Distributed Bayesian Networks for patterns representation

J. M. Medina-Apodaca & J. Figueroa-Nazuno
Centro de Investigación en Computación, Instituto Politécnico Nacional

Abstract

In this paper is presented the design and implementation of a Distributed Bayesian Networks building tool, which can be used for collaborative work in the tasks of human experts knowledge representation. The system allows the users to build a new network, as well as joining an existing one which may be physically distributed along several remote places. Once built, a Bayesian Network contains all the information about conditional probabilities among a set of involved variables. Each group of Bayesian Networks is made accessible to its users by a centralized repository for such group. The interfaces for the system have been specified using CORBA's IDL. At this moment, the system is implemented with the Java language.

Keywords: Bayesian Networks, artificial intelligence, knowledge representation, distributed systems.

1. Introduction.

Let X, Y be random variables for a random experiment with a sample space and probability measure P. Suppose X takes values in sample space S and Y takes values in sample space T. The distributions of X, Y are called marginal distributions.

If X and Y are independent, the probability of A and B is given by the product of their individual probabilities:

$$P(A,B) = P(A)P(B)$$

If A and B are not independent, i.e. the value for A depends on the value of B, the probability of A given that B's occurrence is given by the expression:

$$P(A|B) = \frac{P(A,B)}{P(B)}$$

We have $P(A,B) = P(B|A)P(A)$. Using these equations, we obtain the Bayes Rule:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)}$$

The law of total probability asserts that if $\{A_1, A_2, \dots, A_n\}$ is a set of independent variables whose space is a partition of the sample space, and if a random variable B is dependent of variables $A_1, A_2, \dots, A_n$, then the probability of B is given by:

$$P(B) = P(B|A_1)P(A_1) + P(B|A_2)P(A_2) + \cdots + P(B|A_n)P(A_n)$$

The joint distribution of variables X, Y is a distribution which takes values in a [1]. The importance of the joint distribution is that it stores more information marginal distributions, i.e. the marginal distributions can be obtained from distribution, but the joint distribution cannot be obtained from the marginal distributio

More generally, if we have n variables $V_1, V_2, \ldots, V_n$ with corresponding spaces S_1, where space S_i contains *m* values $v_{i1}, v_{i2}, \ldots, v_{im}$, the joint distribution space is $S_1 x S_2 x$.. The main drawback for using of the joint distribution is that it expands very according with the number of variables and the size of their sample space. If variables are given, each with a sample space of size *m*, the total number of entries joint distribution table is m^n.

A Bayesian Network is an augmented directed acyclic graph, represented by the where:

- *V* is the set of nodes, each of one represent a random variable, along probability distribution table indicating how the probability of this values depend of all the possible combinations of the parents values.
- *E* is the set of directed edges.

A random variable *X* has an incident arc coming from random variable *Y* if and dependent of *Y*, and *X* is said to be a parent of *Y*. Suppose there is a third variable parent is *Y*. In this case *X* and *Z* are said to be independent given *Y* (see figure 1).

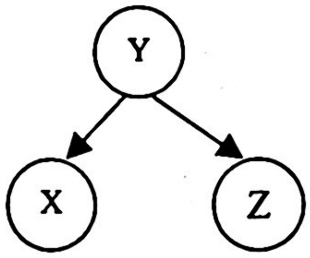

Figure 1. Variables *X* and *Z* are independent given *Y*

Every node in the Bayesian Network has associated a probability distribution depends only of the its parents value. In the above example, *X*'s distribution must entry for every value in the sample space of variable *Y*.

Any entry in the joint probability distribution can be computed from a Bayesian using the independence and conditional probability rules. This way, a Bayesian achieves a smaller representation of the joint distribution by using context informati dependency/independency patterns among the random variables.

There are two approaches for reasoning with a Bayesian Network: Top-Down and Up[2]. The Bottom-Up reasoning (also called diagnostic) goes from effects to this approach, the probability of a parent variable can be computed given the occurre an event in a child variable. The Top-Down reasoning (or causal reasoning), computes the probability of an event given the occurrence of a parent variable.

2. The Distributed Model.

The proposed model for the Distributed Bayesian Network is based in domains. A domain is a well delimited knowledge area which can be represented by a set of random variables semantically related. A consistent Bayesian Network could be constructed using only the variables of the domain. However, some of these variables may be involved with the variables in another domain, forcing inter-domain connections.

As an example consider two domains: *"Weather"* and *"Traffic"*. A representative Bayesian Network can be constructed for each of these domains, containing the random variables of interest. Figure 2 shows a connection between these two domains. If it is raining, the probability of a car accident augments, connecting the random variable "Rains" which has the space of possible values *{True, False}*, and the variable *"Car accident"*, which has the same space. When the probability of slow traffic is computed, the variables in *Weather* should be included.

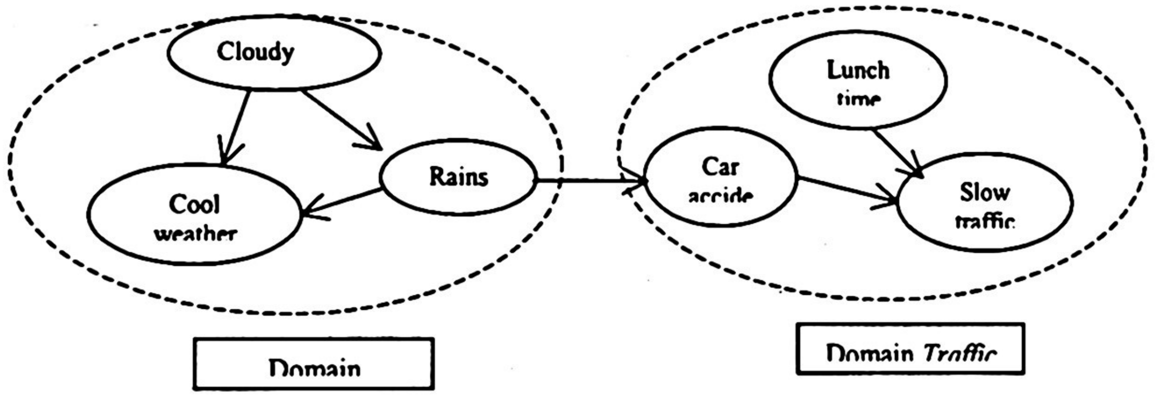

Figure 2. The domains *Weather* and *Traffic*

When two domains are related by one or more connections among their variables, the visualization of each domain could expand to see the whole resultant Bayesian Network. The decision of doing so depends of the specific application and user preferences.

The distribution of each variable respect to its parents should be kept local with the variable. This should be totally changed, though, whenever a parent is added or removed to the variable.

The probability of a random variable in the network is evaluated according to a specific scenario. The scenario is the set of values of the involved variables, according to which the resultant probability will be computed. If the value of a variable is omitted, the resultant probability can whether be computed using the probability distribution of that variable, and considering each possible value this can take, or computing a single probability value of the variable according to the probabilities of its predecessors. This way, the resultant probability may be a new distribution where all the possible values of the omitted varial are considered, or a single probability value.

3. Design of the Distributed System.

The system has been designed using remote objects, specifically CORBA[3]. The objects are: BayesNet, Node, Distribution, Scenario, BayesNetsRepository BayesNetObserver. Some of these remote objects are discussed next.

3.1. BayesNet.

This object represents the complete Bayesian Network. Contains the nodes of the and is responsible of tasks that involve the whole network, such as computing probability of a given node inside the net or finding the root nodes (those nodes which not have any parents).

When an event occurs in the network (i.e. a node is added or removed, or two nodes connected or disconnected), the Bayesian Network must notify all the interested objects. object can register itself to be notified of events generated in the network.

```
interface BayesNet : NodeListener
{
        typedef sequence <BayesNet> BayesNetArray;
        typedef sequence <Node> NodeArray;
        string getName ();
        NodeArray getNodes ();
        Node getNode (in string name);
        void addNode (in Node n);
        void removeNode (in string name);
        NodeArray getRootNodes ();
        Distribution computeProbabilities (in string nodeName, in Scenario scenario)
                                        raises (BayesNetException);
        double computeProbability (in string nodeName, in Scenario scenario)
                                        raises (BayesNetException);
        boolean isMember (in string nodeName);
        void addBayesNetListener (in BayesNetListener nl);
        void removeBayesNetListener (in BayesNetListener nl);
        string toXBN ();
};
```

Figure 3. Definition of the Bayesian Network interface

The BayesNet object listens for actions produced for any of its nodes. This network itself can fire a notification to its listeners and take the adequate actions, adding an external node which has been selected as parent or child of any of the the network.

The network fires an event when a node is moved, added or removed, and when a link (a parent-child relationship) is added or removed to the network.

Every Bayesian Network in the system is identified by an unique name given by the user. Along with its name, the network keeps a description of itself, which can be used for helping the user to select a net among a set of them. The IDL interface for the BayesNet object is shown in figure 3.

3.2. Node.

The Node object represents an independent node in the system. Each node must maintain references to every parent and child node, a reference to its current probability distribution, its position in the plane of the network, its name, and the listeners to the actions of this node. Every node in the system is identified by a name given by the user. If two nodes have the same name but they are in different networks, each network will see only its own node. But if the nodes are in the same network, then they will be indistinguishable of each other.

A node fires actions when a parent or child is added or removed from it, and when the node is moved to other position in the network. The IDL interface for the node is shown in figure 4.

3.3 Distribution.

The Distribution object represents the distribution of a variable according to its parents. The current version of the system supports only distributions with the sample space of values True and False.

This object keeps a mapping between the values of the set of variables and the probability for that specific combination of values. The Distribution object must store the names of every participating variable, except the variable to which the distribution belongs. If there are n variables in the distribution, the object will maintain 2^n probability values. According to this, a non-parents variable will have only one probability value: the marginal probability of the variable to be True.

The Distribution object has two ways to return a probability value: receiving a Scenario object or receiving a number. The former way is discussed below. When the last way is used, the number is interpreted as a set of bits indicating the value of each of the variables. The first variable is positioned in the least significant bit position, and the last in the most significant position. A value of zero in the corresponding bit indicates a False value for the variable, and a value of one indicates a True value.

This object must be able to perform the compute sub-distribution operation. In this operation, the Distribution object receives an Scenario object with the values of the known variables, and the object is expected to return a new distribution with the probability values of the unknown variables (those which are not in the scenario) in those cases where the given scenario is fulfilled. So, if variable v has the value True in the scenario object, the

```
interface Node
{
        typedef sequence <Node> NodeArray;
        string getName ();
        void setName (in string name);
        string getDescription ();
        void setDescription (in string desc);
        Distribution getDistribution ();
        NodeArray getParents ();
        long getParentsLength ();
        NodeArray getChildren ();
        long getChildrenLength ();
        Node getParent (in string name);
        boolean isParent (in string name);
        void addParent (in Node parent);
        void removeParent (in string name);
        void setDistribution (in Distribution newDist);
        Node getChild (in string name);
        boolean isChild (in string name);
        void addChild (in Node child);
        void removeChild (in string name);
        void setPosition (in long x, in long y);
        void addNodeListener (in NodeListener nl);
        void removeNodeListener (in NodeListener nl);
        long getXPos ();
        long getYPos ();
        string varToXBN ();
        string structureToXBN ();
};
```

Figure 4. Definition of the node interface

resultant sub-distribution must contain just the probability values of those cases whe variable is True. This operation is very useful when computing the probability distri of a given node in the Bayesian Network.

The IDL interface for the Distribution object is shown in figure 5. Although it is Distribution object is one of the most important remote objects of the system.

3.4 Scenario.

The Scenario object is used simply as a name-value mapper for a given set of variable used mainly as a friendly interface to retrieve values from the Distribution object. functions allows the user to set the values of the variables as a sequence of bits in in the way described in the Distribution object.

```
interface Distribution
{
        StringArray getNames ();
        long getNamesLength ();
        void setNames (in StringArray names);
        void setValues (in DoubleArray values)
                                raises (DistributionException);
        boolean isElement (in string name);
        double getProbability (in Scenario scenario)
                                raises (ScenarioException);
        double getBinaryProbability (in long bits)
                                raises (DistributionException);
        void setProbability (in Scenario scenario, in double prob)
                                raises (ScenarioException);
        Distribution computeSubdistribution (in Scenario scenario)
                                raises (DistributionException);
        string toXBN (in string nodeName);
};
```

Figure 5. Definition of the distribution interface

3.5 BayesNetsRepository.

The Bayesian Networks repository is the object that stores references and descriptions of every Bayesian Network in a collaborative session. Each network in the repository is identified by its name. There must be exactly one repository for each collaborative group.

3.6 BayesNetObserver.

Sometimes local objects, such as graphic user interfaces, will need to listen for a Bayesian Network's events in order to update their representation or internal state. As local objects, they cannot just register with the Bayesian Network object because they cannot be an argument in a remote object invocation.

The solution used here is to define a listener remote object which listens for the Bayesian Network's events, and whose servant implementation can register (as a local object) the local listeners interested in such events. The events received by the remote observer will be retransmitted to the registered local objects.

4. Probability Computing.

For the computation of the probabilities of a node given a scenario, distributions are computed and merged in a recursive fashion. The system gives the option of computing the probability of the node for each possible configuration of the unspecified parents.

In each stage of the compute, the node's distribution is obtained, and the Distribution's `computeSubdistribution()` method is applied with the given scenario. If a parent is found in the scenario, its value is taken and applied to the actual distribution, eliminating entries in which the parent's value is different from that taken from the scenario. If resultant distribution has only one value (the node's probability), the distribution returned.

After the sub-distribution is computed, the probability of the unknown parents (those are not included in the scenario) is computed with the given scenario in a recursive call. all the possible configurations are to be shown, the distributions are merged in a new that contains the names of all the unknown parents, and the probability entries computed. Otherwise, the parents probabilities are reduced to one probability using probability chain rule[4].

When all the configurations probability is required, for each possible combination of valu in the merged distribution, the probability value when the considered nodes have the of the actual combination is obtained from the sub-distribution. This value is multipl either by the probability value of each parent for that combination if the value of the is positive in the actual combination, or by 1 minus its probability if the value is negati At the end, the merged probability distribution will be completely full and can be returr as the result. If only the total probability is required, these results are added in instead as being put in the merged probability distribution.

5. Local Design.

The local or implementation side of the system has been designed for a collaborat interactive graphic environment. Here are explained some of the most important decisions assumed for the developed system.

5.1. Probability computing methods.

There are several methods for computing the probability of a given variable in a Bayes Network. The most important are the automatic method (used by the system), stochastic sampling method. Because of this variety of methods, the Strategy pattern[5] has been used, allowing for future use of new methods without any changes the Bayesian Network servant implementation.

5.2. Visualization of the networks and other possible options.

When a user is working with a network, he/she should be able to see whether just which is working with or all the involved nodes in the network.

When the user wants to see the complete network (all the involved nodes), then every a parent/child connection is performed between a included node and a not included system must add every connected not included node to the actual network.

This may be just one of several options that may be presented to the user. As Bayesian Network behavior is fixed, and one or more options may be optionally chose the user, the Decorator design pattern has been used. The base network implementation as the basic component, and every extension to this behavior acts as a decorator.

6. Storage of the Networks.

The Bayesian Networks generated using the system can be saved and recovered using the XBN format [6]. This XML based format was proposed by the Decision Theory and Adaptive Systems group of Microsoft Research, and is expected to replace older Bayesian Networks Interchange Formats (BNIFs). This format allows the analysis of networks generated with the system using other existing tools, and the collaborative edition of already created networks.

7. Implementation Issues.

Currently, the system has been completely implemented using the Java language. Despite of the relative simplicity of the distributed system using CORBA, there are some issues which should be addressed during the implementation stage.

7.1 The distributed callback on a synchronized method.

In general, Node events cause the execution of a BayesNet object's method. This method, in turn, will generate an event received by a BayesNetObserver object. This object will communicate the event to the local listeners, which in turn will invoke some methods on the BayesNet object and some Node objects to repaint the network. This sequence of events will produce a callback on the BayesNet object, and probably on the Node object which produced the event, as shown in figure 6. Note that the notifier methods in Node, BayesNet and BayesNetObserver cannot finish before the last listener has finished.

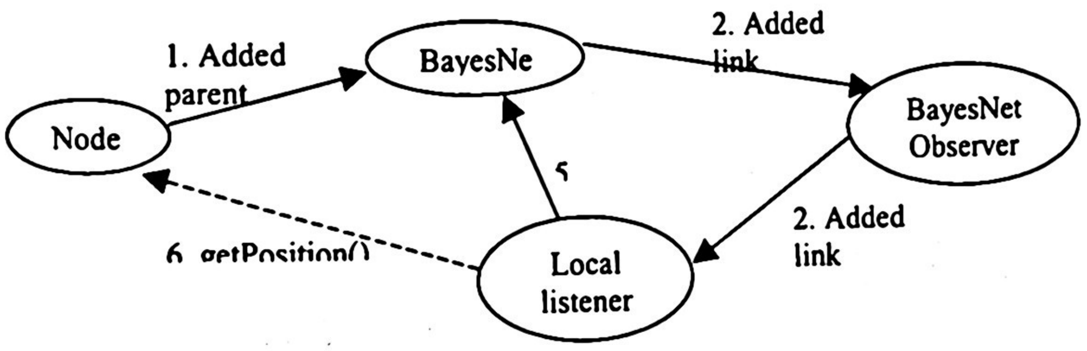

Figure 6. The chain of events causes a callback over the BayesNet and Node objects

As the remote objects can be accessed by several threads at the same time, many of their methods must be synchronized.

The Java monitors are reentrant [7]. This means that a thread will not deadlock itself when it invokes a synchronized method from another synchronized method. But in the distributed case, the reentrant thread is not the same as the one that holds the monitor, so a deadlock is produced.

To overcome this problem, a thread is created for each notification. This way, a method can finish after creating one thread for each listener, freeing the monitor allowing the callback.

8. Test of the System.

The system has been tested using the Serum Calcium model developed by Greg (Cooper, 1986). The system's input is the XBN file specifying such model, and the are compared with the manually computed results. Figure 7 shows the representation for such file.

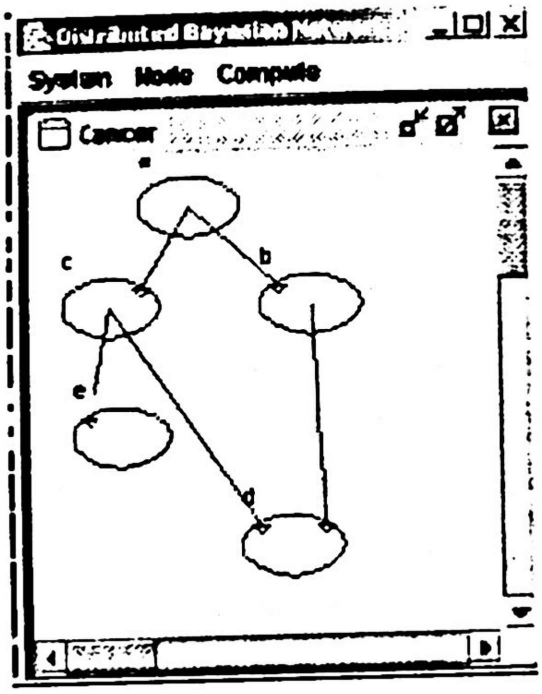

Figure 7. Serium Calcium model represented by the system

The probabilities of each node are shown below:

a. True: 0.2.
b. True given not a: 0.8. True given a: 0.2.
c. True given not a: 0.2. True given a: 0.05.
d. True given not b and not c: 0.8. True given not b and c: 0.9. True given b 0.7. True given b and c: 0.05.
e. True given not c: 0.8. True given c: 0.6.

As a Bottom-Up reasoning test, consider computing the probability of *a=True* *d=False*. According to the Bayes Rule, this probability is:

$$P(a|d) = \frac{P(d|a)P(a)}{P(d)}$$

P(a) is known. To compute *P(d)* we use the law of total probability. According to

P(d)=P(d/b,c)P(b,c)+P(d/b,not c)P(b,not c)+P(d/not b,c)P(not b,c)+P(d/not b,not c)P(not b,not c)

As *b* and *c* are independent given a, we have *P(b,c)=P(b)P(c)*. Using the same reasoning, we have *P(b)=P(b/a)P(a)+P(b/not a)P(not a)* and *P(c)=P(c/a)P(a)+P(c/ not a)P(not a)*.

Computing the numbers we get *P(b)* = 0.68, *P(c)* = 0.17 and *P(b, c)* = 0.1156. Similarly: *P(b, not c)* = 0.5644, *P(not b, c)* = 0.0544 and *P(not b, not c)* = 0.2656. Finally, we get *P(d)* = 0.6623.

To compute *P(d/a)*, we simply consider *P(x/a)=P(x)*. Doing this we get P(b)=0.2, *P(c)*=0.05, *P(b,c)*=0.01, *P(b, not c)*=0.19, *P(not b,c)*=0.04, *P(not b, not c)*=0.76, and *P(d/a)*=0.7775.

Finally, *P(a/d)* = 0.23478786. This result is the same as the computed by the system.

Now, consider a Top-Down test, in which the probability of *d=True* is computed given *a=True*.

Making a partition with the values variables *b* and *c* in conjunction, and using the law of total probability, we get:

P(d/a) = P(d,b,c/a) + P(d,b, not c/a) + P(d, not b,c/a) + P(d, not b,$\neg$*c/a)*

P(d/a) = P(d/b,c,a)P(b,c/a) + P(d/b, not c,a)P(b, not c/a) + P(d/not b,c,a)P(not b,c/a) + P(d/not b, not c,a)P(not b, not c/a)

As *d* does not have a direct dependency on *a*, and as *b* and *c* are independent given *a*, the expression above can be rewritten as:

P(d/a) = P(d/b,c)P(b/a)P(c/a) + P(d/b, not c)P(b/a)P(not c/a) + P(d/not b,c)P(not b/a)P(c/a) + P(d/not b, not c)P(not b/a)P(not c/a)

Those probabilities can be directly obtained from the variables' distributions. After computing the products we get.

P(d/a) = 0.0005 + 0.133 + 0.036 + 0.608 = 0.7775.

Which is the result computed by the system.

9. Conclusion.

Bayesian Networks are very useful tools for representing dependency relationships among random variables. Unfortunately modeling these relations is not an easy task, and may require collaboration among field specialists of the involved areas. This system allows a collaborative constructions of Bayesian Networks using the concept of domains to divide the problem in a set of more manageable parts. Two domains can be joined by establisł a parent-child relationship between one or more pairs of their nodes.

Every Bayesian Network for a group of users is maintained in a repository, where it retrieved by its name. More than one user can access the same network, but the netv belongs always to a single domain, and is physically kept in a lonely site.

The only cache used is the graphic representation of the network for each user. A through semantic is used for every update to the network. The update to every representation (including the source of the update) is delayed until the node and Baye Network are updated, and the last fires the proper event.

Extensions can be made to this work, such as addition of new probability compı methods. If the system is to be used in a slow network, mechanisms for consiste assurance should be included. These mechanisms, though, may impose limitations the need to have a predefined size of the groups when using vector logical clocks. The presented here is a first step for a Distributed Bayesian Networks Pattern Disco System.

References

[1] Papoulis, A. *Probability, Random Variables and Stochastic Processes. Internati Student Edition*. Mc Graw-Hill. 165.

[2] Pearl, J. & Russell, S. (2000). *Bayesian Networks*. pp. 2, 3.

[3] Object Management Group. *The Common Object Request Broker: Architecture Specification*. 312 - 335.

[4] Russell, S. & Norvig, P. (1995). *Artificial Intelligence: A Modern Approach.* Hall. Chapter 5.

[5] Gamma, E. Helm, R. Johnson, R. & Vlissides, J. *Design Patterns. Elemen Reusable Object Oriented Software*. Addison Wesley.

[6] Bengtsson, H. *Bayesian networks. A self contained introduction with implement remarks*. 82, 83.

[7] Campione, M. Walrath K. & Huml, A. *The Java Tutorial*. Third Edition. Microsystems.

[8] Dodier, R. (1999). RISO: An Implementation of Distributed Belief Networks. *Symposium on AI in Equipment Service and Maintenance.*

[9] Dodier, R. (1999). Unified prediction and diagnosis in engineering systems of distributed belief networks. *Ph.D. dissertation, Dept. Civil Engineering, of Colorado at Boulder.*