Efficient Inference for Complex Queries on Complex Distributions

Lili Dworkin
University of Pennsylvania

Michael Kearns
University of Pennsylvania

Lirong Xia
Rensselaer Polytechnic Institute

Abstract

We consider problems of approximate inference in which the query of interest is given by a complex formula (such as a formula in disjunctive formal form (DNF)) over a joint distribution given by a graphical model. We give a general reduction showing that (approximate) marginal inference for a class of distributions yields approximate inference for DNF queries, and extend our techniques to accommodate even more complex queries, and dense graphical models with variational inference, under certain conditions. Our results unify and generalize classical inference techniques (which are generally restricted to simple marginal queries) and approximate counting methods such as those introduced by Karp, Luby and Madras (which are generally restricted to product distributions).

1 Introduction

There is a large body of work on performing marginal inference, both exact and approximate, on complex probability distributions. One line of research considers exact inference on sparse (tree or tree-like) graphical models (Pearl, 1988). A second line of work leverages variational methods to perform approximate inference on densely connected, highly cyclical networks (Jordan et al., 1999; Jaakkola and Jordan, 1999). Meanwhile, a separate body of literature has considered the problem of answering more complex queries, such as the satisfaction probability of a formula in disjunctive normal form (DNF), but over simpler distributions (Karp et al., 1989). In this work, we unify and generalize these lines of work in order to provide efficient approximate inference techniques that can accommodate complex distributions and complex queries simultaneously.

In this context, "query" refers to the computation of the satisfaction probability of a logical formula over the joint distribution. Thus standard marginal inference would correspond to a simple conjunction of literals, specifying the desired setting of each marginal variable of interest. Our methods apply to both DNF formulas and *threshold DNFs*, which are DNF-like formulas but have exponentially large traditional DNF representations. Our algorithm takes as input a formula F and a probability distribution P (given as a graphical model) over a set of variable assignments. In polynomial time, the algorithm outputs an arbitrarily accurate multiplicative approximation of the satisfaction probability of F with high probability.

Our algorithm generalizes the Karp-Luby-Madras (KLM) algorithm (1989) for approximating the cardinality of set unions. The approach consists of three main steps: 1) calculating the probability that a term of the formula is satisfied; 2) generating a satisfying assignment for a term; and 3) checking whether an assignment satisfies a term. If the types of F and P allow us to define an efficient subroutine for each step, then we can apply the algorithm.

We first show that if F is a DNF formula, the approach reduces to performing exact or multiplicatively approximate marginal inference over P. Thus, if P belongs to a class of distributions on which we can efficiently answer marginal queries, then we immediately have an efficient algorithm for answering more complex DNF queries as well. We can therefore take advantage of the large body of existing work on efficient inference to apply our result in a variety of settings.

As a motivating example for DNF queries on complex distributions, consider a hedge fund that maintains multiple portfolios of risky bond or mortgage investments, where the greatest risk in each investment is default and subsequent loss of all capital. Because the failure of any one of its portfolios poses significant risk to the entire fund (typically due to leverage), we are interested in estimating the probability that one or more

Appearing in Proceedings of the 17^{th} International Conference on Artificial Intelligence and Statistics (AISTATS) 2014, Reykjavik, Iceland. JMLR: W&CP volume 33.

	Conjunction	**DNF**	**Threshold DNF**
Product	exact - trivial	(exact - #P-complete) mult. approx. - KLM	(exact - #P-complete[1]) mult. approx. - **new result**
Tree	exact - Pearl's algorithm	(exact - #P-complete) mult. approx. - **new result**	(exact - #P-complete) mult. approx - **new result**
Polytree	exact - Pearl's algorithm	(exact - #P-complete) mult. approx - **new result**	open problem
Two-Layer Parametric	additive approx. - Kearns-Saul	additive approx. - **new result**	open problem

Table 1: Summary of new and prior results. The rows describe classes of joint distributions of increasing complexity, while the columns describe query or formula classes of increasing complexity. Our new results expand the frontier of complex distribution-query pairs.

portfolios fail in a given period. We thus might model the individual assets by boolean variables indicating default, and the failure of a portfolio by a conjunction of defaults in its constituent holdings. The overall failure for the fund is then the disjunction (DNF) of these portfolio terms. Note that different portfolios (terms) may overlap in their holdings (variables), and furthermore the joint distribution over default events may have high-order correlations due to common influencing factors such as the health of an economic sector or region (which can be represented as additional, non-asset variables).

We can also apply the algorithm when F is a *threshold DNF* and P is given as a directed tree Bayesian network. Threshold DNFs would require exponentially many terms if expressed as a standard DNF, but can be efficiently tested for satisfaction on a given assignment. Among other motivations, in the hedge fund example above, threshold DNFs allow us to naturally model the fact that portfolios may have different amounts invested in different assets, and that the threshold for portfolio failure may be less than the default of all of its constituent holdings, neither of which can be succinctly captured by a standard DNF.

Our last result considers the case in which F is a monotone DNF and P is a dense cyclical network. Exact inference is infeasible in such networks, and the best known (variational) algorithms for approximate inference do not have multiplicative guarantees. Thus, it is not possible to apply the approach above. Instead, we use large deviation methods to factor the network into product distributions on which our algorithm can then be applied. The formal guarantees for the resulting approximations are weaker but nontrivial, and provably tight in certain interesting cases. We also provide some experimental evaluation of this approach.

[1]This result follows from the fact that standard DNFs are a subclass of threshold DNFs (see Section 2).

1.1 Related Work

Many combinations of formula and distribution types have been handled in previous work. The algorithmic framework we propose allows us to solve the problem for several new combinations. A comparison of our work to existing results is shown in Table 1.

As mentioned previously, when F is a conjunction, the problem is equivalent to marginal inference. Pearl's algorithm (1988) efficiently computes exact marginal inference over polytrees. In a general Bayesian network, even approximating marginal inference to within $\epsilon < 1/2$ additive error is NP-hard (Dagum and Luby, 1993). However, particular types of networks permit specialized algorithms with better performance. For instance, variational methods have been used successfully on certain classes of dense cyclic networks (Jordan et al., 1999; Jaakkola and Jordan, 1999). A line of work initiated by Kearns and Saul (1998; 1999; Ng and Jordan 2000) provides upper and lower bounds on marginal probabilities with additive error that diminishes rapidly with the size of the network.

When P is the uniform distribution, we can compute the satisfaction probability of F by counting the number of satisfying assignments of F. This problem is known as *model counting*, or #SAT, and is #P-complete. When P is a product distribution, then the problem is equivalent to *weighted model counting* (Sang et al., 2005). The original KLM algorithm applies when P is a product distribution and F is a DNF. To our knowledge, our results are the first to consider cases in which F is a DNF (or DNF variant) and P is not a product distribution.

2 Preliminaries

Let $X = \{x_1, \ldots, x_n\}$ be a set of n binary variables. Let $V = \{0,1\}^n$ denote the set of all possible variable assignments, and let P denote the probability distri-

bution over V. For an assignment $v \in V$, let v_x denote the value of a variable x, and let $P(v)$ denote the probability of v under the distribution P.

A DNF formula consists of a set of k terms, $T_1, \ldots, T_k$, each of which consists of a set of literals. A literal is a tuple $(x, b) \in X \times \{0, 1\}$ where the first element indicates a variable and the second element indicates its desired value. For instance, $(x, 1)$ denotes the literal x and $(x, 0)$ denotes $\neg x$. Given an assignment $v \in V$, a literal (x, b) is satisfied if $v_x = b$. A term T_i of a DNF is satisfied if all literals in T_i are satisfied, and the entire DNF is satisfied if at least one term is satisfied.

A *threshold DNF* consists of a set of k terms, an integer weight $w(l)$ for each literal in each term, and an integer threshold q_i for each term. A term T_i of a threshold DNF is satisfied if the weighted sum of the satisfied literals in T_i is at least q_i. Again, the entire threshold DNF is satisfied if at least one term is satisfied. A standard DNF can be expressed as a threshold DNF in which the thresholds are set equal to the term lengths and all weights are set to one. While a threshold DNF has a succinct representation and can be easily checked for satisfaction on a given assignment v, in general it would take an exponentially large DNF formula to represent a threshold DNF. Proof of this claim is provided in the appendix.

Let $S(T)$ denote the set of assignments that satisfy the term T, and let $P(T)$ denote the probability that T is satisfied. Define $S(F)$ and $P(F)$ similarly with respect to the formula F.

3 General Algorithm

Our first main contribution is an algorithmic framework that outputs a multiplicative approximation of $P(F)$ with high probability. The algorithm takes as input a description of a distribution P, a formula F, and additional parameters α, ϵ, and δ. The parameters α and ϵ control the accuracy of the approximation, and δ controls the confidence with which we achieve this accuracy. The algorithm is *efficient*, by which we mean it runs in time polynomial in P, F, and the inverses of these parameters. Generally, such an algorithm is known as an FPRAS (fully polynomial-time randomized approximation scheme).

For now, we assume the existence of three polynomial-time subroutines, defined below.[2]

[2]We note that these subroutines are assumed to provide the desired behavior only on the unconditioned distribution P. If we make the stronger assumption that they provide the desired behavior conditioned on evidence E, we can also approximate $P(F|E)$. This is directly applicable, for instance, if P is represented by a polytree.

Algorithm 1 Generalized KLM

procedure GeneralizedKLM($\alpha, \epsilon, \delta, P, F$)
  $L \leftarrow \frac{3k}{\alpha^2} \ln(\frac{2}{\delta})$
  $\hat{P}(T_i) \leftarrow \text{TermProb}(P, T_i, \epsilon) \quad \forall i \in [k]$
  $D \leftarrow \sum_{i=1}^{k} \hat{P}(T_i)$
  for $t = 1$ to L **do**
    Choose T_i with probability $\frac{\hat{P}(T_i)}{D}$
    $v \leftarrow \text{GenAssign}(P, T_i, \epsilon)$
    for $j = 1$ to k **do**
      if SatAssign(T_j, v) **then**
        $t(v) \leftarrow j$
        break
    if $t(v) = i$ **then**
      $Z_t \leftarrow 1$
    else
      $Z_t \leftarrow 0$
  $Z = \frac{1}{L} \sum_{t=1}^{L} Z_t$
  return ZD

Definition 1 (Subroutines)**.**

- **TermProb** calculates the approximate probability that a term T is satisfied. The inputs are P, T, and ϵ, and the output is $\hat{P}(T)$, where $(1-\epsilon)P(T) \leq \hat{P}(T) \leq (1+\epsilon)P(T)$.
- **GenAssign** generates a satisfying assignment for a term T. The inputs are P, T, and ϵ, and the output is $v \in V$ such that 1) v satisfies T and 2) the probability that v was generated is equal to $\hat{P}(v)/\hat{P}(T)$, where $\hat{P}(v)$ denotes a $1 \pm \epsilon$ approximation to $P(v)$, and $\hat{P}(T)$ is the same approximation used in TermProb.
- **SatAssign** determines whether an assignment v satisfies a term T. The inputs are v and T, and the output is True in the case of satisfaction and False otherwise.

The framework, shown in Algorithm 1, generalizes the importance sampling scheme of the KLM algorithm. In each iteration, rather than sampling an assignment directly, we first choose a term T_i from the formula F with probability $\hat{P}(T_i)/D$, where $D = \sum_{i=1}^{k} \hat{P}(T_i)$. We calculate $\hat{P}(T_i)$ for $i \in \{1, \ldots, k\}$ by calling TermProb. Next we call GenAssign to generate an assignment v that satisfies T_i. We then find the first term T_j in F that v satisfies by making the necessary calls to SatAssign. If $T_i = T_j$, we set a random variable equal to one, and otherwise, we set it to zero. The intuition behind this step is to correct overcounting an assignment that satisfies multiple terms. Finally, the algorithm outputs ZD, where Z is the average value of the random variable.

As shown in the following theorem, the Generalized

KLM algorithm is efficient, and outputs a multiplicative approximation to $P(F)$ with high probability.

Theorem 1. *Let* TermProb*,* GenAssign*, and* SatAssign *be polynomial-time subroutines as described in Definition 1. Then* Generalized KLM *is efficient, and for any α, ϵ, δ, the algorithm outputs an approximation $\hat{P}(F)$ such that, with probability $1-\delta$,*

$$(1-\alpha)(1-\epsilon)P(F) \leq \hat{P}(F) \leq (1+\alpha)(1+\epsilon)P(F).$$

Proof. The proof closely follows the original proof of Karp, Luby, and Madras (1989). To begin, it is easy to see that the algorithm runs in polynomial time. On line 2, we set the number of iterations L equal to $(3k/\alpha^2)\ln(2/\delta)$. Because we assume that each subroutine runs in polynomial time, each iteration requires only polynomial time as well.

To analyze the correctness of the algorithm, first we will show that the expected value of the output $\mathrm{E}[Z]D$ is a multiplicative approximation of $P(F)$. Then we will use a Chernoff bound to show that the actual output ZD is close to its expectation.

For any trial t, $E[Z_t] = \sum_{(i,v)\in[n]\times V} \Pr[T_i, v \text{ chosen}] \times \mathbb{I}(Z_t = 1)$. Recall that $Z_t = 1$ when $t(v) = i$. Because $t(v)$ is uniquely defined for each v, the number of pairs (i, v) for which $i = t(v)$ is exactly equal to the number of satisfying assignments of F, i.e. $|S(F)|$. Furthermore, we can rewrite $\Pr[T_i, v \text{ chosen}]$ as $\Pr[T_i \text{ chosen}] \times \Pr[v \text{ chosen} \mid T_i \text{ chosen}]$. By the definition of TermProb, $\Pr[T_i \text{ chosen}] = \hat{P}(T_i)/D$, and by the definition of GenAssign, $\Pr[v \text{ chosen} \mid T_i \text{ chosen}] = \hat{P}(v)/\hat{P}(T_i)$. Because we require the $\hat{P}(T_i)$ approximations used in TermProb and GenAssign to be the same, we have

$$\mathrm{E}[Z_t] = \sum_{v\in S(F)} \frac{\hat{P}(T_i)}{D} \cdot \frac{\hat{P}(v)}{\hat{P}(T_i)} = \sum_{v\in S(F)} \frac{\hat{P}(v)}{D}.$$

Note that $\mathrm{E}[Z] = \mathrm{E}[1/L \sum_{t=1}^{L} Z_t] = \mathrm{E}[Z_t]$, and recall $(1-\epsilon)P(v) \leq \hat{P}(v) \leq (1+\epsilon)P(v)$. So we have

$$(1-\epsilon) \sum_{v\in S(F)} \frac{P(v)}{D} \leq \mathrm{E}[Z] \leq (1+\epsilon) \sum_{v\in S(F)} \frac{P(v)}{D}.$$

Because $P(F) = \sum_{v\in S(F)} P(v)$, it follows that $(1-\epsilon)P(F) \leq \mathrm{E}[Z]D \leq (1+\epsilon)P(F)$.

We apply a Chernoff bound to finish the proof. Let $\eta = P(F)/D$. So $(1-\epsilon)\eta \leq E[Z] \leq (1+\epsilon)\eta$. For any $\alpha > 0$, $\Pr[Z > (1+\alpha)(1+\epsilon)\eta] \leq \exp(-\alpha^2\eta L/3)$, and we can similarly bound $\Pr[Z < (1-\alpha)(1-\epsilon)\eta]$. The minimum value of η occurs when all T_i are the same, in which case $P(F) = P(T_i)$ and $D = kP(T_i)$, so $\eta = 1/k$. Thus, $\eta \geq 1/k$, and $\exp(-\alpha^2\eta L/3) \leq \exp(-\alpha^2 L/3k)$. Because the algorithm sets $L = (3k/\alpha^2)\ln(2/\delta)$, we have $\exp(-\alpha^2 L/3k) = \exp(-\ln(2/\delta)) = \delta/2$. So, with probability at least $1-\delta$,

$$\begin{aligned}(1-\alpha)(1-\epsilon)\eta \leq Z \quad &\leq (1+\alpha)(1+\epsilon)\eta \\ (1-\alpha)(1-\epsilon)P(F) \leq ZD &\leq (1+\alpha)(1+\epsilon)P(F).\end{aligned}$$

Because ZD is the output of the algorithm, the proof is complete. □

We have therefore shown that the Generalized KLM algorithm is both efficient and correct, as long as the three subroutines are defined appropriately. In the following sections, we examine particular settings of P and F for which we can provide these subroutines.

4 DNF Formulas

We first apply Generalized KLM to the setting in which F is a DNF. As we will show, if P belongs to a class of networks on which we can perform exact or multiplicatively approximate inference, then it is possible to define the three necessary subroutines. Thus, we will have reduced the problem of calculating the probability of DNF satisfaction to the problem of performing approximate inference.

We begin by assuming the existence of an oracle that can efficiently compute $\Pr[Y = y]$ within a $1 \pm \epsilon$ factor for any set of variables $Y \subseteq X$ and any instantiation $y \in \{0,1\}^{|Y|}$ of these variables. In the following paragraphs, we use this oracle to define the three subroutines TermProb, GenAssign, and SatAssign. These procedures are polynomial in the number of variables in F, the length of the longest term in F, and the running time of marginal inference over P.

TermProb Subroutine The probability that a term of a DNF is satisfied is exactly equal to a marginal probability over the variables of the term. Formally, $P(T) = \Pr[x = b, \forall (x, b) \in T]$. So $P(T)$ (or a multiplicative ϵ-approximation $\hat{P}(T)$) can be calculated by the marginal inference oracle described above.

GenAssign Subroutine The GenAssign subroutine, shown in Algorithm 2, sequentially generates values for variables in X. To start, all variables that appear in the term T are set so that their literal is satisfied. Then, given a set C consisting of pairs of variables and values, we choose the value of a variable x that does not appear in C to be 1 with probability $\hat{P}(C \wedge (x, 1))/\hat{P}(C)$. To calculate these probabilities, we use the marginal inference oracle. We then add the chosen assignment to C, and repeat.

The number of required iterations is $|X|$. Thus, as long as the marginal inference oracle is efficient, this

subroutine will be as well. The first step ensures that the generated assignment will satisfy v, and the following lemma shows that v is generated with the correct probability. The proof can be found in the appendix.

Lemma 1. *The* GenAssign *subroutine for DNF formulas generates an assignment $v \in V$ with probability $\hat{P}(v)/\hat{P}(T)$.*

Algorithm 2 GenAssign for DNFs

```
procedure GenAssign(P, T, ε)
    v_x ← b   ∀(x, b) ∈ T
    C ← T
    for x ∈ {x ∈ X | (x, 0) ∉ C ∧ (x, 1) ∉ C} do
        P̂(C ∧ (x, 1)) ← TermProb(P, C ∧ (x, 1), ε)
        with probability P̂(C∧(x,1)) / P̂(C)
            v_x ← 1
            C ← C ∧ (x, 1)
        else
            v_x ← 0
            C ← C ∧ (x, 0)
    return v
```

SatAssign Subroutine The SatAssign subroutine returns True if $v_x = b, \forall (x, b) \in T$. The running time of the subroutine is linear in $|T|$, and its correctness follows from the definition of DNF satisfaction.

Now that we have described correct and efficient subroutines, the following theorem follows immediately.

Theorem 2. *Let F be a DNF and let P be a distribution over which marginal inference is efficient. Then there is an efficient algorithm that for any α, ϵ, δ, with probability at least $1 - \delta$, will compute an approximation $\hat{P}(F)$ such that $(1 - \alpha)(1 - \epsilon)P(F) \leq \hat{P}(F) \leq (1 + \alpha)(1 + \epsilon)P(F)$.*

Note that if exact marginal inference is efficient on P, we can set $\epsilon = 0$ in the above theorem.

Thus, given a class of distributions on which it is possible to answer simple marginal queries, Generalized KLM makes it possible to answer more complex DNF queries as well. In this way, the algorithm can be viewed as a black box to "boost" the complexity of queries that can be answered on a distribution.

5 Threshold DNFs

In the previous section, we restricted our attention to DNF formulas. We now apply Generalized KLM to the setting in which F is a threshold DNF. Recall that these formulas assign an integer weight $w(l)$ to each literal l, and a term T_i is satisfied if the weighted sum of satisfied literals in the term is at least q_i.

We make one important restriction in this section. We no longer consider any distribution over which inference is efficient, and instead restrict our attention to distributions given as a subset of the nodes of a directed tree Bayes net. This restriction is necessary because a marginal inference oracle cannot be used to calculate the satisfaction probability of a threshold DNF term, and is therefore insufficient for defining the three subroutines. The subroutines instead depend on a dynamic program that operates over a directed tree. To simplify the analysis, we assume the tree is binary, and that P is given over the leaves of the tree only. As we explain in the appendix, both assumptions can be removed. We proceed to define the three subroutines TermProb, GenAssign, and SatAssign.

TermProb Subroutine For a set of literals S, let $W(S) = \sum_{l \in S} w(l)$. For a term T and a subset $S \subseteq T$, let $V^T(S) \subseteq V$ denote the set of variable assignments in which $x = b, \forall (x, b) \in S$, and $x = 1 - b, \forall x \in T - S$. Let T be a term with threshold q. The satisfaction probability of T is $P(T) = \sum_{i=q}^{W(T)} \sum_{\{S \subseteq T | W(S) = i\}} \sum_{v \in V^T(S)} P(v)$.

Let r be the root of the tree. Let u_P denote the parent of a node u, and let u_L and u_R denote the left and right children of u. Let $T_u \subseteq T$ denote the set of literals in the term T whose variables appear as leaves in the subtree rooted at u. Let $\Pr[Q_u^T(q)]$ denote the probability that $W(T_u) = q$. Then $\Pr[Q_u^T(q)] = \sum_{\{S \subseteq T_u | W(S) = q\}} \sum_{v \in V^{T_u}(S)} P(v)$, so $P(T) = \sum_{i=q}^{W(T)} \Pr[Q_r^T(i)]$. We give a dynamic program to compute $\Pr[Q_u^T(i) \mid u_P = z]$ for any u, i, and z, which therefore allows us to calculate $P(T)$ as well.

If $i > W(T_u)$, then $\Pr[Q_u^T(i)] = 0$. If u is a leaf, then there is some $l = (u, b) \in T_u$, and we have

$$\Pr[Q_u^T(i) \mid u_P = z] = \begin{cases} \Pr[u = b \mid u_P = z] & \text{if } i = w(l) > 0 \\ 1 & \text{if } i = w(l) = 0 \\ \Pr[u = 1 - b \mid u_P = z] & \text{if } i = 0 \wedge w(l) > 0 \\ 0 & \text{otherwise.} \end{cases}$$

In the general case,

$$\Pr[Q_u^T(i) \mid u_P = z] = \sum_{b \in \{0,1\}} \Pr[u = b \wedge Q_u^T(i) \mid u_P = z]$$

$$= \sum_{b \in \{0,1\}} \Pr[u = b \mid u_P = z] \times \Pr[Q_u^T(i) \mid u = b]$$

$$= \sum_{b \in \{0,1\}} \Big[\Pr[u = b \mid u_P = z] \times \sum_{\{x,y | x+y=i\}} \Pr[Q_{u_L}^T(x) \mid u = b] \Pr[Q_{u_R}^T(y) \mid u = b] \Big].$$

The last line is the dynamic program, and its correctness is evident from the derivation. To analyze the running time, note that we must compute $\Pr[Q_u^T(i) \mid u_P = z]$ for every possible value of u, i, and z, and each computation requires summing over all pairs x, y such that $x + y = i$. The number of choices for u is equal to the number of nodes in the tree, which we denote $|V|$. The number of choices for i is bounded by $|W(T)|$, and the number of choices for z is two. Finally, the number of possible x, y pairs is bounded by $|W(T)|^2$. So the running time is $O(|V||W(T)|^3)$, which is pseudo-polynomial with respect to $|W(T)|$.

GenAssign Subroutine As before, this subroutine sequentially generates values for variables in X. We start at the top of the tree and work downward, setting values for each internal node as we go. At the root r, we choose a value for the weighted sum of satisfied literals in the term T. We choose this value to be $i \in \{q, \ldots, W(T)\}$ with probability equal to

$$\frac{\Pr[Q_r^T(i)]}{\sum_{i=q}^{W(T)} \Pr[Q_r^T(i)]}.$$

We then choose the value of r to be $b \in \{0, 1\}$ with probability equal to

$$\frac{\Pr[r = b] \Pr[Q_r^T(i) \mid r = b]}{\Pr[Q_r^T(i)]}.$$

Inductively, once we have set the value of an internal node u to be b, we choose the values for the weighted sums of satisfied literals in the subtrees rooted at u_L and u_R. We choose these values to be x, y such that $x + y = i$ with probability equal to

$$\frac{\Pr[Q_{u_L}^T(x) \mid u = b] \Pr[Q_{u_R}^T(y) \mid u = b]}{\Pr[Q_u^T(i) \mid u = b]}.$$

Then we choose the value for u_L to be b' with probability equal to

$$\frac{\Pr[u_L = b' \mid u = b] \Pr[Q_{u_L}^T(x) \mid u_L = b']}{\Pr[Q_{u_L}^T(x) \mid u = b]}.$$

We choose the value for u_R similarly, and we repeat until we have set values for all variables.

The subroutine visits each node once, and all necessary probabilities can be calculated efficiently using the TermProb dynamic program described previously. Thus, the GenAssign subroutine is efficient as well. Its correctness follows from the two lemmas below, for which proofs can be found in the appendix.

Lemma 2. *The* GenAssign *subroutine for threshold DNFs generates an assignment $v \in V$ with probability $P(v)/P(T)$.*

Lemma 3. *The* GenAssign *subroutine for threshold DNFs generates an assignment $v \in V$ such that v satisfies T.*

SatAssign Subroutine The subroutine for SatAssign returns True if $\sum_{\{l=(x,b)\in T \mid v_x = b\}} w(l) \geq q$. The running time of the subroutine is linear in $|T|$, and its correctness follows from the definition of threshold DNF satisfaction.

As before, the following theorem follows immediately. Because TermProb and GenAssign work with exact probabilities, we can set $\epsilon = 0$, and therefore remove this parameter.

Theorem 3. *Let F be a threshold DNF and let P be a directed tree Bayes net. Then there is an efficient algorithm (pseudo-polynomial in the weights of F) that for any α, δ, with probability at least $1 - \delta$, will compute an approximation $\hat{P}(F)$ such that $(1-\alpha)P(F) \leq \hat{P}(F) \leq (1+\alpha)P(F)$.*

In the next section, we return to standard DNFs, and show that these queries can also be answered efficiently on a much different class of networks.

6 Dense Networks

In Section 4, we provided a very general and efficient reduction that takes a class of distributions for which (approximate) marginal (i.e. conjunctive) inference is tractable, and yields an efficient algorithm for approximate inference on more complex DNF queries. The main application of this reduction today would be to the polytree or junction tree algorithms for exact marginal inference, and to various tree decomposition methods for more general graphs. However, future advances in approximate inference might yield more targets for the reduction.

There is another class of distributions for which approximate marginal inference is also thought to be tractable, but in a less formal and rigorous sense than for tree-like graphs. These are dense, cyclical networks with high in-degrees and parametric CPTs acting on weighted sums of parents, for which variational inference algorithms have enjoyed wide interest and success (Jordan et al., 1999; Jaakkola and Jordan, 1999). In this section we show that our methods can also be applied to these networks, even if the formal guarantees are (necessarily) weaker.

Our point of departure is the approach initiated by Kearns and Saul (1998; 1999; Ng and Jordan 2000), one of the few works providing provable upper and lower bounds on marginal probabilities for dense networks. Their basic insight is that in such dense networks, the probability that all incoming weighted sums are near their means is high, and conditioned on this event, the network factors into simple product distributions providing rigorous upper and lower bounds on marginal inferences. In order to adapt these meth-

ods to (monotone[3]) DNF queries, we consider three schemes of increasing sophistication. All three can be viewed as variational algorithms (the variational parameters arising from the trade-off between the tightness of concentration around the means and the failure probability for this event). The first two can be shown to provide provable upper bounds on the true probabilities, and all three can be shown to have exponentially small error in interesting cases. For ease of exposition we focus on two-layered networks (Kearns and Saul, 1998), but the same methods can be applied to the multi-layer case (Kearns and Saul, 1999).

6.1 Preliminaries

The formal definition of a two-layer network is as follows. We have a set X of n input binary variables and a set Y of m output binary variables. For every input x_i and output y_j, there is a real-valued weight θ_{ij}. Let $\Pr[x_i = 1] = q_i$ and $\Pr[y_j = 1] = p_j = f(\sum_{i=1}^n \theta_{ij} x_i)$, where f is a smooth transfer function with bounded derivatives. Let P be the true distribution over the output variables. Let P^μ be the product distribution defined by $p_j = f(\mu_j)$, and let $P^+(\epsilon)$ and $P^-(\epsilon)$ be the product distributions defined by $p_j = f(\mu_j + \epsilon)$ and $p_j = f(\mu_j - \epsilon)$, respectively.

Let $\mu_j = \sum_{i=1}^n \theta_{ij} q_i$ be the mean of the incoming weighted sum for variable y_j. We use Hoeffding's inequality to show that the weighted sums are tightly concentrated around their means. For any j and $\epsilon > 0$,

$$\Pr\left[\frac{1}{n}\left|\left(\sum_{i=1}^n \theta_{ij} x_i - \mu_j\right)\right| > \epsilon\right] \leq 2\exp(-2n\epsilon^2).$$

Taking a union bound, we let $\delta(\epsilon) = 2m\exp(-2n\epsilon^2)$ be an upper bound on the probability of the failure event in which some incoming weighted sum falls more than ϵ from its mean.

6.2 A Simple Upper Bound

We will use factorization arguments to derive various upper bounds on $P(F)$. Let π denote the event in which all weighted sums lie within ϵ of their means, so $P(\pi) = 1 - \delta(\epsilon)$. We write $P(F \mid \pi)$ to denote the probability that F is satisfied conditioned on the event π. Because F is a monotone DNF, $P^-(F) \leq P(F \mid \pi) \leq P^+(F)$. In other words, conditioned on the event π, we can factorize P into upper and lower bound product distributions P^+ and P^-. But without the conditioning, we can only claim $0 \leq P(F) \leq 1$. Therefore, $P(F) \leq (1 - \delta(\epsilon))P^+(F) + \delta(\epsilon)$. Because we cannot calculate $P^+(F)$ directly, we instead use a union bound over the sum of the satisfaction probabilities of each term in the DNF. Thus, our first upper bound on $P(F)$ is

$$P(F) \leq (1 - \delta(\epsilon))\left(\sum_{i=1}^k \prod_{y_j \in T_i} f(\mu_j + \epsilon)\right) + \delta(\epsilon). \quad (1)$$

Note that the summation in this upper bound is over a small number (k) of products that we expect to be exponentially small in the interesting case where the terms T_i are long (if there are short terms in a DNF over a product distribution, the probability of satisfaction can typically be approximated well by naive frequency sampling). The additive term $\delta(\epsilon)$ is also decreasing exponentially with the number n of input-layer variables. So overall we can obtain a provable upper bound that is exponentially small.

More precisely, consider the nontrivial special case in which all terms are of the same length l and all weighted sums have the same mean μ. By applying a Taylor expansion and the binomial theorem to the upper bound above, it can be shown that

$$P(F) \leq kp^l \sum_{j=0}^{l} (l\gamma\epsilon/p)^j + \delta(\epsilon)$$

where $p = f(\mu)$ and γ is a constant involving the first and second derivatives of f. We can make the sum $\sum_{j=0}^{l}(l\gamma\epsilon/p)^j$ smaller by choosing ϵ smaller, but this must be balanced against larger $\delta(\epsilon)$. For instance, choosing $\epsilon = p/(l\gamma)$ yields an upper bound of $klp^l + 2m\exp(-2np^2/(l^2\gamma^2))$. Viewing p and γ as constants, the first term is exponentially small in the term length, which again is large (e.g. linear in m) in the interesting case. The second term becomes exponentially small in the input layer size n, as long as n is $\Omega(l^2)$. So for the case of DNFs with long terms, and two-layer networks where the input layer is sufficiently larger than the output layer, we get an exponentially small upper bound. Obviously other trade-offs are possible.

6.3 More Sophisticated Algorithms

To obtain the closed-form upper bound in the section above, we used the fact that the probability of satisfying a DNF must be less than the union bound over the probabilities of satisfying each term. Of course, this may overestimate the true probability. Rather than applying a union bound to $P^+(F)$, we can instead estimate $P^+(F)$ directly using Generalized KLM. We then turn this into a rigorous upper bound by dividing appropriately to compensate for the $1 \pm \alpha$ error allowed in the algorithm.

[3]We can also handle DNF formulas in which each variable always appears with the same sign, i.e. either negated or unnegated.

More formally, let $\hat{P}^+(F)$ be the output of Generalized KLM on inputs P^+ and F. By Theorem 2, for any α, δ', with probability $1-\delta'$, $(1-\alpha)P^+(F) \leq \hat{P}^+(F) \leq (1+\alpha)P^+(F)$. Therefore, a second upper bound is

$$P(F) \leq (1-\delta(\epsilon))(\hat{P}^+(F)/(1-\alpha)) + \delta(\epsilon). \qquad (2)$$

For α sufficiently small, the term $\hat{P}^+(F)/(1-\alpha)$ will be (potentially much) smaller than the union bound $\sum_{i=1}^{k} \prod_{y_j \in T_i} f(\mu_j + \epsilon)$ from our first upper bound. So theoretically this second approach enjoys at least the same analyses we provided for the union bound approach, but would be expected to be better in practice, since it directly estimates $P^+(F)$ rather than using a known overestimate.

Yet a third approach is to apply Generalized KLM not to the upper product distribution P^+, but to the mean product distribution P^μ. Assuming all output variables fall close to their means, P^μ is a good approximation of P, in which case $P^\mu(F)$ is a good approximation of $P(F)$. We can therefore approximate $P(F)$ by the output of running Generalized KLM on P^μ. This approximation is no longer guaranteed to be an upper bound, but its accuracy can be analyzed along the same lines as before (details omitted).

6.4 Experimental Results

Above we have provided three different approximation schemes for monotone DNFs on dense networks. The accuracy of each scheme depends on multiple factors, including the tuning of the parameters ϵ and δ, as well as the size of F and P. It is natural to question how these approximations perform in practice under various assumptions. Thus, we ran the following experiments. We randomly generated two-layer networks with $m = 25$ outputs, a sigmoidal transfer function, and a varying number of inputs n. For each network, the scaled weights were set to $\theta_{ij} = \tau_{ij}/N$, where each τ_{ij} was chosen from a normal distribution with zero mean and unit variance. Then a random DNF with $k \in [5, 10]$ terms and $l \in [10, 15]$ variables per term was generated.

In each trial, we first approximated $P(F)$ by naive frequency sampling 10^6 times from the distribution over the output variables. Let $\hat{P}(F)$ denote the value of this approximation. We then computed the three approximations we have discussed: the upper bound from Equation 1, which we call *union bound*; the upper bound from Equation 2, which we call *KLM-upper*; and the output of running Generalized KLM on P^μ, which we call *KLM-mean*. Both union bound and KLM-upper require a tradeoff between ϵ and δ, so we varied ϵ over the set $\{10^{-i} \mid i = 1, \ldots, 5\}$ and chose the value that resulted in the smallest bound. Both

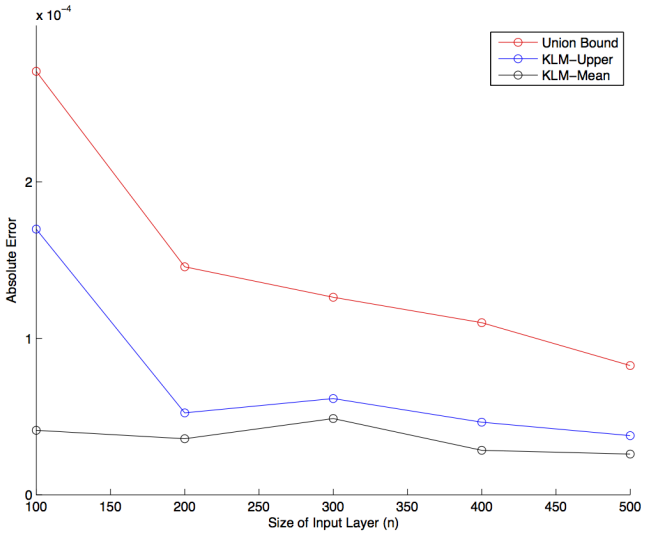

Figure 1: Average absolute errors of three approximation schemes as a function of the size of the input layer n.

KLM-upper and KLM-mean use the Generalized KLM algorithm and therefore require us to set the parameters α and δ'. For KLM-upper, we set $\alpha = 0.01$ and $\delta' = 0.1$. For KLM-mean, we set $\alpha = 0.1$ and $\delta' = 0.1$.

The absolute errors of each approximation are plotted as a function of the size of the input layer n in Figure 1. Each data point represents an average over 25 trials. As expected, since their estimates directly involve a trade-off between ϵ and $\delta(\epsilon)$ as in the theoretical analysis, the error of the union bound and KLM-upper methods decreases with increasing input layer size. The KLM-mean approximation consistently has the best performance, followed by KLM-upper.

The algorithm for KLM-mean is also significantly more efficient than a frequency sampling approach. On average, it took about 3.3 seconds to compute KLM-mean, compared to 266.7 seconds to compute an approximation of comparable accuracy using frequency sampling.

7 Future Work

We have presented an algorithmic framework that approximates $P(F)$ whenever P and F allow us to define three necessary subroutines. We have identified several new combinations of P and F for which these subroutines exist, but there are likely other combinations as well. In particular, we leave as an open question the case in which F is a threshold DNF and P is a polytree or dense cyclic network. Additionally, it would be interesting to consider other types of formulas with compact DNF representations that have more complex term satisfaction rules.

References

Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141 – 153, 1993.

Tommi S. Jaakkola and Michael I. Jordan. Variational probabilistic inference and the QMR-DT network. *Journal of Artificial Intelligence Research*, 10:291–322, 1999.

Michael I. Jordan, Zoubin Ghahramani, Tommi S. Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. In *Machine Learning*, volume 37, pages 183–233. The MIT Press, 1999.

Richard M. Karp, Michael Luby, and Neal Madras. Monte-Carlo approximation algorithms for enumeration problems. *Journal of Algorithms*, 10(3):429–448, 1989.

Michael Kearns and Lawrence Saul. Large deviation methods for approximate probabilistic inference. In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, pages 311–319. Morgan Kaufmann Publishers Inc., 1998.

Michael Kearns and Lawrence Saul. Inference in multilayer networks via large deviation bounds. In *Advances in Neural Information Processing Systems*, volume 11, pages 260–266. The MIT Press, 1999.

Andrew Y. Ng and Michael I. Jordan. Approximate inference algorithms for two-layer Bayesian networks. In *Advances in Neural Information Processing Systems*, volume 12, pages 533–539. The MIT Press, 2000.

Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

Tian Sang, Paul Bearne, and Henry Kautz. Performing Bayesian inference by weighted model counting. In *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 475–481. AAAI Press, 2005.

Technical Appendix

Threshold DNFs An exponentially large standard DNF is required to represent a threshold DNF. Consider a threshold DNF with just a single term containing all n of the binary variables $x_1, \ldots, x_n$, and let the threshold value be $\frac{n}{2}$, with each variable having equal weight. Consider any standard DNF for this $\frac{n}{2}$ threshold function, and let T be any term in it. If T has fewer than $\frac{n}{2}$ variables appearing unnegated, then T has a satisfying assignment with fewer than $\frac{n}{2}$ bits on, which is a contradiction. Furthermore, if T has any variables appearing negated, then these variables can be removed, and all previously satisfying assignments for T will remain satisfying. Thus, the smallest T will contain only unnegated variables, and at least $\frac{n}{2}$ of them. The DNF formula must contain such a term for every possible subset of $\frac{n}{2}$ bits, of which there are exponentially many.

Section 5 Assumptions In Section 5 we assume that the tree is binary and that the distribution P is given over the leaves only. To remove the first assumption, rather than summing over $\{x, y \mid x + y = i\}$, we use another dynamic program that calculates the probability that exactly i amount of weight is distributed among the children of u. The approach is similar to calculating the probability that exactly k out of n biased coins come up heads. To remove the second assumption, we check at each node u (not just the leaves) whether u corresponds to a literal in T, and if so, we make a case analysis similar to the one currently restricted to our base case.

Lemma 1. The GenAssign subroutine for DNF formulas generates an assignment $v \in V$ with probability $\hat{P}(v)/\hat{P}(T)$.

Proof. Let $x_1, \ldots, x_m$ denote variables in the order as they appear in the loop from lines 5 to 12, where $m = n - |T|$. Let v be the assignment generated by GenAssign(P, T, ϵ). Let $l_i = (x_i, v_{x_i})$. The probability that v was generated is

$$\frac{\hat{P}(T \wedge l_1)}{\hat{P}(T)} \times \frac{\hat{P}(T \wedge l_1 \wedge l_2)}{\hat{P}(T \wedge l_1)} \times \ldots \times \frac{\hat{P}(v)}{\hat{P}(T \wedge l_1, \ldots, l_m)}.$$

After cancelling terms, we have $\hat{P}(v)/\hat{P}(T)$. □

Lemma 2. The GenAssign subroutine for threshold DNFs generates an assignment $v \in V$ with probability $P(v)/P(T)$.

Proof. The probability that an assignment v is generated is:

$$\frac{\Pr[Q_r^T(i)]}{\sum_{i=q}^{W(T)} \Pr[Q_r^T(i)]} \times \frac{\Pr[r = v_r] \Pr[Q_r^T(i) \mid r = v_r]}{\Pr[Q_r^T(i)]} \times \frac{\Pr[Q_{r_L}^T(x) \mid r = v_r] \Pr[Q_{r_R}^T(y) \mid r = v_r]}{\Pr[Q_r^T(i) \mid r = v_r]} \times \frac{\Pr[r_L = v_{r_L} \mid r = v_r] \Pr[Q_{r_L}^T(x) \mid r_L = v_{r_L}]}{\Pr[Q_{r_L}^T(x) \mid r = v_r]} \times \frac{\Pr[r_R = v_{r_R} \mid r = v_r] \Pr[Q_{r_R}^T(y) \mid r_R = v_{r_R}]}{\Pr[Q_{r_R}^T(y) \mid r = v_r]} \times \ldots$$

After cancelling terms, we have

$$\frac{\Pr[r = v_r] \Pr[r_L = v_{r_L} \mid r = v_r] \Pr[r_R = v_{r_R} \mid r = v_r] \ldots}{\sum_{i=q}^{W(T)} \Pr[Q_r^T(i)]} = \frac{\Pr[r = v_r \wedge r_L = v_{r_L} \wedge r_R = v_{r_R} \ldots]}{P(T)} = \frac{P(v)}{P(T)}.$$

□

Lemma 3. The *GenAssign* subroutine for threshold DNFs generates an assignment $v \in V$ such that v satisfies T.

Proof. It suffices to prove that the process generates an assignment v in which the weighted sum of satisfied literals is equal to i, where $i \geq q$ is the value we chose in the first step. We will use induction to prove the following more general claim. For any internal node u, if we have chosen the value for the weighted sum of satisfied literals in u's subtree to be i, then the generated assignment will meet this requirement.

Suppose we are at a leaf node u where $l = (u, 1) \in T$ and we have chosen $u_P = b$. We then choose i with probability proportional to $\Pr[Q_u^T(i)]$. The only values for i which correspond to a nonzero probability are $w(l)$ and 0. In order for u's literal to be satisfied, u's value must be 1. So if we have chosen i to be $w(l)$, then we should choose u's value to be 1 with probability 1. According to the subroutine, we choose the value for u to be 1 with probability equal to

$$\frac{\Pr[u = 1 \mid u_P = b] \Pr[Q_u^T(w(l)) \mid u = 1]}{\Pr[Q_u^T(w(l)) \mid u_P = b]} = \frac{\Pr[u = 1 \mid u_P = b](1)}{\Pr[u = 1 \mid u_P = b]} = 1.$$

On the other hand, if we have chosen i to be 0, then we should choose u's value to be 0 with probability 1. According to the process, we choose the value for u to be 0 with probability equal to

$$\frac{\Pr[u = 0 \mid u_P = b] \Pr[Q_u^T(0) \mid u = 0]}{\Pr[Q_u^T(0) \mid u_P = b]}$$
$$= \frac{\Pr[u = 0 \mid u_P = b](1)}{\Pr[u = 0 \mid u_P = b]} = 1.$$

The case where $l = (u, 0) \in T$ follows similarly. For the inductive step, suppose we are at a node u and have chosen the value i. According to the subroutine, we have also chosen values x, y for the subtrees of u_L and u_R, such that $x + y = i$. By the induction hypothesis, we can assume that the conditions were met for u_L and u_R. Thus, the weighted sum of satisfied literals in the subtree of u will be equal to $x + y = i$. □