Approximate and Exact Inference in Bayesian Networks Computer Project - Rough Draft

Josh Squires

November 15, 2022

1 Introduction

Making an inference with a Bayesian Network – computing the posterior $P(X|e)$, for a given set of query nodes X and set of evidence variables e – has been shown to be NP-hard [1]. In fact, general algorithms for approximating this marginal is *also* an NP-hard task [2]. Since interesting real-world applications of Bayes nets, such as medical diagnostic aids, may involve networks of substantial complexity, making exact inference intractable, there's clear value in attempting to understand strengths and weaknesses between different approximate inference schemes for Bayesian networks.

In this paper, three approximate inference methods – likelihood weighting, rejection sampling, and Gibbs sampling – are analyzed and compared to variable elimination, an exact inference method. In sections 2 and 3, background information on all four selected algorithms is provided. Sections 4 and 5 then describe the methodology used to produce the analysis and the results of the analysis, respectively.

2 Exact Inference

A Bayesian network is a graphical representation of a particular joint probability distribution; computing an inference involves adding terms comprised of products of conditional probability distributions:

$$P(X|e) = \frac{P(X,e)}{P(e)} = \frac{\sum_w P(X,e,w)}{\sum_x P(x,e)} \tag{1}$$

where $w \in W$ are all non-query, non-evidence variables. The sums on the right hand side of this expression typically contain repeated elements that may be cached instead of recalculated, opening the door to dynamic programming-based exact inference methods.

2.1 Variable Elimination

One such dynamic programming method for exact inference is variable elimination [3]. The efficiency of variable elimination relies on the idea that, because of the Markov property, any

variable that isn't an ancestor of a query or evidence node is irrelevant to the computation of (1). Remaining variables in w are then summed-out and intermediate results are cached as the computation of (1) proceeds.

As a demonstration of variable-elimination, consider Pearl's Alarm network[1], recreated in Figure 1. Suppose we have the query $P(Earthquake|MaryCalls = True, JohnCalls = True)$. In terms of initial factors, this query may be written

$$P(E|j,m) = \alpha f_1(E) \sum_b f_2(B) \sum_a f_3(A,B,E) f_4(A) f_5(A) \tag{2}$$

with

$$\begin{aligned} f_1(E) =& P(E) \\ f_2(B) =& P(b) \\ f_3(A,B,E) =& P(a|b,E) \\ f_4(A) =& P(j,a) \\ f_5(A) =& P(m,a) \end{aligned}$$

and α a normalization constant. Variable elimination begins by selecting an elimination ordering: here we'll pick $[A, E]$. Then, for each variable, all factors containing that variable are joined into a new factor. Starting with A

$$f_6(B,E) = \sum_a f_3(A,B,E) f_4(A) f_5(A)$$

gives

$$P(E|j,m) = \alpha f_1(E) \sum_b f_2(B) f_6(b,E). \tag{3}$$

Repeating with our last hidden variable E

$$f_7(E) = \sum_b f_2(B) f_6(B,E)$$

results in

$$P(E|j,m) = \alpha f_1(E) f_7(E) \tag{4}$$

which can then be computed by taking the pointwise product [3].

[1]The original publication associated with this network is unclear, but numerous secondary sources, including our course material, attribute it to Pearl. The CPT's shown in Figure 1 were taken from [11]

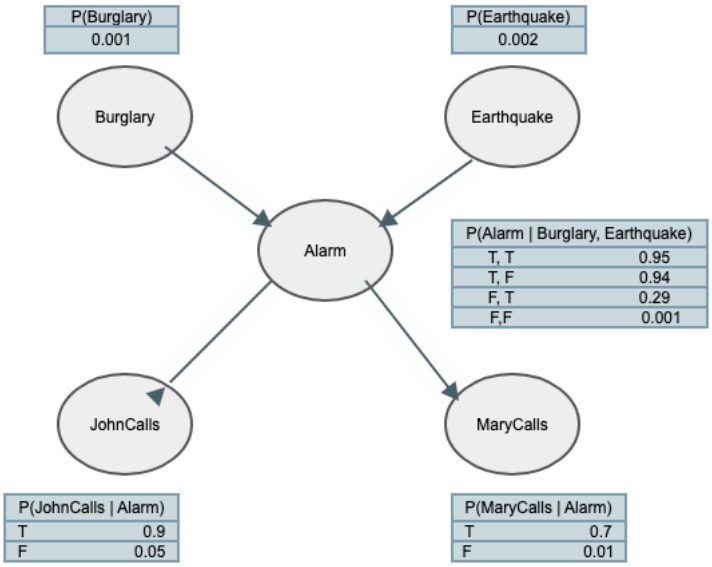

Figure 1: Judea Pearl's Alarm network.

2.1.1 Complexity

The complexity of variable-elimination is highly dependent on the structure of the Bayesian network and the particular ordering of variables used. Each ordering changes the sequence in which variables are marginalized over, which in turn alters the intermediate factors produced. Since the size of a factor product

$$f_3(x, y, z) = f_1(x, y) \times f_2(y, z) \tag{5}$$

scales exponentially with the number of variables in the scope of the factor (f_3 has eight entries, while f_1 and f_2 have four), the complexity of variable-elimination is driven by the factor of largest width. Assuming binary variables, if the largest width factor is k, variable-elimination has both time and space complexity of $2^{O(k)}$. As a result, choosing a good elimination ordering can have a significant effect on the complexity of variable-elimination.

3 Approximate Inference

This section describes three different Monte Carlo (MC) methods for approximate inference. MC methods work by generating random samples in accordance with the conditional probability tables that define the Bayesian network, and approximating the posterior by analyzing the distribution of the samples.

3.1 Rejection Sampling

The aptly named rejection sampling method for approximate inference generates N random samples from the prior distribution and tosses out all samples that disagree with the set of

evidence in order to estimate the posterior:

$$P(X|e) \approx \hat{P}(X|e) = \frac{\vec{N}(X,e)}{N(e)}. \tag{6}$$

Here, $\vec{N}(X,e)$ is the vector of sample counts that agree with the evidence, and $N(e)$ is the total number of samples that agree with the evidence. As more and more samples are drawn, this approximation gets closer and closer to the ground truth posterior distribution. Historically, the idea of rejection sampling dates back to the 1700's but was first used in a Bayesian network context in the 1980's [5]. An illustration of rejection sampling estimating $P(burglary|alarm = True)$ is shown in Figure 2.

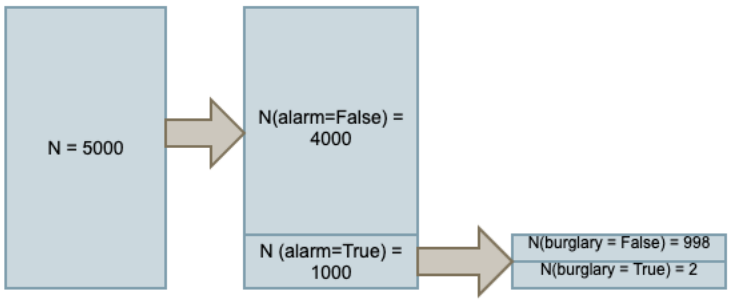

Figure 2: Rejection sampling on Pearl's alarm network predicts $P(burglary|alarm = True) = \frac{2}{1000}$. Note 80% of samples drawn are thrown out and do not contribute to the estimate.

3.1.1 Convergence

Assuming discrete valued random variables and an effectively infinite number of samples, rejection sampling will always converge to the exact answer by the law of large numbers. The trouble with rejection sampling is for inferences with many evidence variables this may take an exceedingly long time. This is because any samples that don't support the evidence are thrown out, which occurs with frequency $1 - P(e)$. This implies rejection sampling will converge very slowly for rare evidential sets – to get a macroscopic number of samples $O(\frac{1}{P(e)})$ iterations are needed. Rejection sampling is a poor inference approach if generating a sample that agrees with the evidence is an unlikely occurrence.

3.2 Likelihood Weighting

Likelihood weighting, attributed to Fung and Chang [9], seeks to correct the inefficiency of rejection sampling by fixing evidence variables to ensure consistency, and then correcting the drawn samples with a weight proportional to the likelihood of the evidence. Denoting the distribution with fixed evidence $Q(X) \equiv P(X, E = \mathbf{e})$, an estimate of the posterior with

likelihood weighting may be written as

$$\hat{P}(X|e) = w(X) \cdot Q(X) \tag{7}$$

$$= \alpha \prod_{i=1}^{m} P(e_i|parents(e_i)) \cdot Q(X). \tag{8}$$

Here $Q(X) \approx \frac{N_Q(\mathbf{x})}{N}$, $N_Q(\mathbf{x})$ is the number of samples with $X = \mathbf{x}$ and α is a normalization factor. In practice, as samples are drawn from Q, $w(x)/N$ is incrementally updated and moves closer and closer to the true likelihood values. To answer a particular query, one simply samples the a priori distribution Q, incrementally updates the likelihood weights, and then computes

$$\hat{P}(X = x, E = e) = \alpha \frac{w(x)}{N}. \tag{9}$$

An example of drawing a single sample given the evidential set $alarm = True$ is shown in Figure 3.

1. w = 1
2. Burglary is sampled. Suppose it returns False.
3. Earthquake is sampled. Suppose it returns False.
4.Alarm is in the evidential set -- fix to true and update
   w= w + P(Alarm |~Burglary, ~Earthquake) = 0.001
5. JohnCalls is sampled. Suppose it returns True.
6. MaryCalls is sampled. Suppose it returns True.

The sample x = (~burglary, ~earthquake, alarm, ~John calls, Mary calls)
has been generated with weight w= 0.001

Figure 3: Applying likelihood weighting to draw a sample and a weight on the Alarm network. This processes is repeated N times, incrementally refining $w(x)$ each time in order to estimate $\alpha w(x)/N$ for a given query x.

3.2.1 Convergence

As with rejection sampling, likelihood weighting is theoretically guaranteed to converge to the exact posterior by the law of large numbers. The advantage of evidence sampling – that only samples consistent with the evidence are sampled – speeds up convergence as compared to rejection sampling *unless* likelihood estimates for all nodes in the evidential set are either near 1.0 or 0.0. In this case, as before, for a meaningful amount of information to accumulate a large amount of samples must be drawn.

3.3 Gibbs Sampling

Similar to likelihood weighting, Gibbs sampling [10] fixes the evidential set at its measured values and then randomly samples to produce a complete sample. For Gibbs sampling, samples are generated with a Markov chain – variables are assigned one at a time by taking random steps, with each new variable's assignment conditioned on the current assignments that comprise its Markov blanket (parents, children, children's parents): $P(x_i|\text{Markov blanket}(x_i))$. This distribution can be computed from the Bayes net's CPTs.

Conditioning the steps of the Markov chain on the state's Markov blanket ensures that the stationary distribution of samples converges to the desired posterior distribution conditioned on the evidence [12]. Therefore, after N samples are drawn the posterior can be estimated with

$$\hat{P}(X = x|e) = \frac{\vec{N}(x)}{N} \tag{10}$$

where $N(x)$ the number of samples drawn with $X = x$.

3.3.1 Convergence

Unlike the previous approximate inference methods considered, Gibb's sampling may not always converge to the true posterior if given infinite compute time. This is becasue Gibb's sampling requires ergodicity[2] which does not occur if there are deterministic transitions encoded in the Bayes net [11].

4 Methodology

4.1 Algorithm Implementation

The analysis below uses open source python implementations of Bayesian inference algorithms developed and packed into the `sorobn` library by Max Halloway [6]. The following pseudocode captures implementation details for each inference method.

[2] A Markov chain is ergodic if all states can be reached from all other states, i.e. the transition probability $p(x, x')$ is greater than zero for links $x \leftrightarrow x'$ in the chain.

Algorithm 1 Variable-Elimination

procedure VARIABLE-ELIMINATION(X, e, net)
    initialize an empty list, f, to hold factors
    select an elimination ordering O for net.
    for for each node v in O **do**
        Create factor, f', for node, given e
        $f \leftarrow f + f'$
        if v is a hidden variable for input query **then**
            sum out v from any factor containing v
    Compute p_f, the pointwise product of f
    return normalized p_f.

Algorithm 2 Rejection Sampling

procedure REJECTION-SAMPLE(X, e, net, N)
    initialize a vector of zeros $\vec{N}$ of length $|X|$
    for $i = 1$ to N **do**
        sample $x \in X$ from net CPT's ▷ topological order; $x[k] = P(X_k|\text{parents}(X_k))$
        if x is consistent with e **then**
            $N[x] \leftarrow N[x] + 1$
    return $\vec{N}/N$

Algorithm 3 Likelihood Weighting

procedure LIKELIHOOD-WEIGHTING(X, e, net, N)
    initialize a vector of zeros $\vec{W}$ of length $|X|$.
    for $i = 1$ to N **do**
        Set $w = 1$, initialize $\vec{x}$ of length n =num nodes in net
        for $j = 1$ to n **do** ▷ loop in topological order
            if X_j is an evidence variable **then**
                Fix $x[j]$ to its value in e
                Update $w \leftarrow w \cdot P(X_j|\text{parents}(X_j))$
            if X_j is not an evidence variable **then**
                Set $x[j]$ by sampling $P(X_j|\text{parents}(X_j))$ from net
        Update entry $\vec{W}[\vec{x}] \leftarrow \vec{W}[\vec{x}] + w$
    return Normalized $\vec{W}$

Algorithm 4 Gibbs Sampling

procedure GIBBS-SAMPLING(X, e, net, N)
    initialize a vector of zeros $\vec{N}$ of length $|X|$
    let Z be the set of variables not in e.
    for $i = 1$ to N **do**
        initialize $\vec{x}$ of length n = num nodes in net randomly
        fix elements of $\vec{x}$ to match e
        while not all Z_i's set **do**
            randomly select a variable Z_j from Z
            set $\vec{x}[Z_i]$ by sampling $P(Z_i|MarkovBlanket(Z_i))$
    Update entry $N[\vec{x}] \leftarrow N[\vec{x}] + 1$
    return Normalized $\vec{N}$

4.2 Data

I've selected two well-known Bayesian networks to analyze: Lauritzen and Spiegelhalter's Asia network [7] and Binder *et al*'s Insurance network [8]. Data for both networks was taken from `bnlearn.com/bnrepository`. The Asia network consists of just 8 nodes, 8 edges, and 18 parameters, and the Insurance network is much larger with 27 nodes, 52 edges, and 984 total parameters. These networks were chosen because of their differing sizes, which ought to help reveal performance deltas between inference methods.

4.3 Analysis Metrics and Experiments

I selected the Kullback-Leibler (KL) divergence

$$D_{KL}(X, e) = \sum_{x \in X} P(x|e) \log \left(\frac{P(x|e)}{\hat{P}(x|e)} \right) \tag{11}$$

to measure the accuracy of the approximate inference methods describe above. Here $P(x|e)$ is the exact conditional distribution computed by variable elimination, $\hat{P}(x|e)$ is the approximate distribution computed by rejection sampling, likelihood weighting, or Gibbs sampling, and D_{KL} quantifies the difference between the approximate and exact conditional distribution. A large KL divergence indicates a weaker approximate inference method.

Each approximate algorithm is assessed using a python testbed `prob_inference.py` that reports accuracy and runtime, averaged over N runs, as a function of number of iterations (number of samples drawn from the prior distribution). For example, a user may choose to run rejection sampling for 100 iterations $N = 10$ times. This experimental framework provides insight into

1. the accuracy of each approximate inference algorithm
2. the computational intensity of each inference algorithm
3. sample efficiency of each algorithm

4.4 Code

A repository containing datasets, the python testbed, and usage instructions can be found at `github.com/jsquires0/approximate_inference`.

5 Analysis and Discussion

I haven't performed the analysis yet, but plots of average runtime, accuracy as a function of iterations will be placed here. I'll comment on trends evident in plots.

5.1 Discussion

Theoretical strengths/weaknesses vs what was actually observed will be discussed here.

5.2 Future Work

Here I'll mention ideas for future work, dependent on what I finish in analysis.

References

[1] G. F. Cooper, "The computational complexity of probabilistic inference using Bayesian belief networks", *Artificial Intelligence* **42** pp. 393-405 (1990)

[2] P. Dagum and M. Luby, "Approximate probabilistic inference in Bayesian networks is NP hard" *Artificial Intelligence*, **60** pp. 141-153 (1993)

[3] N. L. Zhang and D. L. Poole, "A simple approach to Bayesian network computations" (1994)

[4] G. Buffon "Essai d'arithmetique morale", Suppl. to Histoire naturelle, vol. IV (1777)

[5] M. Henrion, "Propagating Uncertainty in Bayesian Networks by Probabilistic Logic Sampling", *Machine Intelligence and Pattern Recognition* **5** pp. 149-163 (1988)

[6] https://github.com/MaxHalford/sorobn

[7] S. Lauritzen, D. Spiegelhalter, "Local Computation with Probabilities on Graphical Structures and their Application to Expert Systems (with discussion)". Journal of the Royal Statistical Society: Series B, **50** pp. 157-224 (1988)

[8] J. Binder, D. Koller et al. "Adaptive Probabilistic Networks with Hidden Variables" Machine Learning **29** pp. 213-244 (1997)

[9] R. Fung, K-C Chang, "Weighting and Integrating Evidence for Stochastic Simulation in Bayesian Networks" Machine Intelligence and Pattern Recognition, **10** pp. 209-219 (1990)

[10] J. Pearl "Evidential Reasoning Using Stochastic Simulation of Causal Models" Artificial Intelligence **32** pp.245-257 (1987)

[11] S. J. Russel and P. Norvig "Artificial Intelligence: a modern approach" 3rd ed. Upper Saddle River, NJ, Prentice Hall. (2010)

[12] J. Speagle "A Conceptual Introduction to Markov Chain Monte Carlo Methods" Computer Science (2019)