm, and Frank Wood. Attention for inference compilation. *arXiv preprint arXiv:1910.11961*, 2019.

[66] L. Martino, V. Elvira, D. Luengo, and J. Corander. Layered adaptive importance sampling. *Statistics and Computing*, 27(3), May 2017.

[67] David Duvenaud, James Lloyd, Roger Grosse, Joshua Tenenbaum, and Ghahramani Zoubin. Structure Discovery in Nonparametric Regression through Compositional Kernel Search. In *International Conference on Machine Learning*, pages 1166–1174. PMLR, May 2013. URL http://proceedings.mlr.press/v28/duvenaud13.html.

[68] David Janz, Brooks Paige, Tom Rainforth, Jan-Willem van de Meent, and Frank Wood. Probabilistic structure discovery in time series data. *arXiv preprint arXiv:1611.06863*, 2016.

[69] G.E.P. Box, G.M. Jenkins, G.C. Reinsel, and G.M. Ljung. *Time Series Analysis: Forecasting and Control*. Wiley Series in Probability and Statistics. Wiley, 2015. ISBN 9781118674925. URL https://books.google.co.uk/books?id=rNt5CgAAQBAJ.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes] Limitations are discussed in relevant sections throughout the paper.
   (c) Did you discuss any potential negative societal impacts of your work? [N/A] The paper provides a generic variational inference algorithm for probabilistic models with stochastic support. As our contributions are methodological and our experiments do not contain any personalized data, we believe that this paper does not introduce any fundamentally new risks.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes] See Proposition 1.
   (b) Did you include complete proofs of all theoretical results? [Yes] See Appendix A.
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] See supplementary material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See Section 6 and Appendix D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Appendix D.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [N/A]
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code in supplementary material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix for Rethinking Variational Inference for Probabilistic Programs with Stochastic Support

Anonymous Authors

A KL Divergence Derivation

A.1 Breaking Down the Global ELBO

The global ELBO is given by

$$\mathcal{L}(\phi, \lambda) = \mathbb{E}_{q(x;\phi,\lambda)} \left[\log \frac{\gamma(x)}{q(x;\phi,\lambda)} \right], \tag{16}$$

$$= \int_{\mathcal{X}} q(x;\phi,\lambda) \log \frac{\gamma(x)}{q(x;\phi,\lambda)} dx, \tag{17}$$

using the fact that the subsets $\mathcal{X}_k$ provide a partition of $\mathcal{X}$ we can write the integral as

$$= \sum_{k=1}^{K} \int_{\mathcal{X}_k} q(x;\phi,\lambda) \log \frac{\gamma(x)}{q(x;\phi,\lambda)} dx, \tag{18}$$

using the factorization of $q(x;\phi,\lambda)$ and the fact that for $x \in \mathcal{X}_k$ the program density satisfies $\gamma(x) = \gamma_k(x)$ we get

$$= \sum_{k=1}^{K} \int_{\mathcal{X}_k} q_k(x;\phi_k) q(k;\lambda) \log \frac{\gamma_k(x)}{q_k(x;\phi_k) q(k;\lambda)} dx \tag{19}$$

then using the fact that $q(k;\lambda)$ does not depend on x we have

$$= \sum_{k=1}^{K} q(k;\lambda) \int_{\mathcal{X}_k} q_k(x;\phi_k) \log \frac{\gamma_k(x)}{q_k(x;\phi_k)} dx - \log q(k;\lambda), \tag{20}$$

which we can write concisely as

$$= \mathbb{E}_{q(k;\lambda)} \left[\mathcal{L}_k(\phi_k) - \log q(k;\lambda) \right], \tag{21}$$

where

$$\mathcal{L}_k(\phi_k) := \mathbb{E}_{q_k(x;\phi_k)} \left[\log \frac{\gamma_k(x)}{q_k(x;\phi_k)} \right].$$

A.2 Optimal Setting of $q(k;\lambda)$

Proposition 1. *Let $L = \{\mathcal{L}_1, \ldots, \mathcal{L}_K\}$ be the set of local ELBOs, defined as per (7), where L is countable but potentially not finite. If $0 < \sum_{k=1}^{K} \exp(\mathcal{L}_k) < \infty$, then the optimal corresponding $q(k;\lambda)$ in terms of the global ELBO (6) is given by*

$$q(k;\lambda) = \exp(\mathcal{L}_k) \,/\, \sum\nolimits_{\ell=1}^{K} \exp(\mathcal{L}_\ell). \tag{8}$$

Proof. By the assumption that $0 < \sum_{k=1}^{K} \exp(\mathcal{L}_k) < \infty$, we have that $\exp(\mathcal{L}_k) / \sum_{k=1}^{K} \exp(\mathcal{L}_k)$ forms a valid probability mass function over $k \in \{1, \ldots, K\}$. We can therefore rewrite (7) as

$$\mathcal{L}(\phi, \lambda) = \mathbb{E}_{q(k;\lambda)} \left[\log \frac{\exp(\mathcal{L}_k)}{\sum_{k=1}^{K} \exp(\mathcal{L}_k)} - \log q(k;\lambda) \right] + \log \sum_{k=1}^{K} \exp(\mathcal{L}_k) \tag{22}$$

$$= -\mathrm{KL} \left(q(k;\lambda) \parallel \frac{\exp(\mathcal{L}_k)}{\sum_{k=1}^{K} \exp(\mathcal{L}_k)} \right) + \log \sum_{k=1}^{K} \exp(\mathcal{L}_k) \tag{23}$$

Now as second term in the above is constant in $q(k;\lambda)$ and a KL divergence is minimized when the two distributions are the same, we can immediately conclude the desired result that the optimal $q(k;\lambda)$ is

$$q(k;\lambda) = \frac{\exp(\mathcal{L}_k)}{\sum_{\ell=1}^{K}\exp(\mathcal{L}_\ell)}. \tag{24}$$

□

Additionally, from (23) it follows that for the optimal setting of the mixture distribution $q(k;\lambda)$ the global ELBO is given by

$$\mathcal{L}(\phi,\lambda^*) = \log\sum_{k=1}^{K}\exp(\mathcal{L}_k).$$

B Details on Resource Allocation

B.1 Background on Successive Halving

Successive Halving (SH) divides a total budget of T iterations into $L = \lceil\log_2(K)\rceil + 1$ phases and starts by optimizing each of K candidates, in our case the SLPs, for $\lfloor T/(KL)\rfloor$ iterations. It then ranks each of the candidates in terms of their performance, in our case the values of $\exp(\mathcal{L}_k)$, before eliminating the bottom half. This process then repeats, with each of the remaining candidates run for $2^{\ell-1}T/(KL)$ iterations at the ℓ-th phase. This results in an exponential distribution of resources allocated to the different candidates, with more resources allocated to those that are more promising after intermediate evaluation.

Adapting it to our setting of treating the problem as a top-m identification is done by simply using $L = \lceil\log_2(K) - \log_2(m)\rceil + 1$ phases instead of $L = \lceil\log_2(K)\rceil + 1$.

B.2 Online Resource Allocation

Here, we present an online version of Algo. 1, where the term 'online' refers to the fact that the algorithm considers more and more SLPs as the computational budget increases. The online variant of the algorithm is useful if a user is unsure about the total iteration budget that they want to spend on the input program. This user might want to run SDVI with an initial iteration budget T_1 and after having observed the results, they might decide that they want to keep further optimizing the guide parameters. We therefore need to adapt Algo. 1 so that it can be 'restarted' after it has terminated. A naive approach to this would be to simply run Algo. 1 again but re-use the q_k's for the SLPs that have already been discovered and only initialize the q_k from scratch for SLPs which have not been seen before. However, this scheme is limited as it disproportionately favours SLPs which were discovered in the previous run. This is because for those SLPs the local ELBOs will already be relatively large compared to the newly added SLPs. As a consequence, SH will not assign significant computational budget to the SLPs that were added after the algorithm was restarted.

To safeguard against this behaviour we instead propose an online version of SDVI in Algo. 2 which is using a modified 'reward' for SH. Instead of ranking the different SLPs according to $\mathcal{L}_k(\phi_k(t_k))$ we instead propose the objective $\exp(\alpha\mathcal{L}_k(\phi_k(t_k)))/t_k$ where $0 < \alpha \leqslant 1$. The reward is scaled by the reciprocal of t_k because we are no longer aiming to select the SLPs with the highest $\mathcal{L}_k(\phi_k(t_k))$ but instead aim to choose the SLPs which have been 'underselected' compared to other SLPs, assuming we should have selected them in proportion to $\exp(\alpha\mathcal{L}_k(\phi_k(t_k)))$. The scaling by the scalar α is a further mechanism to encourage more exploration, with setting $\alpha = 0$ equivalent to uniform sampling in the limit of repeated SH runs. Since this adapted objective takes into account the computational budget that was spent on each SLP, it is a more suitable objective when running SH repeatedly.

Algorithm 2 Online SDVI

Require: Target program γ, iteration budget per SH run T, minimum no. of SH candidates m, parameter controlling $\alpha > 0$ exploration

Extract SLPs $\{\gamma_k\}_{k=1}^{K}$ from γ and set $\mathcal{C} = \{1, \ldots, K\}$
Formulate guide q_k for each SLP and initialize parameters ϕ_k
$t_k = 0$ for all $k \in \mathcal{C}$
while Stopping criteria not satisfied **do**
    $\mathcal{C}' \leftarrow \mathcal{C}$
    Phases in successive halving $L = \lceil \log_2(|\mathcal{C}|) - \log_2(m) \rceil + 1$
    for $l = 1, \ldots, L$ **do**
        Number of iterations $n_l = \lfloor \frac{T}{L|\mathcal{C}'|} \rfloor$
        for $k \in \mathcal{C}'$ **do**
            Perform n_l optimization iterations of ϕ_k targeting $\mathcal{L}_{\text{surr},k}(\phi_k)$
            Estimate $\mathcal{L}_{\text{surr},k}(\phi_k)$ using Monte Carlo estimate of Eq. (11)
            $t_k = t_k + n_l$
        end for
        Remove $\min(\lfloor |\mathcal{C}'|/2 \rfloor, |\mathcal{C}'| - m)$ SLPs from $\mathcal{C}'$ with the lowest $\exp(\alpha\, \mathcal{L}_{\text{surr},k}(\phi_k))/t_k$
    end for
    Extract new SLPs from γ and add them to $\mathcal{C}$, set $t_{k'} = 0$ for each new SLP with index k'
end while
Truncate q_k outside of SLP support, $\mathcal{X}_k$, using Eq. (13)
Estimate each $\mathcal{L}_k(\phi_k)$ using Monte Carlo estimate of Eq. (7)
Calculate $q(k; \lambda)$ according to Eq. (8) and return $q(x; \phi, \lambda)$ as per Eq. (4)

C Details for Training Local Guides

C.1 Density Estimation of the Prior

Before we can define the KL divergence we first have to carefully define global and local prior distributions We first define what we informally call the global 'prior' distribution of the program as the product of all the terms added to the program density by the `sample` statements

$$\pi_{\text{prior}}(x_{1:n_x}) := \prod_{i=1}^{n_x} f_{a_i}(x_i|\eta_i). \tag{25}$$

However, here we are using the term prior only informally, since (25) is not a prior in the conventional Bayesian sense since the η_i can be functions of the observed data y. Note that here n_x in (25) is again a random variable since the raw random draws $x_{1:n_x}$ of the program do not necessarily have fixed length. Then similarly we define local 'prior' distributions

$$\pi_{\text{prior},k}(x_{1:n_k}) := \frac{\mathbb{I}[x_{1:n_k} \in \mathcal{X}_k] \prod_{i=1}^{n_k} f_{A_k[i]}(x_i|\eta_i)}{Z_{\text{prior},k}} = \frac{\mathbb{I}[x_{1:n_k} \in \mathcal{X}_k] \pi_{\text{prior}}(x_{1:n_k})}{Z_{\text{prior},k}},, \tag{26}$$

where

$$Z_{\text{prior},k} := \int_{\mathcal{X}} \mathbb{I}[x \in \mathcal{X}_k]\, \pi_{\text{prior}}(x) dx. \tag{27}$$

Note that for our purposes we will never actually have to estimate $Z_{prior,k}$, we only defined it to ensure that $\pi_{prior,k}$ is a normalized density. This allows us to define the forward KL divergence which we would like to optimize with respect to ϕ_k

$$\text{KL}(\pi_{\text{prior},k}(x) \parallel \tilde{q}_k(x; \phi_k)) = \mathbb{E}_{\pi_{\text{prior},k}(x)} \left[\log \frac{\pi_{\text{prior},k}(x)}{\tilde{q}_k(x; \phi_k)} \right] \tag{28}$$

which we can rewrite as

$$= \mathbb{E}_{\pi_{\text{prior},k}(x)} \left[\log \pi_{\text{prior},k}(x) \right] - \mathbb{E}_{\pi_{\text{prior},k}(x)} \left[\log \tilde{q}_k(x; \phi_k) \right]. \tag{29}$$

The first term is a constant with respect to ϕ_k and therefore does not affect the optimization

$$\propto \mathbb{E}_{\pi_{\text{prior},k}(x)} \left[-\log \tilde{q}_k(x; \phi_k) \right], \tag{30}$$

then by the definition of $\pi_{\text{prior},k}(x)$ in Eq. (26) this is equivalent to

$$= -\frac{1}{Z_{\text{prior},k}} \mathbb{E}_{\pi_{\text{prior}}(x)} \left[\mathbb{I}[x \in \mathcal{X}_k] \log \tilde{q}_k(x; \phi_k) \right]. \tag{31}$$

Finally, $Z_{prior,k}$ is a constant with respect to ϕ_k and can be dropped

$$\propto \mathbb{E}_{\pi_{\text{prior}}(x)} \left[-\mathbb{I}[x \in \mathcal{X}_k] \log \tilde{q}_k(x; \phi_k) \right]. \tag{32}$$

We can estimate the gradients of the objective in Eq. (32) using a Monte Carlo estimator

$$\nabla_{\phi_k} \mathbb{E}_{\pi_{\text{prior}}(x)} \left[-\mathbb{I}[x \in \mathcal{X}_k] \log \tilde{q}_k(x; k, \phi_k) \right] \approx \frac{1}{N} \sum_{j=1}^{N} \mathbb{I}[x^{(j)} \in \mathcal{X}_k] \nabla_{\phi_k} \log \tilde{q}_k(x^{(j)}; k, \phi_k) \tag{33}$$

where $x^{(j)}$ are raw random draws generated by executing the input program forward. These gradient estimates can then be used in a stochastic gradient descent optimization procedure. In our experiments, we generate a fixed set of N samples and re-use the same set of samples for the entire optimization process. Other approaches are also possible such as periodically collecting a new set of samples and using local MCMC moves to collect samples instead of repeatedly sampling from the prior.

C.2 Exploiting Program Structure: Discrete Branching Optimization

In practice, many user-defined programs have structural properties which can be exploited to construct a valid local guide directly and deterministically (without resorting to the stochastic mechanism described in Sec. 4.5). Specifically, consider the class of programs whose program paths are determined by variables sampled from discrete distributions. For these programs, we can assume that for each SLP (kth, say) there is an (ordered) set of indices $I_{\text{branch}} \subset \{1, \ldots, n_k\} = I$ and a set of constants $r_{k,1}, \ldots, r_{k,|I_{\text{branch}}|} \in \mathbb{Z}$ such that the local unnormalized densities are expressible as

$$\gamma_k(x_{1:n_k}) = \gamma(x_{1:n_k}) \prod_{l=1}^{|I_{\text{branch}}|} \mathbb{I}\left[x_{I_{\text{branch}}[l]} = r_{k,l}\right]$$

where $I_{\text{branch}}[j]$ means the jth element in I_{branch}. It follows that we can construct densities for the kth SLP on a subset of variables in $x_{1:n_k}$ by eliminating all the variables given by indices I_{branch} (by instantiating them to constants). This is effectively equivalent to replacing the `sample` statements corresponding to the variables which influence the control flow with `observe` statements which induces a new program density that has the form

$$\tilde{\gamma}_k(x_{1:n'_k}) = \prod_{i=1}^{n'_k} f_{A_k[I'[i]]}(x_i | \eta_i) \prod_{l=1}^{|I_{\text{branch}}|} f_{A_k[I_{\text{branch}}[l]]}(r_{k,l} \mid \eta_l) \prod_{j=1}^{n_y} g_{b_j}(y_j \mid \phi_j) \tag{34}$$

where $I' := [1, \ldots, n_k] \backslash I_{\text{branch}}$, and $n'_k := |I'|$. Furthermore, if all the remaining r.v. are continuous distributions with support in $\mathbb{R}$ (i.e. $\text{supp}(f_{A_k[i]}) = \mathbb{R}$ for $i \in I'$) then $\tilde{\gamma}_k(x_{1:n'_k})$ itself has support in $\mathbb{R}^{n'_k}$. It is then straightforward to construct a guide q_k with support in $\mathbb{R}^{n'_k}$ using existing methods, and we can get gradient estimates using the reparameterization gradient estimator (assuming there are no more discontinuities in $\tilde{\gamma}_k$).

To realize the discrete branching optimization in our Pyro implementation we allow users to annotate the `sample` statements which influence the branching. While it is in principle possible to automatically identify programs with discrete branching using program analysis, formalizing and implementing such a program analysis tool to work with arbitrary Pyro program would be a significant contribution in itself which is out of scope for this paper as we are focused on the statistical evaluation of SDVI. Specifically, the relevant `sample` statements within a Pyro program can be annotated as follows: `pyro.sample("x", dist.Poisson(7), infer={"branching": True})`. Our implementation of SDVI is then able to use these annotations to create the density $\tilde{\gamma}_k$ in (34).

D Additional Details for Experiments

For all experiments that rely on optimization we use the Adam optimizer [70]. The experiments were executed on an internal cluster which uses a range of different computer architectures.

D.1 Model From Figure 1

Listing 1: Pyro Code for Figure 1.

```
import pyro
import pyro.distributions as dist

def model():
    x = pyro.sample("x", dist.Normal(0, 1))
    if x < 0:
        z1 = pyro.sample("z1", dist.Normal(-3, 1))
    else:
        z1 = pyro.sample("z2", dist.Normal(3, 1))

    x = pyro.sample("x", dist.Normal(z1, 2), obs=torch.tensor(2.0))

guide = pyro.infer.autoguide.AutoNormalMessenger(model)
optim = pyro.optim.Adam({"lr": 0.01})
svi = pyro.infer.SVI(
    model, guide, optim, loss=pyro.infer.Trace_ELBO()
)

for j in range(2000):
    svi.step()
```

The full Pyro code for the model in Fig. 1, including automatically generating and training the guide is given in Listing 1. The code for BBVI and SDVI is provided in the code supplementary. For Pyro's AutoGuide and BBVI we run the optimization for 2000 iterations with a learning rate of 0.01. Similarly, for SDVI we have a total iteration budget of $T = 2000$ and use a learning rate of 0.01; we set the minimum number of SH candidates to $m = 2$

D.2 Program with Normal Distributions

For SDVI, we use 10^3 samples from the prior to discover SLPs. To train the local guides to place support within the SLP boundaries we collect 10^2 samples per SLP and optimize the objective in Equation (12) for 10^3 iterations. We run Algorithm 1 with a total budget of $T = 10^5$ with 5 particles for the ELBO and to estimate the final SLP weights we use 10^3 samples per SLP. We use a learning rate of 0.01.

For Pyro AutoGuide, we run the optimization for 10^5 steps with 1 ELBO particle. For BBVI, we run the optimization for 10^4 steps with 10 ELBO particles. For both we use a learning rate of 0.01.

D.3 Infinite Gaussian Mixture Model

For SDVI, we use 10^3 samples from the prior to discover SLPs, run Algorithm 1 with a total budget of $T = 2 * 10^4$ with 10 particles for the ELBO and to estimate the final SLP weights we use 10^2 samples per SLP. We use a learning rate of 0.1.

For BBVI, we run for $2 * 10^4$ iterations using 10 particles for the ELBO and a learning rate of 0.1. In the guide, we use a categorical distribution for number of components K over the range $K \in [1, 25]$. We ran initial experiments with instead using a Poisson distribution paramterized by the rate but we found this leads to an explosion in the number of components in the guide which resulted in the program running out of memory. For each μ_k the variational approximation is a diagonal Normal distribution parameterized by the mean and the diagonal entries in the covariance matrix.

For DCC, we run for 200 iterations, at each iteration we run 10 independent RMH chains generating 10 samples and to get a marginal likelihood estimate we use PI-MAIS [66] which places a proposal distribution (in our case a Gaussian) on the outputs of the RMH chains and samples from this proposal M times; we set $M = 10$.

D.4 Inferring Gaussian Process Kernels

D.4.1 Model Details

Our probabilistic context-free grammar for the kernel structure has the production rules

$$\mathcal{K} \rightarrow \text{ SE} \mid \text{RQ} \mid \text{PER} \mid \text{LIN} \mid \mathcal{K} \times \mathcal{K} \mid \mathcal{K} + \mathcal{K}. \tag{35}$$

with the production probabilities $[0.2, 0.2, 0.2, 0.2, 0.1, 0.1]$. On each base kernel hyperparameter we place an InverseGamma$(\alpha = 2, \beta = 1)$ prior. For each base kernel the specific hyperparameters we wish to do inference over are:[2]

- Squared Exponential (SE): Lengthscale
- Rational Quadratic (RQ): Lengthscale, Scale Mixture
- Periodic (PER): Lengthscale, Period
- Linear (LIN): Bias

Assuming we have N observations with inputs $\mathbf{x} \in \mathbb{R}^N$ and outputs $\mathbf{y} \in \mathbb{R}^N$ our model can then be written as

$$\mathcal{K} \sim \text{PCFG}(), \quad \sigma \sim \text{HalfNormal}(0, 1), \quad \mathbf{y} \sim \mathcal{N}(0, \mathcal{K}(\mathbf{x}) + \sigma^2 \mathrm{I}) \tag{36}$$

where PCFG() samples a kernel (and its hyperparameters) from the probabilistic context-free grammar and $\mathcal{K}(\mathbf{x})$ is the $N \times N$ covariance matrix computed from kernel $\mathcal{K}$.

D.4.2 Algorithm Configurations

For SDVI, we use 10^3 samples from the prior to discover SLPs, run Algorithm 1 with a total budget of $T = 10^6$ with 1 particles for the ELBO and to estimate the final SLP weights we use 10^2 samples per SLP. We use a learning rate of 0.005.

For BBVI, we run for 10^5 iterations using 10 particles for the ELBO and a learning rate of 0.005. The guide uses a log-normal distribution for the kernel hyperparameters and the observation noise, and for the discrete variables which influence the kernel structure we use categorical distributions. For DCC, we run for 10^3 iterations and otherwise use the exact same hyperparameters as in the Gaussian Mixture Model experiment.

E Additional Experimental Results

E.1 Program with Normal Distributions

For completeness we include here the results for DCC on the model from Sec. 6.1. DCC does not have the same fundamental limitations as the BBVI baselines therefore is competitive with SDVI and provides a similar squared error for the SLP weights. In fact, it is quite impressive that SDVI is able to match the performance of DCC because DCC leverages marginal likelihood estimators which asymptotically converge to the true marginal likelihood whereas SDVI calculates the weights based on the ELBO. This is therefore a further indicator that for this model SDVI is able to provide good posterior approximations for each SLP.

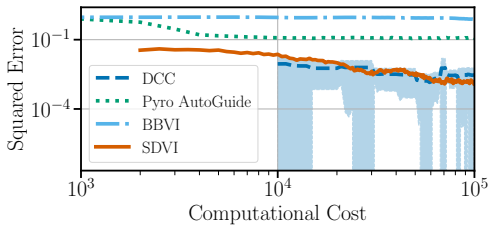


Figure 4: Squared error for the model in § 6.1 with DCC baseline. Conventions as in Fig. 2a.

[2] We use the same naming conventions as the Pyro Docs at https://docs.pyro.ai/en/stable/contrib.gp.html#module-pyro.contrib.gp.kernels.

F Difficulties of Parameter Learning for Models with Stochastic Support

In static support settings, one often uses variational bounds not only as a mechanism for inference, but also for training model parameters themselves [24, 41]. Using our notation from Sec. 2.2, this setting corresponds to having model parameters, θ, that we wish to optimize alongside the variational parameters, ϕ, such that the unnormalized density can be written as $\gamma(x;\theta)$, with corresponding normalization constant $Z(\theta)$. The ELBO then depends on both the variational and model parameters $\mathcal{L}(\phi,\theta) := \mathbb{E}_{q(x;\phi)}\left[\log {}^{\gamma(x;\theta)}/_{q(x;\phi)}\right]$. Provided $Z(\theta)$ is differentiable with respect to θ, both ϕ and θ can then, at least in principle, be simultaneously optimized using stochastic gradient ascent.

However, similar as to the case of pure inference, naively extending this scheme to models with stochastic support is non-trivial and quickly runs into both conceptual and practical problems.

Parameters, θ_k, that are inherently local to only a single SLP can be dealt with straightforwardly: as $\nabla_{\theta_k}\mathcal{L}_\ell = 0\ \forall \ell \neq k$ for such parameters, we can simply ignore parameters not associated with the SLP we are updating, that is we only take a gradient step for $\{\phi_k, \theta_k\}$ on Line 5 of Algo. 1.

Problems start to occur, though, in the more common scenario where parameters are shared between SLPs, in the sense that they influence more than one γ_k. Consider, for example, the GP model from Sec. 6.3 and assume that instead of doing inference over the observation noise, σ, we instead wish to treat this as a learnable parameter instead. Here σ could be seen as a 'global' model parameter as it appears in every SLP, so could be viewed as shared between them.

This now creates an issue in 'balancing' updates from different SLPs; the need to learn a shared θ breaks the separability between inference problems for individual SLPs. Consequently, we can no longer directly treat how often we update each SLPs as just a resource allocation problem: making more updates on a given SLP now increases the influence that SLP has on the θ which are learned. This problem is unlikely to be insurmountable—one could maintain a running estimate of $q(k;\lambda)$ during training and then use this to either directly control the resource allocation or scale the updates of θ depending on how often the corresponding SLP has been used—but it does represent a notable complication that would require its own careful consideration.

Beyond this specific practical challenge, there is also a more fundamental and general issue for parameter learning under stochastic support: should shared parameters be treated globally when we are learning them? Going back to the example of the observation noise, σ, in our GP example, it will actually be quite inappropriate here to learn a single global value for σ, as the optimal observation noise will be different depending on the kernel structure. Thus, though the variable is shared between SLPs in the program itself, it would be advantageous to learn separate values for it for each SLP, regardless of the inference approach we take.

The natural solution to this issue would be to perform parameter learning separately for each SLP, e.g. learning a separate σ_k for each SLP in the GP example above. However, this raises a variety of issues in its own, not least the fact that the inference algorithm will now start to influence the model itself: SDVI and BBVI will learn fundamentally different models. There may also be settings where it is important for a parameter to be truly global and thus shared across the SLPs, e.g. because such sharing is an explicit prior assumption we wish to make.

Further problems occur when we consider that it is also feasible for learnable parameters to influence the control flow of the program, or even the set of possible SLPs. For example, a learnable parameter could impact the maximum possible recursion depth of a recursive program. This will create challenging interactions between SLPs: updates of one will influence the desirable behavior for the variational approximation of another. In turn, this can substantially complicate the resource allocation process and even the SLP discovery process itself.

Together, these aforementioned issues demonstrate that parameter learning for models with stochastic support is a complex issue, requiring specialist consideration beyond the scope of the current paper.

G Issues with Directly Training q_k

A natural question one might ask with the SDVI method is why do we not directly train q_k to (7) by treating it as an implicit variational approximation defined by $\tilde{q}_k$? Namely, we can express (7) in

terms of $\tilde{q}_k$ as follows

$$\mathcal{L}_k(\phi_k) = \log \tilde{Z}_k(\phi_k) + \frac{1}{\tilde{Z}_k(\phi_k)} \mathbb{E}_{\tilde{q}_k(x;\phi_k)} \left[\mathbb{I}\left[x \in \mathcal{X}_k\right] \log \frac{\gamma_k(x)}{\tilde{q}_k(x;\phi_k)} \right], \tag{37}$$

which, in principle, could be directly optimized with respect to ϕ_k.

There are unfortunately two reasons that make this impractical. Firstly, though $\tilde{Z}_k(\phi_k)$ can easily be estimated using Monte Carlo, we actually cannot generate conventional unbiased estimates of $\log \tilde{Z}_k(\phi_k)$ and $1/\tilde{Z}_k(\phi_k)$ (or their gradients) because mapping the Monte Carlo estimator induces a bias. Second, this objective applies no pressure to learn a $\tilde{q}_k$ with a high acceptance rate, i.e. which actually concentrates on SLP k, such that it can easily learn a variational approximation that is very difficult to draw truncated samples from at test time.

By contrast, using our surrogate objective in (11) allows us to produce unbiased gradient estimates. Because of the mode seeking behaviour of variational inference, it also naturally forces us to learn a variational approximation with a high acceptance rate, provided we use a suitably low value of c. If desired, one can even take $c \to 0$ during training to learn an approximation which only produces samples from the target SLP without requiring any rejection. Figure 5 shows that empirically we learn a $\tilde{q}_k$ with a very high acceptance rates for the problem in Section 6.1.

Note that the surrogate and true ELBOs are exactly equal for any variational approximation that is confined to the SLP (as these have $\tilde{Z}_k(\phi_k) = 1$). This does not always necessarily mean that they have the same optima in ϕ_k for restricted variational families, even in the limit $c \to 0$. However, such differences originate from the fact that the trunctation can itself actually generalize the variational family (e.g. if $\tilde{q}_k$ is Gaussian, then q_k will be a truncated Gaussians). As such, any hypothetical gains from targeting (7) directly will always be offset against drops in the acceptance rate of the rejection sampler.

References

[70] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[66] L. Martino, V. Elvira, D. Luengo, and J. Corander. Layered adaptive importance sampling. *Statistics and Computing*, 27(3), May 2017.

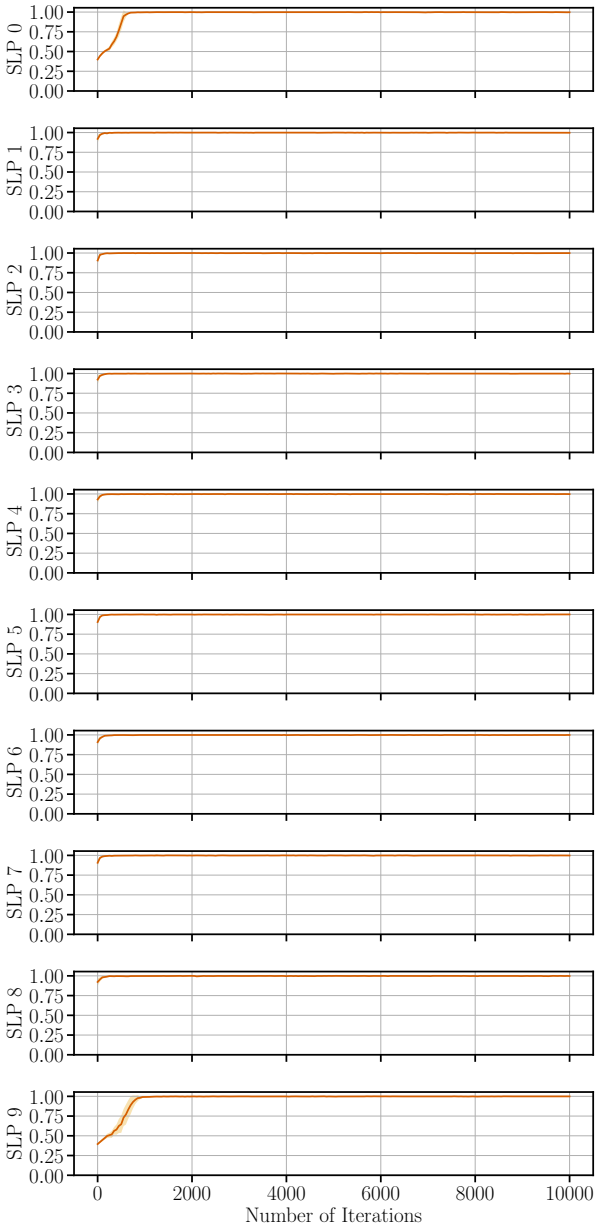


Figure 5: Acceptance rates for evaluating the local ELBOs in each SLP for the model from Sec. 6.1. Each plot represents a separate SLP; the plot with "SLP i" corresponds to the SLP with $z = \mathrm{i}$ in Eq. (15). We can see that for all SLPs the acceptance rate approaches 1 with more iterations, confirming the mode seeking behaviour that arises when maximizing the surrogate ELBO in Eq. (11).