Exact and Approximate Inference Methods for Bayesian Networks

Zachary T. Johnson

Undergraduate Student, Department of Electrical and Computer Engineering
Iowa State University, Ames, Iowa, USA
ztj1@iastate.edu

ABSTRACT

This document highlights some of the methods of exact and approximate probabilistic inference in Bayesian Networks. I will begin by giving an overview of the purpose of Bayesian Networks, describing their numerical and topological semantics, and defining some important terms necessary for understanding later topics. I will then describe some of the foundational methods of probabilistic inference in Bayesian Networks, beginning with exact inference and continuing on to approximate inference. My descriptions will include their computational complexities as well as the primary cases in which they are useful. My information will be drawn largely from [1] and [3]. I have also added some additional information on Markov chain properties and semantics.
Keywords: Bayesian Networks, Inference Methods, Markov Chain Monte Carlo

I. Introduction

Uncertain environments present many problems when it comes to knowledge representation. In these domains, propositional and first order logic can no longer represent relationships without exhaustively describing the relationships between every variable in the universe. To avoid this problem, we introduce the concept of "degree of belief". Using probability theory, we can determine our degree of belief and summarize it with a value between 0 and 1 that represents how strongly we believe a proposition.

Building upon this, we can summarize the conditional relationships between multiple variables using a joint probability distribution. However, in complex problems with many variables, calculating the full joint distribution can become computationally intense; the size of the table grows exponentially with the number of variables.

One way to reduce the complexity is to take advantage of independence properties. If one variable is independent of another, we do not need to include their conditional probabilities in the joint distribution. To avoid computing and storing these unnecessary values, we can instead divide the set of variables into independent subsets, and compute only the conditional probability tables. Using conditional independence, we can break the tables down even further.

With this new information comes the need for a way to succinctly represent the dependencies among variables. In response to this problem, we have Bayesian Networks. Bayesian Networks provide a way to represent complex probabilistic relationships in an intuitive way. However, for reasons that I will describe later, there are many issues associated with probabilistic inference in Bayesian Networks, including computational complexity and mixing rate limitations. To combat these issues, many different approaches have been developed.

In this paper, I will describe some methods of computing and approximating posterior probabilities using Bayesian Networks, as well as the benefits and caveats of each of these methods. For exact inference, I will cover enumeration, variable elimination, and clustering. For approximate inference, I will cover rejection sampling and importance sampling, as well as Markov Chain Monte Carlo methods such as Gibbs and Metropolis-Hastings sampling.

II. Bayesian Network Semantics

In this section I will describe the numerical and topological semantics of Bayesian Networks, as well as some background information on conditional independence relationships.

A. Topological

A Bayesian Network is a directed acyclic graph-based data structure that includes nodes corresponding to variables and directed edges corresponding to dependence relationships. Each node has an associated conditional probability table describing its probability given each possible value combination of its “parents”, which are all the nodes which have an outgoing edge pointed to that node.

An important topological aspect of Bayesian Networks is the condition that a node is independent of its non-descendants given its parents [1]. Therefore, the parents of a node should be all the nodes with a direct effect on its probability.

B. Numerical

A Bayesian Network is only correctly structured if it adheres to the following equation, given in [1]:

$$\boldsymbol{P}(X_i \mid X_{i-1}, \dots, X_1 = \boldsymbol{P}(X_i \mid Parents(X_i))$$

where $Parents(X_i)) \subseteq \{X_{i-1},\dots,X_1\}$, the last condition being satisfied by the nodes being in topological order. This equation mathematically reiterates my earlier statement that each node is conditionally independent of its non-descendants given its parents.

C. Conditional Independence

Another very important aspect of Bayesian Networks is given in [1]: a variable is conditionally independent of all other variables in a network given its **Markov blanket**. The Markov blanket includes a nodes parents, children, and children’s parents. A test used in [1] is to check whether a set of nodes **Z** “d-separates” two sets **X** and **Y**. The test is as follows: convert all edges to undirected edges, add undirected edges between parent nodes that share a child, and check if there are any paths between **X** and **Y** that do not pass through **Z**. If there are not, then **Z** d-separates **X** and **Y**, and **X** is independent of **Y** given **Z**.

III. Exact Inference

Exact inference in Bayesian Networks is the computation of the exact value of a posterior distribution of a set of query variables given the observed values of some evidence variables. This section will describe some of the algorithms that accomplish exact inference in Bayesian Networks: enumeration and variable elimination.

A. Enumeration

The simplest algorithm for computing the posterior distribution is enumeration, or simply the summing of the values from the full joint distribution. This computation can be done using the equation from [2]:

$$\boldsymbol{P}(X \mid \boldsymbol{e}) = \alpha \boldsymbol{P}(X, \boldsymbol{e}) = \alpha \sum_{\boldsymbol{y}} \boldsymbol{P}(X, \boldsymbol{e}, \boldsymbol{y})$$

where the summation is over all possible values **y** of the unobserved variables. Done naively, this computation has complexity $O(n2^n)$, but it can be reduced to $O(2^n)$ by taking advantage of the structure of the network and moving some computations outside of the summation loop. Nevertheless, this is still very computationally inefficient. Many computations are calculated multiple times. The next method will take advantage of these repeated calculations to improve efficiency.

B. Variable Elimination

Variable elimination is a type of dynamic programming algorithm that stores intermediate results to avoid repeating computations. To understand this algorithm, we will first need to understand some terms.

According to [1], a **factor** is a matrix indexed by the values of its argument variables. These matrices are used to store conditional probabilities of a value of a variable given the value of one or more other variables (the arguments).

The **pointwise product**, represented by “×”, is described by [1] as yielding a new factor whose variables are the union of the variables of the operand, and whose entries are given by the product of the corresponding elements of the operands.

The last operation, called “**summing out**”, involves iterating through the values of an argument variable and summing the table entries for that variable, thus creating a new factor that is no longer dependent on that variable.

Using these three operations, here is how the algorithm works:

1) Variables are ordered "right-to-left", i.e. variables that were on the far right of the expression used in enumeration will be processed first, while variables that were on the left will be processed last.
2) For each variable in the new variable order, a factor is made, using the variable and all the observed evidence, and added to the list of factors.
3) If the variable is a hidden variable, i.e. not a query variable or evidence variable, sum out all the factors.
4) Once the loop is complete, take the pointwise product of all the factors and normalize the result.

While variable elimination does not improve the worst-case runtime of exact inference, it performs about 1,000 times faster than enumeration when reverse topological ordering of variables is used [1].

C. Special Cases

For a specific type of tree, called a **polytree**, in which there are one or zero undirected paths between any two nodes, time complexity of variable elimination is linear in the number of conditional probability table entries. However, for multiply connected networks, exact inference has exponential time and space complexity in the worst case. In fact, in [1], it is shown that Bayesian Network exact inference is #P-complete, or strictly harder than NP-complete problems.

IV. Approximate Inference

Approximate inference methods in Bayesian Networks estimate the true value of a posterior probability distribution. These methods are Monte Carlo algorithms, which estimate the answer based on samples, and whose accuracy will improve as more samples are taken. In this section, I will cover rejection sampling and importance sampling, as well as some Markov chain-based methods: Gibbs sampling and Metropolis-Hastings sampling.

A. Direct Sampling

The most basic idea of approximation is to take samples from a known distribution and calculate what proportion of the samples taken satisfy the query. We can do this by sampling:

$$\boldsymbol{P}(X_i \mid Parents(X_i))$$

for each variable (in topological order) and storing the resulting values of each sample in a vector. If we do this many times, we can approximate the probability of a given combination of variables by taking the number of occurrences of that combination and dividing it by the total number of samples.

This approach on its own is quite limited, because it does not take into account any evidence variables, which means we are unable to compute posterior probabilities.

B. Rejection Sampling

Rejection sampling is essentially an extension of direct sampling that allows us to estimate conditional probabilities given some evidence variables. The process of rejection sampling is fairly simple. First, we use direct sampling to generate some samples, and then we remove all samples that don't match the evidence. We can then divide the number of samples that match each value of our query variable by the total number of samples that match the evidence, and that will give us our posterior distribution. The standard deviation of the error in probabilities will be proportional to $1/\sqrt{n}$, where n is the number of samples that matched the evidence. This implies that the estimated probability will converge to the actual probability as n increases to infinity.

The main problem with this approach is how long it takes to converge. As the number of evidence variables increases, the number of samples that match the evidence decreases. In fact, the number of accepted samples decreases exponentially with the number of evidence variables. For this reason, convergence is very slow, and many samples must be generated.

C. Importance Sampling

Importance sampling is based around sampling a distribution Q that is not the true distribution (because the true distribution is hard to sample) and weighting the samples according to some correction factor (weight) P(x)/Q(x). The formula for sampling from Q and applying the correction factor is this [1]:

$$\hat{P}(\boldsymbol{z} \mid \boldsymbol{e}) = \frac{N_Q(\boldsymbol{z})}{N} \frac{P(\boldsymbol{z} \mid \boldsymbol{e})}{Q(\boldsymbol{z})}$$

$$\approx Q(\boldsymbol{z})\frac{P(\boldsymbol{z} \mid \boldsymbol{e})}{Q(\boldsymbol{z})} = P(\boldsymbol{z} \mid \boldsymbol{e})$$

where **Z** is the nonevidence variables and $N_Q(\boldsymbol{z})$ is the number of samples from Q with **Z**=**z**. This equation shows that the estimate converges to the true value regardless of the sampling distribution. However, choosing a good sampling distribution will result in faster convergence, so we want to choose one that is as similar as possible to the true distribution.

One common approach is **likelihood weighting**, where the sampling distribution Q_{WS} shown below is used [1]:

$$Q_{WS}(\boldsymbol{z}) = \prod_{i=1}^{l} P(z_i \mid parents(Z_i))$$

where **Z** = {Z_1, …, Z_l} is the nonevidence variables. The general formula for the weight in importance sampling is shown below [1]:

$$w(\boldsymbol{z}) = \frac{P(\boldsymbol{z} \mid \boldsymbol{e})}{Q_{WS}(\boldsymbol{z})} = \alpha \frac{P(\boldsymbol{z}, \boldsymbol{e})}{Q_{WS}(\boldsymbol{z})}$$

Where the normalization constant α is 1/P(**e**) and is the same for all samples. Since **z** and **e** together are all the variables in the network, we can expand this equation to [1]:

$$w(\boldsymbol{z}) = \alpha \frac{P(\boldsymbol{z}, \boldsymbol{e})}{Q_{WS}(\boldsymbol{z})}$$

$$= \alpha \frac{\prod_{i=1}^{l} P(z_i \mid parents(Z_i)) \prod_{i=1}^{m} P(e_i \mid parents(E_i))}{\prod_{i=1}^{l} P(z_i \mid parents(Z_i))}$$

$$= \alpha \prod_{i=1}^{m} P(e_i \mid parents(E_i))$$

Therefore, the weight is equal to the product of the probabilities of the evidence variables given their parents.

D. Markov Chains

The following approximation methods are largely based on **Markov chains**, so I will use this section to explain what a Markov chain is.

A Markov chain is a state machine in which state transitions are dependent on a probability distribution stored in a **transition matrix**. The transition matrix, which I will refer to as k, defines the probability of transitioning from one state to another. The value k(**x**→**x'**) is the probability of transitioning from state **x** to state **x'**. The transition matrix is a **stochastic matrix**, in which the probabilities in each row add up to one. Intuitively, this makes sense, because the row is selected by the state currently occupied (**x**), while the column corresponds to the state being transitioned to (**x'**). Markov chains also have a vector $\mathbf{x}^{(0)}$ denoting the initial state. Markov chains are also **memoryless** because the probability of transitioning to the next state is dependent only on the current state; there is no memory of previous states retained.

Another term we must define is **ergodic**; if a transition matrix is ergodic, there exists some point in time such that, for all pairs of states **x**, **x'** in the chain, if the initial state was **x**, after the time point there will always be a probability greater than 0 of being in state **x'**. Basically, every state can be reached from every other state, although maybe not in a single state transition.

We will also consider a distribution $\pi_t(\boldsymbol{x})$, which represents the probability of the system being in state **x** at time *t*. Given this, the system has reached its **stationary distribution** when $\pi_t(\boldsymbol{x}) = \pi_{t+1}(\boldsymbol{x})$. If the transition matrix is ergodic, there is exactly one $\pi_t(\boldsymbol{x})$ that satisfies this condition.

The last Markov chains concept we need to introduce is **detailed balance**. The transition matrix is in detailed balance with $\pi_t(\boldsymbol{x})$ when [1]:

$$\pi(\boldsymbol{x})k(\boldsymbol{x} \longrightarrow \boldsymbol{x}') = \pi(\boldsymbol{x}')k(\boldsymbol{x}' \longrightarrow \boldsymbol{x})$$

The following methods of approximation are **Markov Chain Monte Carlo (MCMC)** algorithms, which means instead of collecting every sample from scratch, we will instead generate each new sample by making some random change to the current sample.

E. Gibbs Sampling

Gibbs sampling is the first MCMC algorithm we will explore. This method starts with a random state and then randomly samples a value for a randomly chosen nonevidence variable in the network X_i. The variable to be sampled is chosen based on a probability distribution $\rho(i)$.

As we know, a variable X_i is independent of all other variables given its Markov blanket. Since all other variables are fixed, we can say that X_i is sampled conditioned on its Markov blanket. The distribution of this is given by [1]:

$$P(x_i \mid mb(x_i)) = \alpha P(x_i \mid parents(X_i)) \prod_{Y_j \in Children(X_i)} P(y_j \mid parents(Y_j))$$

This means that each value of X_i is calculated by the product of the probability of its children and the probability of x_i conditioned by its parents.

The most important property of Gibbs sampling is that, due to the way states are changed, the stationary distribution of the sampling process is equivalent to the posterior distribution of the nonevidence variables given the evidence [1]. There are three cases we need to discuss when considering this claim [1]:

1) The states **x** and **x'** differ in two or more variables. In this case, k(**x**→**x'**) = 0, because Gibbs sampling changes no more than one variable at a time.

2) The states differ in exactly one variable X_i that changes value from x_i to x_i'. The probability of this is:

   $$k(\mathbf{x} \to \mathbf{x}') = k\big((x_i, \bar{\boldsymbol{x}}_\iota) \longrightarrow (x_i', \bar{\boldsymbol{x}}_\iota)\big) = \rho(i) P(x_i' \mid \bar{\boldsymbol{x}}_\iota)$$

   where $\bar{\boldsymbol{X}}_\iota$ is all other variables except the evidence variables.

3) The states are the same. Any variable could be chosen, but the value sampled must be the same as before. The probability of this occurrence is:

   $$k(\mathbf{x} \to \mathbf{x}) = \sum_i \rho(i) k\big((x_i, \bar{\boldsymbol{x}}_\iota) \longrightarrow (x_i, \bar{\boldsymbol{x}}_\iota)\big) = \sum_i \rho(i) P(x_i \mid \boldsymbol{x}_i)$$

Gibbs sampling also satisfies detailed balance with $\pi = P(\boldsymbol{x} \mid \boldsymbol{e})$. For the first and third cases above, detailed balance is satisfied trivially, so we will only look at the second here:

$$\begin{aligned}
&\pi(\boldsymbol{x}) k(\boldsymbol{x} \longrightarrow \boldsymbol{x}') \\
&= P(\boldsymbol{x} \mid \boldsymbol{e}) \rho(i) P(x_i' \mid \bar{\boldsymbol{x}}_\iota, \mathbf{e}) \\
&= \rho(i) P(x_i, \bar{\boldsymbol{x}}_\iota \mid \mathbf{e}) P(x_i' \mid \bar{\boldsymbol{x}}_\iota, \mathbf{e}) \\
&= \rho(i) P(x_i \mid \bar{\boldsymbol{x}}_\iota, \mathbf{e}) P(\bar{\boldsymbol{x}}_\iota \mid \mathbf{e}) P(x_i' \mid \bar{\boldsymbol{x}}_\iota, \mathbf{e}) \\
&= \rho(i) P(x_i \mid \bar{\boldsymbol{x}}_\iota, \mathbf{e}) P(x_i', \bar{\boldsymbol{x}}_\iota \mid \mathbf{e}) \\
&= \pi(\boldsymbol{x}') k(\boldsymbol{x}' \longrightarrow \boldsymbol{x})
\end{aligned}$$

Lastly, provided that the conditional probability tables do not contain probabilities of 0 or 1, both the conditions of ergodicity are met. Therefore, the Markov chain will eventually converge to the stationary distribution, which is equivalent to the true posterior distribution.

One major benefit of Gibbs sampling is that the complexity of generating samples is independent of network size, and instead is dependent on the number of children of X_i and its range [1]. Another benefit is that it will outperform likelihood weighting when evidence is downstream, because the evidence variables remain fixed and will be considered even if they are downstream.

Gibbs sampling can encounter limitations, however, in certain network structures. There are certain cases in which not all variables can be reached by only changing one variable at a time. In this case, Gibbs sampling may never converge to the true posterior distribution. Similarly, there may be cases in which all states *can* be reached by Gibbs sampling, but the probability of crossing certain boundaries is very low, hindering the **mixing rate**, or rate of convergence. **Block sampling**, sampling multiple variables simultaneously, can help solve this problem. Metropolis-Hastings sampling, in the next section, also provides a solution.

F. Metropolis-Hastings Sampling

Metropolis-Hastings, similar to Gibbs sampling, operates with the goal of converging to π in order to sample the true posterior distribution. However, there are a few key differences.

Metropolis-Hastings has a two-step sampling process. First, a new state **x'** is sampled from a **proposal distribution** q(**x'** | **x**). Next, the proposed state is accepted based on an **acceptance probability** [1]:

$$a(\boldsymbol{x}' \mid \boldsymbol{x}) = \min\left(1, \frac{\pi(\boldsymbol{x}')q(\boldsymbol{x} \mid \boldsymbol{x}')}{\pi(\boldsymbol{x})q(\boldsymbol{x}' \mid \boldsymbol{x})}\right)$$

If the proposal is rejected, the state remains at **x**.
Besides the added acceptance probability, Metropolis-Hastings also has more flexibility in its proposal distribution. For example, the proposal distribution may include a small probability of generating a new state from scratch using importance sampling. This type of proposal distribution offers a solution to the mixing rate limitations of Gibbs sampling, but still maintains the benefits of the Markov chain-based methodology.

An important piece of Metropolis-Hastings is that it is guaranteed to converge to the correct stationary distribution, given that the transition kernel is ergodic [1]. To prove this, we will once again only focus on the case where **x** transitions to **x'**, as the other two cases are trivial. The probability of this transition is [1]:

$$k(\boldsymbol{x} \to \boldsymbol{x}') = q(\boldsymbol{x}' \mid \boldsymbol{x})a(\boldsymbol{x}' \mid \boldsymbol{x})$$

To prove detailed balance, we show that:

$$\begin{aligned}
&\pi(\boldsymbol{x})q(\boldsymbol{x}' \mid \boldsymbol{x})a(\boldsymbol{x}' \mid \boldsymbol{x}) \\
&= \pi(\boldsymbol{x})q(\boldsymbol{x}' \mid \boldsymbol{x})\min\left(1, \frac{\pi(\boldsymbol{x}')q(\boldsymbol{x} \mid \boldsymbol{x}')}{\pi(\boldsymbol{x})q(\boldsymbol{x}' \mid \boldsymbol{x})}\right) \\
&= \min\left(\pi(\boldsymbol{x})q(\boldsymbol{x}' \mid \boldsymbol{x}), \pi(\boldsymbol{x}')q(\boldsymbol{x} \mid \boldsymbol{x}')\right) \\
&= \pi(\boldsymbol{x}')q(\boldsymbol{x} \mid \boldsymbol{x}')\min\left(\frac{\pi(\boldsymbol{x})q(\boldsymbol{x}' \mid \boldsymbol{x})}{\pi(\boldsymbol{x}')q(\boldsymbol{x} \mid \boldsymbol{x}')}, 1\right) \\
&= \pi(\boldsymbol{x}')q(\boldsymbol{x} \mid \boldsymbol{x}')a(\boldsymbol{x} \mid \boldsymbol{x}')
\end{aligned}$$

A notable property of Metropolis-Hastings is the presence of $\pi(\boldsymbol{x}')/\pi(\boldsymbol{x})$ in the acceptance probability, meaning that if a proposed state is more likely than the current state, it will be accepted. In [3], Tierney highlights that Hastings [4] and Barker [6] explored alternative forms of the acceptance probability, but Peskun [5] later showed that this form is optimal, mostly due to it being less likely to reject candidate states.

Tierney [3] also describes some examples of kernels that can be used in Metropolis-Hastings for estimating posterior distributions, which I will describe below.

G. Random Walk Chains

Random walk chains feature a proposal distribution *q* that samples Z from a density *f* of a Lebesgue measure $\boldsymbol{E} = \mathbb{R}^k$, μ and then generates the new state ***x'*** by adding ***Z*** to the current state ***x*** [3]. Thus, we will have the new state ***x'*** = ***x*** + ***Z***. Intuitively, this means the new state ***x'*** will be a step in some random direction in *k*-dimensional space, sampled from ***E***. Common distributions that ***Z*** may be drawn from include a uniform distribution on a disk, a normal distribution, or a multivariate *t* distribution. The step size can be defined by a constant *c*, which Tierney suggests could reasonably be 1 or ½, and Graves [7] offers an automated solution to.

H. Independence Chains

Independence chains use a proposal distribution that samples candidate states from a fixed density *f*. In other words, *q*(***x'*** | ***x***) = *f*(***x'***). The acceptance probability $a(\boldsymbol{x}' \mid \boldsymbol{x})$ is written as [3]:

$$a(\boldsymbol{x}' \mid \boldsymbol{x}) = \min\left(1, \frac{w(y)}{w(x)}\right)$$

This method is very similar to importance sampling because candidate states with low weights are less likely to be accepted, while those with high weights are likely to be accepted; the process will often remain in these states for multiple steps at a time. If a state has too high a weight value, the process may stay there for too long, so it is important to choose an *f* that is as close to constant as possible. When *f* is constant, candidate states are never rejected, and the samples are drawn from π [3].

I. Rejection Sampling Chains

As a special case of independence chains, we can use rejection sampling to sample from *f* [3]. To accomplish this, we use a density function *h* and constant *c* such that $\pi(\boldsymbol{x}) \le ch(\boldsymbol{x})$ for all $\boldsymbol{x}$, i.e. ch "dominates" *f* for all ***x***. However, it is difficult to choose *c* that is large enough to dominate *f* without making it so large that the algorithm is inefficient. Fortunately, the rejection scheme provides a solution. We can define the acceptance probability as:

$$a(\boldsymbol{x}'|\boldsymbol{x}) = \begin{cases} 1, & \boldsymbol{x} \in C \\ \dfrac{ch(\boldsymbol{x})}{\pi(\boldsymbol{x})}, & \boldsymbol{x} \notin C, \boldsymbol{x}' \in C \\ \min\left(1, \dfrac{\pi(\boldsymbol{x}')h(\boldsymbol{x})}{\pi(\boldsymbol{x})h(\boldsymbol{x}')}\right), & \boldsymbol{x} \notin C, \boldsymbol{x}' \notin C \end{cases}$$

where $C = \{\boldsymbol{x}\colon \pi(\boldsymbol{x}) \le ch(\boldsymbol{x})\}$. With this acceptance probability, candidate states will occasionally be rejected when $\boldsymbol{x} \notin C$, compensating for the deficiency at $\boldsymbol{x}$.

J. Autoregressive Chains

Autoregressive chains are a sort of hybrid between random walks and independence chains. In this method, we generate new states by [3]:

$\boldsymbol{x}' = a + b(\boldsymbol{x} - a) + \boldsymbol{Z}$

where a is a fixed vector and b is a real constant or fixed $k \times k$ matrix. If $b = 1$, this reduces to the random walk kernel. If $a = 0$ and $b = 0$, it reduces to an independence kernel. If $0 < b < 1$, the current state is shrunk toward a before the increment $\boldsymbol{Z}$ is added. If $b = -1$, the current state is reflected about a before the increment $\boldsymbol{Z}$ is added [3].

V. Conclusion

Bayesian Networks offer an elegant representation of uncertain domains. However, performing probabilistic inference in these networks is still very expensive, requiring us to be clever in how we compute our posterior distribution. There are a multitude of methods available for solving this problem, only a few of which I have listed in this paper. As we have seen, each method has its own strengths and weaknesses.

Enumeration and variable elimination can give us the exact value of the posterior distribution, but in many cases are not practical due to their large computational expense.

Rejection sampling often fails due to the challenge of obtaining samples that match the evidence, but importance sampling improves upon this by instead weighting samples based on their usefulness.

Markov Chain Monte Carlo methods are likely the most applicable. Gibbs sampling improves upon importance sampling by including downstream evidence in calculations. Metropolis-Hastings improves upon Gibbs by allowing great flexibility in the choice of kernel, opening the door to methods that improve the mixing rate significantly. This flexibility of Metropolis-Hastings is demonstrated in the later sections describing random walks, independence chains, rejection sampling chains, and autoregressive chains.

Many challenges in the area of probabilistic inference, of course, remain open, but these methods are a foundation for many other methods that have and will continue to be developed to solve these problems.

VI.References

[1] S. J. Russell and P. Norvig, “Probabilistic Reasoning,” in *Artificial intelligence: A Modern Approach*, 4th ed., Hoboken, NJ: Pearson, 2021.

[2] S. J. Russell and P. Norvig, “Quantifying Uncertainty,” in *Artificial intelligence: A Modern Approach*, 4th ed., Hoboken, NJ: Pearson, 2021.

[3] L. Tierney, “Markov Chains for Exploring Posterior Distributions,” *The Annals of Statistics*, vol. 22, no. 4, pp. 1701–1728, 1994.

[4] W. K. Hastings, “Monte Carlo Sampling Methods Using Markov Chains and Their Applications,” *Biometrika*, vol. 57, pp. 97–109, Apr. 1970.

[5] P. H. Peskun, “Optimum Monte-Carlo sampling using Markov chains,” *Biometrika*, vol. 60, no. 3, pp. 607–612, Dec. 1973.

[6] A. A. Barker, “Monte Carlo Calculations of the Radial Distribution Functions for a Proton-Electron Plasma ,” *Australian Journal of Physics*, vol. 18, pp. 119–133, Apr. 1965.

[7] T. L. Graves, “Automatic Step Size Selection in Random Walk Metropolis Algorithms,” 2011.