State estimation in discrete graphical models

Kevin P. Murphy

Last updated November 16, 2006

* Denotes advanced sections that may be omitted on a first reading.

1 Introduction

Graphical models define joint probability distributions

$$p(X_{1:D}|G, \theta) \tag{1}$$

where G is the graph structure (either directed or undirected or both), and θ are the parameters. In Bayesian modeling, we treat the parameters as random variables as well, but they are in turn conditioned on fixed **hyper parameters** α:

$$p(X_{1:D}, \theta|G, \alpha) \tag{2}$$

Clearly this can be represented as in Equation 1 by appropriately redefining X and θ. It will also be notationally helpful to distinguish the hidden nodes X from the observed nodes Y. Without loss of generality, we may assume there is a Y_i node for every X_i node; we sometimes call Y_i the **local evidence** for X_i. If X_i is not observed, then Y_i can be set to something non informative.

There are many quantities of interest we may be interested in inferring. Broadly speaking, they are as follows

1. **State estimation**: inferring $p(X|y, \theta, G)$.
2. **Parameter estimation (learning)**: inferring $p(\theta|y, G)$.
3. **Model selection (structure learning)**: inferring $p(G|y)$.

In this chapter, we focus on the first problem. Furthermore, we restrict our attention to the case where the X_i are discrete random variables. (The observed y_i's may be continuous, however.)

The techniques we describe work equally well for directed and undirected models. A directed model

$$p(X_{1:D}) = \prod_i p(X_i|X_{\pi_i}) \tag{3}$$

can always be written as an undirected model

$$p(X_{1:D}) = \frac{1}{Z} \prod_c \psi_c(X_c) \tag{4}$$

by setting $Z = 1$ and identifying the cliques of the undirected graph with the families in the original DAG. We will use examples of both kinds.

2 Kinds of probabilistic queries

There are several different flavors of state estimation, which we review below using the "water sprinkler" model in Figure 2 as an example.

2.1 Marginals

Often we want to estimate one set of variables given information on another set, e.g.,

$$P(s=1|w=1) = \frac{P(s=1,w=1)}{P(w=1)} \tag{5}$$

$$= \frac{\sum_{c,r} P(s=1,w=1,R=r,C=c)}{\sum_{c,r,s} P(S=s,w=1,R=r,C=c)} \tag{6}$$

$$\propto \sum_{c,r} P(C=c)P(S=1|C=c)P(R=r|C=c)P(W=1|S=s,R=r) \tag{7}$$

In general, computing marginals $p(x_i)$ for some set i involves **sum-product** type computations

$$p(x_i) = \frac{1}{Z} \sum_{x_{-i}} \prod_c \psi_c(x_c) \tag{8}$$

2.2 MAP estimation

Alternatively, we can compute the **most probable explanation (MPE)** of the evidence

$$(s,r,c)^* = \arg\max_{s,r,c} P(S=s,R=r,C=c|w=1) \tag{9}$$

$$= \arg\max_{s,r,c} P(C=c)P(S=1|C=c)P(R=r|C=c)P(W=1|S=s,R=r) \tag{10}$$

which is the most likely setting of *all* the hidden variables given the evidence. This is also (sometimes) called the **maximum a posteriori (MAP)** estimate. In general, To compute the MAP estimate, we use **max-product** computations

$$x^{MAP} = \arg\max_x \prod_c \psi_c(x_c) \tag{11}$$

The case where we max out over all but one of the variables results in the **max marginals**

$$p_{MM}(x_i) = \arg\max_{x_{-i}} \prod_c \psi_c(x_c) \tag{12}$$

2.3 Marginal MAP

Instead of computing the marginal probabilities $P(S|W=1)$ and $P(R|W=1)$, we may want to compute the **marginal MAP** estimate of these nodes:

$$s^* = \arg\max_s P(S=s|W=1) \tag{13}$$

$$= \arg\max_s \sum_{r,c} P(S=s,R=r,C=c|W=1) \tag{14}$$

$$= \arg\max_s \sum_{r,c} P(C=c)P(S=1|C=c)P(R=r|C=c)P(W=1|S=s,R=r) \tag{15}$$

and similarly for r^*. In general, this results in a **max-sum-product** formulation:

$$x_i^{MMAP} = \arg\max_{x_i} \sum_{x_{-i}} \prod_c \psi_c(x_c) \tag{16}$$

The MPE is when we max over all the (hidden) variables and is an easier problem than the mixed max-sum-product case.

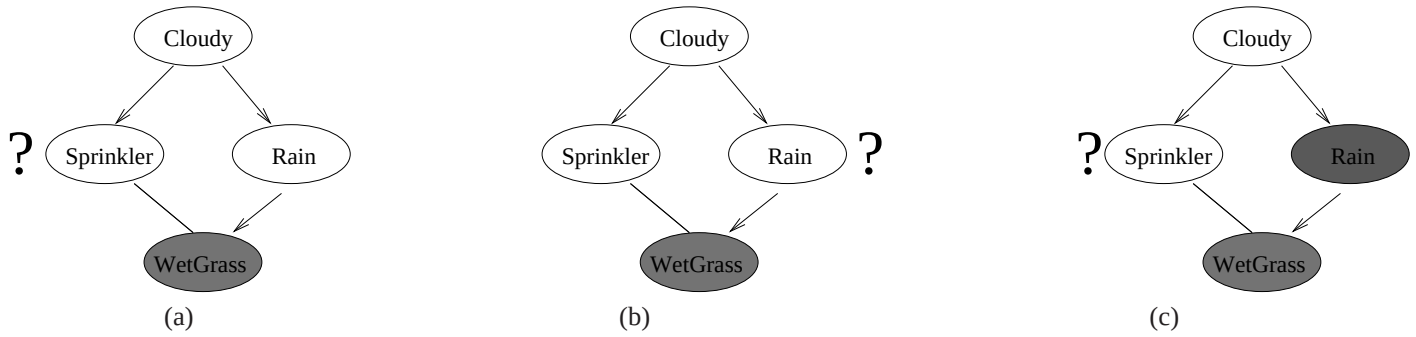

Figure 1: (a) observe W, query S. (b) observe W, query R. (c) Observe W and R, query S.

2.4 Sampling from the distribution

Sometimes we want to draw samples from the distribution, either the prior

$$x \sim p(x) \tag{17}$$

or the posterior

$$x \sim p(x|y) \tag{18}$$

Sampling from the prior for a DGM is straightforward: we can use **ancestral sampling** to sample from root to leaves in topological order. Sampling from the prior for a UGM is hard, because the potentials are not probability distributions, so we need to take into account Z. Sampling from the posterior for both DGMs and UGMs is also hard, again because of the partition function. (For a DGM, the partition function of the posterior is $p(y) = \sum_x p(x, y)$.) However, most of the techniques that are used for computing marginals (sum-product algorithm) can be generalized to draw samples, so the computational complexity is similar.

2.5 Predictive vs diagnostic reasoning in DGMs

Because of the directed/causal semantics of Bayes nets, it is possible to classify probabilsitic queries into several kinds. One kind is **predictive reasoning**: we observe a cause and want to predict its effects. For example, we may want to know the probability the grass becomes wet if we see that it is cloudy: we just compute $p(w = 1|c = 1)$. The most common kind is **diagnostic reasoning**: we observe effects and want to estimate the cause. For example, we observe the grass is wet and want to know if it was caused by the sprinkler or the rain. We can simply compare $P(S = 1|w = 1)$ vs $P(R = 1|w = 1)$: see Figure 1.

Now suppose we observe that the grass is wet *and* it is raining. Intuitively, the probability the sprinkler is on should be less than if we just observed the grass is wet, since the rain has **explained away** the evidence that the grass is wet. In other words, we expect

$$P(S = 1|w = 1, r = 1) < P(S = 1|w = 1) \tag{19}$$

This is indeed the case, as we show numerically below. Note that, a priori, S and R are independent (given C), but once we condition on W, they become dependent, because they compete to explain the evidence.

2.6 Computing the partition function

Since MRFs are undirected, there is no notion of predictive vs diagnostic reasoning, and no explaining away effect. It's simply inferring one variable given evidence on another. One task that is unique to MRFs is computing the partition function. (For DGMs, $Z = 1$.) Not all inference algorithms are capable of computing Z. (For example it is not automatically computed by Gibbs sampling, although there are work arounds [Chi95].)

Being able to compute Z is useful for model selection and for some gradient-based parameter learning algorithms, which need to evaluate the objective function (likelihood) itself. Other learning algorithms just require derivatives of Z, which turn out to be marginal probabilities.

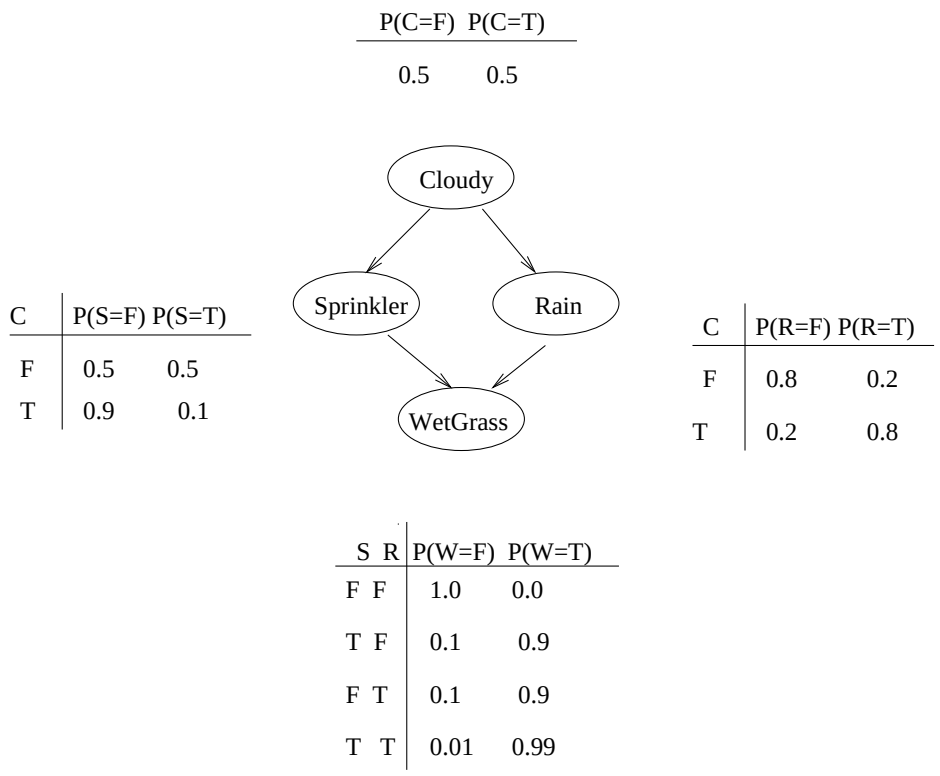

Figure 2: Water sprinkler Bayes net with CPDs shown.

3 Brute force enumeration (naive inference)

If all the nodes are discrete, we can represent this joint explicitly by multiplying all the potentials together elementwise (taking care to match dimensions) and representing the result as a $K \times \cdots \times K = K^D$ table, where there are D nodes each with K states. Call this $T(x_1, \ldots, x_D)$. The partition function is just $Z = \sum_{x_{1:D}} T(x_{1:D})$, and the joint is $p(x_{1:D}) = T(x_{1:D})/Z$.

Of course, constructing the joint explicitly takes $O(K^D)$ time and space, which defeats one of the main advantages of using graphical models. (The other advantage — namely that we need many fewer than $O(K^D)$ parameters to define the model — is not affected by how we represent the joint.) Later we will discuss more efficient techniques for inference that exploit the graph structure.

3.1 DGM example

If all the CPDs are tables, as in Figure 2, we can multiply them altogether to build the joint as a multidimensional array, as in the code below.

```
% Water sprinkler Bayes net
%   C
%  / \
% v  v
% S  R
%  \/
%  v
%  W

C = 1; S = 2; R = 3; W = 4;
false = 1; true = 2;

% Specify the conditional probability tables as cell arrays
% The left-most index toggles fastest, so entries are stored in this order:
% (1,1,1), (2,1,1), (1,2,1), (2,2,1), etc.
```

```
CPD{C} = reshape([0.5 0.5], 2, 1);
CPD{R} = reshape([0.8 0.2 0.2 0.8], 2, 2);
CPD{S} = reshape([0.5 0.9 0.5 0.1], 2, 2);
CPD{W} = reshape([1 0.1 0.1 0.01 0 0.9 0.9 0.99], 2, 2, 2);

% naive method
joint = zeros(2,2,2,2);
for c=1:2
  for r=1:2
    for s=1:2
      for w=1:2
        joint(c,s,r,w) = CPD{C}(c) * CPD{S}(c,s) * CPD{R}(c,r) *  CPD{W}(s,r,w);
      end
    end
  end
end

% vectorized method
joint2 = repmat(reshape(CPD{C}, [2 1 1 1]), [1 2 2 2]) .* ...
         repmat(reshape(CPD{S}, [2 2 1 1]), [1 1 2 2]) .* ...
         repmat(reshape(CPD{R}, [2 1 2 1]), [1 2 1 2]) .* ...
         repmat(reshape(CPD{W}, [1 2 2 2]), [2 1 1 1]);

assert(approxeq(joint, joint2));

pSandW = sumv(joint(:,true,:,true), [C R]); % 0.2781
pW = sumv(joint(:,:,:,true), [C S R]); % 0.6471
pSgivenW = pSandW / pW;  % 0.4298

pRandW = sumv(joint(:,:,true,true), [C S]); % 0.4581
pRgivenW = pRandW / pW; % 0.7079

% P(R=t|W=t) > P(S=t|W=t), so
% Rain more likely to cause the wet grass than the sprinkler

pSandRandW = sumv(joint(:,true,true,true), [C]); % 0.0891
pSgivenWR = pSandRandW / pRandW; % 0.1945

% P(S=t|W=t,R=t) << P(S=t|W=t)
% Sprinkler is less likely to be on if we know that
% it is raining, since the rain can "explain away" the fact
% that the grass is wet.
```

Having computed the joint, we can answer any probabilistic query we want. For example, we can compute

$$p(S=2|W=2) = \frac{p(S=2,W=2)}{p(W=2)} \tag{20}$$

$$= \frac{\sum_{c,r} p(C=c,R=r,S=2,W=2)}{\sum_{c,r,s} p(C=c,R=r,S=s,W=2)} \tag{21}$$

$$= \frac{0.2781}{0.6471} = 0.4298 \tag{22}$$

(where 2 denotes true and 1 denotes false).

Similarly, we can compute

$$p(R=2|W=2) = \frac{p(R=2,W=2)}{p(W=2)} \tag{23}$$

$$= \frac{0.4581}{0.6471} = 0.7079 \tag{24}$$

Since $0.7079 = P(R=t|W=t) > P(S=t|W=t) = 0.4298$, rain is more likely the cause of the wet grass than the sprinkler. Finally, we can illustrate explaining away by showing $P(S=t|W=t,R=t) = 0.1945 \ll P(S=t|W=t) = 0.4298$.

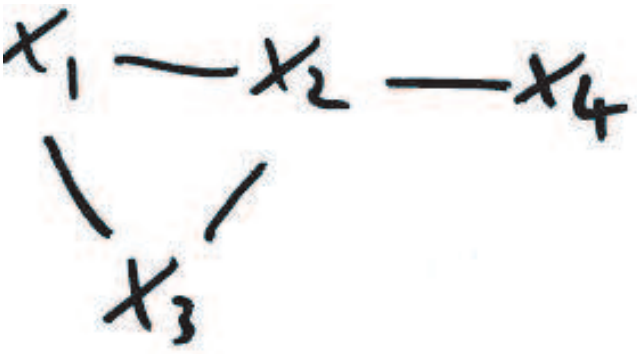

Figure 3: A simple MRF

3.2 UGM example

Now consider the MRF in Figure 3. All nodes are binary (have values 1 or 2). The model defines the following joint distribution

$$p(X_{1:4}) = \frac{1}{Z}\psi_{123}(X_1, X_2, X_3)\psi_{24}(X_2, X_4) \tag{25}$$

where the potentials are defined as follows

X_1	X_2	X_3	Ψ_{123}
1	1	1	1
2	1	1	2
1	2	1	3
2	2	1	4
1	1	2	5
2	1	2	6
1	2	2	7
2	2	2	8

X_2	X_4	Ψ_{24}
1	1	0.5
2	1	1
1	2	1.5
2	2	2

Hence the unnormalized joint $p'(x_{1:4})$ and the normalized joint $p(x_{1:4}) = p'(x_{1:4})/Z$ is given below, where $Z = 94$.

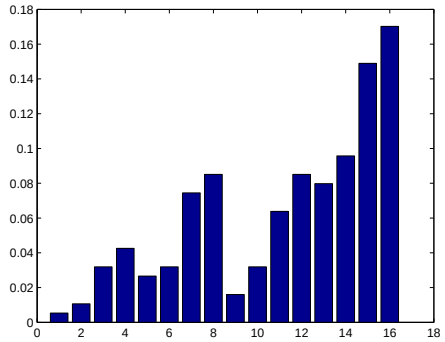

Figure 4: The "flattened" joint distribution encoded by the MRF in Figure 3. This represents a distribution on a $2 \times 2 \times 2 \times 2$ table.

X_1	X_2	X_3	X_4	$p'(X_{1:4})$	$p(X_{1:4})$
1	1	1	1	$1 \times 0.5 = 0.5$	0.0053
2	1	1	1	$2 \times 0.5 = 1$	0.0106
1	2	1	1	$3 \times 1 = 3$	0.0319
2	2	1	1	$4 \times 1 = 4$	0.0426
1	1	2	1	$5 \times 0.5 = 2.5$	0.0266
2	1	2	1	$6 \times 0.5 = 3$	0.0319
1	2	2	1	$7 \times 1 = 7$	0.0745
2	2	3	1	$8 \times 1 = 8$	0.0851
1	1	1	2	$1 \times 1.5 = 1.5$	0.0160
2	1	1	2	$2 \times 1.5 = 3$	0.0319
1	2	1	2	$3 \times 2 = 6$	0.0638
2	2	1	2	$4 \times 2 = 8$	0.0851
1	1	2	2	$5 \times 1.5 = 7.5$	0.0798
2	1	2	2	$6 \times 1.5 = 9$	0.0957
1	2	2	2	$7 \times 2 = 14$	0.1489
2	2	2	2	$8 \times 2 = 16$	0.1702

The corresponding joint distribution is shown in Figure 4.

We could write code to multiply these tables together, but to avoid the need for all the `repmat`'s and `reshape`'s, I have developed some Matlab code that automates these kinds of calculations. The class `tabularPot` represents discrete potentials, and is basically an array in which each dimension has a 'tag' associated with it, representing the 'domain' of the array, so that when we multiply two potentials together, the corresponding dimensions can be lined up. The class has a constructor and various methods, which are illustrated below. We omit the implementation of these functions since they involve a lot of uninteresting book keeping. (See [HD96] for some of the details.) However, we illustrate how to use them below by constructing the joint as a 4 dimensional table.

```
% mrfJointDemo.m
%
% Constructor is pot = tabularPot(domain, sizes, T)
% domain = variables in the potential
% sizes = number of states for each variable
% T = table of numbers
f1 = tabularPot([1 2 3], [2 2 2], [1:8]);
f2 = tabularPot([2 4], [2 2], [0.5 1 1.5 2]);

% Combine potentials
J = tabularPot(1:4, [2 2 2 2]);
J = multiplyByPot(f, f1);
J = multiplyByPot(f, f2);

% Or more directly
J = multiplyPots(f1, f2)
```

```
% Convert object to array
unnormalizedJoint = J.T(:)'

% Normalize the array
[joint, Z] = normalize(unnormalizedJoint) % Z=94
bar(joint)

% alternative method of computing Z
m = marginalizePot(f, []);
Z = m.T; % m.T = 58
```

Given the joint, we can easily find the MAP assignments

$$x^* \in \arg\max_x p(x) \tag{26}$$

We can also draw samples from the joint by treating it as a histogram with K^D bins. Finally, we can compute any marginal or conditional we want. For example, the code below computes

$$p(x_1|x_4) = \frac{\sum_{x_2,x_3} p(x_{1:4})}{\sum_{x_1,x_2,x_3} p(x_{1:4})} \tag{27}$$

```
% mrfJointDemo2

% run mrfJointDemo first!

% find the MAP state
[junk, xMAP] = max(joint) % 16
xMAPbits = ind2subv(2*ones(1,4), xMAP) % 2,2,2,2

% sample from the distribution (with replacement)
S = 1000;
K = length(joint);
%samples = randsample(1:K, S, true, joint)
samples = sample_discrete(joint, 1, S);
h = hist(samples,1:K);
bar(normalize(h))

% compute cond14(x4, x1) = p(x1|x4)
J14 = marginalizePot(J, [1 4]);
cond14 = mk_stochastic(J14.T'); % satisfies sum_x1 cond14(x4,x1) = 1
sum(cond14,2) % column of 1s
```

4 Variable elimination

The **variable elimination** algorithm uses the principle of **(non-serial) dynamic programming** and can be much more efficient than the naive approach of brute force enumeration. Dynamic programming is applicable whenever the optimal solution to a problem can be divided into pieces that can be solved separately and then reused. A classic example is **Dijkstra's shortest path algorithm**. Later we will see the **forwards backwards** algorithm, which is closely related.

Consider the example model in Figure 5. This can either be interpreted as a directed graphical model (i.e., a Bayesian network)

$$P(C,D,I,G,S,L,J,H) = P(C)P(D|C)P(I)P(G|I,D)P(S|I)P(L|G)P(J|L,S)P(H|G,J) \tag{28}$$

or as an undirected graphical model (i.e., as an MRF)

$$P(C,D,I,G,S,L,J,H) = \psi_C(C)\psi_D(D,C)\psi_I(I)\psi_G(G,I,D)\psi_S(S,I)\psi_L(L,G)\psi_J(J,L,S)\psi_H(H,G,J) \tag{29}$$

where the potentials (factors) are $\psi_C(C) = p(C)$, $\psi_D(D,C) = p(D|C)$, etc. (Since all the potentials are locally normalied (sum to one), we find $Z = 1$. This is always the case when we convert from a Bayes net to an MRF.)

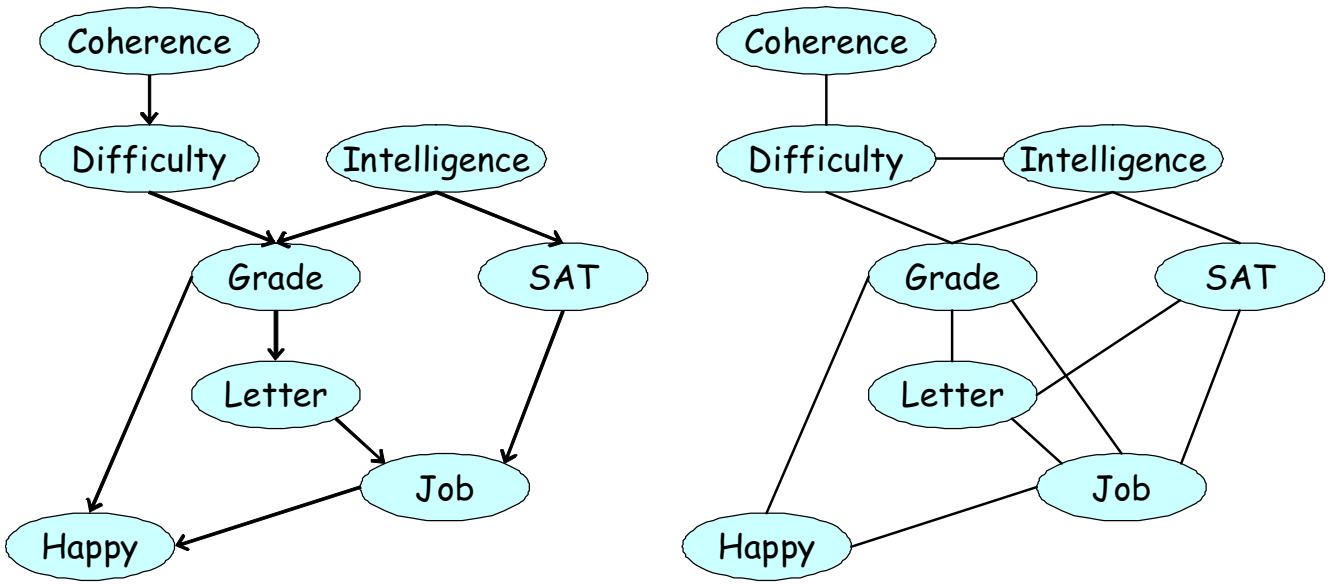

Figure 5: Left: The student Bayes net. Right: the equivalent Markov network. We add moralization arcs D-I, G-J and L-S. Note that this graph is not triangulated. Source: [KF06].

Suppose we want to compute $P(J)$, the marginal probability that a person will get a job. The key idea of variable elimination is to **push sum inside products**, which is valid because of the **distributive law** of sums and products.

$$P(J) = \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C P(C, D, I, G, S, L, J, H) \tag{30}$$

$$= \sum_L \sum_S \sum_G \sum_H \sum_I \sum_D \sum_C \psi_C(C)\psi_D(D, C)\psi_I(I)\psi_G(G, I, D)\psi_S(S, I)\psi_L(L, G)\psi_J(J, L, S)\psi_H(H, G, J) \tag{31}$$

$$= \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I)\psi_I(I) \sum_D \psi_(G, I, D) \sum_C \psi_C(C)\psi_D(D, C) \tag{32}$$

Now we work right to left (this is called **peeling**), as shown in Figure 6. At step i, we create an intermediate factor τ_i which gets combined with the original factors. We can think of each $\sum$ term as a **bucket** containing all the factors immediately to its right (in its immediate lexical scope); then the τ_i factors act as **messages** that are sent from bucket to bucket. Hence the variable eliminatio algorithm is also called **bucket elimination**.

We explain these steps in more detail below.

- We first multiply together all factors that mention C to create $\psi_1(C, D)$, and store the result in C's bucket:

$$P(J) = \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I)\psi_I(I) \sum_D \psi_(G, I, D) \sum_C \underbrace{\psi_C(C)\psi_D(D, C)}_{\psi_1(C,D)} \tag{40}$$

- Then we sum out C to make $\tau_1(D)$:

$$P(J) = \sum_L \sum_S \psi_J(J, L, S) \sum_G \psi_L(L, G) \sum_H \psi_H(H, G, J) \sum_I \psi_S(S, I)\psi_I(I) \sum_D \psi_(G, I, D) \underbrace{\sum_C \psi_1(C, D)}_{\tau_1(D)} \tag{41}$$

$$P(J) = \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \sum_D \psi_(G,I,D) \underbrace{\sum_C \psi_C(C)\psi_D(D,C)}_{\tau_1(D)} \quad (33)$$

$$= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \underbrace{\sum_D \psi_(G,I,D)\tau_1(D)}_{\tau_2(G,I)} \quad (34)$$

$$= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \underbrace{\sum_I \psi_S(S,I)\psi_I(I)\tau_2(G,I)}_{\tau_3(G,S)} \quad (35)$$

$$= \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \underbrace{\sum_H \psi_H(H,G,J)}_{\tau_4(G,J)} \tau_3(G,S) \quad (36)$$

$$= \sum_L \sum_S \psi_J(J,L,S) \underbrace{\sum_G \psi_L(L,G)\tau_4(G,J)\tau_3(G,S)}_{\tau_5(J,L,S)} \quad (37)$$

$$= \sum_L \underbrace{\sum_S \psi_J(J,L,S)\tau_5(J,L,S)}_{\tau_6(J,L)} \quad (38)$$

$$= \underbrace{\sum_L \tau_6(J,L)}_{\tau_7(J)} \quad (39)$$

Figure 6: Eliminating variables from Figure 5 in the order C, D, I, H, G, S, L.

- and multiply into D's bucket to make $\psi_2(G,I,D)$:

$$P(J) = \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \sum_D \underbrace{\psi_(G,I,D)\tau_1(D)}_{\psi_2(G,I,D)} \quad (42)$$

- Then we sum out D to make $\tau_2(G,I)$:

$$P(J) = \sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J) \sum_I \psi_S(S,I)\psi_I(I) \underbrace{\sum_D \psi_2(G,I,D)}_{\tau_2(G,I)} \quad (43)$$

- and multiply into I's bucket to make $\psi_3(G,S,I)$, etc.
- And so on.

4.1 Dealing with evidence

So far we have computed the unconditional distribution $P(J)$. To compute conditional distributions, we take the ratios of unconditionals e.g.

$$P(J|I=1,H=0) = \frac{P(J,I=1,H=0)}{P(I=1,H=0)} \quad (44)$$

where $P(I=1,H=0) = \sum_j P(J=j,I=1,H=0)$ is the normalizing constant. The numerator is gotten by running VE where we have evidence on nodes I and H. There are two methods.

In the first method, we instantiate observed variables to their observed values, by taking the appropriate "slices" of the factors (only works for discrete observations):

$$
\begin{aligned}
P(J) &= \sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \sum_S \psi_J(J,L,S) \sum_I \psi_I(I)\psi_S(S,I) \underbrace{\sum_G \psi_G(G,I,D)\psi_L(L,)\psi_H(H,G,J)}_{\tau_1(I,D,L,J,H)} && (49)\\
&= \sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \sum_S \psi_J(J,L,S) \underbrace{\sum_I \psi_I(I)\psi_S(S,I)\tau_1(I,D,L,J,H)}_{\tau_2(D,L,S,J,H)} && (50)\\
&= \sum_D \sum_C \psi_D(D,C) \sum_H \sum_L \underbrace{\sum_S \psi_J(J,L,S)\tau_2(D,L,S,J,H)}_{\tau_3(D,L,J,H)} && (51)\\
&= \sum_D \sum_C \psi_D(D,C) \sum_H \underbrace{\sum_L \tau_3(D,L,J,H)}_{\tau_4(D,J,H)} && (52)\\
&= \sum_D \sum_C \psi_D(D,C) \underbrace{\sum_H \tau_4(D,J,H)}_{\tau_5(D,J)} && (53)\\
&= \sum_D \underbrace{\sum_C \psi_D(D,C)\tau_5(D,J)}_{\tau_6(D,J)} && (54)\\
&= \underbrace{\sum_D \tau_6(D,J)}_{\tau_7(J)} && (55)
\end{aligned}
$$

Figure 7: Eliminating variables from Figure 5 in the order G, I, S, L, H, C, D.

$$P(J, I = 1, H = 0) = \tag{45}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G)\psi_H(H=0,G,J)\psi_S(S,I)\psi_I(I=1) \sum_D \psi_(G,I=1,D) \sum_C \psi_C(C)\psi_D(D,C) \tag{46}$$

In the second method, we multiply in local evidence factors $\phi_i(X_i)$ for each node.

$$P(J, I = 1, H = 0) = \tag{47}$$

$$\sum_L \sum_S \psi_J(J,L,S) \sum_G \psi_L(L,G) \sum_H \psi_H(H,G,J)\phi_{\mathbf{H}}(\mathbf{H}) \sum_I \psi_S(S,I)\psi_I(I)\phi_{\mathbf{I}}(\mathbf{I}) \sum_D \psi_(G,I,D) \sum_C \psi_C(C)\psi_D(D,C) \tag{48}$$

If X_i is observed to have value x_i^*, we set $\phi_i(X_i) = I(X_i = x_i^*)$. If y_i is a noisy observaton of X_i, we set $\phi_x(X_i) = p(y_i|X_i)$ (sometimes called **soft evidence** or **virtual evidence**).

4.2 Computational complexity

The time to answer any query is exponential in the size (number of terms) in the largest factor (table) that is encountered. The factors come from the original model, but new factors are created in the process of summing out. The order in which we perform the summation (the **elimination order**) can have a large impact on the size of the intermediate factors. For example, consider the ordering in Figure 6: the largest factor is $\tau_5(J, L, S)$. Now consider the ordering in Figure 7: now the largest factor $\tau_1(I, D, L, J, H)$ or $\tau_2(D, L, S, J, H)$.

We can determine the size of the largest factor graphically, without worrying about the actual numerical values of the factors. When we eliminate a variable X_i, we connect it to all variables that share a factor with X_i (to reflect new factor τ_i). Such edges are called **fill-in edges**. For example, Figure 8 shows the fill-in edges introduced when we eliminate in the order $C, D, I, \ldots$. The first two steps do not introduce any fill-ins, but when we eliminate I, we

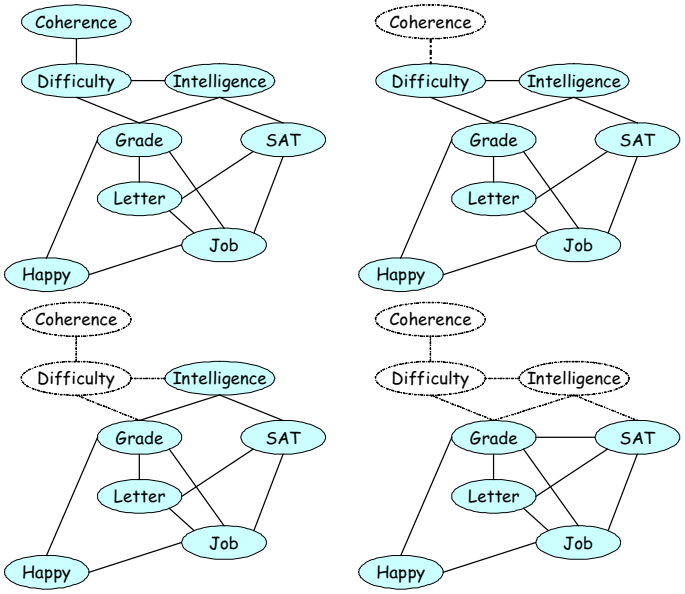

Figure 8: Fill in edges. When we eliminate I (bottom right), we connect G and S. From [KF06].

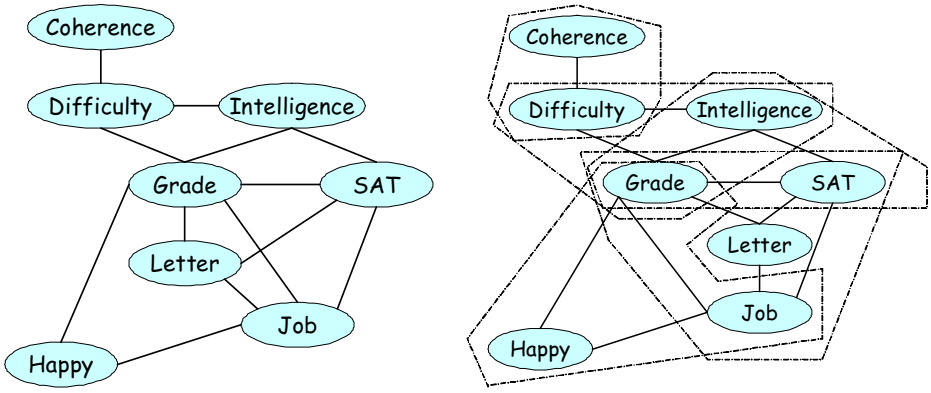

Figure 9: Maximal cliques. Source [KF06].

connect G and S, since they co-occur in factor $\tau_3(G, S)$:

$$\tau_3(G, S) = \sum_I \psi_S(S, I)\psi_I(I)\tau_2(G, I) \tag{56}$$

Let $I_{G,\prec}$ be the (undirected) graph induced by applying variable elimination to G using ordering $\prec$. The factors generated by VE correspond to **cliques** in $I_{G,\prec}$ and vice versa. For example, with ordering (C, D, I, H, G, S, L), the **maximal cliques**, shown in Figure 9, are

$$\{C, D\}, \{D, I, G\}, \{G, L, S, J\}, \{G, J, H\}, \{G, I, S\} \tag{57}$$

Let us define the **induced width** of a graph given elimination ordering $\prec$, denote $W_{G,\prec}$, as the size of the largest factor (induced clique) minus 1. Then it is easy to show

Theorem 1 *The complexity of VarElim with ordering $\prec$ is $O(DK^{W_{G,\prec}+1})$, where we assume all D nodes have K possible states each.*

Note that if the graph is chordal (triangulated), then variable elimination not introduce any extra fill-in egdes, and $W_{G,\prec} = W_G$.

We define the **tree width of a graph** as the minimal induced width:

$$W_G = \min_{\prec} \max_i |\tau_i| - 1 \tag{58}$$

What is the order that produces this minimal tree width? Unfortunately, one can show

Theorem 2 *Finding an elimination ordering $\prec$ which minimizes $W_{G,\prec}$ is NP-hard.*

A standard approach to finding $\prec$ is greedy search. The **min-fill** heuristic says: choose as the next node to eliminate the one that introduces the least number of fill-in edges (breaking ties randomly). The **min-weight** heuristic says: choose as the next node to eliminate the one that introduces the factor of smallest weight, where the weight of a factor is the size of its state space (product of the cardinalities of all variables within it).

5 Belief propagation *

If the graph is a tree or a chain (so there are no undirected cycles), one can use a **dynamic programming** algorithm called **belief propagation (BP)** to perform exact inference. For a chain in which all (hidden) nodes are discrete, BP inference takes $O(DK^2)$ time, where D is the number of nodes and K is the number of states. This algorithm is also called the **forwards backwards algorithm**. (The max-product version is called the **Viterbi algorithm**.) For a chain in which all nodes are Gaussian, inference takes $O(D(2K)^3)$ time, where all (vector valued) nodes have size K. (The cubic terms arises because we have to invert matrices of size $2K \times 2K$.) This algorithm is also called the **Kalman filter/ RTS smoother** algorithm (RTS = Rauch Tung Streibel). We will discuss these algorithms later.

If the graph is not a tree (so it has loops), but all nodes are discrete or Gaussian, one can still run the BP algorithm; this is called **loopy belief propagation**. Although it often works well, it is not guaranteed to work (e.g., it may oscillate); see [YFW01] for details.

6 Junction tree algorithm *

If the graph is decomposable, then it is possible to convert it into a **junction tree (jtree)**, also called a **join tree**, whose nodes correspond to cliques in the triangulated graph. If all nodes are discrete or Gaussian, one can then perform BP on the jtree. (Some modifications are required to handle the fact that the variables in the tree correspond to sets of variables in the original model.) If all hidden nodes are discrete, then the jtree algorithm takes $O(DK^{w+1})$ time, where w is the treewidth of the graph. If all hidden nodes are Gaussian, then the jtree algorithm takes $O(D(K(w+1))^3)$ time. See [CDLS99, Jor06, KF06] for details.

If the graph is not decomposable, it can always be made so by triangulating it, but the resulting treewidth w may be so large that inference becomes intractable. For example, for a 100×100 grid, we have $w = 100$, so inference takes $O(2^{100})$ time for binary nodes. In cases where the treewidth is too large, one must resort to approximate inference techniques. The other situation in which approximate inference is necessary is when not all the nodes are discrete or Gaussian (e.g., in hierarchical Bayesian models). We discuss some approximation methods below.

7 Gibbs sampling

One general purpose technique for sampling from distributions (discrete or continuous or mixed) which are hard to normalize is MCMC. (MRFs can be hard to normalize since computing Z takes $O(K^w)$ time, where w is the treewidth of the graph. DGMs are hard to normalize when there is evidence, since computing $p(y) = \sum_x p(x, y)$ also takes $O(K^w)$ time.) Although one can use Metropolis Hastings, it is usually simpler and more efficient to use Gibbs sampling. Gibbs sampling for graphical models is particularly simple because the full conditionals $p(x_i|x_{-i})$ only depend on the state of the nodes in i's Markov blanket, where x_{-i} are all the other nodes except x_i.

Gibbs sampling can be applied to DGMs or UGMs. For simplicity, we consider a pairwise MRF. Let N_i be the

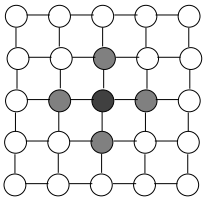

Figure 10: The Markov blanket of the central dark node in a 2D grid MRF are its nearest neighbors (shaded).

neighbors (Markov blanket) of node i, and $F_i = N_i \cup \{i\}$ be the family of node i (node and its neighbors). Then

$$p(X_i = \ell | x_{-i}) = \frac{p(x_i = \ell, x_{-i})}{\sum_{\ell'} p(X_i = \ell', x_{-i})} \tag{59}$$

$$= \frac{(1/Z)[\prod_{j \in N_i} \psi_{ij}(X_i = \ell, x_j)][\prod_{<jk>:j,k \notin F_i} \psi_{jk}(x_j, x_k)]}{(1/Z)\sum_{\ell'}[\prod_{j \in N_i} \psi_{ij}(X_i = \ell', x_j)][\prod_{<j,k>:j,k \notin F_i} \psi_{jk}(x_j, x_k)]} \tag{60}$$

$$= \frac{\prod_{j \in N_i} \psi_{ij}(X_i = \ell, x_j)}{\sum_{\ell'} \prod_{j \in N_i} \psi_{ij}(X_i = \ell', x_j)} \tag{61}$$

The notation $\prod_{<jk>:j,k \notin F_i}$ means a product over all edges $j - k$ where neither j nor k is in F_i. See Figure 10.

In the special case of an Ising model, where $\psi(x_i, x_j) = e^{J x_i x_j}$, this simplifies to

$$p(X_i = +1 | x_{-i}) = \frac{\prod_{j \in N_i} \psi_{ij}(X_i = +1, x_j)}{\prod_{j \in N_i} \psi_{ij}(X_i = +1, x_j) + \prod_{j \in N_i} \psi_{ij}(X_i = -1, x_j)} \tag{62}$$

$$= \frac{\exp[J \sum_{j \in N_i} x_j]}{\exp[J \sum_{j \in N_i} x_j] + \exp[-J \sum_{j \in N_i} x_j]} \tag{63}$$

$$= \frac{\exp[J w_i]}{\exp[J w_i] + \exp[-J w_i]} \tag{64}$$

$$= \sigma(2 J w_i) \tag{65}$$

where $w_i = \sum_{j \in N_i} x_j$ and $\sigma(u) = 1/(1 + e^{-u})$ is the sigmoid function.

When we combine an Ising prior with a local evidence term (as in the image denoising example), the full conditional becomes

$$p(X_i = +1 | x_{-i}, y) = \frac{\exp[J w_i] \phi_i(+1, y_i)}{\exp[J w_i] \phi_i(+1, y_i) + \exp[-J w_i] \phi_i(-1, y_i)} \tag{66}$$

For the case of a Gaussian observation model, $\phi_i(x_i, y_i) = \mathcal{N}(y_i | x_i, \sigma)$.

Thus implementing Gibbs sampling on a 2D lattice is particularly easy: the code below is all that is needed to produce the image denoising example in Figure **??**. (Note that a pixel at location (i, j) in a $D \times D$ grid corresponds to a linear index of $i + (D - 1) \times j$.)

```
% gibbsDemoDenoising
% Denoising of letter A using Gibbs sampling
% with an Ising Prior and a Gaussian likelihood
% Based on code originally written by Brani Vidakovic

seed = 3;
randn('state',seed)
rand ('state',seed)

sigma = 2; % noise level

% input matrix consisting of letter A. The body of letter
% A is made of 1's while the background is made of -1's.
img = imread('lettera.bmp');
```

```
[M,N] = size(img);
img = double(img);
m = mean(img(:));
img2 = +1*(img>m) + -1*(img<m); % -1 or +1
y = img2 + sigma*randn(size(img2)); %y = noise signal


% observation model
offState = 1; onState = 2;
mus = zeros(1,2);
mus(offState) = -1; mus(onState) = +1;
sigmas = [sigma sigma];
Npixels = M*N;
localEvidence = zeros(Npixels, 2);
for k=1:2
  localEvidence(:,k) = normpdf(y(:), mus(k), sigmas(k));
end

[junk, guess] = max(localEvidence, [], 2);  % start with best local guess
X = ones(M, N);
X(find(guess==offState)) = -1;
X(find(guess==onState)) = +1;
Xinit = X;

doPrint = 0;

figure;
imagesc(y);colormap gray; axis square; axis off
title(sprintf('sigma=%2.1f', sigma))
fname = sprintf('figures/gibbsDemoDenoisingOrigS%2.1f.eps', sigma);
if doPrint, print(gcf, '-depsc', fname); end

figure;
imagesc(Xinit);colormap gray; axis square; axis off
title('initial guess')
fname = sprintf('figures/gibbsDemoDenoisingInitS%2.1f.eps', sigma);
if doPrint, print(gcf, '-depsc', fname); end

fig = figure; clf
pause

J = 1;
avgX = zeros(M,N);
X = Xinit;
maxIter = 100000;
for iter =1:maxIter
  % select a pixel at random
  ix = ceil( N * rand(1) ); iy = ceil( M * rand(1) );
  pos = iy + M*(ix-1);
  neighborhood = pos + [-1,1,-M,M];
  neighborhood(find([iy==1,iy==M,ix==1,ix==N])) = [];
  % compute local conditional
  wi = sum( X(neighborhood) );
  p1  = exp(J*wi) * localEvidence(pos,onState);
  p0  = exp(-J*wi) * localEvidence(pos,offState);
  prob = p1/(p0+p1);
  if rand < prob
    X(pos) = +1;
  else
    X(pos) = -1;
  end
  avgX = avgX+X;
  % plotting
  if rem(iter,10000) == 0,
    figure(fig);
    imagesc(X);  axis('square'); colormap gray; axis off;
    title(sprintf('sample %d', iter));
    drawnow
  end
  if doPrint % iter==10000 | iter==50000 | iter==100000
    figure;
    imagesc(X);colormap gray; axis square; axis off
    title(sprintf('sample %d', iter))
    fname = sprintf('figures/gibbsDemoDenoisingIter%dJ%3.2fS%2.1f.eps', iter, J, sigma);
    print(gcf, '-depsc', fname);
  end
end
```

Figure 11: A taxonomy of different methods for approximate inference. P is the true distribution, Q is an approximate distribution, $KL(P||Q)$ is the KL-divergence from truth to approximation, and $KL(Q||P)$ is the KL-divergence from approximation to truth. EM = expectation maximization. ICM = iterative conditional modes. VarBayes = variational Bayes. EP = expectation propagation. BP = belief propagation. ADF = assumed density filtering. BK = Boyen-Koller algorithm.

```
figure;
imagesc(avgX);colormap gray; axis square; axis off
title(sprintf('posterior mean after %d samples', iter))
fname = sprintf('figures/gibbsDemoDenoisingMean%dJ%3.2fS%2.1f.eps', iter, J, sigma);
if doPrint, print(gcf, '-depsc', fname); end
```

8 Approximate inference *

We saw above that exact inference using the variable elimination algorithm takes $O(K^w)$ time, where K is the number of discrete states per node and w is the treewidth of the graph. Is there a better algorithm that does not take exponential time? In general, no, because of the following theorem.

Theorem 3 *Exact inference in discrete graphical models is NP-hard.*

Proof (sketch): Just show that 3-SAT is equivalent to inference in a deterministic Bayes net.

A large variety of approximate inference algorithms have been developed and this is a very active research area. A few methods have formal guarantees on solution quality[1] , but most are just heuristics whose quality is hard to assess. We list a few examples below and provide a bigger picture in Figure 11.

8.1 Loopy belief propagation

As mentioned above, one can always run BP on a graph even if it has loops. However, it may not always work well, or even converge. LBP is an example of **variational inference**. The **mean field** is another example. See [YFW01] for a discussion.

8.2 Graph cuts

In general, it is NP-hard to compute the minimal energy (most probable) state in an Ising model, but it can be solved exactly in polynomial time in the ferromagnetic case, i.e., if $J_{ij} > 0$, using linear programming or **grapchuts** [BVZ01]. For the multi-state case (Potts models/ associative Markov networks), one can get good approximate answers using similar techniques.

8.3 MCMC

What if we want to draw samples instead of computing the MAP state? This is strictly harder than finding the MAP state, e.g., it is #P-hard, even in the ferromagnetic case. The reason is that it is a **counting problem**: to compute probabilities (or samples), we have to know $Z = \sum_x p'(x, y)$, which requires summing over all x. Whereas finding the MAP state just means a single best state x. Despite the #P-hardness, we can use MCMC to get appproximate answers.

References

[BVZ01] Yuri Boykov, Olga Veksler, and Ramin Zabih. Fast approximate energy minimization via graph cuts. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 23(11), 2001.

[CDLS99] R. G. Cowell, A. P. Dawid, S. L. Lauritzen, and D. J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.

[Chi95] S. Chib. Marginal likelihood from the Gibbs output. *J. of the Am. Stat. Assoc.*, 90:1313–1321, 1995.

[DL93] P. Dagum and M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60:141–153, 1993.

[HD96] C. Huang and A. Darwiche. Inference in belief networks: A procedural guide. *Intl. J. Approx. Reasoning*, 15(3):225–263, 1996.

[Jor06] M. I. Jordan. *An Introduction to Probabilistic Graphical Models*. 2006. In preparation.

[KF06] D. Koller and N. Friedman. *Bayesian networks and beyond*. 2006. To appear.

[YFW01] J. Yedidia, W. T. Freeman, and Y. Weiss. Understanding belief propagation and its generalizations. In *Intl. Joint Conf. on AI*, 2001.

[1] One can also show it is NP-hard to approximate inference to within any fixed constant, either additive or multiplicative [DL93]. So any formal guarantees must come with additional assumptions.