Black-Box Policy Search with Probabilistic Programs

Jan-Willem van de Meent **Brooks Paige** **David Tolpin** **Frank Wood**
Department of Engineering Science, University of Oxford

Abstract

In this work we show how to represent policies as programs: that is, as stochastic simulators with tunable parameters. To learn the parameters of such policies we develop connections between black box variational inference and existing policy search approaches. We then explain how such learning can be implemented in a probabilistic programming system. Using our own novel implementation of such a system we demonstrate both conciseness of policy representation and automatic policy parameter learning for a set of canonical reinforcement learning problems.

1 Introduction

In planning under uncertainty the objective is to find a policy that selects actions, given currently available information, in a way that maximizes expected reward. In many cases an optimal policy can neither be represented compactly nor learned exactly. Online approaches to planning, such as Monte Carlo Tree Search [Kocsis and Szepesvári, 2006], are nonparametric policies that select actions based on simulations of future outcomes and rewards, also known as rollouts. While policies like this are often able to achieve near optimal performance, they are computationally intensive and do not have compact parameterizations. Policy search methods (see Deisenroth et al. [2011] for a review) learn parameterized policies offline, which then can be used without performing rollouts at test time, trading off improved test-time computation against having to choose a policy parameterization that may be insufficient to represent the optimal policy.

In this work we show how probabilistic programs can represent parametric policies in a both more general and compact manner. We also develop automatic inference techniques for probabilistic programming systems to do model-agnostic policy search. Our proposed approach, which we call black box policy learning (BBPL), is a variant of Bayesian policy search [Wingate et al., 2011, 2013] in which policy learning is cast as stochastic gradient ascent on the marginal likelihood.

In contrast to languages that target a single domain-specific algorithm [Andre and Russell, 2002, Srivastava et al., 2014, Nitti et al., 2015], our formulation emphasizes the use of general-purpose techniques for Bayesian inference, in which learning is used for inference amortization. To this end, we adapt black-box variational inference (BBVI), a technique for approximation of the Bayesian posterior [Ranganath et al., 2014, Wingate and Weber, 2013] to perform (marginal) likelihood maximization in arbitrary programs. The resulting technique is general enough to allow implementation in a variety of probabilistic programming systems. We show that this same technique can be used to perform policy search under an appropriate planning as inference interpretation, in which a Bayesian model is weighted by the exponent of the reward. The resulting technique, BBPL is closely related to classic policy gradient methods such as REINFORCE [Williams, 1992].

We present case studies in the Canadian traveler problem, the RockSample domain, and introduce a setting inspired by Guess Who [Coster and Coster, 1979] as a benchmark for optimal diagnosis problems.

2 Policies as Programs

Probabilistic programming systems [Milch et al., 2007, Goodman et al., 2008, Minka et al., 2014, Pfeffer, 2009, Mansinghka et al., 2014, Wood et al., 2014, Gordon et al., 2014] represent generative models as programs in a language that provides specialized syntax to instantiate random variables, as well as syntax to impose conditions on these random variables. The goal of inference in a probabilistic program is to characterize the distribution on its random variables subject to the imposed conditions, which is done using one or more generic methods provided by an inference backend.

Appearing in Proceedings of the 19^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2016, Cadiz, Spain. JMLR: W&CP volume 41.

```
(defquery ctp
 "Probabilistic program representing an agent
  solving the Canadian Traveler Problem"
 [graph src tgt base-prob make-policy]
 (let [sub-graph
        (sample-weather graph base-prob src tgt)
       [path dist counts]
        (dfs-agent sub-graph src tgt (make-policy))]
  (factor (- dist))
  (predict :path path)
  (predict :distance dist)
  (predict :counts counts)))

(defm dfs-agent
 "Run depth-first-search from start to target,
  prioritizing edges according to policy"
 [graph start target policy]
   ... )
```

```
(defm make-random-policy
 "Policy: Select edge at random"
 []
 (fn policy [u vs]
  (sample
   (categorical
    (zipmap vs (repeat (count vs) 1.))))))

(defm make-edge-policy
 "Policy: learn priorities for each edge"
 []
 (let [Q (mem (fn [u v]
               (sample [u v]
                (tag :policy
                 (gamma 1. 1.)))))]
  (fn policy [u vs]
   (argmax
    (zipmap vs (map (fn [v] (Q u v)) vs))))))
```

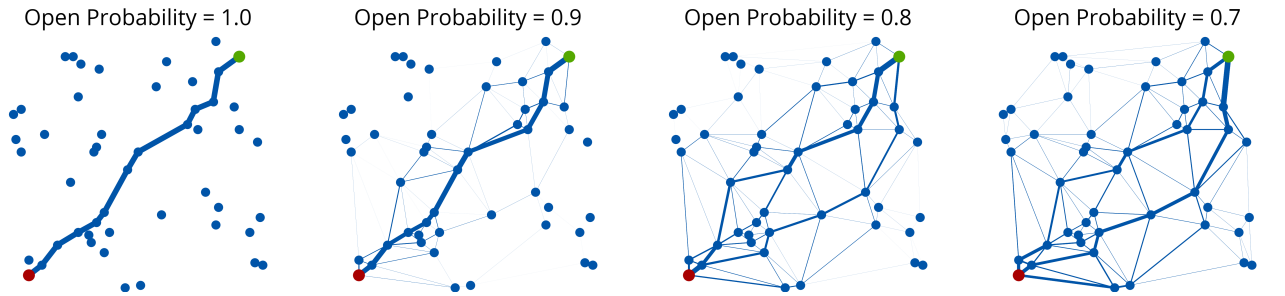

Figure 1: A Canadian traveler problem (CTP) implementation in Anglican. In the CTP, an agent must travel along a graph, which represents a network of roads, to get from the start node (green) to the target node (red). Due to bad weather some roads are blocked, but the agent does not know which in advance. Upon arrival at each node the agent observes the set of open edges. The function `dfs-agent` walks the graph by performing depth-first search, calling a function `policy` to choose the next destination based on the current and unvisited locations. The function `make-random-policy` returns a `policy` function that selects destinations uniformly at random, whereas `make-edge-policy` constructs a `policy` that selects according to sampled edge preferences `(Q u v)`. By learning a distribution on each value `(Q u v)` through gradient ascent on the marginal likelihood, we obtain a heuristic offline policy that follows the shortest path when all edges are open, and explores more alternate routes as more edges are closed.

In sequential decision problems we must define a stochastic simulator of an agent, which chooses actions based on current contextual information, and a stochastic simulator of the world, which may have some internal variables that are opaque to the agent, but provides new contextual information after each action. For sufficiently simple problems, both the agent and the world simulator can be adequately described as graphical models. Here we are interested in using probabilistic programs as simulators of both the world and the agent. The trade-off made in this approach is that we can incorporate more detailed assumptions about the structure of the problem into our simulator of the agent, which decreases the size of the search space, at the expense of having to treat these simulators as black boxes from the perspective of the learning algorithm.

In Figure 1 we show an example of a program, written in the language Anglican [Wood et al., 2014], which simulates an agent in the Canadian traveler problem (CTP) domain. This agent traverses a graph using depth first search (DFS) as a base strategy, choosing edges either at random, or according to sampled preferences. Probabilistic programs can describe a family of algorithmic policies, which may make use of programming constructs such as recursion, and higher-order functions and arbitrary deterministic operations. This allows us to define structured policies that enforce basic constraints, such as the rule that you should never travel the same edge twice.

Given a base policy program, we can define different parametrizations that encode additional structure, such as the typical travel distance starting from each edge. We can then formulate a Bayesian approach to policy learning, in which we place a prior on the policy

parameters and optimize its hyperparameters to maximize the reward. To do so we employ a planning as inference interpretation [Toussaint et al., 2006, Rawlik et al., 2012, Neumann, 2011, Hoffman et al., 2009a,b, Levine and Koltun, 2013] that casts policy search as stochastic gradient ascent on the marginal likelihood.

A challenge in devising methods for approximate inference in probabilistic programs is that such methods must deal gracefully with programs that may not instantiate the same set of random variables in each execution. For example, the random policy in Figure 1 will generate a different set of categorical variables in each execution, depending on the path followed through the graph. Similarly, the edge based policy samples values `(Q u v)` lazily, depending on the visited nodes.

In this paper we develop an approach to policy learning based on black box variational inference (BBVI) [Ranganath et al., 2014, Wingate and Weber, 2013], a technique for variational approximation of the posterior in Bayesian models. We begin by reviewing planning as inference formulations of policy search. We then show how BBVI can be adapted to perform hyperparameter optimization. In a planning as inference interpretation this method, which we call black box policy learning (BBPL), is equivalent to classic policy gradient methods. We then describe how BBPL may be implemented in the context of probabilistic programs with varying numbers of random variables, and provide a language-agnostic definition of the interface between the program and the inference back end.

3 Policy Search as Bayesian Inference

In sequential decision problems, an agent draws an action u_t from a policy distribution $\pi(u_t \,|\, x_t)$, which may be deterministic, conditioned on a context x_t. The agent then observes a new context x_{t+1} drawn from a distribution $p(x_{t+1} \,|\, u_t, x_t)$. In the finite horizon case, where an agent performs a fixed number of actions T, resulting in a sequence $\tau = (x_0, u_0, x_1, u_1, x_2, \ldots, u_{T-1}, x_T)$, which is known as a trajectory, or roll-out. Each trajectory gets a reward $R(\tau)$. Policy search methods maximize the expected reward $J_\theta = \mathbb{E}_{p_\theta}[R(\tau)]$ for a family of stochastic policies π_θ with parameters θ

$$J_\theta = \int R(\tau) p_\theta(\tau)\, d\tau, \tag{1}$$

$$p_\theta(\tau) := p(x_0) \prod_{t=0}^{T-1} \pi(u_t \,|\, x_t, \theta) p(x_{t+1} \,|\, u_t, x_t). \tag{2}$$

We are interested in performing upper-level policy search, a variant of the problem defined in terms of the hyperparameters λ of a distribution $p_\lambda(\tau, \theta)$ that places a prior $p_\lambda(\theta)$ on the policy parameters

$$J_\lambda = \int R(\tau) p_\lambda(\tau, \theta)\, d\tau\, d\theta, \tag{3}$$

$$p_\lambda(\tau, \theta) := p_\lambda(\theta) p(\tau \,|\, \theta). \tag{4}$$

Upper-level policy search can be interpreted as maximization of the normalizing constant Z_λ of an unnormalized density

$$\gamma_\lambda(\tau, \theta) = p_\lambda(\tau, \theta) \exp(\beta R(\tau)), \tag{5}$$

$$Z_\lambda = \int \gamma_\lambda(\tau, \theta)\, d\tau\, d\theta \tag{6}$$

$$= \mathbb{E}_{p_\lambda}[\exp(\beta R(\tau))]. \tag{7}$$

The constant $\beta > 0$ has the interpretation of an 'inverse temperature' that controls how strongly the density penalizes sub-optimal actions. The normalization constant Z_λ is the expected value of the exponentiated reward $\exp(\beta R(\tau))$, which is known as the desirability in the context of optimal control [Kappen, 2005, Todorov, 2009]. It is not a priori obvious that maximization of the expected reward J_λ yields the same policy hyperparameters as maximization of the desireability Z_λ, but it turns out that the two are in fact equivalent, as we will explain in section 5.

In planning as inference formulations, $\gamma_\lambda(\tau, \theta)/Z_\lambda$ is often interpreted as a posterior $p_\lambda(\tau, \theta \,|\, r)$ conditioned on a pseudo observable $r = 1$ that is Bernoulli distributed with probability $p(r = 1 \,|\, \tau) \propto \exp(\beta R(\tau))$, resulting in a joint distribution that is proportional to $\gamma_\lambda(\tau, \theta)$,

$$p(r = 1, \tau, \theta) \propto p_\lambda(\tau, \theta) \exp(\beta R(\tau)) = \gamma_\lambda(\tau, \theta). \tag{8}$$

Maximization of Z_λ is then equivalent to the maximization of the marginal likelihood $p_\lambda(r = 1)$ with respect to the hyperparameters λ. In a Bayesian context this is known as empirical Bayes (EB) [Maritz and Lwin, 1989], or type II maximum likelihood estimation.

4 Black-box Variational Inference

Variational Bayesian methods [Wainwright and Jordan, 2008] approximate an intractable posterior with a more tractable family of distributions. For purposes of exposition we consider the case of a posterior $p(z, \theta \,|\, y)$, in which y is a set of observations, θ is a set of model parameters, and z is a set of latent variables. We write $p(z, \theta \,|\, y) = \gamma(z, \theta)/Z$ with

$$\gamma(z, \theta) = p(y \,|\, z, \theta) p(z \,|\, \theta) p(\theta), \tag{9}$$

$$Z = \int \gamma(z, \theta)\, dz\, d\theta. \tag{10}$$

Variational methods approximate the posterior using a parametric family of distributions q_λ by maximizing a

lower bound on $\log Z$ with respect to λ

$$\mathcal{L}_\lambda = \mathbb{E}_{q_\lambda}[\log \gamma(z,\theta) - \log q_\lambda(z,\theta)] \quad (11)$$
$$= \log Z - D_{\mathrm{KL}}(q_\lambda(z) \,||\, \gamma(z)/Z) \leq \log Z. \quad (12)$$

This objective may be optimized with stochastic gradient ascent [Hoffman et al., 2013]

$$\lambda_{k+1} = \lambda_k + \rho_k \nabla_\lambda \mathcal{L}_\lambda\big|_{\lambda=\lambda_k}, \quad (13)$$

$$\nabla_\lambda \mathcal{L}_\lambda = \mathbb{E}_{q_\lambda(z)}\left[\nabla_\lambda \log q_\lambda(z) \log \frac{\gamma(z,\theta)}{q_\lambda(z,\theta))}\right]. \quad (14)$$

Here ρ_k is a sequence of step sizes that satisfies the conditions $\sum_{k=1}^{\infty} \rho_k = \infty$ and $\sum_{k=1}^{\infty} \rho_k^2 < \infty$. The calculation of the gradient $\nabla_\lambda \mathcal{L}_\lambda$ requires an integral over q_λ. For certain models, specifically those where the likelihood and prior are in the conjugate exponential family [Hoffman et al., 2013], this integral can be performed analytically.

Black box variational inference targets a much broader class of models by sampling $z^{[n]}, \theta^{[n]} \sim q_\lambda$ and replacing the gradient for each component i with a sample-based estimate [Ranganath et al., 2014]

$$\hat{\nabla}_{\lambda_i} \mathcal{L}_\lambda = \sum_{n=1}^{N} \nabla_{\lambda_i} \log q_\lambda(z^{[n]}, \theta^{[n]})(\log w^{[n]} - \hat{b}_i), \quad (15)$$

$$w^{[n]} = \gamma(z^{[n]}, \theta^{[n]})/q_\lambda(z^{[n]}, \theta^{[n]}), \quad (16)$$

in which $\hat{b}_i$ is a control variate that reduces the variance of the estimator

$$\hat{b}_i = \frac{\sum_{n=1}^{N} (\nabla_{\lambda_i} \log q_\lambda(z^{[n]}, \theta^{[n]}))^2 w^{[n]}}{\sum_{n=1}^{N} (\nabla_{\lambda_i} \log q_\lambda(z^{[n]}, \theta^{[n]}))^2}. \quad (17)$$

5 Black-box Policy Search

The sample-based gradient estimator in BBVI resembles the one used in classic likelihood-ratio policy gradient methods [Deisenroth et al., 2011], such as REINFORCE [Williams, 1992], G(PO)MDP [Baxter and Bartlett, 1999, Baxter et al., 1999], and PGT [Sutton et al., 1999]. There is in fact a close connection between BBVI and these methods, as has been noted by e.g. Dayan et al. [1995], Mnih and Gregor [2014] and Ba et al. [2014].

To make this connection precise, let us consider what it would mean to perform variational inference in a planning as inference setting. In this case, we can define a lower bound $\mathcal{L}_{\lambda,\lambda_0}$ on $\log Z_{\lambda_0}$ in terms of a variational distribution $q_\lambda(\tau,\theta)$ with parameters λ and an unnormalized density $\gamma_{\lambda_0}(\tau,\theta)$ of the form in equation 5, with parameters λ_0

$$\mathcal{L}_{\lambda,\lambda_0} = \mathbb{E}_{q_\lambda}[\log \gamma_{\lambda_0}(z,\theta) - \log q_\lambda(z,\theta)] \quad (18)$$
$$= E_{q_\lambda}\left[\beta R(\tau) + \log \frac{p_{\lambda_0}(\tau,\theta)}{q_\lambda(\tau,\theta)}\right] \quad (19)$$

If we now choose a variational distribution with the same form as the prior, then $q_\lambda(\tau,\theta) = p_{\lambda_0}(\tau,\theta)$ whenever $\lambda = \lambda_0$. Under this assumption, the lower bound at $\lambda = \lambda_0$ simplifies to

$$\mathcal{L}_{\lambda,\lambda_0}\Big|_{\lambda=\lambda_0} = E_{q_\lambda}\left[\beta R(\tau)\right]\Big|_{\lambda=\lambda_0} = \beta J_\lambda\Big|_{\lambda=\lambda_0}. \quad (20)$$

In other words, the lower bound $\mathcal{L}_{\lambda,\lambda_0}$ is proportional to the expected reward J_λ when the variational posterior is equal to the prior.

The gradient of the lower bound similarly simplifies to

$$\begin{aligned}\nabla_\lambda \mathcal{L}_{\lambda,\lambda_0}\Big|_{\lambda=\lambda_0} &= E_{q_\lambda}\left[\nabla_\lambda \log q_\lambda(\tau,\theta) \log \frac{\gamma_{\lambda_0}(\tau,\theta)}{q_\lambda(\tau,\theta)}\right]\Big|_{\lambda=\lambda_0} \\ &= E_{q_{\lambda_0}}\left[\nabla_\lambda \log q_\lambda(\tau,\theta)\Big|_{\lambda=\lambda_0} \beta R(\tau)\right] \\ &= \int d\tau d\theta \; \nabla_\lambda q_\lambda(\tau,\theta)\Big|_{\lambda=\lambda_0} \beta R(\tau) \\ &= \nabla_\lambda J_\lambda\Big|_{\lambda=\lambda_0}.\end{aligned}$$

The implication of this identity is that we can perform gradient ascent on J_λ by making a slight modification to the update equation

$$\lambda_{k+1} = \lambda_k + \rho_k \hat{\nabla}_\lambda \mathcal{L}_{\lambda,\lambda_k}\big|_{\lambda=\lambda_k}. \quad (21)$$

The difference in these updates is that instead of calculating the gradient $\hat{\nabla}_\lambda \mathcal{L}_{\lambda,\lambda_0}$ estimate relative to a fixed set of prior parameters λ_0, we update the parameters of the prior $p_{\lambda_k}(\tau,\theta)$ after each gradient step, and calculate the gradient $\nabla_\lambda \mathcal{L}_{\lambda,\lambda_k}$. We note that the constant β is simply a scaling factor on the step sizes ρ_k, and will from here on assume that $\beta = 1$.

When BBVI is performed using the update step in equation 21, and the variational family q_λ is chosen to have the same form as the prior p_λ, we obtain a procedure for EB estimation, which maximizes the normalizing constant Z_λ with respect to the parameters λ of the prior. The difference between the EB and maximum likelihood (ML) methods is that the first calculates the gradient relative to hyperparameters λ, whereas the other calculates the gradient relative to the parameters θ. Because this difference relates only to the assumed model structure, EB estimation is sometimes referred to as Type II maximum likelihood.

As is evident from equation 20, EB estimation in the context of planning as inference formulations maximizes the expected reward J_λ. In the context of a probabilistic programming system this means that we can effectively get three algorithms for the price of one: If we can provide an implementation of BBVI, then this implementation can be adapted to perform EB estimation, which in turn allows us to perform policy search by simply defining models where exponent

of the reward takes the place of the likelihood terms. This results in a method that we call black box policy learning (BBPL), which is equivalent to variants of REINFORCE applied to upper-level policy search.

6 Learning Probabilistic Programs

An implementation of BBVI and BBPL for probabilistic program inference needs to address two domain-specific issues. The first is that probabilistic programs need not always instantiate the same set of random variables, the second is that we need to distinguish between distributions that define model parameters θ and those that define latent variables z, or variables that are part of the context x in the case of decision problems.

Let us refer back to the program in Figure 1. The function `dfs-agent` performs a recursive loop until a stopping criterion is met: either the target node is reached, or there are no more paths left to try. At each step `dfs-agent` makes a call to `policy`, which is created by either calling `make-random-policy` or `make-edge-policy`. A random policy samples uniformly from unexplored directions. When depth first search is performed with this policy, we are defining a model in which the number of context variables is random, since the number of steps required to reach the goal state will vary. In the case of the edge policy, we use a memoized function to sample edge preference values as needed, choosing the unexplored edge with the highest preference at each step. In this case the number of parameter variables is random, since we only instantiate preferences for edges that are (a) open, and (b) connect to the current location of the agent.

As has been noted by Wingate and Weber [2013], BBVI can deal with varying sets of random variables quite naturally. Since the gradient is computed from a sample estimate, we can compute gradients for a each random variable by simply averaging over those executions in which the variable exists. Sampling variables as needed can in fact be more statistically efficient, since irrelevant variables that never affect the trajectory of the agent will not contribute to the gradient estimate. BBVI has the additional advantage of having relatively light-weight implementation requirements; it only requires differentiation of the log proposal density, which is a product over primitive distributions of a limited number of types, for which derivatives can be computed analytically. This is in contrast to implementations based on (reverse-mode) automatic differentiation [Pearlmutter and Siskind, 2008], as is used in Stan [Kucukelbir et al., 2015], which store derivative terms for the entire computation graph.

To provide a language-agnostic definition of BBVI and BBPL, we formalize learning in probabilistic programs as the interaction between a program $\mathcal{P}$ and an inference back end $\mathcal{B}$. The program $\mathcal{P}$ represents all deterministic steps in the computation and has internal state (e.g. its environment variables). The back end $\mathcal{B}$ performs all inference-related tasks.

A program $\mathcal{P}$ executes as normal, but delegates to the inference back end whenever it needs to instantiate a random variable, or evaluate a conditioning statement. The back end $\mathcal{B}$ then supplies a value for the random variable, or makes note of the probability associated with the conditioning statement, and then delegates back to $\mathcal{P}$ to continue execution. We will assume that the programming language provides some way to differentiate between latent variables z, which are simply to be sampled, and parameters θ for which a distribution is to be learned. In Anglican the syntax `(sample (tag :policy d))`, as used in Fig. 1, is used as a general-purpose mechanism to label distributions on random variables. An inference back end can simply ignore these labels, or implement algorithm-specific actions for labeled subsets.

In order for the learning algorithm to be well-defined in programs that instantiate varying numbers of random variables, we require that the each random variable z_a is uniquely identified by an address a, which may either be generated automatically by the language runtime, or specified by the programmer. Each model parameter θ_b is similarly identified by an address b.

In BBVI, the interface between a program $\mathcal{P}$ and the back end $\mathcal{B}$ can be formalized with the following rules:

- Initially $\mathcal{B}$ calls $\mathcal{P}$ with no arguments $\mathcal{P}()$.
- A call to $\mathcal{P}$ returns one of four responses to $\mathcal{B}$:
  1. $(\texttt{sample}, a, f, \phi)$: Identifies a latent random variable (not a policy parameter) z_a with unique address a, distributed according to $f_a(\cdot \,|\, \phi_a)$. The back end generates a value $z_a \sim f_a(\cdot \,|\, \phi_a)$ and calls $\mathcal{P}(z_a)$.
  2. $(\texttt{learn}, b, f, \eta)$: For policy parameters, the address b identifies a random variable θ_b in the model, distributed according to a distribution f_b with parameters η_b. The back end generates $\theta_b \sim f_b(\cdot \,|\, \lambda_b)$ conditioned on a learned variational parameter λ_b and registers an importance weight $w_b = f_b(\theta_b \,|\, \eta_b)/f_b(\theta_b \,|\, \lambda_b)$. Execution continues by calling $\mathcal{P}(\theta_b)$.
  3. $(\texttt{factor}, c, l)$: Here c is a unique address for a factor with log probability l_c and importance weight $w_c = \exp(l_c)$. Execution continues by calling $\mathcal{P}()$.
  4. $(\texttt{return}, v)$: Execution completes, returning a value v.

Because each call to $\mathcal{P}$ is deterministic, an execution history is fully characterized by the values for each random variable that are generated by $\mathcal{B}$. However the set of random variables that is instantiated may vary from execution to execution. We write A, B, C for the set of addresses of each type visited in a given execution. The program $\mathcal{P}$ now defines an unnormalized density $\gamma_{\mathcal{P}}$ of the form

$$\gamma_{\mathcal{P}}(z, \theta) := p_{\mathcal{P}}(z, \theta) \prod_{c \in C} \exp(l_c), \tag{22}$$

$$p_{\mathcal{P}}(z, \theta) := \prod_{a \in A} f_a(z_a \,|\, \phi_a) \prod_{b \in B} f_b(\theta_b \,|\, \eta_b) \ . \tag{23}$$

Implicit in this notation is the fact that the distribution types $f_a(\cdot \,|\, \phi_a)$ and $f_b(\cdot \,|\, \eta_b)$ are return values from calls to $\mathcal{P}$, which implies that both the parameter values and the distribution type may vary from execution to execution. While $f_a(\cdot \,|\, \phi_a)$ and $f_b(\cdot \,|\, \eta_b)$ are fully determined by preceding values for z and θ, we assume they are opaque to the inference algorithm, in the sense that no analysis is performed to characterize the conditional dependence of each ϕ_a or η_b on other random variables in the program.

Given the above definition of a target density $\gamma_{\mathcal{P}}(z, \theta)$, we are now in a position to define the density of a variational approximation $\mathcal{Q}_\lambda$ to the program. In this density, the runtime values η_b are replaced by variational parameters λ_b

$$p_{\mathcal{Q}_\lambda}(z, \theta) := \prod_{a \in A} f_a(z_a \,|\, \phi_a) \prod_{b \in B} f_b(\theta_b \,|\, \lambda_b) \ . \tag{24}$$

This density corresponds to that of a mean-field probabilistic program, where the dependency of each θ_b on other random variables is ignored.

Repeated execution of $\mathcal{P}$ given the interface described above results in a sequence of weighted samples $(w^{[n]}, \theta^{[n]}, z^{[n]})$, whose importance weight $w^{[n]}$ is defined as

$$\begin{aligned} w^{[n]} &:= \gamma_{\mathcal{P}}(z^{[n]}, \theta^{[n]}) \ / \ p_{\mathcal{Q}_\lambda}(z^{[n]}, \theta^{[n]}) \\ &= \prod_{b \in B} \frac{f(\theta_b^{[n]} \,|\, \eta_b)}{f(\theta_b^{[n]} \,|\, \lambda_b)} \prod_{c \in C} \exp l_c^{[n]}. \end{aligned} \tag{25}$$

With this notation in place, it is clear that we can define a lower bound $\mathcal{L}_{\mathcal{Q}_\lambda, \mathcal{Q}_{\lambda_k}}$ analogous to that of Equation 19, and a gradient estimator analogous to that of Equation 15, in which the latent variables z take the role of the trajectory variables τ. In summary, we can describe a sequential decision problem as a probabilistic program $\mathcal{P}$ in which the log probabilities l_c are interpreted as rewards, parameters θ_b define the policy and all other latent variables z_a are trajectory variables. EB inference can then be used to learn the hyperparameters λ that maximize the expected reward, as described in Algorithm 1.

Algorithm 1 Black-box Policy Learning

initialize parameters $\lambda_{0,b} \leftarrow \eta_b$, iteration $k = 0$
repeat
  Set initial $\lambda_{k+1} = \{\lambda_{k,b}\}_{b \in B}$
  Run N executions of program $\mathcal{Q}_{\lambda_k}$, generating $(w^{[n]}, \theta^{[n]}, z^{[n]})$ according to Eqns. 24, 25
  for each address b **do**
    Let $N_b \leq N$ be the # of runs containing b
    Let $g_b^{[n]} := \nabla_{\lambda_{k,b}} \log f(\theta_b^{[n]} | \lambda_{k,b})$
    Compute baseline $\hat{b}_{\lambda_{k,b}}$ from Eq. 17
    $\hat{\nabla}_{\lambda_{k,b}} J_{\lambda_k} \leftarrow {N_b}^{-1} \sum g_b^{[n]} (\log w^{[n]} - \hat{b}_{\lambda_{k,b}})$
    Update $\lambda_{k+1,b} \leftarrow \lambda_{k,b} + \rho_k \hat{\nabla}_{\lambda_{k,b}} J_{\lambda_k}$
  end for
  $k \leftarrow k + 1$
until parameters λ_b converge

An assumption that we made when deriving BBPL is that the variational distribution $q_\lambda(\tau, \theta)$ must have the same analytical form as the prior $p_{\lambda_0}(\tau, \theta)$. Practically this requirement means that a program $\mathcal{P}$ must be written in such a way that the values of the hyperparameters η_b have the same constant values in every execution, since their values may not depend on those of random variables. One way to enforce this is to pass η as a parameter in the initial call $\mathcal{P}(\eta)$ by $\mathcal{B}$, though we do not formalize such a requirement here.

7 Case Studies

We demonstrate the use of programs for policy search in three problem domains: (1) the Canadian Traveler Problem, (2) a modified version of the RockSample POMDP, and (3) an optimal diagnosis benchmark inspired by the classic children's game Guess Who.

These three domains are examples of *deterministic* POMDPs, in which the initial state of the world is not known, and observations may be noisy, but the state transitions are deterministic. Even for discrete variants of such problems, the number of possible information states $x_t = (u_0, o_1, \ldots, u_{t-1}, o_t)$ grows exponentially with the horizon T, meaning that it is not possible to fully parameterize a distribution $\pi(u \,|\, x, \theta)$ in terms of a conditional probability table $\theta_{x,u}$. In our probabilistic program formulations for these problems, the agent is modeled as an algorithm with a number of random parameters, and we use BBPL to learn the distribution on parameters that maximizes the reward.

We implement our case studies using the probabilistic programming system Anglican [Wood et al., 2014]. We use the same experimental setup in each of the three

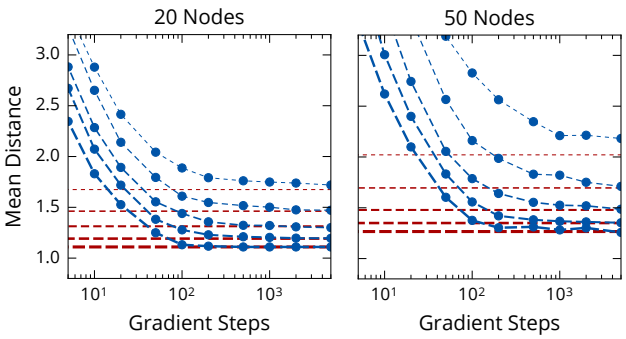

Figure 2: Convergence for CTP domains of 20 and 50 nodes. Blue lines show the mean traveled distance using the learned policy, averaged over 5 domains. Red lines show the mean traveled distance for the optimistic heuristic policy. Dash length indicates the fraction of open edges, which ranges from 1.0 to 0.6.

domains. A trial begins with a learning phase, in which BBPL is used to learn the policy hyperparameters, followed by a number of testing episodes in which the agent chooses actions according to a fixed learned policy. At each gradient update step, we use 1000 samples to calculate a gradient estimate. Each testing phase consists of 1000 episodes. All shown results are based on test-phase simulations.

Stochastic gradient methods can be sensitive to the learning rate parameters. Results reported here use a RMSProp style rescaling of the gradient [Hinton et al.], which normalizes the gradient by a discounted rolling decaying average of its magnitude with decay factor 0.9. We use a step size schedule $\rho_k = \rho_0/(\tau + k)^\kappa$ as reported in Hoffman et al. [2013], with $\tau = 1$, $\kappa = 0.5$ in all experiments. We use a relatively conservative base learning rate $\rho_0 = 0.1$ in all reported experiments. For independent trials performed across a range $1, 2, 5, 10, \ldots, 1000$ of total gradient steps, consistent convergence was observed in all runs using over 100 gradient steps.

The source code for the case studies, as well as the BBPL implementation, is available online.[1]

7.1 Canadian Traveler Problem

In the Canadian Traveler Problem (CTP) [Papadimitriou and Yannakakis, 1991], an agent must traverse a graph $G = (V, E)$, in which edges may be missing at random. It is assumed the agent knows the distance $d : E \to \mathbb{R}+$ associated with each edge, as well as the probability $p : E \to (0, 1]$ that the edge is open, but has no advance knowledge of the edges that are blocked. The problem is NP-hard [Fried et al., 2013],

[1] https://bitbucket.org/probprog/black-box-policy-search

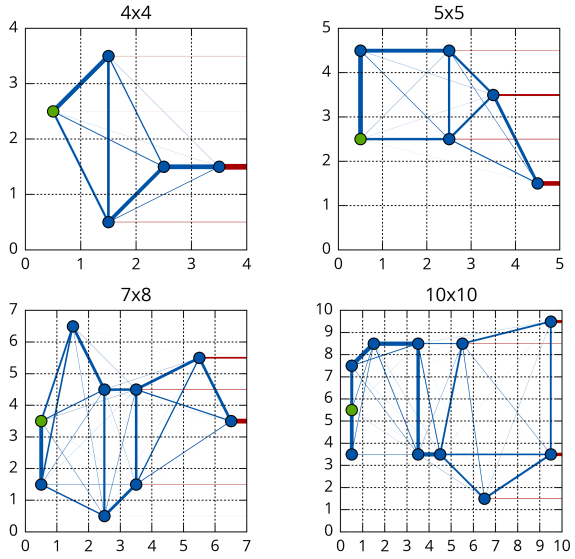

Figure 3: Learned policies for the Rock Sample domain. Edge weights indicate the frequency at which the agent moves between each pair of rocks. Starting points are in green, exit paths in red.

and heuristic online and offline approaches [Eyerich et al., 2010] are used to solve problem instances.

The results in Figure 1 show that the learned policy behaves in a reasonable manner. When edges are open with high probability, the policy takes the shortest path from the start node, marked in green, to the target node, marked in red. As the fraction of closed edges increases, the policy makes more frequent use of alternate routes. Note that each edge has a fixed probability of being open in our set-up, resulting in a preference for routes that traverse fewer edges.

Figure 2 shows convergence as a function of the number of gradient steps. Results are averaged over 5 domains of 20 and 50 nodes respectively. Convergence plots for each individual domain can be found in the supplementary material. We compare the learned policies against the optimistic policy, a heuristic that selects edges according to the shortest path, assuming that all unobserved edges are open. We observe that mean traveled distance for the learned policy converges to that of the optimistic policy, which is close to optimal.

7.2 RockSample POMDP

In the RockSample POMDP [Smith and Simmons, 2004], an $N \times N$ square field with M rocks is given. A rover is initially located in the middle of the left edge of the square. Each of the rocks can be either good or bad; the rover must traverse the field and collect samples of good rocks while minimizing the traveled distance. The rover can sense the quality of a rock remotely with

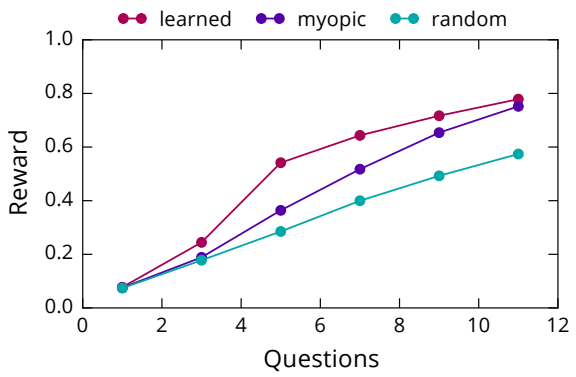

Figure 4: (left) Average reward in Guess Who as a function of number of questions. (right) Convergence of rewards as function number of gradient steps. Each dot marks an independent restart.

an accuracy decreasing with the distance to the rock. We consider a finite-horizon variant of the RockSample domain, described in the supplementary material, with a structured policy in which a robot travels along rocks in a left-to-right order.

The policy plots in Figure 3 show that this simple policy results in sensible movement preferences. In particular we point out that in the 5×5 instance, the agent always visits the top-left rock when traveling to the top-middle rock, since doing so incurs no additional cost. Similarly, the agent follows an almost deterministic trajectory along the left-most 5 rocks in the 10×10 instance, but does not always make the detour towards the lower rocks afterwards.

7.3 Guess Who

Guess Who is a classic game in which players pick a card depicting a face, belonging to a set that is known to both players. The players then take turns asking questions until they identify the card of the other player [Coster and Coster, 1979]. We here consider a single-player setting where an agent asks a pre-determined number of questions, but the responses are inaccurate with some probability. This is sometimes known as a measurement selection, or optimal diagnosis problem. We make use of a feature set based on the original game, consisting of 24 individuals, characterized by 11 binary attributes and two multi-class attributes, resulting in a total of 19 possible questions. We assume a response accuracy of 0.9. By design, the structure of the domain is such that there is no clear winning opening question. However the best question at any point is highly contextual.

We assume that the agent knows the reliability of the response and has an accurate representation of the posterior belief $b_t(s) = p(s \mid x_t)$ for each candidate s in given questions and responses. The agent selects randomly among the highest ranked candidates after the final question. We consider 3 policy variants, two of which are parameter-free baselines. In the first baseline, questions are asked uniformly at random. In the second, questions are asked according to a myopic estimate of the value of information [Hay et al., 2012], i.e. the change in expected reward relative to the current best candidates, which is myopically optimal in this setting. Finally, we consider a policy that empirically samples questions q according to a weight $v_q = \gamma^{n_q}(Ab)_q$, based on the current belief b, a weight matrix A, and a discount factor γ^{n_q} based on the number of times n_q a question was previously asked. Intuitively, this algorithm can be understood as learning a small set of α-vectors, one for each question, similar to those learned in point-based value iteration [Pineau et al., 2003]. The discounting effectively "shrinks" the belief-space volume associated with the α-vector of the current best question, allowing the agent to select the next-best question.

The results in Figure 4 show that the learned policy clearly outperforms both baselines, which is a surprising result given the complexity of the problem and the relatively simplistic form of this heuristic policy. While these results should not be expected to be in any way optimal, they are encouraging in that they illustrate how probabilistic programming can be used to implement and test policies that rely on transformations of the belief or information state in a straightforward manner.

8 Discussion

In this paper we put forward the idea that probabilistic programs can be a productive medium for describing both a problem domain and the agent in sequential decision problems. Programs can often incorporate assumptions about the structure of a problem domain to represent the space of policies in a more targeted manner, using a much smaller number of variables than would be needed in a more general formulation. By combining probabilistic programming with black-box variational inference we obtain a generalized variant of well-established policy gradient techniques that allow us to define and learn policies with arbitrary levels of algorithmic sophistication in moderately high-dimensional parameter spaces. Fundamentally, policy programs represent some form of assumptions about what contextual information is most relevant to a decision, whereas the policy parameters represent domain knowledge that generalizes across episodes. This suggests future work to explore how latent variable models may be used to represent past experiences in a manner that can be related to the current information state.

Acknowledgements

We would like to thank Thomas Keller for his assistance with Canadian traveler problem, and Rajesh Ranganath for helpful feedback on configuring RMSProp for black-box variational inference. Frank Wood is supported under DARPA PPAML through the U.S. AFRL under Cooperative Agreement number FA8750-14-2-0006, Sub Award number 61160290-111668.

References

D. Andre and S. J. Russell. State Abstraction for Programmable Reinforcement Learning Agents. In *AAAI*, 2002.

J. Ba, V. Mnih, and K. Kavukcuoglu. Multiple object recognition with visual attention. In *Proceedings of the International Conference on Learning Representations*, 2014. arXiv:1412.7755.

J. Baxter and P. Bartlett. Direct gradient-based reinforcement learning: I. Gradient estimation algorithms. Technical report, Computer Sciences Laboratory, Australian National University, 1999.

J. Baxter, L. Weaver, and P. Bartlett. Direct gradient-based reinforcement learning: II. Gradient ascent algorithms and experiments. Technical report, Computer Sciences Laboratory, Australian National University, 1999.

T. Coster and O. Coster. Guess Who? `http://theoradesign.com`, 1979.

P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel. The Helmholtz machine. *Neural Computation*, 7(5):889–904, 1995.

M. P. Deisenroth, G. Neumann, and J. Peters. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2(2011):1–142, 2011.

P. Eyerich, T. Keller, and M. Helmert. High-quality policies for the Canadian traveler's problem. In *AAAI*, 2010.

D. Fried, S. E. Shimony, A. Benbassat, and C. Wenner. Complexity of Canadian traveler problem variants. *Theor. Comput. Sci.*, 487:1–16, 2013.

N. Goodman, V. Mansinghka, D. M. Roy, K. Bonawitz, and J. B. Tenenbaum. Church: a language for generative models. In *Uncertainty in Artificial Intelligence*, pages 220–229, 2008.

A. D. Gordon, T. A. Henzinger, A. V. Nori, and S. K. Rajamani. Probabilistic programming. In *International Conference on Software Engineering (ICSE, FOSE track)*, 2014.

N. Hay, S. Russell, D. Tolpin, and S. Shimony. Selecting Computations: Theory and Applications. In *Uncertainty in Artificial Intelligence*, 2012.

G. Hinton, N. Srivastava, and K. Swersky. `http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf`.

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

M. W. Hoffman, N. d. Freitas, A. Doucet, and J. R. Peters. An expectation maximization algorithm for continuous Markov Decision Processes with arbitrary reward. In *International Conference on Artificial Intelligence and Statistics*, pages 232–239, 2009a.

M. W. Hoffman, H. Kueck, N. de Freitas, and A. Doucet. New inference strategies for solving Markov decision processes using reversible jump MCMC. In *Uncertainty in Artificial Intelligence*, pages 223–231. AUAI Press, 2009b.

H. J. Kappen. Path integrals and symmetry breaking for optimal control theory. *Journal of Statistical Mechanics: Theory and Experiment*, 2005(11), 2005. P11011.

L. Kocsis and C. Szepesvári. Bandit based Monte-Carlo planning. In *European Conference on Machine Learning*, pages 282–293, 2006.

A. Kucukelbir, R. Ranganath, A. Gelman, and D. M. Blei. Automatic Variational Inference in Stan. *Neural Information Processing Systems*, 2015.

S. Levine and V. Koltun. Guided Policy Search. In *International Conference on Machine Learning*, volume 28, pages 1–9, 2013.

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

J. S. Maritz and T. Lwin. *Empirical Bayes methods*, volume 35. Chapman and Hall, London, 1989. ISBN 0412277603.

B. Milch, B. Marthi, S. Russell, D. Sontag, D. L. Ong, and A. Kolobov. Blog: Probabilistic models with unknown objects. *Statistical relational learning*, page 373, 2007.

T. Minka, J. Winn, J. Guiver, S. Webster, Y. Zaykov, B. Yangel, A. Spengler, and J. Bronskill. Infer.NET 2.6, 2014. Microsoft Research Cambridge. http://research.microsoft.com/infernet.

A. Mnih and K. Gregor. Neural variational inference and learning in belief networks. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1791–1799, 2014.

G. Neumann. Variational Inference for Policy Search in Changing Situations. In *International Conference on Machine Learning*, 2011.

D. Nitti, V. Belle, and L. De Raedt. Planning in Discrete and Continuous Markov Decision Processes by Probabilistic Programming. In *ECML PKDD*, Lecture Notes in Computer Science, pages 327–342, Cham, 2015. Springer International Publishing.

C. H. Papadimitriou and M. Yannakakis. Shortest paths without a map. *Theor. Comput. Sci.*, 84(1):127–150, July 1991.

B. A. Pearlmutter and J. M. Siskind. Using programming language theory to make automatic differentiation sound and efficient. *Advances in Automatic Differentiation*, pages 79–90, 2008.

A. Pfeffer. Figaro: An object-oriented probabilistic programming language. Technical report, 2009.

J. Pineau, G. Gordon, and S. Thrun. Point-based value iteration: An anytime algorithm for POMDPs. In *International Joint Conference on Artificial Intelligence*, pages 1025–1030, 2003.

R. Ranganath, S. Gerrish, and D. M. Blei. Black Box Variational Inference. In *Artificial Intelligence and Statistics*, 2014.

K. Rawlik, M. Toussaint, and S. Vijayakumar. On Stochastic Optimal Control and Reinforcement Learning by Approximate Inference. *On Stochastic Optimal Control and Reinforcement Learning by Approximate Inference*, (2), 2012.

T. Smith and R. Simmons. Heuristic Search Value Iteration for POMDPs. In *Uncertainty in Artificial Intelligence*, pages 520–527, Arlington, Virginia, United States, 2004. AUAI Press.

S. Srivastava, S. Russell, P. Ruan, and X. Cheng. First-Order Open-Universe POMDPs. In *Uncertainty in Artificial Intelligence*, 2014.

R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy Gradient Methods for Reinforcement Learning with Function Approximation. *Neural Information Processing Systems*, pages 1057–1063, 1999.

E. Todorov. Efficient computation of optimal actions. *Proc. Nat. Acad. Sci. of America*, 106(28):11478–11483, 2009.

M. Toussaint, S. Harmeling, and A. Storkey. Probabilistic inference for solving (PO)MDPs. *Neural Computation*, 31(December):357–373, 2006.

M. J. Wainwright and M. I. Jordan. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning*, 1(1âĂŞ2):1–305, 2008.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, 1992.

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

D. Wingate, N. D. Goodman, D. M. Roy, L. P. Kaelbling, and J. B. Tenenbaum. Bayesian policy search with policy priors. In *International Joint Conference on Artificial Intelligence*, pages 1565–1570, 2011.

D. Wingate, C. Diuk, T. O. Donnell, J. Tenenbaum, S. Gershman, L. Labs, and J. B. Tenenbaum. Compositional Policy Priors. Technical report, Computer Science and Artificial Intelligence Laboratory, Cambridge, MA, 2013.

F. Wood, J. van de Meent, and V. Mansinghka. A new approach to probabilistic programming inference. In *Artificial Intelligence and Statistics*, pages 1024–1032, 2014.