An Algorithm for Inferences in a Polytree with Heterogeneous Conditional Distributions

Robert Dodier
robert_dodier@yahoo.com
http://riso.sourceforge.net

Abstract

This paper describes a general scheme for accomodating different types of conditional distributions in a Bayesian network. The algorithm is based on the polytree algorithm for Bayesian network inference, in which "messages" (probability distributions and likelihood functions) are computed. The posterior for a given variable depends on the messages sent to it by its parents and children, if any. In this scheme, an exact result is computed if such a result is known for the incoming messages, otherwise an approximation is computed, which is a mixture of Gaussians. The approximation may then be propagated to other variables. Approximations for likelihood functions (λ-messages) are not computed; the approximation step is put off until the likelihood function is combined with a probability distribution — this avoids certain numerical difficulties. In contrast with standard polytree algorithms, which can only accomodate distributions of a few types at most, this heterogeneous polytree algorithm can, in principle, handle any kind of continuous or discrete conditional distribution. With standard algorithms, it is necessary to construct an approximate Bayesian network, in which one then computes exact results; the heterogeneous polytree algorithm, on the other hand, computes approximate results in the original Bayesian network. The most important advantage of the new algorithm is that the Bayesian network can be directly represented using the conditional distributions most appropriate for the problem domain.

1 Introduction

Bayesian networks are successfully used in many fields to model joint probability distributions. A Bayesian network directly encodes independence relations, so that one can immediately determine whether two variables are independent given some evidence. These independence relations are crucial to making tractable the computation of posterior distributions. Almost as important, the computation of posterior distributions is much simplified if the conditional distribution for each variable comes from a certain small set of distributions. The best-known case is that of discrete conditional distributions; several algorithms have been developed (see, for example, Ref. [8]) for computing posterior distributions, and there are algorithms which can handle Bayesian networks which contain loops. Likewise, conditional Gaussian distributions are also well known [8]. An algorithm is also known [4] for a Bayesian network in which each conditional distribution is a mixture of conditional Gaussians (with a linear dependence of the mean and no dependence of the variance), but this algorithm applies only to a polytree network. This algorithm has been extended [5] to a polytree containing both continuous variables (with mixtures of conditional Gaussian distributions) and discrete variables.

In many problems, though, the most natural probability distributions are neither discrete nor Gaussian, and constructing approximations within those well-known classes may yield a representation which is verbose and which obscures the conceptual basis of the natural model. It seems desirable, then, to represent the natural probability distributions directly, and to only compute approximations (from the class of mixtures of Gaussians, for example) when necessary. The representation problem is easily solved by equipping each variable with type information and suitable parameters, and the difficult part is the computation of posterior distributions. Only for certain kinds of conditional distributions will it be possible to compute the

partial results necessary for the posterior, so in many cases some kind of approximation is necessary. The following approach is taken in this report:

> Directly represent the conditional distributions from the problem domain; compute an exact result for the posterior when possible, and otherwise compute an approximation to the exact result.

Note that this is different from the approach implicit in present Bayesian network implementations, which is this: if a distribution does not fall into a well-behaved category (discrete or Gaussian), make a well-behaved approximation, then compute an exact posterior using the approximation.

We want to keep representations in a Bayesian network close to the original problem domain. This will make it easier to describe and comprehend a Bayesian network, especially for domain experts who are not particularly well-acquainted with Bayesian networks; if one views a Bayesian network as a kind of probabilistic database [8] then it is important that such people find it intuitive and productive to work with Bayesian networks. As an important part of this approach, the conditional distributions in the network must match the ones which domain experts are accustomed to working with.

The reader will note that the scheme described in this report is similar in spirit to the adaptive discretization algorithm described in Ref. [7] and the combination of exact inference and Gibbs sampling in HUGS [6]. In each case, exact and numerical methods are combined to extend Bayesian network inference to a larger class of problems.

2 The polytree algorithm

The polytree algorithm for computing the posterior of any variable in a Bayesian network (without loops) exploits the fact that each variable X separates the network into two disjoint parts, one above X and one below X. The computation of the distribution of X given all the evidence $\mathbf{e}$ in the network, $p_{X|\mathbf{e}}$, can be expressed in terms of "predictive" messages $\pi_{U,X}$ which are passed down from each parent U of X, and "likelihood" messages $\lambda_{Y,X}$ which are passed up from each child Y of X. These messages are combined to yield the "predictive support" π_X, and the "likelihood support" λ_X, and the posterior for X is just the normalized product of π_X and λ_X. The polytree algorithm computes the posterior of one variable at a time, by computing the predictive and likelihood support for that variable, and computing any messages needed. So long as the evidence variables in the network do not change, the same messages may be re-used to compute the posterior distributions for other variables as well.

The polytree algorithm is defined in terms of π's and λ's as follows.[1] Denote the distribution of X given its parents $U_1, ..., U_m$ as q_X. That is, q_X is a shorthand for $p_{X|U_1,...,U_m}$. Let $\mathbf{e}$ be the set of all evidence variables within the network, with $\mathbf{e}_X^+$ denoting all the evidence above X in the polytree and $\mathbf{e}_X^-$ denoting all the evidence below X in the polytree. Also, let $\mathbf{e}_{X,U}^+$ denote the evidence above X which is also above its parent U, and let $\mathbf{e}_{X,Y}^-$ denote the evidence below X which is also below its child Y. Then the π and λ functions are defined in terms of probabilities involving X, the parents and children of X, and the evidence in the network.

1. Posterior for variable X:

$$p_{X|\mathbf{e}}(x) \propto \pi_X(x)\, \lambda_X(x) \tag{1}$$

2. Predictive support for X:

$$\begin{aligned} \pi_X(x) &= p_{X|\mathbf{e}_X^+}(x) \\ &= \int du_1 \ldots \int du_m \; q_X(x, u_1, ..., u_m) \\ &\quad \times \pi_{U_1,X}(u_1) \ldots \pi_{U_m,X}(u_m) \end{aligned} \tag{2}$$

3. Likelihood support for X:

$$\lambda_X(x) \propto p_{\mathbf{e}_X^-|X}(x) = \prod_{j=1}^{n} \lambda_{Y_j,X}(x) \tag{3}$$

4. Predictive message sent to child Y_k:

$$\pi_{X,Y_k}(x) = p_{X|\mathbf{e}\setminus\mathbf{e}_{X,Y}^-}(x) \propto \pi_X(x) \prod_{\substack{j=1 \\ j\neq k}}^{n} \lambda_{Y_j,X}(x) \tag{4}$$

5. Likelihood message sent to parent U_k:

$$\begin{aligned} \lambda_{X,U_k}(u_k) &= p_{\mathbf{e}\setminus\mathbf{e}_{X,U_k}^+|U_k}(u_k) \\ &\propto \int dx \int du_1 \ldots \int du_{k-1} \int du_{k+1} \ldots \int du_m \\ &\quad \times \lambda_X(x)\, q_X(x, u_1, ..., u_m) \prod_{\substack{j=1 \\ j\neq k}}^{m} \pi_{U_j,X}(u_j) \end{aligned} \tag{5}$$

Note that since a likelihood function is not a probability density, it need not be normalized to 1; a likelihood

[1] A particularly clear exposition of the polytree algorithm was given by Driver and Morrell [4].

function may integrate to any positive number, or, indeed, it need not be integrable at all. Any positive multiple of a likelihood function is again a likelihood function; likelihood functions are unique only up to a positive constant factor. For this reason, expressions involving likelihood functions are given as porportionalities instead of equalities in Eqs. 1 through 5; the constants of proportionality for Eqs. 1, 2, and 4 are simply those numbers which make the left-hand sides integrate to 1. The constants of proportionality for Eqs. 3 and 5 are arbitrary; it is sometimes convenient to make the likelihood function integrate to 1. However, for some purposes, the normalization of likelihood functions does matter, and incorrect results obtain if the constants of proportionality are ignored. This problem arises in the computation of π's and λ's when the distributions involved are mixtures.

For some combinations of types of functions, the result can be computed symbolically. Otherwise, an approximate result must be computed. A general scheme for computing such approximations is presented in Sec. 4. However, it seems likely that symbolic results could be obtained for a wider range of distributions than the ones mentioned already (discrete, conditional Gaussian, mixtures of conditional Gaussian). A catalog of the π and λ computations which can be carried out exactly, or with a close approximation, would be very useful in Bayesian network implementations. It would be useful even if not all the functions of interest (π_X, λ_X, etc.) can be computed exactly; if some result can indeed be computed exactly, let us do so, and postpone approximations until they are finally necessary.

3 Implementation

Software for representation and inference in heterogeneous Bayesian networks has been developed as a part of the RISO project [3]. RISO employs a polytree algorithm, which lends itself well to the following "lazy" or "just in time" computational scheme:

> To compute the posterior for X (or to compute π_X, or λ_X, or a predictive or likelihood message), compute only those functions which are required, then use those functions to compute the quantity of interest.

Probability distributions are represented within RISO as classes in the Java programming language with reasonably descriptive names, such as `Gaussian` and `ConditionalDiscrete`. To compute the posterior (or π_X, etc.), the inference code first computes any necessary partial results, such as incoming messages. Then the inference code uses the types of the partial results to search the local filesystem for a helper class which contains a function to compute the quantity of interest. The helper classes are grouped together into "packages" according to their purpose; all classes to compute a posterior will be found in the package `computes_posterior`. Likewise, classes to compute π_X will be found in `computes_pi`, and so on for classes to compute λ_X, etc.

For example, if the posterior is to be computed and π_X is represented by an object of class A and λ_X is represented by an object of class B, then the inference code attempts to find a helper class named `computes_posterior.A_B`. If no such class exists, the inference code attempts to locate a class named `computes_posterior.S_T` where `S` is `A` or a superclass of `A` and `T` is `B` or a superclass of `B`. This scheme makes it possible to construct code which handles both special cases (for the subclasses) or handles general cases (for the superclasses).

All classes which represent a probability distribution are subclasses of the abstract class `ConditionalDistribution` (if it is conditional) or `Distribution` (if it is unconditional). If an exact symbolic result is known for some combination of required functions, that result should be handled by a helper class named by the subclasses. Otherwise, we fall back on a helper class named according to the superclasses. The approximation scheme described in this document is implemented by superclass helpers; exact results are implemented by subclass helpers. At present, RISO can compute exact results in networks with discrete distributions, and rules for inference with Gaussian and Gaussian mixture distributions are under development. An approach for computing approximate results is described in the next section.

Since each type of distribution and each type of helper is represented by a different class, new types can be added without requiring modification of existing type definitions, and, above all, without requiring modification of the inference algorithm. This allows one to create the types suitable for particular applications, as in the example described in Sec. 6.

4 Approximating π's and λ's on the fly

To construct an approximation, RISO minimizes the cross-entropy between the target (the posterior for X, or π_X, or a π-message) and a Gaussian mixture. Likelihoods are not targets because they need not be normalized nor normalizeable, and so there may be no well-defined approximation. The cross-entropy calculation is just

$$H_{p,q}(\theta) = -\int p(x)\ \log q(x|\theta)\ dx \qquad (6)$$

denoting a target density as p and its Gaussian mixture approximation as q; the parameters of the approximation are denoted θ. The cross-entropy can be considered the continuous analog of the negative log-likelihood which appears in approximation problems based on measured data. Values of $p(x)$ are computing by directly evaluating the appropriate equation — in the case of π_X and λ_{X,U_k}, this requires numerical evaluation of integrals. Evaluating the cross-entropy itself also requires a numerical integration.

The cross-entropy is minimized by an expectation-maximization (EM) algorithm, also employed by Poland [9]; the following discussion of convergence of the algorithm is based on Wu [11]. EM algorithms are usually described in terms of discrete data, but the development can be extended readily to the approximation of a continuous function. Let x denote the variable or variables on which the target p and its approximation q are defined, and let y denote the "unobserved" variable or variables; in the mixture estimation problem, the mixture selector is the unobserved variable. Consider the expectation of the logarithm of the joint distribution of x and y under the approximation model with respect to possible instantiations of the unobserved variable,

$$\begin{aligned} Q(\theta,\theta') &= \int p(x) \int q(y|x,\theta') \log q(x,y|\theta)\, dy\, dx \\ &= -H_{q,q}(\theta,\theta') - H_{p,q}(\theta) \end{aligned} \tag{7}$$

writing the cross-entropy of $q(y|x,\theta')$ and $q(y|x,\theta)$ as

$$H_{q,q}(\theta,\theta') = -\int p(x) \int q(y|x,\theta') \log q(y|x,\theta)\, dy\, dx \tag{8}$$

Thus Q is related to the cross-entropy as

$$H_{p,q}(\theta) = -H_{q,q}(\theta,\theta') - Q(\theta,\theta') \tag{9}$$

Applying Gibbs' inequality to

$$\int q(y|x,\theta') \log \frac{q(y|x,\theta')}{q(y|x,\theta)}\, dy \tag{10}$$

we find

$$H_{q,q}(\theta',\theta') \leq H_{q,q}(\theta,\theta') \tag{11}$$

Let us suppose we have some initial value for the parameters, denoted θ'. The EM algorithm can be stated as these two steps:

- *E step:* Compute $Q(\theta,\theta')$.
- *M step:* Maximize Q over θ and assign the result to θ'.

These two steps are repeated until θ' seems not to change much, or Q seems not to change much, or we run out of patience. Each two-step iteration decreases the cross-entropy $H_{p,q}$: suppose we have found θ such that $Q(\theta,\theta') > Q(\theta',\theta')$. Then from Eqs. 9 and 11 we find

$$\begin{aligned} H_{p,q}(\theta) &< -H_{q,q}(\theta',\theta') - Q(\theta',\theta') \\ &= H_{p,q}(\theta') \end{aligned} \tag{12}$$

If, in a sequence of parameter updates we consider θ' to be a previous value and θ to be new value, the EM algorithm decreases the cross-entropy $H_{p,q}$. If the entropy of the target exists, the cross-entropy must have a point of accumulation, since the cross-entropy decreases with EM iterations and is bounded below by the entropy of the target distribution p.[2] It does not necessarily follow that the parameters θ likewise converge, although if the cross-entropy is unchanged, whether the parameters converge is inconsequential in the function approximation problem.

The EM algorithm is applied to the mixture approximation problem as follows. Let i denote the component selector; the number of components is m. The parameters are $\theta = (\alpha_1, ..., \alpha_m, \mu_1, ..., \mu_m, \sigma_1, ..., \sigma_m)$. The joint distribution of x and i is

$$q(x,i|\theta) = \alpha_i g(x;\mu_i,\sigma_i) \tag{13}$$

Denote the mixture approximation as

$$q(x|\theta) = \sum_{i=1}^{m} \alpha_i g(x;\mu_i,\sigma_i) \tag{14}$$

with g being the Gaussian density function, $g(x;\mu,\sigma) = \exp(-(x-\mu)^2/(2\sigma^2))/(\sigma\sqrt{2\pi})$. With these definitions, we have

$$\begin{aligned} Q(\theta,\theta') &= \int p(x) \sum_{i=1}^{m} \frac{q(x,i|\theta')}{q(x|\theta')} \log q(x,i|\theta)\, dx \\ &= \sum_{i=1}^{m} \int p(x) \frac{\alpha_i' g(x;\mu_i',\sigma_i')}{q(x|\theta')} \log \alpha_i\, dx \\ &+ \sum_{i=1}^{m} \int p(x) \frac{\alpha_i' g(x;\mu_i',\sigma_i')}{q(x|\theta')} \log g(x;\mu_i,\sigma_i)\, dx \end{aligned} \tag{15}$$

To maximize Q over the α_i, we need consider only the first term in Eq. 15. Let us define the "integrated responsibility" as

$$IR_i(\theta) = \int p(x) \frac{\alpha_i g(x;\mu_i,\sigma_i)}{q(x|\theta)}\, dx \tag{16}$$

[2] However, there exist proper distributions for which the entropy does not exist, such as $p(x) = x^{-1}(\log x)^{-2}$ for $x > e$. It is not clear how large a class of distributions this is.

Applying Gibbs' inequality again, we see that

$$\sum_{i=1}^{m} \int p(x) \frac{\alpha_i' g(x; \mu_i', \sigma_i')}{q(x|\theta')} \log \alpha_i \, dx \leq \sum_{i=1}^{m} IR_i(\theta') \log IR_i(\theta') \tag{17}$$

thus to maximize Q we choose

$$\alpha_i \leftarrow IR_i(\theta') \tag{18}$$

As for the other parameters μ_i and σ_i, we seek a stationary point of the second term in Eq. 15. Computing the gradient with respect to the μ_i and σ_i and setting the gradient to zero, we obtain

$$\mu_i \leftarrow \frac{1}{IR_i(\theta')} \int x \, p(x) \, \frac{\alpha_i' g(x; \mu_i', \sigma_i')}{q(x|\theta')} \, dx \tag{19}$$

$$\sigma_i^2 \leftarrow \frac{1}{IR_i(\theta')} \int (x - \mu_i')^2 \, p(x) \, \frac{\alpha_i' g(x; \mu_i', \sigma_i')}{q(x|\theta')} \, dx \tag{20}$$

In general, one should consider a maximum of Q at the boundaries of the parameter space as well as any stationary points within the parameter space. Since the μ_i may take on any real value, the only boundary to worry about is $\sigma_i = 0$. If the target density p is smooth, none of the components of the mixture approximation will have zero variance and we can safely ignore the boundary. On the other hand, if the target density has nonzero mass at some points (for example, if it is a mixture of discrete and smooth densities) then Q will reach a maximum for some $\sigma_i = 0$; the required derivatives do not all exist, and the assignment in Eq. 20 is not applicable. In this case, it seems the best course of action will be to place zero-width components at the points supporting discrete masses, and carry out the EM algorithm on the smooth density which remains.

Since over the course of several iterations of the cross-entropy minimization algorithm the target function will be evaluated repeatedly at the same or nearly the same argument, we can speed up the calculations by cacheing values of the target function. The cacheing algorithm is based on a self-balancing binary tree called a "top-down splay tree" [10]. Each node in the splay tree stores a key x and its associated function value $f(x)$; the nodes are ordered by increasing values of x. When a value of $f(x)$ is needed, the splay tree is searched for x. If x is contained in some node, the associated $f(x)$ is returned. Otherwise, if x is between two nearby values, the values associated with the neighbors are interpolated and the result is returned. Otherwise x is less than the least key in the tree or greater than the greatest key, or the neighbors of x are too far away; the value of $f(x)$ is computed, stored in the tree with key x, and returned.

On the average, searching a top-down splay tree requires a number of operations proportional to the logarithm of the number of keys in the tree. These operations are relatively fast, such as dereferencing memory addresses and comparing numbers. Since the target function may be defined in terms of numerical integrations which are relatively time-consuming, using a splay tree as a cache can yield a significant speed-up.

In this scheme which computes approximations to explicitly computed target functions, there is substantial bookkeeping. Target functions are constructed by keeping references to the necessary components functions (π- or λ-messages and the conditional distribution for a given variable), and then evaluating the components (integrating them if necessary) when an output of the target function is needed.

5 Numerical subleties

Finding the "effective support." To make numerical integrations easier, RISO tries to make the region over which we integrate as small as possible. Let us refer to a region which contains at least a mass $1 - \epsilon$, for a small number ϵ, as an "effective support" of the integrand. Note that a region which contains a mass $1 - \epsilon$ is not unique; RISO makes an effort to find the smallest effective support. It is important to obtain a small effective support because numerical integrations may fail if the integrand varies on scales that are much larger or much smaller than the region over which the integration is carried out. Also, to initialize a Gaussian mixture approximation (as described below), RISO attempts to find peaks in the target distribution over the effective support, and this search is more efficient when the effective support is as small as possible.

Integration algorithm. Numerical integrations in more than one dimensions are difficult. In the current implementation, multidimensional integrations are reduced to repeated one-dimensional integrations. The one-dimensional integrations are carried out by an adaptive region-splitting algorithm based on a Gauss-Kronrod 21-point rule. (The code is a translation of the QAGS algorithm from QUADPACK, a collection of quadrature algorithms available from `www.netlib.org`.) The adaptive quadrature algorithm can be "fooled" if the integrand varies on a scale much smaller than $I/42$, where I is the length of the interval of integration. For this reason, RISO tries to find the smallest effective support of the integrand, as described under the preceding heading.

Initial approximations. Since the EM algorithm yields only a local minimum of the cross-entropy, the initial approximation should be not too far from correct — otherwise we might come to a high local mini-

mum of cross-entropy. In particular, it pays to search for peaks in the target density, and initialize the approximation with corresponding peaks. RISO constructs an initial mixture approximation with a certain number of components with equal mass and variance, with their centers placed at regularly spaced intervals; these components are called "pavers," since they cover the support of the target like flagstones. Also, a mixture component is allocated for each peak which is found in the target density, as described in the following paragraphs.

RISO implements the following scheme for locating peaks in the target density. A uniform grid x_0, $x_1 = x_0 + h$, $x_2 = x_0 + 2h, ..., x_n$ is placed over the effective support (see below) of the target density. If a point seems to be a local maximum (i.e., if the density is greater at x_i than at x_{i-1} and x_{i+1}), then a component is added to the initial approximation mixture with mean equal to x_i and standard deviation calculated by fitting a Gaussian bump to the curvature of the target density. For convenience, translate from x_i to 0, with $u = x - x_i$. Let q_0 denote a function proportional to the Gaussian density with mean 0 and variance σ^2. Denote the mass $\int q_0(u)\,du$ as α. Then q_0 and its first two derivatives are

$$q_0(u) = \frac{\alpha}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{u^2}{\sigma^2}\right) \quad (21)$$

$$q_0'(u) = -\frac{u}{\sigma^2}\, q_0(u) \quad (22)$$

$$q_0''(u) = \frac{1}{\sigma^2}\left(\frac{u^2}{\sigma^2} - 1\right) q_0(u) \quad (23)$$

From this it follows that

$$q_0''(0) = -(1/\sigma^2)\, q_0(0) \quad (24)$$

Now given the curvature of the target density, estimated as

$$q_0''(0) \approx \frac{p(x_{i+1}) - 2p(x_i) + p(x_{i-1})}{h^2} \quad (25)$$

we can solve for σ to get an approximate standard deviation:

$$\sigma \approx \sqrt{-q_0(0)/q_0''(0)} \quad (26)$$

The mass of the peak is estimated as

$$\alpha \approx p(x_i)\,\sigma\,\sqrt{2\pi} \quad (27)$$

The corresponding mixing parameter is set to somewhat more than the estimated mass of the peak: a mixture component is allocated with its weight proportional to

$$\frac{1}{n} + p(x_i)\,\sigma\,\sqrt{2\pi} \quad (28)$$

where n is the number of peaks plus the number of pavers in the initial mixture. A paver is assigned a weight proportional to $1/n$.

Delaying integration of likelihood functions. Since likelihoods need not be normalizeable, they may not have an effective support smaller than all the reals. RISO does not integrate over a likelihood until there is a predictive distribution (which is guaranteed to be normalized) in the integrand. Thus approximations are never computed for λ_X or a λ-message, but only for π_X, a π-message, or the posterior of a variable. This is perhaps too pessimistic; there are examples of likelihood functions which do have bounded support, and which are therefore amenable to approximation.

Removing "unneeded" components. If an accurate approximation can be constructed using some number of components, then over iterations of the EM algorithm any additional components often become nearly the same (i.e., having the same mean and variance) as some other component. RISO makes an effort to find and remove such redundant components, by comparing the mean and variance of each component against the mean and variance of every other component. If the means are close enough and the variances are close enough, the components are redundant: one of them is removed and its weight is given to the other component. "Close enough" is assessed as follows:

> Let $r = \sigma_1/\sigma_2$, $\sigma^2 = 1/(1/\sigma_1^2 + 1/\sigma_2^2)$, and $\Delta\mu = \mu_1 - \mu_2$. If $\Delta\mu/\sigma < \beta$ and $|r - 1| < \gamma$, then components 1 and 2 are redundant.

Of course, the numerical factors β and γ can be adjusted to suit one's tastes; there does not appear to be a principled means of adjusting these parameters. At present, these are assigned the values $\beta = 0.25$ and $\gamma = 0.2$.

If, at the end of an EM iteration, a component has a mass less than some threshold, it is removed. In the present implementation, the mass threshold is 0.005.

Other heuristics for reducing the number of components in a mixture have been described [2, 1]. The application in these papers was to mixtures of discrete distributions, but the basic ideas should also apply to Gaussian mixtures as well.

Stopping criterion for cross-entropy minimization. It is not yet clear how to determine when an approximation computed by minimizing cross-entropy is "close enough." At present, the approximation algorithm runs for a fixed number of cycles; this yields acceptable results.

6 An example – radar cross-section

The radar cross-section (RCS) network is taken from another paper [5], which describes a polytree algorithm for Bayesian networks with continuous nodes (having mixtures of conditional Gaussians) and discrete nodes. In the original paper, a Gaussian mixture approximation for each distribution in the RCS network was computed, and then the inference algorithm was applied to the resulting approximations. In the present report, the RCS network is represented using the distributions natural to the problem, and approximations are computed only as needed to obtain the posterior distributions for the variables of interest.

The conditional distribution for the variable *RCS* depends on the discrete parent T and the continuous parent θ:

$$p_{RCS|T,\theta}(RCS, T, \theta) = p_\epsilon(RCS - F(T, \theta)) \qquad (29)$$

where the noise distribution p_ϵ is a unit-variance Gaussian centered on zero, and the cross-section function F is

$$\begin{aligned} F(T,\theta) \quad &= \quad A[T](\exp(-B[T](\theta - \pi/2)) \\ &\quad + \exp(-B[T](\theta - 3\pi/2))) - C[T] \end{aligned} \qquad (30)$$

The variable T indexes the parameters A, B, and C, as follows.

T	A	B	C
1	30	2	0
2	30	10	20
3	20	1.5	10

The prior distribution of θ is uniform over the interval $[0, 2\pi]$. The prior of T is also a uniform distribution, $[1/3, 1/3, 1/3]$. A Bayesian network in the RISO format which encodes this description of the RCS network can be found on the web.[3]

Let us first consider the posterior distribution of θ, given $RCS = 10$. This is a "diagnostic" inference, in that the observation is downstream from the variable of interest. First note that $F(T = 2, \theta)$ and $F(T = 3, \theta)$ both reach a maximum equal to 10 at $\theta = \pi/2$ and $3\pi/2$, while $F(T = 1, \theta)$ reaches a maximum equal to 30 at $\theta = \pi/2$ and $3\pi/2$; $F(T = 1, \theta)$ is equal to 10 at approximately $\theta = 0.9, 2.3, 4$, and 5.5. These features are reflected by Figure 1: the two large bumps at $\theta = \pi/2$ and $3\pi/2$ are due to the maxima of F for $T = 2$ and $T = 3$, while the smaller bumps occur where $F(T = 1, \theta)$ equals 10. The posterior computed by RISO contains six components, one for each of the peaks; all of the pavers were automatically removed from the mixture approximation. The non-Gaussian

[3] `http://riso.sourceforge.net/examples/rcs.riso`

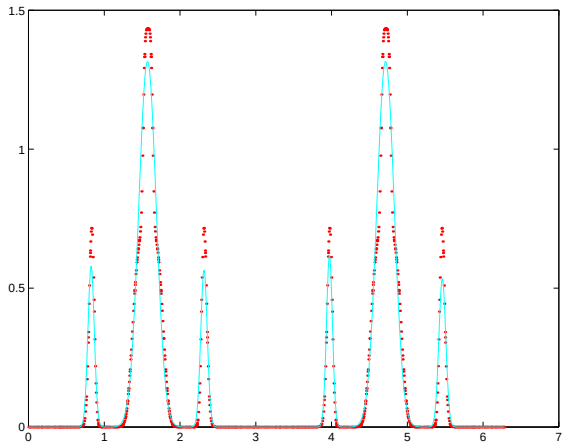

Figure 1: Posterior distribution of θ, given $RCS = 10$. Dotted line represents the approximation; dots are points at which the target (the posterior) was evaluated. Note the distinctly non-Gaussian bumps in the target at $\theta = \pi/2$ and $\theta = 3\pi/2$. The target was evaluated at about 1200 points.

peaks at $\pi/2$ and $3\pi/2$ could have been better captured by two components each — perhaps this indicates the heuristic for removal of duplicate components is overly zealous.

Now consider the posterior of *RCS* given $\theta = \pi$. This is a "predictive" inference. In this case there are three obvious components in the posterior, corresponding to $T = 1, 2, 3$, as shown in Figure 2. The masses of the components are very nearly the same because the prior over T is uniform.

Figures 1 and 2 show that the mixture approximations, while acceptable, slightly misrepresent the features of the posteriors. Perhaps one could represent the posterior or π_X or π-message by simply giving a list of points, such as are plotted in the figures. A list $(x_i, p(x_i))$ would be more verbose than a Gaussian mixture, but easier to compute, and perhaps more accurate as well.

7 Concluding remarks

At present, the most important unsolved problem in RISO is handling networks which contain loops. A conditioning algorithm is planned, perhaps using algorithms proposed by Xiang [12] to cooperatively detect loops in a distributed environment. However, it is not clear that it will be feasible to carry out conditioning in the presence of hetergeneous distributions — in particular, conditioning on continuous variables may well be troublesome.

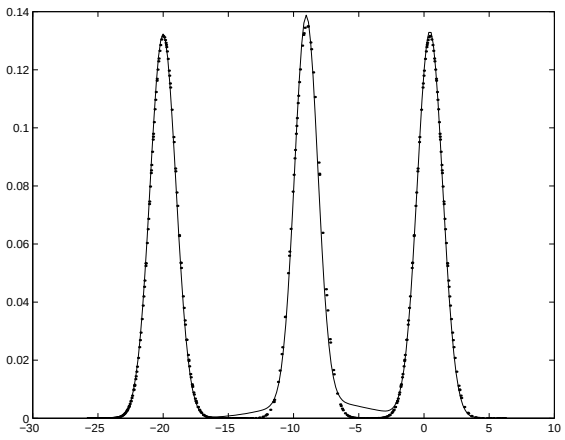

Figure 2: Posterior distribution of *RCS*, given $\theta = \pi$. Solid line represents the approximation; dots are points at which the target (the posterior) was evaluated. The target was evaluated at about 500 points.

There remain some numerical problems. While convergence of the EM algorithm is comforting, other algorithms such as the quasi-Newton algorithm might be faster. Sometimes the EM algorithm finds a local optimum which is clearly inferior; perhaps several random starting points should be tried, and the least cross-entropy approproximation kept. Also, computing repeated one-dimensional integrals is known to be inefficient in three or more dimensions. Quasi-Monte Carlo algorithms are often superior in high dimensions, and these algorithms are relatively easy to implement. It might also be profitable to combine exact inferences with Gibbs sampling, in the style of HUGS [6].

In closing, let us review the contributions of this report. A general scheme for inference in Bayesian networks containing distributions of various types, both continuous and discrete, has been described. Exact results are computed when such results are known, and approximations are computed otherwise. The approximation is applied only to the π_X, π-message, or posterior distribution required for a given inference, and the network is represented using the distributions most natural and convenient for the problem. New distribution types, as appropriate for the problem at hand, may be defined without disturbing the existing software. A number of implementation details, especially numerical problems arising in the computation of approximations, are described. It is hoped that other researchers will find this approach useful and inspiring.

Acknowledgements. This research has been supported by grants from the Link Foundation and the American Society of Heating, Refrigeration, and Air-conditioning Engineers.

References

[1] M. Baioletti. *Metodi computazionali per l'inferenza bayesiana con dati incompleti*. PhD thesis, Universitá degli Studi di Perugia, 1996.

[2] R.G. Cowell, A.P. Dawid, and P. Sebastiani. A comparison of sequential learning methods for incomplete data. In J.M. Bernardo, editor, *Bayesian Statistics 5: Proceedings of the Fifth Valencia International Meeting*, pages 533–542. Oxford: Clarendon Press; New York: Oxford University Press, 1996.

[3] R. Dodier. RISO: An implementation of distributed belief networks. In *Proc. AAAI Symposium on AI in Equipment Service*, 1999. To appear.

[4] E. Driver and D. Morrell. Implementation of continuous Bayesian networks using sums of weighted Gaussians. In P. Besnard and S. Hanks, editors, *Proc. 11th Conf. Uncertainty in Artificial Intelligence*. San Francisco: Morgan Kaufmann, 1995.

[5] E. Driver and D. Morrell. A new method for implementing hybrid Bayesian networks. Unpublished technical report, 1998.

[6] U. Kjaerulff. HUGS: Combining exact inference and Gibbs sampling in junction trees. In P. Besnard and S. Hanks, editors, *Proc. 11th Conf. Uncertainty in Artificial Intelligence*. San Francisco: Morgan Kaufmann, 1995.

[7] A. Kozlov and D. Koller. Nonuniform dynamic discretization in hybrid networks. In D. Geiger and P. Shenoy, editors, *Proc. 13th Conf. Uncertainty in Artificial Intelligence*. San Francisco: Morgan Kaufmann, 1997.

[8] J. Pearl. *Probabilistic reasoning in intelligent systems*. San Francisco: Morgan Kaufmann, 1988.

[9] W. Poland. *Decision analysis with continuous and discrete variables*. PhD thesis, Stanford University, Dept. of Engineering-Economic Systems, 1994.

[10] D. Sleator and R. Tarjan. Self-adjusting binary search trees. *J. Assoc. Computing Machinery*, 32(3):652–686, 1985.

[11] C.F.J. Wu. On the convergence properties of the EM algorithm. *Annals of Statistics*, 11(1):95–103, 1983.

[12] Y. Xiang. Verification of DAG structures in cooperative belief network based multi-agent systems. *Networks*, 31:183–191, 1998.