RISO: An Implementation of Distributed Belief Networks

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

Abstract

This paper describes RISO, an implementation of distributed belief network software. Distributed belief networks are a natural extension of ordinary belief networks in which the belief network is composed of subnetworks running on separate processors. In keeping with the distributed computational model, no single processor has information about the structure of the entire distributed belief network, and inferences are to be computed using only local quantities. A general policy is proposed for publishing information as belief networks. A modeling language for the representation of distributed belief networks has been devised, and software has been implemented to compile the modeling language and carry out inferences. Belief networks may contain arbitrary conditional distributions, and new types of distributions can be defined without modifying the existing inference software. In inference, an exact result is computed if a rule is known for combining incoming partial results, and if an exact result is not known, an approximation is computed on the fly — this scheme allows the belief networks to directly represent the distributions that arise in practice. Some features of Java, the implementation language, have been found to be extremely useful, namely the classloader and Remote Method Invocation. The paper concludes with a small example which shows how a monitoring system can be implemented as a distributed belief network.

Introduction

Distributed belief networks are useful models for diagnosis and prediction in geographically or functionally distributed systems. Belief networks for such systems have lately been the object of study by Xiang (Xiang 1996) and his students (Chu & Xiang 1997; Hu & Xiang 1997) who refer to "multiply-sectioned Bayesian networks" which have a certain strict definition. In this paper the term "distributed belief network" will be used rather loosely to mean a probability model represented as an acyclic directed graph and implemented on multiple processors which communicate by means of a standard networking protocol; that part of a distributed belief network implemented on a particular processor will be referred to as a "component network" or "subnetwork." These definitions are deliberately vague, so as to include a wide range of interesting architectures and implementations. Broadly speaking, we will be as much interested in the "distributedness" as in the "belief-networkness" of distributed belief networks, and the interaction between these two aspects will lead to interesting problems.

Figure 1 shows a typical distributed belief network. Each one of the components A, B, C, and D is itself a belief network. The variables within the belief networks are identified by *network.variable*. For example, variable t in subnetwork A is referred to as "*A.t*." This scheme avoids name conflicts when variables in two networks have the same name. Note that in the example network, there is one loop (that is, an undirected cycle) wholly contained within subnetwork A, but there is another loop induced by the dependencies of B and C on variables in A. Such induced dependencies make life difficult for inference algorithms; a local algorithm for inference in a distributed system is desirable, but may not be feasible in the presence of loops.

The representation of geographically distributed systems is perhaps the prototypical application of distributed belief networks, although by no means the only application. It is natural to represent each geographical unit with a belief network, and to represent the flow of information from one locale to another by edges connecting variables in separate belief networks; different kinds of messages travel with the arrows and against the arrows. Such a scheme is all the more reasonable if the cost, time lapse, or difficulty of transferring information from one locale to another is large compared to the computations required for inference within one belief network. In that case, we will want to locally carry out as much computation as possible, and only transmit messages when necessary.

In addition to modeling geographically separate systems as separate belief networks, one can use distributed belief networks as a means of decomposing the modeling problem so that modeling of the pieces can proceed independently. For the same reasons that other kinds of software are customarily constructed piece by piece, it could be convenient to construct sub-networks separately, then connect them together through common variables. A decomposition into sub-networks will make it easier to comprehend the model and present

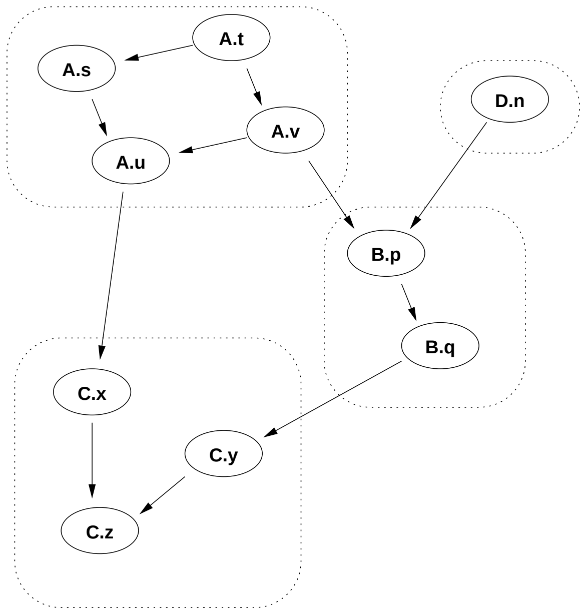

Figure 1: An example of a distributed belief network, composed of four subnetworks, A, B, C, and D.

it to other people. The components of a distributed belief network can be treated as independent software modules and reused for purposes other than those for which they were originally constructed. A great deal of work may go into the design and implementation of a belief network for a particular engine, reactor, pump, compressor, etc., and it will greatly speed development for new applications if the belief networks previously devised for similar systems can be reused. Generic belief networks for some special purposes, such as sensor models, time-series filtering, and hidden Markov models, can be easily constructed; it is foreseen that one could make available a library or archive of belief network components, which could be used as templates or building blocks from which particular applications could be constructed.

It is assumed that the reader is familiar with belief networks. Should this not be the case, the reader is referred to the introductory texts by Pearl (1988) and Castillo et al. (1996). An excellent introduction to probability and its use as a reasoning tool is given by Jaynes (1996).

The RISO project thus far

A prototype implementation of distributed belief networks, named RISO,[1] has been constructed. Progress has been made in several areas, in both theoretical and practical matters. On the whole, the theoretical development is mostly complete, as is most of the software development, but specific applications of RISO to practical prediction and diagnosis problems are still underway. Let us now survey the progress of the RISO project.

Representation of belief networks with heterogeneous distributions. A necessary first step towards a flexible and extensible modeling system for general applications is the ability to represent belief networks composed of arbitrary conditional probability distributions. In order to make the belief network more comprehensible to domain experts, and to simplify the specification and modification of the belief network, probability distributions from the problem domain should be directly represented in the belief network. Software has been implemented in RISO which makes it possible to represent several kinds of basic distributions, and RISO is designed so that additional types of distributions can be implemented without changing the existing code at all. In contrast, most currently available belief network software can only handle a few well-known types of distributions, and it is assumed that if a distribution arising in a problem does not fall into a known category, then that distribution will be represented by an approximation belonging to a known category.

Inference in polytrees with arbitrary distributions. RISO implements a "just in time" approximation scheme for inference with arbitrary conditional distributions described in another report (Dodier 1998a). The RISO inference algorithm is based on the polytree algorithm for belief network inference, in which "messages" (predictive distributions called π-messages and likelihood functions called λ-messages) are computed; the equations for π- and λ-messages are given, for example, in Dodier (1998d). 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 usually 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 at most a few types, this heterogeneous polytree algorithm can, in principle, handle any kind of continuous or discrete conditional distribution.[2] With standard algorithms, it is necessary to construct an approximate belief network, in which one then computes exact results; the heterogeneous polytree algorithm, on the other hand, computes approximate results in the original belief network. The advantages are that the approximations computed by the new algorithm are all one-dimensional and thus easier to compute, and, more importantly, that the belief

[1]So called because the nodes in belief network diagrams look like nice plump grains of rice; see Figure 1.

[2]A different approach to solve the problem of inference in heterogeneous belief networks is taken by the BUGS ("Bayesian inference Using Gibbs Sampling") software, described by Gilks et al. (1994).

network can be directly represented using the conditional distributions most appropriate for the problem domain.

Implementation of distributed belief networks. In extending the usual single-processor computational model to multiple processors, several interesting problems arise, which must be solved for the successful implementation of a distributed belief network. In keeping with the distributed computational model, no single processor has information about the structure of the entire distributed belief network, and inferences are to be computed using only local quantities. However, this leads to difficulties (which have not yet been resolved) when there are loops in the distributed belief network which contain nodes in two or more component networks. Also, temporal dependencies in one network lead to temporal dependencies in any other network connected to some of its variables; this may lead to intractable dependencies, especially if some of the dependencies involve different time scales. While these problems of induced dependencies have only been identified, and not resolved, progress has been been made in other areas. A general policy is proposed for publishing information as belief networks. A modeling language for the representation of distributed belief networks has been devised, and software has been implemented to compile the modeling language and carry out inferences.

Communication in distributed belief networks

In this section, let us briefly review some of the issues surrounding communication in distributed belief networks. These topics are explored at greater length in a technical report (Dodier 1998a).

Local versus global control of communication. Throughout this paper it will be assumed that communications between the components of a distributed belief network will be restricted to immediate neighbors. That is, belief network A can communicate with another, B, only if some variable in A has a parent or child in B. This local communication model, inspired by the idea of belief networks representing independent agents, is to be contrasted with a global communication model in which a central mechanism organizes communications between all components of a distributed belief network. The local communication model is better suited to a system in which component belief networks are created and executed independently, for potentially different purposes.

Publishing information as distributed belief networks. There is considerable promise in the idea of publishing information on the Internet in the form of belief networks. Just as people can now connect to data sources in the form of files or Common Gateway Interface (CGI) and then reprocess the information for their own purposes, it should be possible to connect to belief network interfaces to obtain information in the form of probability and likelihood messages, which can then be used for purposes not specifically foreseen by the originators of the belief network. The original belief network together with any variables which are created as descendants of its interface variables will comprise a larger belief network. The new child variables may in turn be linked to still others through their own interface variables; a belief network of any size could be constructed in this piecemeal fashion.

Allowing a belief network to reference variables in another raises interesting questions concerning the permission to pass messages within the overall belief network, a topic considered at greater length in a technical report (Dodier 1998b).

Computing inferences in distributed belief nets. To accomodate the geographically and functionally distributed belief networks which are the focus of this paper, it is of great important that the inference algorithms used allow for locality of computations and heterogeneous conditional distributions. Toward this end, the central inference algorithm of RISO is the polytree algorithm (Pearl 1988). Unfortunately, this algorithm has limited applicability to belief networks which contain loops (that is, undirected cycles). In RISO loops will be handled by a conditioning algorithm (Pearl 1988; Castillo, Gutierrez, & Hadi 1996), using a loop-cutset algorithm (Becker & Geiger 1994), but the details of this scheme have not yet been worked out. On the brighter side, the polytree algorithm does lend itself well to accomondating different types of conditional distributions.

Software implementation of distributed belief networks

A prototype implementation of some of these ideas about distributed belief networks has been constructed using the Java programming language, in particular the Remote Method Invocation (RMI) mechanism for passing information between sub-networks. As a Java application, RISO will run on any hardware that supports the Java virtual machine. Furthermore, RISO has been designed without a graphical user interface, to make it possible to run the software on machines without a display; a rudimentary user interface has been constructed for testing purposes.

In contrast to some other implementations of distributed systems for inference (e.g., the Integrated Diagnostic System (Wylie *et al.* 1997)) RISO is not designed to link diagnostic systems based on different software architectures and different reasoning paradigms. Probability was chosen as the sole medium for the expression and exchange of information, for theoretical reasons (Cox 1946; Jaynes 1996) which suggest that other paradigms for reasoning under uncertainty are either essentially the same as probability or else fundamentally weaker.[3] Choosing only one representation

[3]This argument does assume the acceptance of a certain set of desiderata; see Jaynes (1996) for details. Reasoning systems satisfying other desiderata might be considered for

of uncertainty also greatly decreases the complexity of the software implementation. Choosing a single implementation language, and a single collection of classes within that language, also greatly simplifies the implementation. The choice of a strictly probabilistic system implemented in Java has the advantages of unity, simplicity, and comprehensibility, both conceptual and practical.

A parser for the RISO belief network grammar has been implemented; the RISO grammar is a revision of the proposed Belief Network Interchange Format (BNIF) grammar. The RISO grammar is described at greater length in a technical report (Dodier 1998c). In the RISO grammar, each belief network is allocated a namespace. Within each namespace, identifiers must be unique. However, the same local identifier may occur in two different namespaces. To resolve an ambiguous reference, an identifier is qualified with the "." operator; an unqualified identifier is assumed to exist in the current namespace. Namespaces cannot be nested. The name of a belief network coincides with the name of the file which contains it. The extension of the filename is "`riso`". For example, a belief network named "*sensor-diagnosis*" is contained in a file named "`sensor-diagnosis.riso`". Each belief network file contains exactly one belief network.

A qualified variable name of the form *some-bn.x* refers to a variable in a belief network on the same host as the network in which the reference occurs. If the belief network *some-bn* is not already loaded, then it is loaded from the file `some-bn.riso` on the local filesystem, and the variable x is sought within it. A qualified variable name of the form *some-host/some-bn.x* refers to a variable in a belief network on the same or a different host. The belief network *some-bn* is located by connecting to an RMI daemon listening on a specified port on *some-host*, and x is sought within the belief network. The hostname can be a fully-qualified symbolic Internet address, although one need specify only enough to locate the host. The address can include a port number, e.g. *cedar.colorado.edu:2099*. These rules for identifiers allow software to automatically locate belief networks referred to by another. Belief network developers can break networks into separate components, thus increasing comprehensibility and distributing computations.

The polytree algorithm lends itself well to the following "lazy" computational scheme:

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

Probability distributions are represented within the belief network code as Java classes with descriptive names, such as `Gaussian` and `ConditionalDiscrete`. To compute the posterior (or π_X, etc.), the inference code first computes the required functions by some means. Then the inference code uses the types of the required functions to search the local filesystem for a helper class which contains a function to compute the quantity of interest. The helper classes will be grouped together into "packages" according to their purpose; all classes to compute a posterior will be found in the package `computes_posterior`, likewise other functions will be computed by classes in other packages.

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 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 special cases (for the subclasses) and handles general cases (for the superclasses).

Each class which represents a probability distribution is a subclass 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 mentioned in the next section and described in more detail by Dodier (1998b) is implemented by superclass helpers; exact results are implemented by subclass helpers.

Approximating π's and λ's on the fly

To construct an approximation, RISO minimizes the cross-entropy between the target (which is the posterior for a variable X, or π_X, or a π-message, but not λ_X or a λ-message) and a Gaussian mixture. (Approximations are not computed for likelihoods because likelihoods need not be normalized nor normalizeable, and so there is no well-defined approximation.) An algorithm reminiscent of the expectation-maximization algorithm is employed, as described by Poland (1994). The cross-entropy calculation is just

$$\int p(x)\ \log q(x)\ dx,$$

denoting a target function as p and its Gaussian mixture approximation as q. 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.

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

a distributed diagnostic system, but such a system would be fundamentally incompatible with a probabilistic system.

is based on a self-balancing binary tree called a "top-down splay tree" (Sleator & Tarjan 1985). 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.

There are various numerical subtleties to consider, which are described in a technical report (Dodier 1998d). An initial approximation is constructed by searching for "bumps" in the target density; finding the bumps in an arbitrary density can be difficult. Numerical integration in more than a few dimensions is also difficult. RISO uses the QAGS algorithm from QUADPACK.[4] This is an adaptive one-dimensional quadrature algorithm based on a 21-point Gauss-Kronrod rule. Integrations in two or more dimensions are carried out as repeated one-dimensional integrations. To make integrations easier, RISO tries to find an "effective support" for the integrand which is as small a region as possible; an effective support is an interval or collection of intervals which contains at least $1 - \epsilon$ of the mass of the distribution, with ϵ a suitable small number. Since a likelihood function need not have a bounded support, computations involving likelihoods are delayed until the integrand contains the product of probability density with the likelihood — this guarantees a bounded effective support.

Communications problems, and solutions

Several interesting problems arise when a belief network is implemented in a distributed computing system. The usual network communication problems take on forms peculiar to belief network computations. Among these problems, the following have been handled in the design of RISO.

Locating and connecting belief networks on different hosts. When a belief network is loaded, RISO advertises its name in a globally–visible list, called the "registry." Belief networks in other processes on the same host or on different hosts can use the registry to obtain a pointer to any registered belief network. If the parent *spruce/weather.humidity* referred to by a belief network cannot be located in the registry on the host *spruce*, an attempt will be made to have the *weather* belief network loaded onto *spruce* so that *weather* becomes available; this is similar in spirit to the resolution of function references in libraries. If the host name is omitted, the host of the belief network in which the reference occurs is assumed. There will usually be a process running on each host which can load any belief network from a description on the file system of that host. However, only a short program need be installed on each host, and additional software can be loaded as needed from another host. In particular, classes named in a belief network description will be copied (by the Java runtime software) to the host on demand; the complete RISO software need be installed on just one machine.

Communicating π- and λ-messages between belief networks. The messages transferred between belief networks are probability distributions and likelihood functions. These are expressed in parametic forms, which may be very short (e.g., a Gaussian can be described by just a few numbers) or very long (e.g., a mixture of Gaussians may contain an arbitrary number of parameters). Probabilities and likelihoods are represented as objects within the RISO software, and these objects are automatically converted into a block of data which is transferred across a socket connection by RMI. Thus a request for a message from a remote belief network is implemented as a function call, and the message which is returned appears to be the return value of the function. As part of the general "lazy" computational policy of RISO, messages are requested only when they are needed; this cuts down on the relatively large overhead of passing messages between remote belief networks.

Other kinds of messages are passed in distributed belief networks. When evidence is entered or removed from a node X in the network, messages are sent to all parents and children of X, telling them that the π- and λ-messages originating from X are no longer valid; these "invalid π- or λ-message" messages are propagated as appropriate to other nodes not d-separated from X. Any node receiving such a message knows that its posterior must be recomputed, but the computation is postponed until a request for the posterior is received.

Coping with communication failures, which occur, for instance, when a host crashes or the process running a belief network is killed. When a π- or λ-message is required and an attempt to communicate with the corresponding parent or child node fails, RISO attempts to re-establish the link using the same registry look-up algorithm by which the link was originally established. If the attempt to reconnect fails as well, in order to make some progress RISO assumes that no evidence is available through the lost parent or child. Due to the distinction between parents and children in the

[4] QUADPACK is a collection of quadrature algorithms; it is available from Netlib, `www.netlib.org`.

computation of a posterior distribution, lost parents are treated differently from lost children. A lost child is simply dropped from the list of children of the variable which requested the λ-message, since in the absence of evidence from the child, the child has no effect on the computation of the posterior. A lost parent must be kept on the list of parents, but until the parent becomes available again (through restarting the process running the belief network), the prior for the parent will be substituted for any request for a π-message from that parent, since in the absence of evidence the π-message from the parent will simply be the parent's prior. The parent's prior is sent from the parent to any remote child when the child first makes contact with the parent.

Security issues. In distributed belief networks, there are the usual problems of who can access which data, and these problems can be handled by well-known authentication, authorization, and encryption algorithms. However, there is at least one security problem which is peculiar to belief networks, namely that the naming X as the parent of Y implies (according to the laws of probability) the transfer of information from X to Y but also from Y to X. This suggests that one could affect degrees of belief in a network maintained by the Weather Service (let us say) by connecting some child variables and then introducing evidence into our sub-network; this problem and others are discussed at greater length in a technical report (Dodier 1998b). It is foreseen that the each belief network should be able to specify which others can propagate information up from child nodes, but this policy has not been implemented yet in RISO.

Example: Monitoring a Distributed System

Consider the problem of monitoring a geographically distributed system. Let us model each component of the system as a separate belief network, each of which has a discrete status variable and one or two variables on which we can make measurements; in what follows we'll call these "measurable" variables. (We will distinguish between the variable we are trying to measure and the measurement itself. The measurement may be more or less accurate, and can be affected by a failure of the sensor or measuring device. The *monitor3* belief network shows a simple measurement model, comprising a measurable variable, the sensor status, and the measurement.) The monitor is represented as a belief network which contains only one variable, a "or" gate which computes the probability that at least one of the status variables is non-zero. We adopt the convention that status equal to 0 represents the normal status, and any other value represents an abnormal status.

The RISO code for the monitor subnetwork names the parents of the "or" gate as the status variables of the three individual monitor subnetworks.

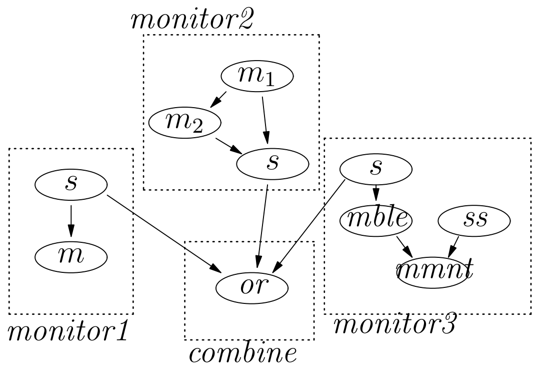

Figure 2: A distributed belief network for monitoring geographically distributed equipment. Key: s = "status," m or *mble* = "measurable," *mmnt* = "measurement," and *ss* = "sensor status."

```
BeliefNetwork combine
{
  Variable or
  {
    type discrete { "all OK"
       "at least one failure" }
    parents { rtt/monitor1.s
       beethoven/monitor2.s
       civil/monitor3.s }
    distribution OrGate
  }
}
```

In this belief network and others, the "`BeliefNetwork`" tag not only begins the belief network description, but names the Java class which knows how to parse the description and which implements the various message-passing functions necessary for computing inferences. Likewise, "`Variable`" begins the description of a variable within the belief network and also names the class which parses the description and implements the functions needed for a variable. In this scheme, one could implement software which accepts an alternative description by simply extending the `BeliefNetwork` or `Variable` class. For example, no provision has been made in RISO for representing display information such as the color and position of variables' nodes, but such information could be stored by a `FancyVariable` which extends `Variable`. More importantly, a conditional distribution represented in a RISO belief network description is tagged with the name of the class which implements it, and that class contains the code to parse the description and implement the probability functions needed for the inference algorithm. Thus special-purpose distributions can be created as the need arises in an application, and existing code for belief network objects and for variables need not be changed, including, above all, the inference algorithm.

Let us briefly consider the distributions encoded in the belief networks in this example.[5] In the interest

[5]The belief networks discussed in this example are in-

of brevity, only the description of *monitor2* is shown. The *monitor1* belief network contains a simple "naïve Bayes" model. In *monitor2*, variable *mmnt* has a conditional Gaussian dependence on *mble*, and *s* is a logistic discriminant model with two classes. Thus *monitor2* shows a conventional classification model, which computes class probabilities conditional on its inputs *m1* and *m2*, integrated into the monitoring system.

```
BeliefNetwork monitor2
{
  Variable s {
    type discrete { "OK" "goofed" }
    parents { m1 m2 }
    distribution SquashingNetworkClassifier
      { ...  } }

  Variable m1 { distribution Gaussian
      { mean 30 std-deviation 4 } }

  Variable m2 {
    parents { m1 }
    distribution ConditionalGaussian {
      conditional-mean-multiplier { 0.3 }
      conditional-mean-offset { 8 }
      conditional-variance { 28 }
    }
  }
}
```

In *monitor3*, a naïve Bayes model is extended with a simple measurement model. The measurable variable *mble* is not directly known; only the measurement *mmnt* is observed. The measurement depends on the status of the sensor *ss*, as well as the measurable variable. The measurement is a noisy function of the measurable variable when the sensor is working correctly, and the measurement is just zero when the sensor is broken; this is common in sensors which output a voltage. However, in the model specified in *monitor3*, a measurement of zero can also occur when the sensor is working correctly. Finally, *combine* contains a single variable, *or*, which represents the probability that any of its parents (the status variables *monitor1.s*, *monitor2.s*, and *monitor3.s*) is non-zero.

Each belief network is running on a different host: *monitor1* on *rintintin*, *monitor2* on *beethoven*, *monitor3* on *civil*, and *combine* on *sonero*. The first three are Solaris machines, and the last is a GNU/Linux machine. The description of *combine* specifies, in the list of parents for the *or* variable, which belief network is running on which host.

First, let us set *monitor2.m1* to -150. Querying *monitor2.s* (that is, computing the its posterior), we find $p(s = \text{"OK"} | m1 = -150) = 0.2974$. This result averages over values of the missing variable *monitor2.m2*; querying *m2* we see that its posterior is a Gaussian distribution with mean -37 and standard deviation 5.2.

tended for illustration only, in particular the transmission of messages between variables. The complete RISO description files for these belief networks can be found on the web.[6] For clarity, some identifiers have been abbreviated in this paper.

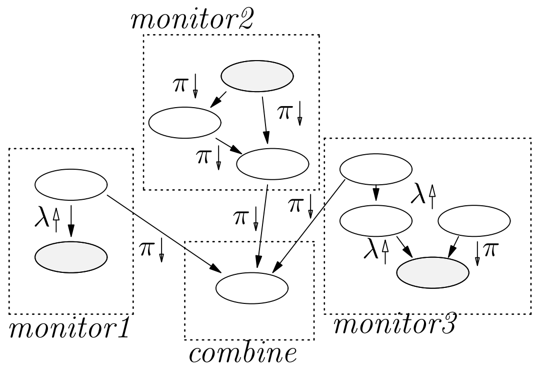

Figure 3: π- and λ-messages transmitted within a distributed belief network to satisfy a query on *combine.or*. Evidence nodes are shaded.

So far, no messages have been passed from one belief network to another. Now let's set *monitor3.mmnt* to 23 and query *combine.or*. A π-message is sent from each monitoring belief network to the "or" gate. For *monitor1.s*, this is just its prior since no evidence has been introduced, and for *monitor2.s*, the π-message is just the posterior which was computed a moment ago. But for *monitor3.s*, computing the π-message to *combine.or* requires that evidence be propagated up from *monitor3.mmnt*; *ss* sends a π-message to *mmnt*, which sends a λ-message to *mble*, which sends a λ-message to *s*, which then sends the π-message to *combine.or*. Note that none of the messages in *monitor3* were computed until *combine.or* asked for a π-message. We find the posterior of *combine.or* has the probability that *or* = "all OK" is 0.2759, given the upstream evidence in *monitor2* and *monitor3*.

Introducing evidence *monitor1.m* $= 57$, the posterior probability of *or* = "all OK" drops to 0.01705. Now there is evidence in all three monitor belief networks. Figure 3 shows the messages which have been communicated between the variables in the four belief networks comprising our distributed monitoring system. Each message is requested from the variable which needs the it — a message is not constructed unless it is required; this is the "lazy" computational policy. Also, messages within a belief network (that is, within the boxes shown in Figure 3) are transferred as return values from functions, while messages between belief networks are transmitted as blocks of data on a socket connection, and reconstituted into program objects by their recipient.

Let's see what happens in two scenarios involving a failure. The first scenario is a communication failure: *beethoven* has crashed, leaving *monitor2* inaccessible. If we query *combine.or*, an attempt is made to contact *monitor2* to supply a π-message. The attempt fails, so *combine.or* uses the marginal prior for *monitor2.s* which was computed when *combine.or* first connected to *monitor2.s*. The prior for *monitor2.s* gives probability of "OK" equal to 0.03227, and this value is used to update *combine.or*, yielding the probability of "all OK" equal to 0.001833. Note that the prior over *monitor2.s* must be computed by integrating over its parents *m1* and *m2*; priors for status variables in *monitor1* and *monitor3*, which are root variables, are specified directly.

The second scenario is a sensor failure in *monitor3*. The sensor fails, and its output is zero. Querying *monitor3.ss*, we find that $p(\text{“sensor OK”} \mid mmnt = 0) = 0.2309$. Although the likelihood for "sensor OK" indicates that the measurement $mmnt = 0$ is much more typical of a failed sensor than one operating correctly, with

$$\frac{p(mmnt = 0 \mid \text{“sensor OK”})}{p(mmnt = 0 \mid \text{“sensor not OK”})} = 0.003032$$

the posterior for "sensor OK" is appreciably greater than zero due to the 99 : 1 prior odds in favor of "sensor OK." Since *mmnt* is the common descendent of *s* and *ss*, information can travel from *ss* to *s* via *mmnt*. Querying *s*, we see that the posterior probability of "OK" is 0.9213, not much different from its prior value of 0.9000. However, if we set the sensor status *ss* equal to "OK," we find a greater change in the posterior of *s*. Now the probability of *s* = "OK" is 0.9755. The effect of the measurement on the status variable *s* was weakened because the evidence favored a failed sensor.

Continuing research

The "just in time" inference scheme is, regrettably, not as robust as one might hope; however, the difficulties are numerical, so there is hope. An investigation into more sophisticated integration algorithms, perhaps of the quasi Monte Carlo type, will be made. RISO will be extended with a greater number of predefined conditional distributions (which will reduce the number of cases in which approximate results need to be computed), and a conditioning algorithm will be implemented to handle loops, although it is not yet clear how such an algorithm can be implemented without global connectivity information. Finally, nontrivial applications involving modeling and diagnosis in engineering problems will be implemented as RISO belief networks.

References

Becker, A., and Geiger, D. 1994. Approximation algorithms for the loop cutset problem. In de Mantaras, R. L., and Poole, D., eds., *Proc. 10th Conf. on Uncertainty in Artificial Intelligence*, 60–68. San Francisco: Morgan Kaufmann.

Castillo, E.; Gutierrez, J.; and Hadi, A. 1996. *Expert Systems and Probabilistic Network Models.* New York: Springer-Verlag.

Chu, T., and Xiang, Y. 1997. Exploring parallelism in learning belief networks. In Geiger, D., and Shenoy, P., eds., *Proc. 13th Conf. on Uncertainty in Artificial Intelligence.* San Francisco: Morgan Kaufmann.

Cox, R. 1946. Probability, frequency, and reasonable expectation. *Am. J. Physics* 14(1):1–13. Reprinted in (Shafer & Pearl 1990).

Dodier, R. 1998a. An algorithm for inferences in a polytree with arbitrary conditional distributions. Technical report, U. Colorado at Boulder. URL `http://civil.colorado.edu/~dodier/-publications.html`.

Dodier, R. 1998b. An overview of distributed belief networks in engineering systems. Technical report, U. Colorado at Boulder. URL `http://civil.-colorado.edu/~dodier/publications.html`.

Dodier, R. 1998c. A revision of the Bayesian network interchange format. Technical report, U. Colorado at Boulder. URL `http://civil.colorado.-edu/~dodier/publications.html`.

Dodier, R. 1998d. Tools for unified prediction and diagnosis in HVAC systems: The RISO project. Technical report, U. Colorado at Boulder. URL `http://civil.colorado.edu/~dodier/-publications.html`.

Gilks, W.; Thomas, A.; and Spiegelhalter, D. 1994. A language and program for complex Bayesian modelling. *The Statistician* 43:169–178.

Hu, J., and Xiang, Y. 1997. Learning belief networks in domains with recursively embedded pseudo independent submodels. In Geiger, D., and Shenoy, P., eds., *Proc. 13th Conf. on Uncertainty in Artificial Intelligence.* San Francisco: Morgan Kaufmann.

Jaynes, E. 1996. *Probability theory: the logic of science.* Unpublished MS; URL `ftp://bayes.wustl.-edu/Jaynes.book/`.

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

Poland, W. 1994. *Decision analysis with continuous and discrete variables.* Ph.D. Dissertation, Stanford University, Dept. of Engineering-Economic Systems.

Shafer, G., and Pearl, J., eds. 1990. *Readings in uncertain reasoning.* San Mateo, CA: Morgan Kaufmann.

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

Wylie, R.; Orchard, R.; Halasz, M.; and Dub, F. 1997. IDS: Improving aircraft fleet maintenance. In *Proc. 14th Nat'l Conf. Innovative Applications of Artificial Intelligence (IAAI-97)*, 1078–1085.

Xiang, Y. 1996. A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. *Artificial Intelligence* 87(1):295–342.