

Is there an efficient algorithm for approximate posterior inference in all Bayesian networks on typical-case observations?

George Matheos | Final project report for MIT course 6.541 | May 14, 2024

Abstract

In this paper, I extend Dagum and Luby’s 1993 result [1], strengthening the sense in which it is known that under standard complexity-theoretic conjectures, there are no efficient general-purpose algorithms for approximate posterior inference in Bayesian networks. Dagum and Luby showed that if $\mathbf{P} \neq \mathbf{NP}$, then there is no algorithm which can do approximate posterior inference in any Bayesian network, conditioning on worst-case observed values. I extend this to show that if pseudorandom generators or one-way functions exist, then there is no algorithm which can do approximate posterior inference in every Bayesian network, on random observations distributed according to the Bayesian network, with probability greater than $3/4$. Under the aforementioned assumptions, this implies that there is no general-purpose approximate inference algorithm which succeeds with very high probability when conditioning on observations generated by random processes in the world which are well described by the probabilistic model on hand.

1 Introduction

A central computational problem in science, artificial intelligence, and statistics, is posterior inference in probabilistic models. This problem’s input is the description of a probability distribution P on random values X and Y , and an assignment y to Y . Its output is a description of the posterior distribution $P(X|Y = y)$. A natural question for complexity theorists, therefore, is how hard variants of this problem are. One family of probabilistic models in which to study the complexity of inference is the family of Bayesian networks. A Bayesian network is a directed acyclic graph whose nodes are random variables, and whose edges represent probabilistic dependencies between the variables. Roughly, a Bayesian network is the probabilistic analog of a circuit (whereas a general probabilistic program [2] is the probabilistic analog of a Turing machine). In the early 1990s, Cooper [3] showed that if there were a general-purpose algorithm for exact probabilistic inference in Bayesian networks, then $\mathbf{P} = \mathbf{NP}$. Although Cooper did not note this in his paper, it follows from his result that such an algorithm would in fact imply $\mathbf{P} = \mathbf{PP}$, a stronger statement.¹ In 1993, noting that exact computation of posterior distributions is often unnecessary, and computing approximations is often sufficient, Dagum and Luby [1] extended Cooper’s result to show that in a strong sense, a general purpose algorithm for approximate posterior inference in Bayesian networks would also imply $\mathbf{P} = \mathbf{NP}$.

These proofs work by showing that for any instance of an NP-hard problem, there exists a Bayesian network P and an observation y such that computing the posterior $P(X = \cdot | Y = y)$ would yield a solution to the NP-hard problem. Just as exact inference is often unnecessary in practice, the ability to do inference under any worst-case observed value y is also unnecessary in many settings, and hence the worst-case nature of the value y in these constructions is unsatisfying.

Frequently, the probability distributions P under which it is relevant to do inference in practice are those which to some extent describe the distribution of things occurring in the world. Thus in many settings, if the model P is well-specified, the observed values y which will present themselves and under which we will need to do inference will be generated by a process in the world with a similar distribution to P . In

¹The reason for this is as follows. The canonical problem in \mathbf{PP} is to determine whether more than half of the inputs y of an upper-bounded size to a Turing machine $M(x, y)$ cause it to output 1, for a given x . For any given x we can construct a Bayesian network which posits that the bitstring y is generated from a uniformly random distribution, and another variable z has a value exactly equal to $M(x, y) = 1$. This involves encoding a circuit for M on inputs of this length into the Bayesian network. The \mathbf{PP} counting problem boils down to determining if $P(z = 1) \geq 1/2$. Say that there were indeed a general-purpose exact posterior inference algorithm for Bayesian networks. Then by Cooper’s result, $\mathbf{P} = \mathbf{NP}$, so for any value x it would be possible to find a value y such that $M(x, y) = 1$ in polynomial time. Then, if it were possible to compute $P(Y = y | z = 1)$ exactly, because $P(Y = y | z = 1) = P(Y = y, z = 1) / P(z = 1)$, and the value $P(Y = y, z = 1)$ is simply $\frac{1}{2^{|y|}}$ in this case, an algorithm could compute $P(z = 1)$ exactly in polynomial time. This would allow it to solve the counting problem in \mathbf{PP} in polynomial time.

such a situation, if one needed to do inference under P but there were a set $\mathcal{Y}_{\text{hard}}$ of observations y under which inference were computationally difficult, this would probably not be a problem if it were the case that $P(\mathcal{Y}_{\text{hard}}) < 0.0001$. Dagum and Luby’s hardness results therefore leave open the possibility that there exist general-purpose inference algorithms which succeed on typical-case observations, so long as the probabilistic model in question is well-specified (that is, matches the distribution of observations in the world). If it were possible to develop an inference algorithm which almost always worked in any well-specified Bayesian network model, that would be extremely valuable. This possibility has not been ruled out by existing results on the complexity of probabilistic inference.

In this paper, I extend Dagum and Luby’s 1993 result [1] to show that not only is approximate inference hard on worst-case observations, but in fact, if standard complexity theoretic conjectures hold, then there exist Bayesian networks in which inference under a randomly-generated observation is hard with probability $1/4$. That is, if these standard conjectures hold, then for any inference algorithm A , there exists a Bayesian network P and a set of observations $\mathcal{Y}_{\text{hard}}$ such that $P(\mathcal{Y}_{\text{hard}}) \geq 1/4$, and the inference algorithm fails to approximately compute $P(X|Y = y)$ on any $y \in \mathcal{Y}_{\text{hard}}$. Theorem 2 shows that one sufficient condition for this result is the existence of a pseudorandom generator. From this follows Corollary 3, stating that the existence of one-way functions also imply the nonexistence of universal inference algorithms which succeed on a $3/4$ -typical set of observations. Roughly, this means that if general-purpose typical-case approximate inference is possible, then public key cryptography is impossible.

This result provides new evidence that there are no general-purpose inference algorithms which succeed in every probabilistic model. However, note that the notion of typicality here is only with respect to observations in a given worst-case model. The possibility remains that there are meaningful senses in which for typical probabilistic models, inference is computationally easy.

1.1 Summary of the proof of the main result

The key idea in the proof of Theorem 2 is of a fairly different character from the reductions used in proofs of the hardness of inference from Cooper [3] and Dagum and Luby [1]. The starting point for this proof is not the assumption that $\mathbf{P} \neq \mathbf{NP}$, but instead the assumption of the existence of a pseudorandom generator. This is a function $G : \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that for any polynomial-time algorithm A , the distribution of A ’s outputs on uniformly random inputs is very close to the distribution of A ’s outputs on the outputs of G .

The proof of Theorem 2 involves the construction of a Bayesian network such that inference under typical case observations would allow one to distinguish the outputs of the pseudorandom generator from random strings. The Bayesian network implements a probabilistic process which first randomly decides whether to output a uniformly random bitstring, or to output a value computed by the pseudorandom generator G . Let x be a bit indicating which branch was chosen, let y denote the output of the process, and consider the posterior distribution $P(x = 1|y)$. This gives the posterior probability that the value y was generated by the pseudorandom generator, as opposed to uniformly at random. Under the specific Bayesian network constructed in my proof, I show that it cannot be easy to compute $P(x = 1|y)$ up to ϵ additive error for any $\epsilon < 1/2$ on a large typical set of y values. Specifically, I show that if there is an algorithm A and a set of bitstrings $\mathcal{Y}_{\text{good}}$ with more than $3/4$ probability under the generative process P , ie. if $P(\mathcal{Y}_{\text{good}}) > 3/4$, and A computes an ϵ -approximation of $P(x = 1|y)$ for all $y \in \mathcal{Y}_{\text{good}}$, then A distinguishes the outputs of G from random strings with nontrivial probability, in such a way that the distribution of the output of A on random bitstrings differs noticeably from the distribution of outputs of A on bitstrings output by G . If A runs in polynomial time, this contradicts the assumption that G is a pseudorandom generator.

1.2 Paper outline

Section 2 defines the problem of posterior inference and several computational formulations of it. Subsection 2.4 describes several hardness relationships between these formulations. These relationships indicate that if it is computationally hard to perform a particular form of approximate probabilistic inference, called additive-error probability mass function approximation, then it is computationally difficult to perform probabilistic inference per any of the other standard formulations. Section 3 provides background on Bayesian

networks and inference problems. In this section I define the problem of typical-case PMF approximation up to additive error. It follows from subsection 2.4 that if it is this is a computationally hard problem, inference on typical-case observations is hard under any of the other formulations outlined herein. Section 3 also states Dagum and Luby’s 1993 hardness result, and the conjectures of the existence of one-way functions and secure pseudorandom generators. Section 4 states the main result of this paper, Theorem 2, which shows that if secure pseudorandom generators exist, then there is no algorithm nor probabilistic algorithm which, for any Bayesian network, can perform additive PMF approximation, up to error ϵ for any $\epsilon < 1/2$, on a $(1 - \rho)$ -typical set of observations, for any $\rho < 1/4$.

2 Posterior inference and computation

2.1 Posterior inference

Consider a probability distribution P on a space $X \times Y \times Z$. For the purposes of this paper, we will take $X, Y, Z \subseteq \{0, 1\}^*$. The problem of posterior inference is to characterize the posterior distribution

$$P(X = \cdot | Y = y)$$

for a value $y \in Y$ s.t. $P(Y = y) > 0$.

I first define the marginal distributions

$$P(X = x, Y = y) := \sum_{z \in Z} P(x, y, z); \quad P(Y = y) := \sum_{(x, z) \in X \times Z} P(x, y, z)$$

The posterior $P(X = \cdot | Y = y)$ is the probability distribution such that for all $x \in X$,

$$P(X = x | Y = y) := \frac{P(X = x, Y = y)}{P(Y = y)}$$

Stated informally, the problem of posterior inference is: *given a probabilistic model P and a value y with $P(Y = y) > 0$, characterize $P(X = x | Y = y)$.*

Representing a probability distribution involves representing real-valued quantities, raising the question of how these values are stored in a computer. In this paper, I assume all real values are represented in computers as rational numbers stored exactly using a finite number of bits. The results in this paper are independent from the exact encoding, so I will not specify the representation of real values further.

2.2 Computational formulations of exact inference

To formalize the problem of posterior inference as a specification a computer algorithm could satisfy, we must specify what it would mean for an algorithm to “characterize” a probability distribution on the space X , as this is the type of mathematical object that $P(X = x | Y = y)$ is. There are several distinct algorithmic formulations:

1. **Probability mass function (PMF) computation.** Given a value $x \in X, y \in Y$, compute the number $P(X = x | Y = y)$.
2. **Cumulative distribution function (CDF) computation.** Given a value $x \in X, y \in Y$, and letting strings in $x \in X \subseteq \{0, 1\}^*$ have lexicographic ordering, compute the number $P(X \leq x | Y = y)$.
3. **Sampling.** Given a value $y \in Y$, sample a random value in X according to the distribution $P(X = x | Y = y)$. This requires the use of a probabilistic Turing machine. Under standard formulations, this means we want a Turing machine A which, given $y \in Y$ and an uniformly random bit sequence $r \in \{0, 1\}^*$, computes a value $A(y, r) \in X$ such that for all $x \in X$, $P(X = x | Y = y) = \Pr_r[x = A(y, r)]$.²

²The definition of exact sampling is more complicated when there are probabilities which do not have the form $m/2^k$ for some $m, k \in \mathbb{N}$. For the purposes of this paper on approximate inference, this can be ignored.

2.3 Computational formulations of approximate inference

It usually suffices for algorithms to perform approximate probabilistic inference. There are several standard formulations of approximate probabilistic inference as computational problems:

1. **PMF evaluation up to additive error.** Fix $\epsilon > 0$. Algorithm A does PMF evaluation in P up to additive error ϵ if for any $x \in X, y \in Y$, it computes a number $A(x, y)$ where $|A(x, y) - P(X = x|Y = y)| < \epsilon$.
2. **PMF evaluation up to relative error.** Fix $\epsilon > 0$. Algorithm A does PMF evaluation in P up to relative error ϵ if for any $x \in X, y \in Y$, it computes a number $A(x, y) \in (\frac{1}{1+\epsilon}P(X = x|Y = y), (1+\epsilon)P(X = x|Y = y))$.
3. **CDF evaluation up to additive error.** Fix $\epsilon > 0$. Algorithm A does CDF evaluation in P up to additive error ϵ if for any $x \in X, y \in Y$, it computes a number $A(x, y)$ where $|A(x, y) - P(X \leq x|Y = y)| < \epsilon$.
4. **CDF evaluation up to relative error.** Fix $\epsilon > 0$. Algorithm A does CDF evaluation in P up to relative error ϵ if for any $x \in X, y \in Y$, it computes a number $A(x, y) \in (\frac{1}{1+\epsilon}P(X \leq x|Y = y), (1+\epsilon)P(X \leq x|Y = y))$.
5. **Approximate sampling with pointwise probability bounds.** There are several formulations of approximate sampling. One formulation is to say that probabilistic algorithm $A(y, r)$ (where the probabilistic nature of the algorithm is derived from its consumption of a uniformly random bitstring r) approximately samples from $P(X = x|Y = y)$ up to pointwise error δ if for all $x \in X$, $|\Pr_r[A(y, r) = x] - P(X = x|Y = y)| < \delta$.

For any of the PMF or CDF computation problems, a probabilistic version can be defined in which a probabilistic algorithm A is said to do PMF or CDF computation iff it satisfies the above specification with probability $\geq 2/3$ on any input.

2.4 Some hardness relationships among these formulations

A number of relationships are known between the hardness of inference according to these different formulations. Below, I will articulate a subset of these which indicate that if it is computationally hard to do probabilistic PMF computation up to additive error, then it is computationally difficult to solve any of the aforementioned inference problems. Note that the following reductions can be applied for any given observation y , so the results in this paper indicating that additive PMF approximation is hard on typical sets of observations imply that all of these inference problems are hard to solve on typical sets of observations.

1. **Relative approximation for is at least as hard as additive approximation for equal ϵ .** Consider a true PMF or CDF value p . If $\epsilon \geq 1$ then additive PMF or CDF approximation is trivial, so consider an $\epsilon < 1$. Since $p \in [0, 1]$, if \hat{p} satisfies $\hat{p} \in (p/(1+\epsilon), (1+\epsilon)p)$, then

$$|\hat{p} - p| \leq \max(|(1+\epsilon)p - p|, |p/(1+\epsilon) - p|) \leq \max(|1+\epsilon-1|, |1/(1+\epsilon)-1|) = \epsilon$$

Thus if it is possible to compute a relative ϵ approximation to either CDF or PMF, it is also possible to compute an additive ϵ approximation to the same quantity.

2. **CDF computation is at least as hard as PMF computation.**

- (a) *Exact computation.* Say $\text{CDF}(x, y) := P(X \leq x|Y = y)$ can be computed exactly. Then we can compute $\text{PMF}(x, y) := P(X = x|Y = y) = \text{CDF}(x, y) - \text{CDF}(x-1, y)$.
- (b) *Additive approximation.* Say we can compute an additive ϵ approximation $\text{CDF}_\epsilon(x, y)$ to $P(X \leq x|Y = y)$. Then $\text{PMF}_{2\epsilon}(x, y) := \text{CDF}_\epsilon(x, y) - \text{CDF}_\epsilon(x-1, y)$ is a 2ϵ additive approximation to $P(X = x|Y = y)$.

3. Sampling is at least as hard as probabilistic PMF computation up to additive error.

Say that it is possible to sample with pointwise error $\epsilon/2$ from $P(X = \cdot | Y = y)$. Let Q be the exact distribution which can be sampled from, which has pointwise error $\leq \epsilon/2$ from the posterior. Consider an algorithm which generates N independent samples $x_i \sim$ and computes $\hat{p} := \frac{1}{N} \sum_{i=1}^N 1_{x_i=x}$. By the Hoeffding theorem, since $\mathbb{E}[\hat{p}] = Q(x)$, $\Pr[|\hat{p} - Q(x)| \geq \epsilon/2] \leq 2e^{-\epsilon^2/(2N)}$. Thus if $N = 100/\epsilon^2$, $\Pr[|\hat{p} - Q(x)| \geq \epsilon/2] \ll 1/3$. Because $|Q(x) - P(X = x | Y = y)| < \epsilon/2$, this means that $\Pr[|\hat{p} - P(X = x | Y = y)| > \epsilon] \ll 1/3$. Thus it is possible to compute an ϵ approximation to PMF at a cost $O(1/\epsilon^2)$ times the cost of performing approximate pointwise sampling. Therefore a polynomial time approximate sampling algorithm implies the existence of a polynomial time additive PMF approximation algorithm.

See [4] or [5] for a more detailed discussion of some of these relationships, and for more information about complexity theoretic formulations of computing aspects of probability distributions.

3 Background for the main results

3.1 Bayesian networks and inference problems

Definition 1. A **Bayesian network on binary variables** is a tuple (V, E, P) , where V is an ordered, finite set of variables $V = \{v_1, \dots, v_n\}$, E is a set of directed edges between the variables, and P is a *conditional probability table*. The directed graph (V, E) must be acyclic. For $v \in V$, $\text{Pa}(v)$ denotes the set of parent variables of v : $\text{Pa}(v) = \{u : (u \mapsto v) \in E\}$. Given any assignment $a_{\text{Pa}(v)} \in \{0, 1\}^{|\text{Pa}(v)|}$ to the parent variables of v , the conditional probability table P stores value $P(v = \cdot; a_{\text{Pa}(v)})$, which is a probability vector $[p_{v=0}, p_{v=1}]$ in \mathbb{R}^2 .

A general Bayesian network lifts the restriction that each variable v_i is binary, and allows it to have arbitrary finite domain. In this report, I will focus on binary Bayesian networks. Because a variable with a domain of size k can be represented using $\log k$ binary variables, the results proven herein carry to the general case. Henceforth, the phrase “Bayesian network” should be understood as a Bayesian network on binary variables.

Given a Bayesian network (P, E, V) , we can define a joint distribution on all the variables in V , with probability mass function

$$P(a) = \prod_{v_i \in V} P(v_i = a_i; a_{\text{Pa}(v_i)}) \quad \forall a \in \{0, 1\}^{|V|}$$

where $a_{\text{Pa}(v_i)}$ is the assignment to the parent variables of v_i in a . I will often write P to refer to the whole Bayesian network, the joint distribution on all its variables, and also marginal and conditional distributions on subsets of its variables.

A Bayesian network can be regarded as defining a probabilistic process which generates assignments to the variables in V . A probabilistic computer with the ability to sample from all of the conditional distributions in the Bayesian network³ can generate samples from the joint distribution P by traversing the graph in topological order, and sampling a value for each variable v from $P(v = \cdot | a_{\text{Pa}(v)})$ when v is encountered in the traversal. A Bayesian network can be regarded as the probabilistic analog of a circuit on no inputs, where deterministic logic gates are replaced by conditional probability distributions.

Definition 2. An **inference problem** consists of a Bayesian network (V, E, P) , a set of *observed variables* $Y \subseteq V$, a set of *query variables* $X \subseteq V$, and an assignment $y \in \{0, 1\}^{|Y|}$ to the observed variables such that $P(Y = y) > 0$.

The goal of an inference problem is to compute some piece of information about the posterior distribution $P(X = \cdot | Y = y)$, which is a probability distribution on $\{0, 1\}^{|X|}$. For convenience, I will use the notation

³This is always possible if, for instance, the conditional distributions have probability values which are dyadic rationals, that is rationals of the form $m/2^k$ for $m, k \in \mathbb{N}$.

$P(X = x|Y = y)$, $P(X = x, Y = y)$, and $P(Y = y)$ to refer to conditional, joint, and marginal probability values on the assignments to the variables in variable sets X and Y , even though this is slightly nonstandard notation as X and Y here are subsets of V here, and are not formally defined as random variables.

Definition 3. An **inference problem schema** is the tuple $I = (V, E, P, X, Y)$ as in an inference problem, but not fixing an assignment y to the observed variables.

3.2 Worst and typical case inference algorithms

Definition 4. A **deterministic, worst-case additive PMF approximation algorithm with tolerance ϵ** is a Turing machine A which on input (I, y, x) , where I is any inference problem schema, y is any assignment to the observed variables, and x is an assignment to the query variables, outputs a rational number $A(I, y, x)$ such that

$$|A(I, y, x) - P(X = x|Y = y)| < \epsilon \quad (1)$$

Definition 5. A **probabilistic, worst-case additive PMF approximation algorithm with tolerance ϵ** is a probabilistic Turing machine A which for every I, y, x , satisfies condition 1 with probability $\geq 2/3$.

In 1993, Dagum and Luby [1] showed that if there exists a deterministic worst-case additive PMF approximation algorithm with tolerance $< 1/2$, and it has polynomial runtime, then $\mathbf{P} = \mathbf{NP}$. Further, if a probabilistic polynomial-time worst-case additive PMF approximation algorithm exists with tolerance $< 1/2$, then $\mathbf{NP} \subseteq \mathbf{RP}$.

Bayesian networks are typically used to model aspects of the world. That is, each variable in V represents some aspect of the world; Y represents a set of values which we have observed; and X represents a set of values which we wish to infer. In this setting, the probability distribution P is a description of our beliefs about how probable different joint outcomes of events in the world are. Therefore, given a Bayesian network P in which we must do inference, it may be acceptable to us if there exist assignments y under which computing the posterior distribution is very expensive, so long as these instances are extremely rare. Since we have a probability distribution P on hand which ought to roughly correspond to the distribution of y values which will occur in the world, and on which we will have to run inference, a natural notion of “rare” is available. Say there is a small set of observation assignments $\mathcal{Y}_{\text{hard}} \subseteq \{0, 1\}^{|Y|}$ such that $P(\mathcal{Y}_{\text{hard}}) < \rho$ for very small ρ (e.g. $\rho = 0.00001$), such that for all $y \notin \mathcal{Y}_{\text{hard}}$, we can compute the posterior distribution $P(X = \cdot | Y = y)$ efficiently. Then we can say that inference is easy in the typical case, and for many purposes this is sufficient.

Definition 6. A **deterministic, typical-case additive PMF approximation algorithm with tolerances (ϵ, ρ)** is a Turing machine A which accepts inputs of the form (I, y, x) , where I is any inference problem schema, y is any assignment to the observed variables, and x is an assignment to the query variables, and outputs a rational number $A(I, y, x)$ with the following property. For any inference schema I and any value x ,

$$\Pr_{y \sim P(Y=\cdot)} [|A(I, x, y) - P(X = x|Y = y)| \geq \epsilon] < \rho$$

Given such an algorithm A , a given inference schema I , and a value x , I will write $\mathcal{Y}_{\text{good}}$ to denote the set

$$\mathcal{Y}_{\text{good}} := \{y : |A(I, x, y) - P(X = x|Y = y)| < \epsilon\}$$

Note that $P(\mathcal{Y}_{\text{good}}) > 1 - \rho$.

Definition 7. A **probabilistic, typical-case additive PMF approximation algorithm with tolerances (ϵ, ρ)** is a probabilistic Turing machine A which accepts inputs (I, y, x) as above, such that for any inference schema I and any value x , there exists a set $\mathcal{Y}_{\text{good}} \subseteq \{0, 1\}^{|Y|}$ where $P(\mathcal{Y}_{\text{good}}) > 1 - \rho$ and

$$y \in \mathcal{Y}_{\text{good}} \implies \Pr[|A(I, x, y) - P(X = x|Y = y)| < \epsilon] > 2/3$$

where the probability is taken over the randomness of the algorithm.

In this report, I prove a new result showing that if either a deterministic or probabilistic typical-case additive PMF approximation algorithm exists with tolerances $\epsilon < 1/2$, $\rho < 1/4$, then one-way functions do not exist. This hardness result suggests that typical-case inference is computationally difficult, and it is not merely the worst-case nature of the observations considered in Dagum and Luby's reduction which made inference difficult in their setting. Due to the typical-case formulation studied here, a different proof strategy is needed to show this hardness result.

3.3 One-way functions and pseudorandom generators

Proving the hardness of a computational problem C is often done by reducing from an **NP**-hard problem like 3SAT to problem C , thereby showing that if C could be solved in polynomial time, $\mathbf{P} = \mathbf{NP}$. However, problems like 3SAT are stated in terms of behavior on all, and thus worst-case, inputs. Therefore, to prove the hardness of a computational problem on typical-case inputs, it is preferable to reduce to a hardness conjecture stated directly in terms of typical-case behavior.

In this report I will reference two such conjectures, which are known to be related. The first conjecture is the existence of *one-way functions*, functions which can be efficiently computed, but not efficiently inverted for the majority of inputs. It is widely thought likely that such functions exist [6], as there are a number of functions like multiplication of prime numbers, for which no inversion algorithms are known which are efficient in the typical case. Such functions are widely used in public-key cryptography.

Definition 8. (Arora and Barak Def. 9.4) A **one-way function** f is a function $f : \{0, 1\}^* \rightarrow \{0, 1\}^*$ such that f can be computed in polynomial time, and for every probabilistic polynomial time algorithm A , for every $c \in \mathbb{R}$,

$$n^c \cdot \Pr_{x \sim \text{Uniform}(\{0, 1\}^n)} [A(f(x), 1^n) \in f^{-1}(f(x))] \xrightarrow{n \rightarrow \infty} 0 \quad (2)$$

where $f^{-1}(f(x))$ is the set $\{x' \in \{0, 1\}^* : f(x') = f(x)\}$.

The second conjecture is the existence of *secure pseudorandom generators*, which are functions which take a short random seed and expand it into a long string which is indistinguishable from a truly random string. This is known to follow from the existence of one-way functions, and this weaker conjecture is all that is needed for the main result of this report. Specifically, I consider PRGs which are secure against all polynomial-time adversaries, once the strings being generated are sufficiently long.

Definition 9. (Weakened variant of Arora and Barak Def. 9.8) A *secure pseudorandom generator* (PRG) is a polynomial time computable function $\{0, 1\}^* \rightarrow \{0, 1\}^*$ such that $|G(x)| = |l(|x|)|$ for some function $l : \mathbb{N} \rightarrow \mathbb{N}$, such that for every probabilistic polynomial time algorithm A ,

$$\left| \Pr_{s \sim \text{Uniform}(\{0, 1\}^n)} [A(G(s)) = 1] - \Pr_{y \sim \text{Uniform}(\{0, 1\}^{l(n)})} [A(y) = 1] \right| \xrightarrow{n \rightarrow \infty} 0 \quad (3)$$

The function l is called the stretch of the PRG.

This definition is weaker than the definition given in Arora and Barak [6], as it only requires the expression in eq. 3 to tend to zero, rather than to zero at a superpolynomial rate, as in eq. 2, and as in Arora and Barak's definition of a secure PRG. The reduction in this report shows that the existence of a secure PRG, as defined here, with superlinear stretch implies the non-existence of typical-case inference algorithms with certain tolerances. The existence of secure PRGs as defined in Arora and Barak is a strictly stronger statement than the existence of secure PRGs as defined here, and their PRGs are instances of secure PRGs as defined here.

It is known that the existence of one-way functions implies the existence of secure pseudorandom generators. Thus, to the extent that it is believed that one-way functions exist, it is also believed that secure pseudorandom generators exist.

Proposition 1. (Arora and Barak Thm. 9.9) If there exists a one-way function, then there exists a secure pseudorandom generator with stretch $l(n) = n^c$.

The reason I include the superpolynomial convergence rate in Def. 8 is to ensure that this theorem goes through. As far as I am aware, it may be that given my weaker definition of a secure PRG, a weaker definition of one-way function would also suffice to imply the existence of such PRGs, but I have not verified this.

4 Main result

Theorem 2. If there exists a polynomial-time typical-case additive PMF approximation algorithm A with tolerances $\epsilon < \frac{1}{2}$, $\rho < \frac{1}{4}$, then there does not exist a secure pseudorandom generator with an invertible stretch function l such that $\lim_{n \rightarrow \infty} [l(n) - n] = \infty$. This holds whether A is deterministic or probabilistic. This holds even if we only require A to be capable of typical-case inference in Bayesian networks with in-degree ≤ 2 and with probability values in the set $\{0, 1/2, 1\}$.

Corollary 3. If there exists a polynomial-time typical-case additive PMF approximation algorithm with tolerances $\epsilon < \frac{1}{2}$, $\rho < \frac{1}{4}$, then one-way functions do not exist.

The corollary follows because if one-way functions exist, by Proposition 1, there exists a secure PRG with stretch $l(n) = n^2$.

These results suggest that it is unlikely that there exists a universal typical-case inference algorithm which succeeds on a $3/4$ typical set of observations in all Bayesian networks.

Proof. (Theorem 2.) For contradiction, suppose A is a deterministic polynomial-time typical-case additive PMF approximation algorithm with tolerances $\epsilon < \frac{1}{2}$ and $\rho < \frac{1}{4}$. (The case where A is probabilistic will be covered at the end of the proof.) Let $\delta > 0$ be s.t. $\epsilon < \frac{1}{2} - \delta$. Suppose G is a secure PRG with invertible stretch function $l(n)$ s.t. $\lim_{n \rightarrow \infty} [l(n) - n] = \infty$. Note that we can assume l is upper-bounded by a polynomial in n , because if l grows superpolynomially we can obtain a new PRG G' which outputs the first n^2 bits of the output of G . Under the assumption that l is upper-bounded by a polynomial, note that l^{-1} is computable in $\text{poly}(n)$ time, because given m , one can iterate through the values $n = 1, 2, \dots, m$ to find when $l(n) = m$.

Proof outline. I will construct a polynomial time Turing machine B which can distinguish the distributions $\text{Uniform}(\{0, 1\}^n) \circ G^{-1}$ and $\text{Uniform}(\{0, 1\}^{l(n)})$ for all $n > N$, where N is a constant. This contradicts that G is a secure PRG.

Given input $y \in \{0, 1\}^{l(n)}$, Turing machine B will guess whether it was sampled from the pseudo-random generator, or from the uniform on $\{0, 1\}^{|y|}$. If a string y is given such that $|y|$ is not in the range of l , then B can immediately reject. Otherwise, B will compute $n = l^{-1}(|y|)$ to find the seed length needed for G to produce y .

Second, B will construct the description of an inference schema $I = (P, V, E, X, Y)$ where $|Y| = l(n)$ and $|X| = 1$, such that if $n > N$,

$$y' \in G(\{0, 1\}^n) \implies P(X = 1 | Y = y') > 1 - \delta \quad (*)$$

and

$$y' \notin G(\{0, 1\}^n) \implies P(X = 1 | Y = y') = 0 \quad (**)$$

Third, B will compute $a \leftarrow A(I, 1, y)$. If $a < 1/2$, B will output 0; otherwise B will output 1. I will first show that this implies that on typical y values, B exactly decides whether $y \in \text{range}(G)$. Specifically, there is a set $\mathcal{Y}_{\text{good}} \subseteq \{0, 1\}^{l(n)}$ with $P(\mathcal{Y}_{\text{good}}) > 3/4$ such that $y \in \mathcal{Y}_{\text{good}} \implies B(y) = 1_{y \in \text{range}(G)}$. Using this, I will then show that, because we can guarantee $P(y \in \mathcal{Y}_{\text{good}}) > 3/4 = 1 - \rho$, this implies that there exists a $\gamma > 0$ such that

$$\left| \Pr_{s \sim \text{Uniform}(\{0, 1\}^n)} [B(G(s)) = 1] - \Pr_{\bar{y} \sim \text{Uniform}(\{0, 1\}^{l(n)})} [B(\bar{y}) = 1] \right| \geq \gamma \quad (***)$$

for all $n > N$, contradicting that G is a secure pseudorandom generator.

I now proceed to complete the proof by (1) proving that B can construct an inference schema with properties $(*)$ and $(**)$, and (2) proving that $(***)$ holds.

Construction of the inference schema. Let (P, V, E) be the Bayesian network which implements the following process.

1. P samples $s = s_1 s_2 \dots s_n$ uniformly at random from $\{0, 1\}^n$.
2. P samples $\bar{y} = \bar{y}_1 \bar{y}_2 \dots \bar{y}_{l(n)}$ uniformly at random from $\{0, 1\}^{l(n)}$.
3. P samples x uniformly at random from $\{0, 1\}$.
4. If $x = 0$, P computes $y = \bar{y}$. Otherwise, P computes $y = G(s)$.

The set X is defined to contain only x , and Y is defined to contain the $l(n)$ bits of y . This yields schema (P, V, E, X, Y) .

Clearly, writing a Bayesian network which does the sampling and multiplexing steps can be done in polynomial time. (This just requires adding a bunch of binary variables to the vertex set V , and adding a probability table for each to P specifying that each of these variables is uniformly distributed.) Writing out the part of the Bayesian network which computes $G(s)$ can be done in polynomial time in n using the Cook-Levin reduction for writing a circuit which implements the same behavior as the Turing machine for G , on inputs of length n . Each and gate, or gate, or not gate in the circuit can be represented by a binary variable in the Bayesian network with one or two parent variables, and with a deterministic probability table that given a parent assignment, outputs the correct value for the gate.

Analysis of the posteriors (proof of $(*)$ and $()$).** First, note that if $y \notin \text{range}(G)$,

$$P(X = 1|Y = y) = \frac{P(X = 1, Y = y)}{P(Y = y)} = \frac{0}{P(Y = y)} = 0$$

because it is impossible for P to sample $x = 1$ yet output a value not in the range of G . This proves $(**)$.

Now, say $y \in \text{range}(G)$. Let $m = |G^{-1}(y)|$, the number of seeds in $\{0, 1\}^n$ that get mapped to y . Then

$$P(X = 1|Y = y) = \frac{P(X = 1, Y = y)}{P(X = 0, Y = y) + P(X = 1, Y = y)} = \frac{\frac{1}{2} \frac{m}{2^n}}{\frac{1}{2} \frac{1}{2^{l(n)}} + \frac{1}{2} \frac{m}{2^n}}$$

The numerator here is $P(X = 1, Y = y) = \frac{1}{2} \frac{m}{2^n}$ because $X = 1$ with probability $\frac{1}{2}$, and given that $X = 1$, we will have $Y = y$ iff the seed s is in $G^{-1}(y)$, which occurs with probability $\frac{m}{2^n}$. The denominator contains the term $P(X = 0, Y = y) = \frac{1}{2} \frac{1}{2^{l(n)}}$ because there is a $1/2$ probability that $X = 0$, and if it is 0, $Y = y$ only if one of the $2^{l(n)}$ equally probable values in $\{0, 1\}^{l(n)}$ is chosen as \bar{y} .

Simplifying this,

$$P(X = 1|Y = y) = \frac{m/2^n}{1/2^{l(n)} + m/2^n}$$

so

$$P(X = 0|Y = y) = \frac{1/2^{l(n)}}{1/2^{l(n)} + m/2^n} = \frac{1}{1 + m2^{l(n)-n}}$$

Since $2^{l(n)-n} > 0$, this expression decreases in m . Since $y \in \text{range}(G)$, $m \geq 1$. Thus this expression is maximized when $m = 1$. Therefore, for all m ,

$$P(X = 0|Y = y) \leq \frac{1}{1 + 2^{l(n)-n}} < \frac{1}{2^{l(n)-n}}$$

Let N_1 be an integer such that $n > N_1 \implies l(n) - n > \log(1/\delta)$. Then $n > N_1 \implies 2^{l(n)-n} > 1/\delta$, so for all such n ,

$$P(X = 0|Y = y) < \delta$$

and thus

$$P(X = 1|Y = y) > 1 - \delta$$

This proves (*).

Proof that on $\mathcal{Y}_{\text{good}}$, B exactly identifies $\text{range}(G)$. Let $\mathcal{Y}_{\text{good}}$ be the good set of observations for inference schema I constructed above, and assignment $x = 1$, as defined in Definition 6. Then if the given y satisfies $y \in \mathcal{Y}_{\text{good}}$, $|A(I, x, y) - P(X = x|Y = y)| < \epsilon < \frac{1}{2} - \delta$. Thus if $y \notin \text{range}(G)$, by (**), $|A(I, x, y) - 0| < \frac{1}{2} - \delta$ so $A(I, x, y) < 1/2$. And if $y \in \text{range}(G)$ but $A(I, x, y) < 1/2$, by (*) we would have $|P(X = 1|Y = y) - A(I, x, y)| \geq |(1 - \delta) - 1/2| \geq 1/2 - \delta > \epsilon$, so it must be the case that $A(I, x, y) > 1/2$. That is,

$$y \in \mathcal{Y}_{\text{good}} \implies [(y \notin \text{range}(G) \implies A(I, x, y) < 1/2) \wedge (y \in \text{range}(G) \implies A(I, x, y) > 1/2)]$$

Thus for $y \in \mathcal{Y}_{\text{good}}$, algorithm B exactly decides whether $y \in \text{range}(G)$: $y \in \mathcal{Y}_{\text{good}} \implies B(y) = 1_{y \in \text{range}(G)}$.

Proof that B identifies outputs from the PRG with nontrivial probability (proof of (*)).** In this section I will write \Pr_s as shorthand for $\Pr_{s \sim \text{Uniform}(\{0,1\}^n)}$ and $\Pr_{\bar{y}}$ as shorthand for $\Pr_{\bar{y} \sim \text{Uniform}(\{0,1\}^{l(n)})}$.

The goal of this section is to establish that there exists a $\gamma > 0$ such that

$$|\Pr_s[B(G(s)) = 1] - \Pr_{\bar{y}}[B(\bar{y}) = 1]| \geq \gamma$$

Let $p_s := \Pr_s[B(G(s)) = 1]$ and $p_{\bar{y}} := \Pr_{\bar{y}}[B(\bar{y}) = 1]$. Let $p_{s,1} := \Pr_s[B(G(s)) = 1 \wedge G(s) \in \mathcal{Y}_{\text{good}}]$, $p_{s,2} := \Pr_s[B(G(s)) = 1 \wedge G(s) \notin \mathcal{Y}_{\text{good}}]$, $p_{\bar{y},1} = \Pr_{\bar{y}}[B(\bar{y}) = 1 \wedge \bar{y} \in \mathcal{Y}_{\text{good}}]$, and $p_{\bar{y},2} = \Pr_{\bar{y}}[B(\bar{y}) = 1 \wedge \bar{y} \notin \mathcal{Y}_{\text{good}}]$. Observe that $p_s = p_{s,1} + p_{s,2}$ and $p_{\bar{y}} = p_{\bar{y},1} + p_{\bar{y},2}$. By the triangle inequality,

$$|p_{s,1} + p_{s,2} - p_{\bar{y},1} - p_{\bar{y},2}| + |p_{\bar{y},2} - p_{s,2}| \geq |p_{s,1} + p_{s,2} - p_{\bar{y},1} - p_{\bar{y},2} + p_{\bar{y},2} - p_{s,2}|$$

Canceling terms on the right hand side, we get

$$|p_{s,1} + p_{s,2} - p_{\bar{y},1} - p_{\bar{y},2}| + |p_{\bar{y},2} - p_{s,2}| \geq |p_{s,1} - p_{\bar{y},1}|$$

Plugging in $p_s = p_{s,1} + p_{s,2}$ and $p_{\bar{y}} = p_{\bar{y},1} + p_{\bar{y},2}$, and rearranging, we get

$$|p_s - p_{\bar{y}}| = |p_{s,1} + p_{s,2} - p_{\bar{y},1} - p_{\bar{y},2}| \geq |p_{s,1} - p_{\bar{y},1}| - |p_{s,2} - p_{\bar{y},2}| \quad (4)$$

We have

$$\begin{aligned} |p_{s,2} - p_{\bar{y},2}| &= \left| \Pr_s[G(s) \notin \mathcal{Y}_{\text{good}}] \Pr_s[B(G(s)) = 1|G(s) \notin \mathcal{Y}_{\text{good}}] - \Pr_{\bar{y}}[\bar{y} \notin \mathcal{Y}_{\text{good}}] \Pr_{\bar{y}}[B(\bar{y}) = 1|\bar{y} \notin \mathcal{Y}_{\text{good}}] \right| \\ &\leq \max(\Pr_s[G(s) \notin \mathcal{Y}_{\text{good}}], \Pr_{\bar{y}}[\bar{y} \notin \mathcal{Y}_{\text{good}}]) \end{aligned} \quad (5)$$

We also have

$$\begin{aligned} p_{s,1} - p_{\bar{y},1} &= \Pr_s[B(G(s)) = 1 \wedge G(s) \in \mathcal{Y}_{\text{good}}] - \Pr_{\bar{y}}[B(\bar{y}) = 1 \wedge \bar{y} \in \mathcal{Y}_{\text{good}}] \\ &= \Pr_s[G(s) \in \text{range}(G) \wedge G(s) \in \mathcal{Y}_{\text{good}}] - \Pr_{\bar{y}}[\bar{y} \in \text{range}(G) \wedge \bar{y} \in \mathcal{Y}_{\text{good}}] \\ &= \Pr_s[G(s) \in \mathcal{Y}_{\text{good}}] - \Pr_{\bar{y}}[\bar{y} \in \text{range}(G) \wedge \bar{y} \in \mathcal{Y}_{\text{good}}] \\ &\geq \Pr_s[G(s) \in \mathcal{Y}_{\text{good}}] - \Pr_{\bar{y}}[\bar{y} \in \text{range}(G)] = \Pr_s[G(s) \in \mathcal{Y}_{\text{good}}] - 2^n/2^{l(n)} \end{aligned} \quad (6)$$

The second equality here follows from the fact established above, that for $y \in \mathcal{Y}_{\text{good}}$, algorithm B exactly decides whether $y \in \text{range}(G)$.

We now need to bound $\Pr_s[G(s) \in \mathcal{Y}_{\text{good}}]$ and $\Pr_{\bar{y}}[\bar{y} \in \mathcal{Y}_{\text{good}}]$. Observe that

$$P(\mathcal{Y}_{\text{good}}) = \frac{1}{2} \Pr_s[G(s) \in \mathcal{Y}_{\text{good}}] + \frac{1}{2} \Pr_{\bar{y}}[\bar{y} \in \mathcal{Y}_{\text{good}}]$$

because under the model P , y is set equal to $G(s)$ half the time and equal to \bar{y} the other half of the time. Since each of these probability terms are no greater than 1, we have

$$\Pr_s[G(s) \in \mathcal{Y}_{\text{good}}] \geq 2P(\mathcal{Y}_{\text{good}}) - 1; \quad \Pr_{\bar{y}}[\bar{y} \in \mathcal{Y}_{\text{good}}] \geq 2P(\mathcal{Y}_{\text{good}}) - 1 \quad (7)$$

By subtracting each side of these inequalities from 1, we also obtain

$$\Pr_s[G(s) \notin \mathcal{Y}_{\text{good}}] \leq 2 - 2P(\mathcal{Y}_{\text{good}}); \quad \Pr_{\bar{y}}[\bar{y} \notin \mathcal{Y}_{\text{good}}] \leq 2 - 2P(\mathcal{Y}_{\text{good}}) \quad (8)$$

Combining equations 5 and 8, we get

$$|p_{s,2} - p_{\bar{y},2}| \leq 2 - 2P(\mathcal{Y}_{\text{good}}) \quad (9)$$

and combining equations 6 and 7, we get

$$|p_{s,1} - p_{\bar{y},1}| \geq 2P(\mathcal{Y}_{\text{good}}) - 1 - 2^{n-l(n)} \quad (10)$$

Plugging in these bounds to equation 4, we obtain

$$|p_s - p_{\bar{y}}| \geq 2P(\mathcal{Y}_{\text{good}}) - 1 - 2^{n-l(n)} - (2 - 2P(\mathcal{Y}_{\text{good}})) = 4P(\mathcal{Y}_{\text{good}}) - 3 - 2^{n-l(n)} \quad (11)$$

Thus,

$$P(\mathcal{Y}_{\text{good}}) \geq \frac{3}{4} + \frac{1}{4 \cdot 2^{l(n)-n}} + \frac{1}{4} \gamma \implies |p_s - p_{\bar{y}}| \geq \gamma$$

Since $l(n) - n \rightarrow \infty$, the term $\frac{1}{4 \cdot 2^{l(n)-n}} \rightarrow 0$. Thus since $P(\mathcal{Y}_{\text{good}}) > \frac{3}{4}$, there exists a value $\gamma > 0$ and an N_2 such that for all $n > N_2$,

$$|\Pr_s[B(G(s)) = 1] - \Pr_{\bar{y}}[B(\bar{y}) = 1]| \geq \gamma$$

Finally, it suffices to take $N = \max(N_1, N_2)$.

Extending the proof to the case where the inference algorithm A is probabilistic. This extension is conceptually straightforward; we simply need to track several additional terms in the analysis, to control the probability that algorithm B fails to indicate whether $y \in \text{range}(G)$ for a value $y \in \mathcal{Y}_{\text{good}}$. Since we can upper-bound this probability arbitrarily tightly by boosting the probabilistic version of algorithm B through replication, the ultimate conclusion is the same as before.

Say that the inference algorithm $A(I, x, y)$ is probabilistic. Write $A(I, x, y, r)$ to denote a run of algorithm A that uses internal random bits r . By saying A is a probabilistic typical-case additive-error PMF approximation algorithm, we mean that for $y \in \mathcal{Y}_{\text{good}}$, we have

$$\Pr_r[|A(I, x, y, r) - P(X = x|Y = y)| \geq \epsilon] < 1/3$$

In the procedure described above, the only place where algorithm B uses algorithm A is to check whether $P(X = 1|Y = y)$ is greater than $1/2$. I will write $B(y, r)$ for a call to B that uses random bitstring r when calling A . Our preceding analysis shows that for $y \in \mathcal{Y}_{\text{good}}$,

$$|A(I, 1, y, r) - P(X = 1|Y = y)| < \epsilon \implies B(y, r) = 1_{y \in \text{range}(G)}$$

Thus for $y \in \mathcal{Y}_{\text{good}}$,

$$\Pr_r[B(y, r) \neq 1_{y \in \text{range}(G)}] < 1/3$$

By calling B $O(k)$ times with different random bitstrings r , and taking the most frequent output, we can obtain an algorithm B^* , consuming a longer random bitstring r^* , where for $y \in \mathcal{Y}_{\text{good}}$,

$$\Pr_{r^*}[B^*(y, r^*) \neq 1_{y \in \text{range}(G)}] < 1/2^k$$

Except for the final section dedicated to proving statement (**), the preceding proof remains essentially unchanged. We must, however, redo part of the analysis in the final section, replacing the old deterministic algorithm B with our new stochastic algorithm B^* . Much of the analysis remains unchanged, except that the equations should be rewritten so that probability expressions of the form $\Pr_s[\dots]$ and $\Pr_{\bar{y}}[\dots]$ which include a $B(\dots)$ term inside the brackets, are replaced by expressions of the form $\Pr_{s, r^*}[\dots]$ and $\Pr_{\bar{y}, r^*}[\dots]$, with the $B(\dots)$ terms replaced by $B^*(\dots, r^*)$. The first place where the analysis diverges nontrivially is in equation 6.

To do the new version of the analysis, I will introduce the following notation. For $y \in \mathcal{Y}_{\text{good}}$, let $\mathcal{R}_{\text{good}}^y$ be the set of bitstrings r^* such that $B^*(y, r^*) = 1_{y \in \text{range}(G)}$, and for $y \notin \mathcal{Y}_{\text{good}}$ let $\mathcal{R}_{\text{good}}^y$ be the set of all bitstrings r^* .

Using this, in place of Equation 6, we now have

$$\begin{aligned} p_{s,1} - p_{\bar{y},1} &= \Pr_{s, r^*}[B^*(G(s), r^*) = 1 \wedge G(s) \in \mathcal{Y}_{\text{good}}] - \Pr_{\bar{y}, r^*}[B^*(\bar{y}, r^*) = 1 \wedge \bar{y} \in \mathcal{Y}_{\text{good}}] \\ &= \Pr_{s, r^*}[G(s) \in \text{range}(G) \wedge G(s) \in \mathcal{Y}_{\text{good}} \wedge r^* \in \mathcal{R}_{\text{good}}^{G(s)}] + \Pr_{s, r^*}[B^*(G(s), r^*) = 1 \wedge G(s) \in \mathcal{Y}_{\text{good}} \wedge r^* \notin \mathcal{R}_{\text{good}}^{G(s)}] \\ &\quad - (\Pr_{\bar{y}, r^*}[\bar{y} \in \text{range}(G) \wedge \bar{y} \in \mathcal{Y}_{\text{good}} \wedge r^* \in \mathcal{R}_{\text{good}}^{\bar{y}}] + \Pr_{\bar{y}, r^*}[B^*(\bar{y}, r^*) = 1 \wedge \bar{y} \in \mathcal{Y}_{\text{good}} \wedge r^* \notin \mathcal{R}_{\text{good}}^{\bar{y}}]) \\ &\geq \Pr_{s, r^*}[G(s) \in \text{range}(G) \wedge G(s) \in \mathcal{Y}_{\text{good}} \wedge r^* \in \mathcal{R}_{\text{good}}^{G(s)}] \\ &\quad - (\Pr_{\bar{y}, r^*}[\bar{y} \in \text{range}(G) \wedge \bar{y} \in \mathcal{Y}_{\text{good}} \wedge r^* \in \mathcal{R}_{\text{good}}^{\bar{y}}] + \Pr_{\bar{y}, r^*}[B^*(\bar{y}, r^*) = 1 \wedge \bar{y} \in \mathcal{Y}_{\text{good}} \wedge r^* \notin \mathcal{R}_{\text{good}}^{\bar{y}}]) \\ &\geq (1 - \frac{1}{2^k}) \Pr_{s, r^*}[G(s) \in \text{range}(G) \wedge G(s) \in \mathcal{Y}_{\text{good}}] - \Pr_{\bar{y}, r^*}[\bar{y} \in \text{range}(G)] - \Pr_{\bar{y}, r^*}[r^* \notin \mathcal{R}_{\text{good}}^{\bar{y}}] \\ &\geq (1 - \frac{1}{2^k}) \Pr_s[G(s) \in \mathcal{Y}_{\text{good}}] - 2^n/2^{l(n)} - 1/2^k \quad (12) \end{aligned}$$

Rewriting inequality 10 using this new bound yields

$$|p_{s,1} - p_{\bar{y},1}| \geq (1 - \frac{1}{2^k})(2P(\mathcal{Y}_{\text{good}}) - 1) - 2^{n-l(n)} - 1/2^k \quad (13)$$

Equation 11 becomes

$$\begin{aligned} |p_s - p_{\bar{y}}| &\geq (1 - \frac{1}{2^k})(2P(\mathcal{Y}_{\text{good}}) - 1) - 2^{n-l(n)} - 1/2^k - (2 - 2P(\mathcal{Y}_{\text{good}})) \\ &= 4P(\mathcal{Y}_{\text{good}}) - 3 - 2^{n-l(n)} - \frac{2P(\mathcal{Y}_{\text{good}})}{2^k} \geq 4P(\mathcal{Y}_{\text{good}}) - 3 - 2^{n-l(n)} - \frac{2}{2^k} \quad (14) \end{aligned}$$

where the last inequality follow from the fact that $P(\mathcal{Y}_{\text{good}}) \leq 1$. Therefore

$$P(\mathcal{Y}_{\text{good}}) \geq \frac{3}{4} + \frac{1}{4}(\frac{1}{2^{l(n)-n}} + \frac{2}{2^k}) + \frac{1}{4}\gamma \implies |p_s - p_{\bar{y}}| \geq \gamma \quad (15)$$

Since we in fact have $P(\mathcal{Y}_{\text{good}}) > \frac{3}{4}$ with strict inequality, letting $\eta := P(\mathcal{Y}_{\text{good}}) - \frac{3}{4}$, we have $\eta > 0$. Choose N_2, k large enough so that $n \geq N_2 \implies \frac{1}{2^{l(n)-n}} + \frac{2}{2^k} < \eta/2$, and set $\gamma = 2\eta$. Because ρ is a property of algorithm A , this choice of k is independent of the length of the string y input to algorithm B^* , so k is a constant for the purposes of characterizing the complexity of algorithm B^* . Thus setting k to have this value does not affect the assessment that B^* is a polynomial-time algorithm. Then (15) holds for this γ , so there exists $\gamma > 0$ such that for all $n > N := \max(N_1, N_2)$,

$$|\Pr_{s, r^*}[B^*(G(s), r^*) = 1] - \Pr_{\bar{y}, r^*}[B^*(\bar{y}, r^*) = 1]| \geq \gamma$$

□

5 Related work

Marginal probability computation and counting. In this paper, I focused on the problem of posterior inference, that of computing values relating to a posterior distribution $P(X|Y = y)$. A related problem is that of computing the marginal probability $P(Y = y)$. This problem is widely studied in complexity theory, as for a certain class of probabilistic models, it is essentially isomorphic to counting the number of witnesses y such that a polynomial-time Turing machine M outputs $M(x, y) = 1$ on a given x . This counting problem defines the complexity class $\#\mathbf{P}$, the class of functions on from x to $|\{y : M(x, y) = 1\}|$ for polynomial time Turing Machines M . This class is a widely studied object in complexity theory. The isomorphism in question involves considering a probability distribution on $|y| + 1$ binary values. The first $|y|$ bits are a witness y for the Turing machine, which are posited to each have an independent Bernoulli($1/2$) distribution. The final bit z is a flag bit, which is set to 1 if $M(x, y) = 1$ and 0 otherwise (for a given x). The marginal probability $P(z = 1)$ is exactly $1/2^{|y|}$ times the number of witnesses y such that $M(x, y) = 1$. Note that while this assumes every witness y to x has an equal length, this assumption requires no loss of generality for time-bounded Turing machines. For an overview of counting complexity, see Chapter 17 of Arora and Barak’s textbook [6].

PMF computation of and sampling from marginal probability distributions. As $P(X = x|Y = y) := P(X = x, Y = y)/P(Y = y)$, the ability to exactly compute the marginal probability values $P(X = x, Y = y)$ and $P(Y = y)$ implies the ability to compute posterior PMF value $P(X = x|Y = y)$. Likewise, the ability to compute each of these values up to a multiplicative approximation error implies the ability to compute posterior PMF values up to the same error. A related question is whether marginal probability computation enables sampling from posterior distributions. In partial answer to this, Jerrum, Valiant, and Vazirani showed in 1986 that in important families of probabilistic models, the ability to sample approximately from a distribution is equally hard up to a polynomial factor as stochastically computing marginal probability values [7]. In general, the ability to compute marginal probabilities is \mathbf{PP} -complete. However, there are special cases where it has been shown that this problem is tractable. Recently, work by Akmal and Williams and Till Tantau showed that computing the marginal probability that a k -CNF formula is satisfied, when the boolean variables are uniformly distributed, is computable in polynomial time [8, 9].

Tractability results. In addition to past hardness results on the complexity of posterior inference in Bayesian networks [3, 1], several tractability results are also known. In 1997, Dagum and Luby showed that probabilistic polynomial time inference is possible in any inference schema with two restrictions: the number of observation variables is upper bounded by a constant, and the minimum probability value occurring in the Bayesian network is strictly greater than 0 [10]. For a survey of such parametrized complexity results, and several new results, see [11]. Restrictions like those in [10] in effect limit the amount of information that the observation variables can carry about the query variables, and I conjecture that in the typical-case setting considered here, these restrictions can in fact be stated explicitly as an upper bound on the Shannon mutual information between the query and observed variables. Another setting in which some practitioners believe posterior inference is likely be tractable is in probabilistic models with bounded coupling between different latent variables [12]. It is indeed known that approximate marginal probability computation, and approximate sampling, are tractable in such settings where conditions as in the Lovasz local lemma hold [13].

Complexity of probabilistic inference in continuous models. Bayesian networks are broad family of probabilistic models in which to study probabilistic inference. The main restriction of such models is that all the random variables are discrete with finite domains. Discreteness is a core features of computational models like Turing machines and circuits, and in this sense Bayesian networks are a natural probabilistic analog to these models. Many statisticians also consider probabilistic inference in classes of probabilistic models on continuous random variables. Computations in these models can be represented in Turing machines, using algorithms which compute numerical approximations to quantities like posterior probability density function or cumulative distribution function values, and receive as input parameters controlling the requested precision for the computation. Ackerman, Freer, and Roy have shown that under standard formulations of precision-controlled real-valued computation in Turing machines, there are probabilistic models in which posterior cumulative distribution function and probability mass function computation is noncomputable

[14]. They also show that under many restrictions on the model family, often obeyed by statistical models on continuous random variables, inference is computable. For a survey of models of real-valued computation using Turing machines in general, see [15].

References

- [1] Paul Dagum and Michael Luby. “Approximating probabilistic inference in Bayesian belief networks is NP-hard”. In: *Artificial Intelligence* 60.1 (1993), pp. 141–153.
- [2] Vikash Kumar Mansinghka et al. “Natively probabilistic computation”. PhD thesis. Citeseer, 2009.
- [3] Gregory F Cooper. “The computational complexity of probabilistic inference using Bayesian belief networks”. In: *Artificial Intelligence* 42.2-3 (1990), pp. 393–405.
- [4] Tomoyuki Yamakami. “Polynomial time samplable distributions”. In: *Journal of Complexity* 15.4 (1999), pp. 557–574.
- [5] Alex Edmonds. *Concepts of Computationally Efficient Samplability*. Master’s degree thesis, University of Toronto. 2017.
- [6] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.
- [7] Mark R Jerrum, Leslie G Valiant, and Vijay V Vazirani. “Random generation of combinatorial structures from a uniform distribution”. In: *Theoretical computer science* 43 (1986), pp. 169–188.
- [8] Shyan Akmal and Ryan Williams. “majority-3sat (and related problems) in polynomial time”. In: *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*. IEEE. 2022, pp. 1033–1043.
- [9] Till Tantau. “On the satisfaction probability of k-CNF formulas”. In: *Proceedings of the 37th Computational Complexity Conference. CCC ’22*. Philadelphia, Pennsylvania: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2022. ISBN: 9783959772419. DOI: 10 . 4230 / LIPIcs . CCC . 2022 . 2. URL: <https://doi.org/10.4230/LIPIcs.CCC.2022.2>.
- [10] Paul Dagum and Michael Luby. “An optimal approximation algorithm for Bayesian inference”. In: *Artificial Intelligence* 93.1-2 (1997), pp. 1–27.
- [11] Johan Kwisthout. “Approximate inference in Bayesian networks: Parameterized complexity results”. In: *International Journal of Approximate Reasoning* 93 (2018), pp. 119–131.
- [12] Cameron E Freer, Vikash K Mansinghka, and Daniel M Roy. “When are probabilistic programs probably computationally tractable?” In: *NIPS Workshop on Monte Carlo Methods for Modern Applications*. 2010.
- [13] Ankur Moitra. “Approximate counting, the Lovász local lemma, and inference in graphical models”. In: *Journal of the ACM (JACM)* 66.2 (2019), pp. 1–25.
- [14] Nathanael L Ackerman, Cameron E Freer, and Daniel M Roy. “On the computability of conditional probability”. In: *Journal of the ACM (JACM)* 66.3 (2019), pp. 1–40.
- [15] Klaus Weihrauch. *Computable analysis: an introduction*. Springer Science & Business Media, 2000.