Probabilistic Inference Using Program Analysis

Andrew D. Gordon
Microsoft Research Cambridge
adg@microsoft.com

Aditya V. Nori
Microsoft Research India
adityan@microsoft.com

Sriram K. Rajamani
Microsoft Research India
sriram@microsoft

1. Introduction

Though probabilistic model checkers such as PRISM [10] have been built to analyze a variety of probabilistic models such as discrete and continuous Markov chains, they primarily check for conformance of satisfaction of probabilistic temporal logic formulas (such as PCTL or PLTL) by these models, which limits their applications to the formal methods community, and in analyzing system reliability and performance. In contrast, the machine learning community at large has focused on computing quantities such as maximum likelihood estimates or posterior probabilities for models, and such inferences have applications in a variety of domains such as information retrieval, speech recognition, computer vision, coding theory and biology. In this paper, we argue that static and dynamic program analysis techniques can indeed be used to infer such quantities of interest to the machine learning community, thereby providing a new and interesting domain of application for program analysis.

Probabilistic models, particularly those with causal dependencies, can be succinctly written as probabilistic programs. Recent years have seen a proliferation of languages for writing such probabilistic programs, as well as tools and techniques for performing inference over these programs [5, 6, 8, 9, 11, 13]. Inference approaches can be broadly classified as static or dynamic —static approaches compile the probabilistic program to a probabilistic model such as a graphical model and then perform inference over the graphical model [8, 9, 11] exploiting its structure. On the other hand, dynamic approaches work by running the program several times using sampling to generate values, and perform inference by computing statistics over the results of several such runs [6, 13]. We believe that ideas from the areas of static and dynamic analysis of programs can be profitably used for the purpose of inference over probabilistic programs, and hence can find applications in inference for machine learning.

2. Probabilistic Programs

We first introduce probabilistic programs and probabilistic inference in this section. The next section argues that inference can be performed using static and dynamic program analysis.

Probabilistic programs are imperative programs with two added constructs (1) the ability to draw values at random from distributions, and (2) the ability to condition values of variables in a program through observations. We introduce probabilistic programs and inference using a series of examples.

```
bool c1, c2;
c1 := Bernoulli(0.5);
c2 := Bernoulli(0.5);
```

Example 1.

```
bool c1, c2;
c1 := Bernoulli(0.5);
c2 := Bernoulli(0.5);
observe(c1 || c2);
```

Example 2.

Figure 1. Two probabilistic programs.

```
bool b, c;
b := true;
c := Bernoulli(0.5);
while (c){
  b := !b;
  c := Bernoulli(0.5);
}
```

Example 3.

```
bool c1, c2;
c1 = Bernoulli(0.5);
c2 = Bernoulli(0.5);
while !(c1 || c2) {
   c1 = Bernoulli(0.5);
   c2 = Bernoulli(0.5);
}
```

Example 4.

Figure 2. Probabilistic programs with loops.

Consider Example 1 in Figure 1. Intuitively, this program tosses two fair coins (simulated by drawing from a Bernoulli random variable with mean 0.5), assigns the outcomes of these coin tosses to c1 and c2 respectively, and returns the values of the two variables c1 and c2. The program represents a probability distribution over Bernoulli variables c1 and c2, where Pr(c1=false,c2=false) = Pr(c1=false,c2=true) = Pr(c1=true,c2=false) = Pr(c1=true,c2=true) = 1/4.

Next, consider Example 2 in Figure 1. In this program, in addition to tossing the two coins and assigning the outcomes to c1 and c2, we have the statement observe(c1||c2). The semantics of the observe statement classifies runs which satisfy the boolean expression c1||c2 as *valid* runs. Runs that do not satisfy c1||c2 are classified as *invalid* runs. The program specifies the generated distribution over the values of the variables (c1, c2) conditioned over valid runs, which is given by Pr(c1=false,c2=false) = 0, and Pr(c1=false,c2=true) = Pr(c1=true,c2=false) = Pr(c1=true,c2=true) = 1/3.

Next, we consider probabilistic programs with loops. Consider Example 3 in Figure 2. This program initializes b to true and c to the outcome of tossing a coin. Then, it loops until c becomes false, toggling b and assigning to c the result from a fresh coin-toss in every iteration of the loop. The while-loop terminates with probability 1, since for the loop to not terminate, c should be always assigned true from the coin toss, the probability of which decreases exponentially with the number of iterations. The program specifies the generated distribution over the values of the variables (b, c) given by: Pr(b=true,c=true) = 0, and Pr(b=false,c=true) = false, and Pr(b=true,c=false) = 2/3, and Pr(b=true,c=false) = 1/3. The probability Pr(b = true, c = false) is the probability that the program executes the while loop an even number of times, given the summation $(1/2) + (1/8) + (1/32) + \cdots$, which converges to $2/3$, and the probability $\mathsf{Pr}(\mathtt{b} = \mathtt{true}, \mathtt{c} = \mathtt{false})$ is the probability that the program executes the while loop an odd number of time, given by the summation $(1/4) + (1/16) + (1/64) + \cdots$, which converges to $1/3$.

Consider Example 4 in Figure 2. This program repeatedly assigns to `c1` and `c2` outcomes of fair coin tosses, in a loop, until the condition $(c1||c2)$ becomes true. Thus, this program specifies the generated distribution over the variables $(\mathtt{c1}, \mathtt{c2})$ given by: Pr(`c1=false,c2=false`) = 0, and Pr(`c1=false,c2=true`) = Pr(`c1=true,c2=false`) = Pr(`c1=true,c2=true`) = 1/3. The alert reader would notice that this distribution is identical to the distribution specified by Example 2. Though `observe` statements can be equivalently represented using `while` loops using a simple program transformation illustrated in this example, our inference algorithm handles `observe` statements more efficiently, when compared to loops. Also, there is no simple transformation that converts any `while` loop to an `observe` statement. Consequently, we have both `observe` statements and `while` loops in our language.

2.1 Inference

Calculating the distribution specified by a probabilistic program is called *inference*. The inferred probability distribution is called *posterior probability* distribution, and the initial guess made by the program is called the *prior probability* distribution. One way to perform inference is to compile the probabilistic program to a probabilistic model [9] over which inference is performed using message passing algorithms such as belief propagation and its variants [12]. Alternatively, one can execute the program several times using sampling to execute probabilistic statements, and observe the values of the desired variables in valid runs [6], and compute statistics on the data to infer an approximation to the desired distribution.

3. Inference is analysis of probabilistic programs

Our key message is: *inference is analysis of probabilistic programs*. We propose new directions for efficient inference of probabilistic programs based on techniques from static and dynamic analysis of programs:

Static analysis The semantics of a probabilistic program can be calculated exactly via the classic idea of symbolic execution. To achieve efficiency, we use Algebraic Decision Diagram (ADD) [1], a graphical data structure for compactly representing finite functions. Our algorithm consists of doing symbolic execution of the probabilistic program, maintaining at each step a symbolic representation of the joint distribution. Prior techniques for static inference are restricted to loop-free programs [4]. We are able to statically analyze probabilistic programs with loops using the idea of fixpoints from the program analysis and verification communities. Even for loop-free programs, we show that symbolic execution offers performance benefits over existing techniques. We are able to perform exact inference, and hence compute an answer with better precision than current techniques which use marginal distributions to scale. For very large programs, we show how to perform symbolic execution approximately and scale, retaining the scaling benefits of approximate techniques, and in many cases obtaining answers with better precision [3, 7].

Dynamic analysis Dynamic approaches (which are also called sampling based approaches) are widely used, since running a probabilistic program is natural, regardless of the programming language used to express the program. However, there are two main challenges with sampling based approaches. The first challenge is the quality and diversity of samples obtained from the joint probability distribution represented by the program. The main issue here is that there are many interdependent choices to be made during sampling, and choices that are unlikely apriori, may be highly likely aposteriori in light of observations or evidence. In the context of probabilistic programs, these choices correspond to exploring distinct paths in the program. Straightforward sampling of the program fails to sufficiently explore these paths, leading to poor estimated results. The second challenge for sampling from probabilistic programs (even along a single path) is that many samples that are generated during execution are ultimately rejected for not satisfying the observations. This is analogous to rejection sampling algorithms. In order to improve efficiency, it is desirable to avoid generating samples that are later rejected, to the extent possible. Our sampling algorithm uses program analysis in order to address both the above challenges. We choose paths using a path selection heuristic, which ensures that as we explore more paths the residual probability mass converges to zero. We show how to hoist observations using Dijkstra's weakest preconditions to sampling statements, in order to avoid generating samples that are later rejected. Together, these techniques enable us to improve the quality and efficiency of sampling based estimation [2].

References

[1] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, 1997.

[2] A. T. Chaganty, A. V. Nori, and S. K. Rajamani. Efficiently sampling probabilistic programs via program analysis. In *NIPS Workshop on Probabilistic Programming (to appear)*, 2012.

[3] G. Claret, S. K. Rajamani, A. V. Nori, A. D. Gordon, and J. Borgström. Bayesian inference for probabilistic programs via symbolic execution. Technical Report MSR-TR-2012-86, Microsoft Research, 2012.

[4] A. Darwiche. SamIam. Software available from `http://reasoning.cs.ucla.edu/samiam`.

[5] W. R. Gilks, A. Thomas, and D. J. Spiegelhalter. A language and program for complex Bayesian modelling. *The Statistician*, 43(1): 169–177, 1994.

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

[7] A. D. Gordon, M. Aizatulin, J. Borgstrom, G. Claret, T. Graepel, A. Nori, S. Rajamani, and C. Russo. A model-learner pattern for bayesian reasoning. In *Principles of Programming Languages (POPL 2013, to appear)*.

[8] S. Kok, M. Sumner, M. Richardson, P. Singla, H. Poon, D. Lowd, and P. Domingos. The Alchemy system for statistical relational AI. Technical report, University of Washington, 2007. http://alchemy.cs.washington.edu.

[9] D. Koller, D. A. McAllester, and A. Pfeffer. Effective Bayesian inference for stochastic programs. In *National Conference on Artificial Intelligence (AAAI)*, pages 740–747, 1997.

[10] M. Z. Kwiatkowska. Model checking for probability and time: from theory to practice. In *LICS*, pages 351–. IEEE Computer Society, 2003. ISBN 0-7695-1884-2.

[11] T. Minka, J. Winn, J. Guiver, and A. Kannan. Infer.NET 2.3, Nov. 2009. Software available from `http://research.microsoft.com/infernet`.

[12] J. Pearl. *Probabilistic reasoning in intelligent systems – networks of plausible inference*. I-XIX. Morgan Kaufmann, 1989.

[13] A. Pfeffer. *Statistical Relational Learning*, chapter The design and implementation of IBAL: A General-Purpose Probabilistic Language. MIT Press, 2007.