Building Intelligent Sensor Networks With Multiagent Graphical Models

Yang Xiang

University of Guelph, Canada

1 Introduction

Modern society relies heavily on various equipment (food production processes, assembly lines, transportation vehicles, airplanes, electricity grids, etc.) Consider monitoring a piece of complex equipment. To ensure productive operation, avoid downtime and reduce maintenance cost, engineers must constantly determine whether the equipment is operating normally. If the equipment is determined to be abnormal, the faulty devices must be replaced. Very often, a detected abnormal behavior of the equipment may be caused by one or more faulty devices from a large number of candidates. It is simply too costly to replace them all. Hence, the next decision is to determine a small number of devices that, if relaced, is highly probable to bring the equipment back to normal. This is equivalent to answer the query: what is the small set of devices that is highly likely the culprit of abnormality?

A sensor network is often deployed to gather and process the key information in this decision process. Complex equipment consists of multiple components, each of which is further composed of multiple devices. For some devices, sensors can be deployed to observe their inputs and outputs, but sensor observations are noisy and unreliable. For other devices, no sensors can be deployed to observe their inputs and outputs due to accessibility or cost. No sensor can directly observe whether a device is normal and hence the state of the device must be inferred. We refer to sensors deployed in the equipment and their transmission media collectively as the *sensor network*. To infer the state of a device from sensor observations, knowledge about intended and faulty behavior of the device as well as knowledge about other devices it interfaces with are necessary. Often, components are manufactured by different vendors, who may be unwilling to disclose internals of their components. In such cases, no single entity has all the knowledge needed: an issue that arises when the sensor network crosses technical and economical boundaries.

Traditional approach for sensor network monitoring is centralized, where all sensor observations are transmitted to a central location for processing.

Transmission introduces delay; centralized processing creates a bottleneck; and the central unit must have access to all the knowledge needed.

As the cost of computing and networking continues to decrease, distributed processing becomes a more promising alternative, where sensor observations are processed locally and processing units exchange only partial information through message passing. Each processing unit is abstracted as an intelligent *agent*, embodying its subset of sensors, its computing resources, its local knowledge, and its inference procedures. The collection of these agents as well as the sensors that they manage forms an *intelligent sensor network*. The task to process sensor obserations by these agents and to answer the query "what is the small set of highly probable faulty devices" becomes a task of *multiagent inference*.

Alternative approaches exist for multiagent inference. The early approach is logic-based. Logic has intrinsic limitations in handling uncertainty. Effort to overcome these limitations leads to default logic based approach. This approach relies on default and model minimization to handle uncertainty. There are situations, however, where the minimal model is not the most probable. More recent approach is based on Markov decision processes (MDPs) for its strength in handling uncertainty. It suffers, however, from high computational complexity.

In this chapter, we introduce the approach based on multiply sectioned Bayesian networks (MSBNs). Built upon the success of Bayesian Networks (BN) [1, 2, 3, 4, 5, 6], MSBNs [7] provide a probabilistic framework for reasoning about uncertain domains in cooperative multiagent systems (MAS). Under the framework, a complex, uncertain problem domain is partitioned into overlapping subdomains so that each can be managed by a single intelligent agent. The agent holds a partial perspective of the domain in terms of a Bayesian subnet over the subdomain. These agents reason autonomously as well as through limited communication. The distributed inference operations defined under the MSBN framework ensure that their beliefs are *exact* as governed by Bayesian probability theory. These beliefs form a distributed assessment of the current state of the domain and answer the query "what is the small set of highly probable faulty devices" in the context of sensor network. When the subnet dependency structures are sparse, the inference computation is *efficient*.

Several advances have been made in recent years on modeling, compilation and inference under the MSBN framework, making it even closer to field applications. Before a general technological framework can be turned into deployed applications, practitioners must understand sufficiently well how theoretical intricacies are mapped into practical reality. The levels of such understanding can be described as follows:

1. Mathematical and algorithmic level.
2. Application development level.
3. Operation level.

This chapter is intended to facilitate understanding at the application development level. It links together key technological steps involved in applying the MSBN framework to intelligent sensor networks through a case study (in a laboratory setting).

The problem domain of case study is a moderately sized sensor network for monitoring a combinational digital system. The choice of a digital system is due to the common knowledge (among readers) on digital circuits. Once how to apply the MSBN framework to such a system is understood, its applications to monitoring other equipment (electrical, mechanical, chemical or other nature) will be within one's grasp. We demonstrate how such a problem domain can be modeled as an MSBN-based MAS, how the model can be compiled into an efficient run-time representation, and how agents can cooperate to monitor the digital system and isolate faults. We explain intuitively the rationals behind each operation. The operations are demonstrated using a state of the art software toolkit, WebWeavr [8], developed by the author and freely available to researchers and educators.

To serve its purpose, the chapter is kept as informal as possible, with pointers to references containing mathematical and algorithmic details. In short, the chapter addresses the following questions: What technical steps are involved in building an MSBN-based intelligent sensor network? Why are these steps necessary? How can these steps be performed using a software toolkit, such as WebWeavr? How does the resultant intelligent sensor network answer the query "what is the small set of highly probable faulty devices"? What are the benefits of adopting this framework?

The remainder of the chapter is organized as follows: Section 2 surveys the literature. Section 3 specifies the problem domain of the case study. Section 4 describes the knowledge representation and integration of the MSBN-based MAS for the case study. Section 5 presents MAS system verification. Enhancement of agent interface for improved inference efficiency is addressed in Section 6. How to compile the MAS into an efficient run-time representation is demonstrated in Section 7. The decision making computation, how multiagent inference isolates faulty devices, is illustrated in Sections 8 and 9.

2 Related Work

Several alternative frameworks for generic multiagent inference exist. The earliest one is the *blackboard* architecture [9], a distributed rule-based system. It is essentially a logic-based approach.

The BDI architecture [10] has been very influential in building multiagent systems. It primarily deals with representation of an agent's mental state for practical reasoning where an agent's belief is represented as atoms of first-order logic [11].

The main limitations of logic in handling uncertainty are summarized by Russell and Norvig [12] as the following: (1) Logic relies on exhaustive disjunction to ensure exceptionless rules and such disjunction can become unbounded in practical uncertain domains. (2) In practical uncertain domains, existing knowledge is not deterministic as suitably represented by logic rules. (3) In practical decision making, it is often too costly to gether all the facts needed for firing the necessary logic rules.

The limitations of logic lead to several extensions known as *default logic* [13], *circumscription* [14], *nonmonotonic logic* [15], and *truth maintenance systems* [16]. Distributed assumption-based truth maintenance system (DATMS) [17] and distributed truth maintenance system(DTMS) [18] extend centralized truth maintenance systems to distributed agents. The basic idea is to begin the knowledge base with a set of default assumptions which are uncertain. Inference proceeds as if these assumptions were true until some are found to be false due to new observations. The falsified assumptions as well as conclusions drawn from them will be retracted from the knowledge base to regain consistency. The computational complexity of these extensions is at least NP-hard [12].

Some frameworks have been developed specially for data fusion in sensor networks. Roos et al [19] propose a multiagent framework for sensor network monitoring based on logical consistency. They showed that establishing a global diagnosis under the framework is NP-Hard and therefore their protocol does not guarantee one.

Guestrin et al [20] propose distributed regression for efficient interpretation of sensor data. Their method assumes that the data can be fit into a linear function.

None of the above frameworks maintains agents' beliefs in terms of Bayesian probability. A recent trend has focused on modeling multiagent decision making using Markov decision processes (MDP) [21, 22, 23]. However, it has been shown [24] that the computation for solving general distributed MDPs is intractable. The state of the art algorithms can handle only very small problem domains currently (much smaller than the domain in the case study to be presented here).

In the remainder of this chapter, we present a case study based on the MSBN framework. The key advantages of the framework are the following: It maintains agents' beliefs in terms of exact Bayesian probability and is well suited to modeling of uncertain domains. It works well with non-linear dependence relations among sensor observations. It guarantees globally consistent diagnosis. Due to its graphical modeling, the computation is efficient when the graphical structure is sparse.

3 Sensor Network for Digital System Monitoring

Our case study involves monitoring a combinational digital system. It consists of remotely located components $U_0, ..., U_4$ supplied by five independent venders and integrated by a sixth vendor. Each component is further composed of a number of devices (logic gates) as shown in Figures 1 through 5.

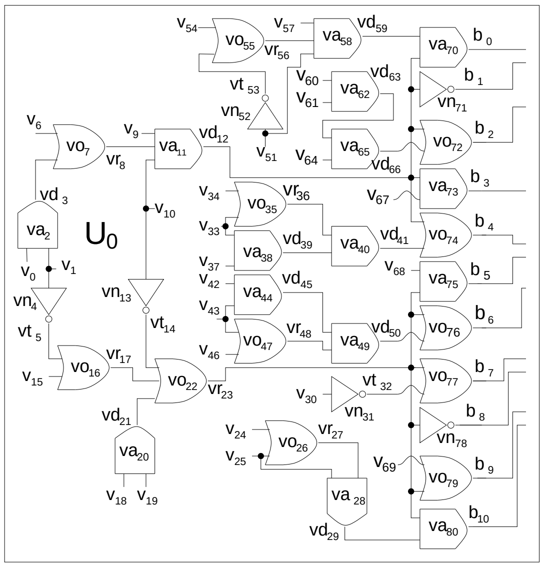


Fig. 1. Components U_0.

Each component has some external inputs, such as signal v_{54} (see top of Figure 1) in U_0. It may accept signals from other components. For instance, U_1 accepts signal b_0 (see top left of Figure 2) from U_0 (see top right of Figure 1). It may output signals to others. For example, U_1 outputs signal c_0 (see bottom left of Figure 2) to U_2 (see top left of Figure 3).

Signals exchanged between components are labeled identically, e.g., $c_0, ..., c_9$ between U_1 (see bottom of Figure 2) and U_2 (see top of Figure 3). In the case study, all signals are assumed binary (taking values of logic 0 or 1). In general, each signal can take a finite number of discrete values, or even be continuous (see, for instance, [25]).

Each device is in one of two states, normal or faulty, although more states can also be represented (e.g., two normal states each at a different operating mood). We assume that each device may be in the faulty state at any given time with a probability of 0.01. A faulty NOT gate produces incorrect output 50% of time. The corresponding probabilities for AND and OR gates are assumed to be 0.8 and 0.3, respectively. These parameters and Figures 1 through 5 *completely* specify the problem domain and allow replication/verification of our case study. These details also give readers a feel of the complexity of the problem domains that the MSBN framework is capable of handling. The digital system is used here as an example of any complex system made of multiple components, each of which further decomposes into simpler units. Collectively,

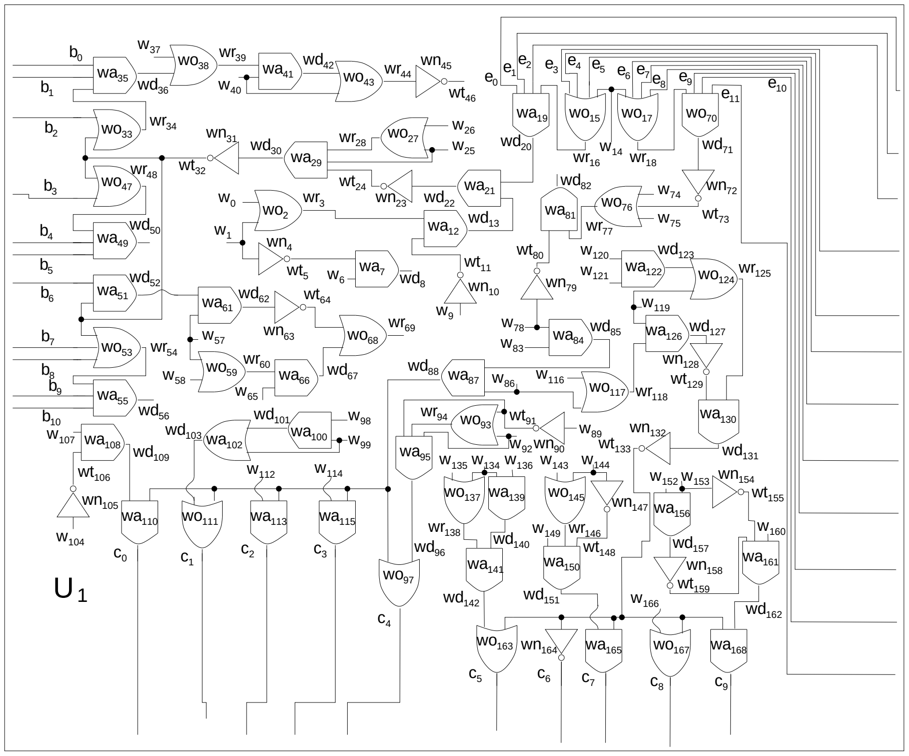


Fig. 2. Component U_1.

these components implement some useful functionalities that may be electrical, mechanical, chemical, and so on.

There are two types of decisions to be made in monitoring such a system. The first decision involves a *normality* query: Is the current system operating normally? The answer to the query is binary. A positive answer (the system is normal) requires no intervention. A negative answer (the system is abnormal) requires intervention in order to bring the system back to normal.

In the case of a negative answer, the second decision must be made to decide which faulty gates should be replaced. Very often, a faulty gate produces incorrect output signal which propagates to other gates and causes their output signals to be incorrect as well. It is simply too costly and unwise to replace them all. It is desirable to replace only a small number of devices that are highly probable to be the culprit of abnormality. Hence, the second decision involves a *culprit* query: What is the small set of devices that is highly likely the culprit of abnormality? The answer to this query has multiple potential values.

To answer these queries, sensors can be deployed to collect necessary raw information. We assume that in general each external input signal and the

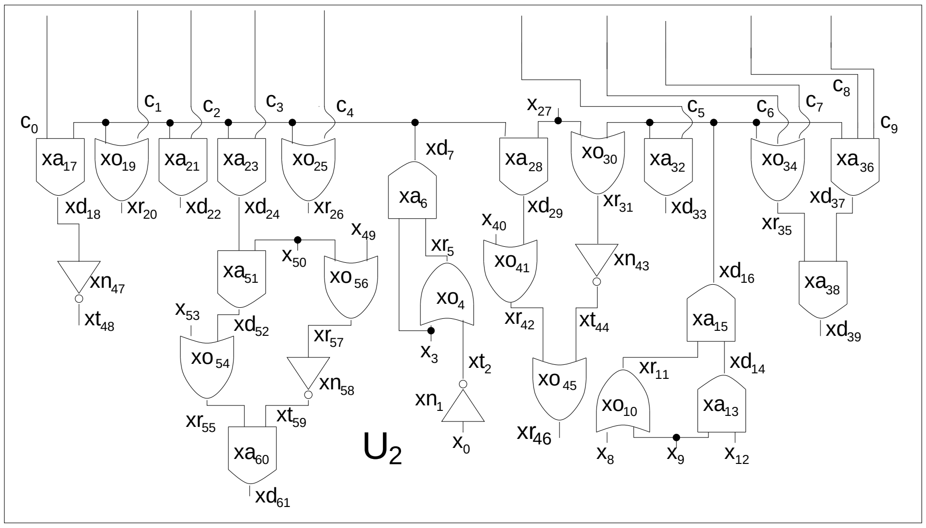


Fig. 3. Component U_2.

output signal of each gate can be observed through a sensor, although in practice no sensors are deployed to observe some signals due to cost involved or other constraints. We assume that whether a logical gate is faulty can never be observed directly and can only be inferred from other observations. These assumptions are consistent with the partial observability of practical systems. We refer to the collection of sensors and the signal transmission media deployed to monitor a given system as a *sensor network*.

Although a sensor network provides the raw information about the behavior of the monitored system, the information must be processed in order to answer the normality query and culprit query. In theory, the processing can be performed by an intelligent agent through reasoning based on its knowledge on the monitored system as well as sensor observations. However, this paradigm has a number of practical difficulties. First of all, transmitting sensor observations distributed over space to a central location requires high bandwidth and introduces time delay. Second, the dependence on a single agent creates a bottleneck and processing fails completely when the agent fails. Third, the need to process all relevant knowledge and observations at the single agent places heavy load of computation at the agent and introduces additional computational delay. Fourth, it is difficult to develop a single agent capable of monitoring a large and complex problem domain, due to the amount of domain knowledge to be gathered and encoded. Vert often, no single person or single technical entity possesses all the knowledge needed. For example, we have assumed that the digital system consists of five components supplied by independent vendors. Although each vendor has detailed knowledge about the

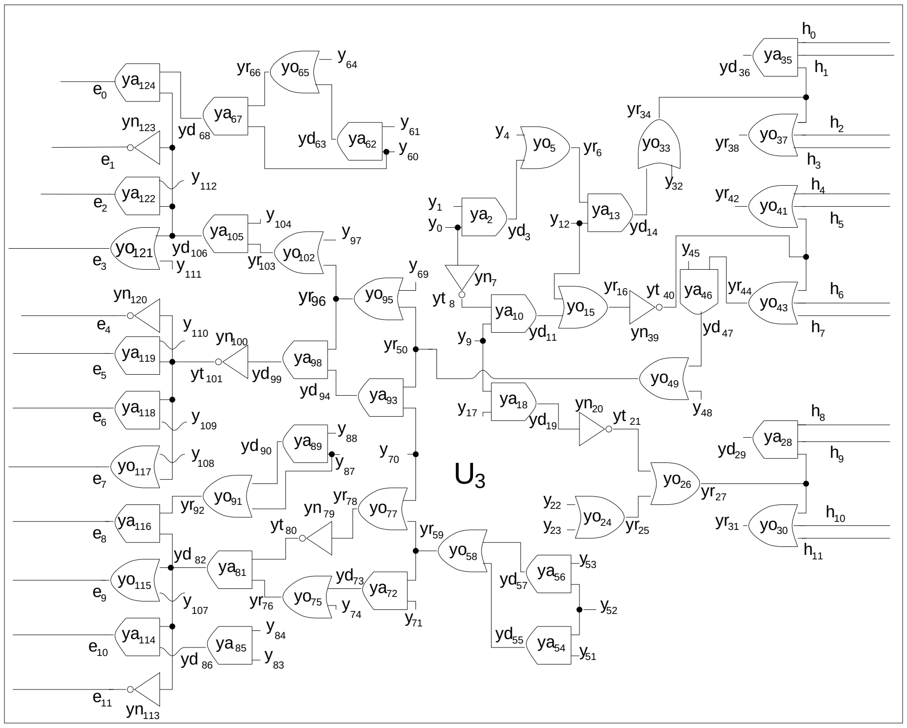


Fig. 4. Component U_3.

composition of the corresponding component, it may not want to disclose this knowledge due to competition.

One alternative to the single-agent paradigm is the multiagent paradigm. The large domain is partitioned into subdomains. In our case study, each subdomain corresponds to one component. Sensor observations for each subdomain are collected and processed by a separate agent, and a set of agents are responsible for the entire domain. The advantages can be understood as follows: First, the agent responsible for a given component can be deployed at the same location as the component, eliminating the need of high bandwidth and time delay due to transmission of observations to a central location. Second, even when a single agent fails, the other agents can still function. Hence, the monitoring system fails gracefully instead completely. Third, each agent needs to process mainly knowledge and observations on its subdomain (plus some communication with other agents). Since different agents process their local information in parallel, the overall computation is more efficient. Fourth, each agent encodes only knowledge about its subdomain and the agent development is easier. When the subdomains are partitioned naturally, a natural technical entity exists to supply the relevant knowledge. For our case study, the vendor who supplies the component becomes the natural agent developer.

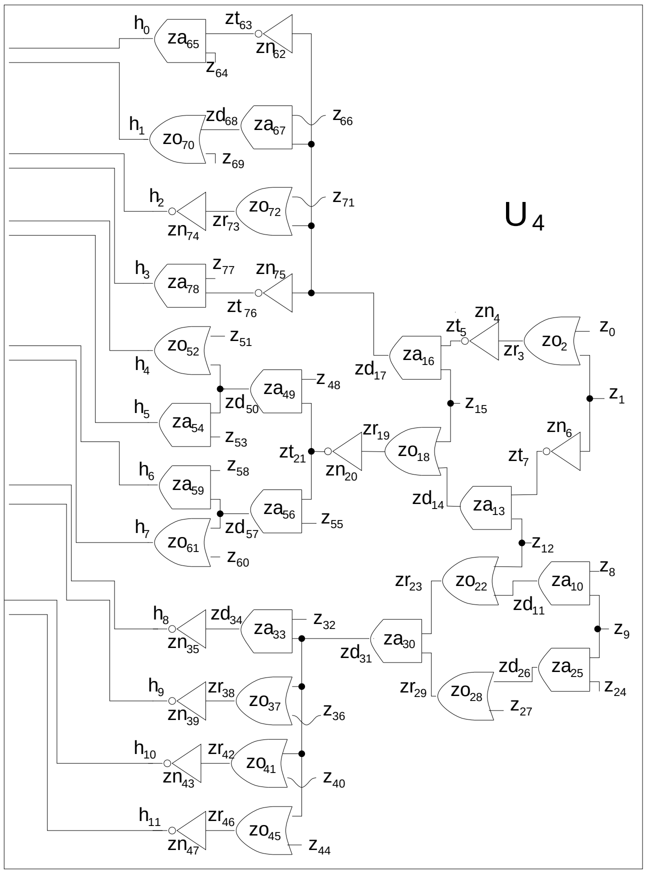


Fig. 5. Component U_4.

In this chapter, we adopt the multiagent paradigm for the case study. The agent responsible for the component U_i $(i = 0, ..., 4)$ is denoted by A_i. We refer to the collection of the sensor network, the local signal transmission media, and the agents as an *intelligent sensor network*.

4 Integration of MSBN-based Multiagent System

Given the knowledge on a problem domain and sensor observations, agents can answer the normality and culprit queries based on their beliefs. For example, if every agent believes that each gate in its subdomain is currently normal, then agents collectively can answer the normality query positively. On the other hand, if at least one agent believes that some gates in its subdomain is currently abnormal, then agents collectively can answer the normality query negatively. In that case, the set of gates, each of which is believed highly probable to be faulty by at least one agent, constitutes the answer to the culprit query.

How, then, should agents represent their beliefs? It has been shown [26] that under some reasonable assumptions, the correct belief must be consistent with Bayesian probability. Furthermore, if one's belief deviates from Bayesian probability, then actions consistent with that belief will lead to guaranteed failure in an malicious uncertain environment [27]. Belief maintained by earlier multiagent reasoning systems do not satisfy this criterion (see Section 2). The MSBN framework is developed with the objective that agents' beliefs are *exact* according to Bayesian probability theory. To this end, the framework employs two levels of knowledge representation: the individual agent level and the agent society level.

At the individual agent level, the knowledge is represented as a BN as in the single-agent paradigm. Under that paradigm, a BN is a concise encoding of the single agent's probabilistic knowledge of its domain through a graphical model. Under the multiagent paradigm, since each agent only has the knowledge about a subdomain and encodes that knowledge into a BN, the graphical model is referred to as the *subnet* of the agent.

A subnet consists of three components: a set of variables, a graph, and a set of conditional probability distributions. The set of variables corresponds to the *subdomain* of the agent. The graph is a directed acyclic graph (DAG), where each node corresponds to a subdomain variable (hence we refer to the nodes and variables interchangeably) and each arc corresponds to a causal dependence relation. The DAG encodes conditional independence relations among the variables. If two sets X and Y of nodes are graphically separated by a third set Z, then the dependency between X and Y is mediated by Z. Once the value of Z is known, X is no longer dependent on Y, and X and Y are said to be *conditionally independent* given Z. The set of probability distributions consists of one conditional probability distribution (CPT) for each node x in the form of $P(x|\pi(x))$, where $\pi(x)$ is the parent nodes of x. Due to the encoding of conditional independence in the DAG, a probability distribution over all subdomain variables is well defined as the product of all CPTs. For fundamentals on representation of conditional independence in BNs, see [1, 2, 3, 4, 5, 6].

In general, the subdomain of an agent in our case study may contain the following types of variables: *gate*, *signal*, *sensor* and *sensor observation*. Figure 6 (a) illustrates a logic gate, its input and output signals, the sensor that monitors the gate output, and the sensor observation. How to encode the dependency among these variables in a subnet is shown in (b).

The illustration and the representation capture the unreliability of the sensor: When the sensor is functioning normally, the sensor observation is identical to the gate output ($signal_3$). When the sensor fails, its output may differ from that of the gate. From this, we see that sensors and logic gates that they monitor are not much different. They are both devices that are subject to failure and can be modeled in the same way.

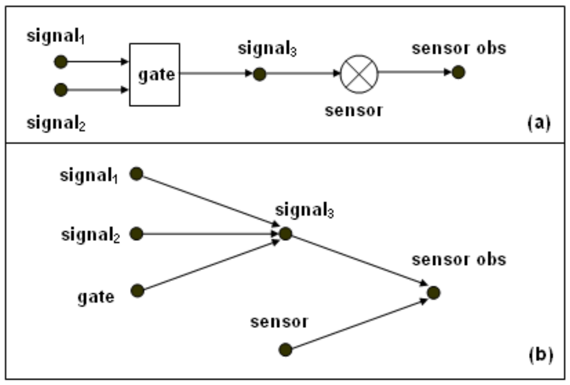
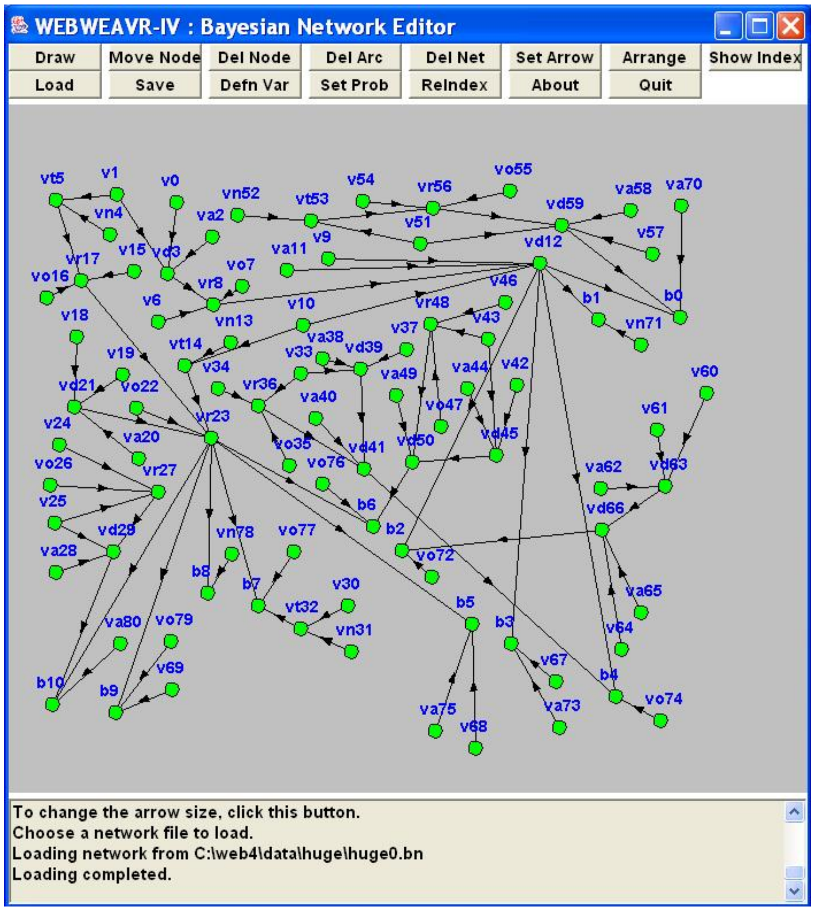


Fig. 6. (a) Illustration of a logic gate and a sensor. (b) Representation in a subnet.

One might point out their difference in observability: The output signal of the gate cannot be directly perceived by the agent, but the sensor observation can. However, although the sensor observations are directly perceivable, the perception requires the sensor output to be transmitted to the agent, which takes time and bandwidth. When a large number of sensors exist, the agent must choose the sensors to perceive selectively. Those sensors not being chosen at a given time are effectively not observable. We now see that sensors and devices (logic gates) that they monitor are not any different at all, from the modeling perspective. To simply our presentation, we assume the following in the case study:

- All sensors are reliable.
  Hence, sensor observations are always equal to the signals they monitor. This makes representation of sensors and sensor observations redundant. We therefore omit the sensor and sensor observation types of variables from the subnet and regard the corresponding signal variables directly observable subject to the following restriction.
- Some signals do not have associated sensors.
  This assumption divides signal variables into those that are observable and those that are not. Due to the omission of sensor variables, the difference is not explicit in the subnet representation. We make observability as the default and we indicate explicitly when a signal variable is not observable.

Figure 7 shows the subnet S_0 (for agent A_0) constructed by the vendor of U_0 using the tool Network Editor from the WebWeavr software toolkit [8]. It shares common nodes b_0 through b_{10} with subnet S_1 (not shown) for agent A_1. Due to the above assumption, the subdomain V_0 of S_0 consists of only *gate* and *signal* variables. A gate variable represents the state of a digital gate: whether it is normal or faulty (simply denoted as *good* and *bad*). For instance, vn_4 (see middle left of Figure 1) represents a NOT gate. A signal

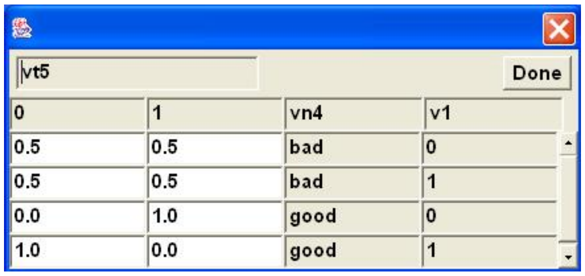


To change the arrow size, click this button.
Choose a network file to load.
Loading network from C:\web4\data\huge\huge0.bn
Loading completed.

Fig. 7. Subnet S_0 for component U_0 and the CPT for node vt_5.

variable represents the logic value of a signal if it is not observed by a sensor or the correctly observed value of the signal by a sensor. In either case, its value is either logic 0 or logic 1. For instance, vt_5 represents the sensed output signal of gate vn_4. The knowledge encoded in S_0 is private to its developer, the vendor of U_0. For simplicity, we say that S_0 is private to agent A_0 and this privacy will be maintained through the lifetime of A_0, as we will see. The exception is the variables that A_0 shares with other agents, e.g., b_0 through b_{10}. Since these variables are known to another component and its vendor, they are public anyway.

Figure 8 illustrates the CPT for variable vt_5. The last two rows, where $vn_4 = good$, encode the knowledge on the normal behavior of NOT gate vn_4. The two rows where $vn_4 = bad$ state that, when the NOT gate is faulty, its output is random.



vt5

Done

0	1	vn4	v1
0.5	0.5	bad	0
0.5	0.5	bad	1
0.0	1.0	good	0
1.0	0.0	good	1

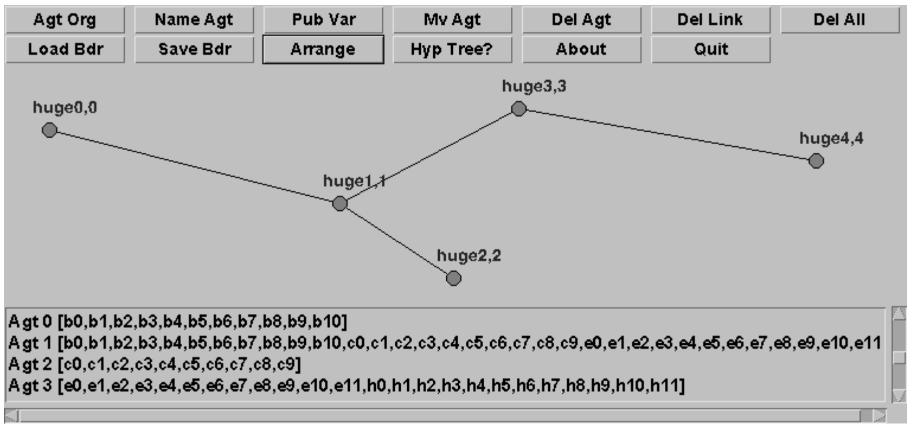
Fig. 8. CPT for node vt_5 in subnet S_0.

Next, we consider the knowledge representation at the agent society level. The key issues addressed at this level are agent organization and agent interface. Agent organization specifies, for each agent, which other agents it can communicate directly. Agent interface specifies, for each pair of agents who can communicate directly, what are the pubic variables between them as these variable determine the content of messages they exchange. Agent organization is presented in the remainder of this section. Agent interface is partly described here and is continued into the next two sections.

The chief concerns of agent organization are to support exact and efficient probabilistic inference. Through formal analysis, it has been shown that the organization must be a underdirected tree structure [28] (called a *hypertree*). In the hypertree, each hypernode corresponds to an agent and each hyperlink corresponds to a direct communication link between the agents connected (through their interface). That is, according to the organization, each agent can only communicate directly with agents adjacent on the hypertree.

Intuitively, in a hypertree organization, each hyperlink defines two separate agent communities (one on each end) and potentially allows information to be fully exchanged between the two communities through exactly two messages over the hyperlink (one in each direction). For a society of n agents, this amonts to exactly $2\ (n-1)$ messages along the hypertree, which is efficient. If the agent interfaces are adequately composed (as will be addressed in the next section), such message passing can also ensure exact probabilistic inference.

Who is responsible to specify the agent organization? As we mentioned, there exists a sixth independent vendor, referred to as *Assembler*, who assembles the five components into the final digital system. Assembler is also the natural candidate to assemble the five corresponding agents into an MAS. Operationally, it uses the tool Integrator from the WebWeavr toolkit illustrated in Figure 9 (through the function buttons "Agt Org" and "Name Agt" at the top left of the figure). In the figure, the hypertree topology is shown where each hypernode is labeled with the nickname of an agent (*huge0* for A_0, *huge1* for A_1, and so on).



```
Agt 0 [b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10]
Agt 1 [b0,b1,b2,b3,b4,b5,b6,b7,b8,b9,b10,c0,c1,c2,c3,c4,c5,c6,c7,c8,c9,e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11
Agt 2 [c0,c1,c2,c3,c4,c5,c6,c7,c8,c9]
Agt 3 [e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11,h0,h1,h2,h3,h4,h5,h6,h7,h8,h9,h10,h11]
```

Fig. 9. Integrating agents into an MAS. Middle: agent organization. Bottom: agent public variables.

Next, we consider the agent interface. The chief concerns here are to support exact and efficient probabilistic inference, and to protect agent privacy. The efficiency and privacy concerns demand that a minimum amount of information to be exchanged between agents. The exactness concern demands that a sufficient amount of information to be exchanged over an interface. Recall that an agent interface (corresponding to a hyperlink in the hypertree) is the unique information channel between the two agent communities. Mathematical analysis [28] shows that the agent interface should consist of a set of variables that renders the two agent communities conditionally independent, and a message from an agent should be the agent's subjective probability distribution over the interface variables. This message contains all relevant information to inform the other agent community. Anything less is not sufficient in general.

The agent interfaces are specified through the tool Integrator by specifying, for each agent, a set of *public* variables (using the "Pub Var" function button at top of Figure 9). From the set of public variables in each agent, the agent interfaces can be be derived as the intersection of these variables between adjacent agents. For instance, A_0 has public variables $b_0, ..., b_{10}$: signals exchanged between U_0 and U_1 (see the first line in the bottom of Figure 9).

From the general requirement of the agent interface, a number of implied requirements can be derived and are enforced by the tool Integrator. Each public variable in an agent must be associated with at least another adjacent agent (otherwise, the variable is not really public). For each pair of adjacent agents, the two corresponding sets of public variables must have a non-empty intersection (otherwise, the content of message to be exchanged between them is undefined). If non-adjacent agents A_i and A_j have a common public variable, then it must be a public variable in each agent along the hypertree pathway between A_i and A_j. Otherwise, information from A_i on the variable cannot be communicated to A_j. This is because any information to be delivered between them must be conveyed indirectly through the agents between them on the hypertree, as dictated by the agent organization. The tool Integrator automatically enforces the above requirements during specification and gives feedbacks to Assembler until all conditions mentioned above are satisfied.

The MAS is now *logically* specified. However, in order for agents to communicate according to the specified organization, the MAS has to be *physically* set up. That is, for each agent to be able to communicate directly to its hypertree neighbors, it must know their physical addresses in the computer network. To do so, Assembler uses the tool Binder from WebWeavr:

Binder is a special agent and its physical address is known to all agents. When it starts, it is given access to the organization specification of the MAS. It then waits for each agent to register. An agent registers itself by sending its physical address to Binder. For instance, agent A_0 (nicknamed *huge0*) sends its host computer IP address and port number to Binder. After all agents have registered, Binder notifies each of them with the physical address of each adjacent agent on the hypertree as well as the set of public variables shared between them, the agent interface.

The successful termination of the binding and agent registration mark the integration of the MSBN-based MAS. Each agent now knows to whom it can communicate directly, how to reach them, and what message content should be exchanged with them.

5 Model Verification

To ensure exact inference, the knowledge representation of the MAS must satisfy two additional conditions. Both conditions are related to the directions of arcs in agents' subnets. One of them has a global scope and the other is restricted to interface variables only.

First, we consider the global condition. When agents' subnets are viewed as a whole (by merging their public variables), it must be a DAG. This requirement is implied by the causal interpretation of the graphical structure of subnets. If we start from a variable in a subnet, traverse subnets through a directed path, and finally return to the same variable, then the graphical structure has violated the causal interpretation.

As we mentioned before, each subnet is a DAG, specified by the corresponding agent developer. However, when multiple DAGs (one for each subnet) are merged together, it may be cyclic. This is illustrated in Figure 10. When the three DAGs G_1, G_2 and G_3 are merged through their public nodes (labeled identically in each DAG), a directed path $(a, c, d, b, n, k, g, j, l, a)$ is formed.

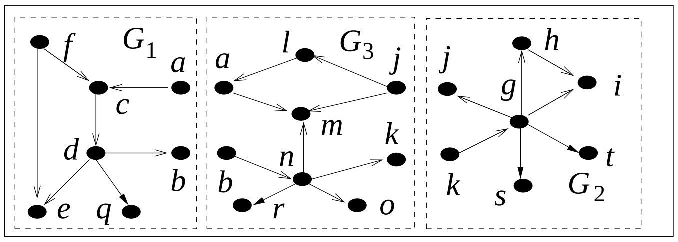


Fig. 10. A directed cycle is formed after the three DAGs are merged.

The possible cyclicity from merging multiple DAGs means that just ensuring acyclicity at each subnet is not sufficient. That is, the global acyclicity cannot be enforced at the level of individual agent developers. Verification performed at the level of agent society is necessary. However, as each subnet is *private* (built by an independent vendor), the global acyclicity cannot be verified by physically merging individual subnets (which requires disclosure of the internal structure of each subnet to a centralizing agent). Despite its seeming impossibility, a verification method has been developed [29] that only requires each agent to pass messages to their hypertree neighbors on whether its subnet contains any parent or child of their shared variables (but not how many and what they are). Nothing else about the internal structure of its subnet is disclosed. Based on such messages, agents can cooperate to detect global cyclicity whenever it occurs and to verify global acyclicity whenever it holds. The verification tool DVerify in WebWeavr toolkit implements the method, whose operation will be briefly illustrated below.

Next, we consider the directionality of arcs that connect public variables. As mentioned in Section 4, variables in an agent interface should render the two corresponding agent communities conditionally independent. In other words, no matter whether or not the interface variables have been observed, passing the subjective probability distribution over these variables from one community (through the corresponding agent) should sufficiently inform the other community. When a public variable is involved in a particular type of dependency, termed *induced dependence* [1], it can cause violation of the requirement.

In Figure 11, for instance, the fragment of a digital system in (a) has been partitioned and represented in two agents A_1 and A_2 as shown in (b). The interface between the agents contains variable sig_5 that corresponds to the output signal sig_5 of AND gate g_4. Suppose that agent A_1 observed input signal of NOT gate g_1 to be $sig_1 = 0$ as well as signal $sig_5 = 0$. A_2 observed input signal of NOT gate g_3 to be $sig_3 = 0$. From the intended functions of NOT and AND gates, we know that at least one of g_1, g_3 and g_4 is faulty. However, if A_1 passes its probability distribution on sig_5 to A_2, it is not possible for A_2 to realize that g_3 or g_4 might be faulty, since A_2 has no

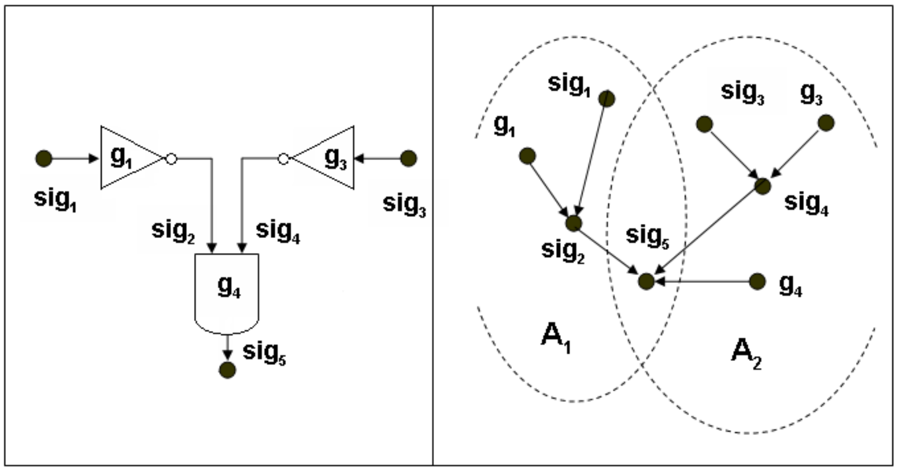


Fig. 11. (a) A fragment of a digital system. (b) Representation of the fragment in subnets of adjacent agents.

information about the expected value of sig_2. In fact, A_2 does not even know the existence of variable sig_2 since it is private to A_1.

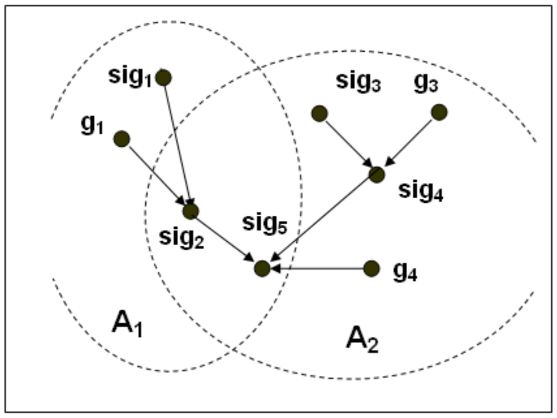


Fig. 12. Modified subnet representation that satisfies the d-sepnode condition.

This problem lies in the fact that none of the agents has all the parent variables of sig_5, namely, sig_2, sig_4 and g_4. To avoid such problem, it is required that every public variable is a *d-sepnode*: A public variable x is a d-sepnode if at least one subnet contains all parent nodes of x from all subnets. The d-sepnode condition can be satisfied in the above example by making variable sig_2 public, as shown in Figure 12.

With this new subnet representation, A_2 can encode the dependence of sig_5 on sig_2, sig_4 and g_4 within its subnet. If A_1 is to pass a message to A_2, the message not only include the information $sig_5 = 0$, it also include A_1's expectation on the value of sig_2. From the dependency, A_1's expectation on sig_2 (namely, $sig_2 = 1$), and A_2's own expectation on sig_4 (namely, $sig_4 = 1$), A_2 will be able to identify the abnormal behavior of the digital system.

Again, because each subnet is *private* and the parent variables of a public variable may also be private (e.g., the variable sig_2 in Figure 11), the d-sepnode condition cannot be verified by agents working independently. Due to the need to protect agent privacy, it cannot be verified by physically merging individual subnets at a centralizing agent either. A method has been developed [30] that essentially requires an agent to tell its neighbors, for each of its public variable x, whether it contains any private parent variables of x (but not how many and what they are). Based on these messages, agents can cooperate to detect every non-d-sepnode and to verify every d-sepnode. The method is also implemented in the tool DVerify in WebWeavr toolkit.

To cooperate in verification of global acyclicity and d-sepnode condition, each agent executes a copy of DVerify. One agent, arbitrarily chosen as the coordinator initiates verification. During verification, messages will be passed among agents along the hypertree, interleaved with local computation at each agent. At the end of the cooperation, the coordinator agent is able to announce whether the MAS has passed the global acyclicity test and d-sepnode test.

6 Agent Interface Enhancement

An MSBN-based MAS that has passed the above verification can support autonomous and exact multiagent probabilistic reasoning. However, communication between agents may not be efficient. An agent communicates with an adjacent agent by sending its subjective probability distribution over their interface. For instance, the interface between A_1 and A_3 has 12 binary variables $(e_0, ..., e_{11})$ and a message between them contains 4096 probability values. In general, if the interface consists of m variables and each has k possible values, the message contains k^m probability values.

To reduce the message size while supporting exact inference, factorization of the probability distribution over the interface can be explored. For instance, if variables $e_0, ..., e_4$ are conditionally independent of $e_8, ..., e_{11}$ given e_5, e_6, e_7, then the message between A_1 and A_3 can be encoded into two distributions over $e_0, ..., e_7$ and $e_5, ..., e_{11}$ with a total size of 256+128 = 384: a reduction of factor 10.

How to explore the conditional independence existing in the agent interface through subnet compilation is presented in the next section. In this section, we address the situation where no conditional independence relations can be found within the natural agent interface or those that exist do not yet offer sufficient efficiency gain. One solution is to enhance the interface with additional variables that can bring conditional independence relations from the subdomain into the interface.

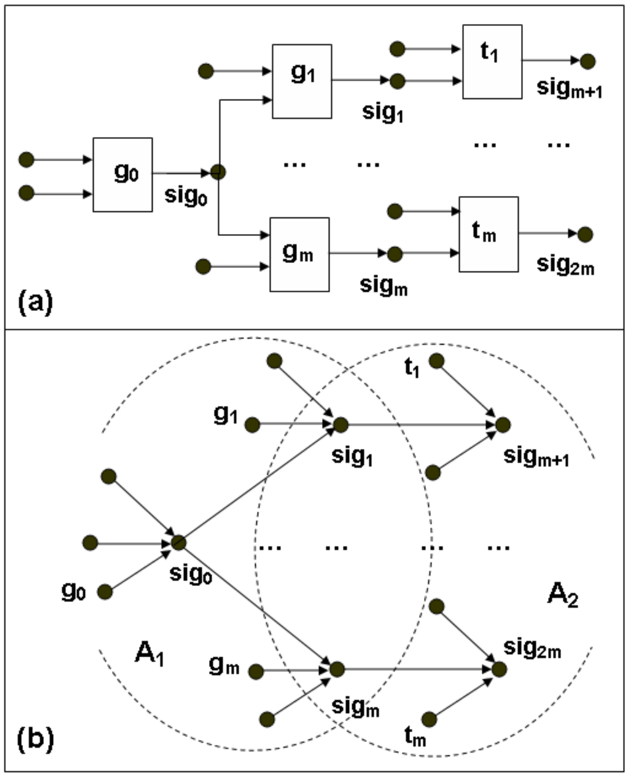


Fig. 13. (a) A fragment of a digital system. (b) Corresponding adjacent subnets.

Figure 13 illustrates the idea with a fragment of a digital system (a). It is represented as adjacent subnets in (b). The agent interface consists of m variables $sig_1, ..., sig_m$. There is no conditional independence relations within the interface. This can be understood as follows: For any three signals sig_i, sig_j and sig_k $(1 \leq i < j < k \leq m)$, if the value of sig_i is known, it helps to generate expectation on the value of sig_0, which in turn generates expectation on the value of sig_j. Even if the value of sig_k is known, it cannot diminish this dependency between sig_i and sig_j. Hence, there is no conditional independence relations among them and a message over the interface has a size of 2^m.

Figure 14 shows new subnets where the agent interface is enhanced by adding the variable sig_0. Now, if the value of sig_0 is known, it helps to generate expectation on the value of sig_i. Knowing in addition the value of sig_j

cannot change that expectation at all. Hence, sig_i and sig_j are conditionally independent given sig_0: an independence relation has been introduced into the interface. Furthermore, the independence relation holds for every pair of i and j. This allows the probability distribution over the interface to be factorized into m distributions each defined over two variables sig_0 and sig_i ($i = 1, ..., m$). The total size of the inter-agent message is reduced to $4m$ from the original size of 2^m.

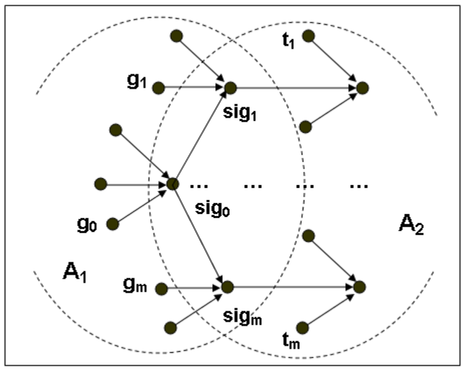


Fig. 14. New subnets with enhanced agent interface.

To explore this idea, suitable variables (such as sig_0) need to be identified. Identification of these variables among a large number of alternatives is non-trivial. The process often requires cooperation between agents. For instance, if sig_{m+1} through sig_{2m} all feed into a common gate in A_2, then unless its output signal is also added into the agent interface, the above mentioned reduction in message size cannot be achieved. Instead of burdening the agent developers with the enhancement task, it can be delegated to agents.

The enhancement involves search through many alternatives, including disclosure of some originally private variables, to neighbor agents, as promising enhancement candidates. To protect agent privacy during enhancement, each agent classifies variables in its subnet into three groups: private, public, and preferably private. The *public* group forms the natural initial agent interface. The *private* group will be kept so absolutely. The *preferably private* group is initially private, but the agent is allowed to make some elements of this group public if it believes that the disclosure may improve efficiency. The agent is required, however, to keep the disclosed variables as fewer as possible. That is, any disclosed candidate variable should be highly promising through the agent's local evaluation. The actual efficiency improvement of an enhancement can only be determined by agents' cooperative evaluation.

A suite of algorithms for multiagent interface enhancement has been developed [31]. Through multiagent heuristic search, each of the four agent interfaces are enhanced. For example, the interface between A_1 and A_3 (consisting of $e_0, ..., e_{11}$) is enhanced with additional variables

$$yd_{82}, yd_{101}, yd_{106}, w_{14}, wr_{16}, wr_{18}.$$

These variables bring several independence relations into the interface. For instance, e_0, e_1, e_2 are independent of e_3, e_4, e_5 given wr_{16}, yd_{106}. As the result of enhancement, the message size between each pair of adjacent agents is reduced significantly, as shown in Table 1, with the new message size to be as low as about 4% of the original (between A_3 and A_4).

Table 1. The message size between each pair of adjacent agents before and after interface enhancement.

Interface	$A_0 - A_1$	$A_1 - A_2$	$A_1 - A_3$	$A_3 - A_4$
Before	2048	1024	4096	4096
After	136	136	336	160

Agent interface enhancement is the only technical step where information about variables that are initially private (those that are preferably private) may be disclosed. This step is not necessary for exact inference using the MSBN-based MAS and should be regarded as an option for trading privacy with efficiency.

7 Compilation into Linked Cluster Trees

Inference computation in an MSBN-based MAS consists of local inference at individual agents and communication among agents. Local inference involves updating the agent's belief (subjective probability distribution) over its subdomain based on local sensor observations. During communication, the basic operation of an agent involves passing to another agent its subjective probability distribution over their interface (the message). The two computations are intertwined: A message for communication must be derived from the sending agent's local distribution over its subdomain, and a message received should be processed for updating the local distribution over the receiving agent's subdomain.

Suppose that an agent's subdomain consists of n variables and each has up to k possible values. The probability distribution over the subdomain has a size of k^n. To make the local inference efficient, the agent must avoid direct manipulation of the distribution. The idea is to explore conditional independence and factorization of the distribution. Each agent compiles its subnet into a cluster tree, where variables are grouped into *clusters* with intersections of adjacent clusters referred to as *separators*. The cluster tree is so constructed such that the intersection of any two clusters is contained in every cluster on the path between them. The property ensures that any update on the probability of a variable located in a cluster can be propagated to every other cluster that contains the same variable. This idea of using such cluster trees for probabilistic inference was proposed first in the single-agent paradigm (see [32, 33, 34, 3]). It has been extended into operations under the multi-agent paradigm [35, 7]. Details on compilation can be found from the given reference. Figure 15 shows the cluster tree compiled from the subnet of agent A_0 through cooperation with other agents using the tool Structure Compiler in WebWeavr toolkit.

Each cluster is associated with a probability distribution over its member variables obtained from the CPTs in the subnet. The cluster tree encodes the conditional independence relations existing in the subnet: Two adjacent clusters are conditional independent given their separator. These independence relations allow factorization of the the agent's subjective probability distribution over its subdomain. The cluster distributions are more efficient spacewise, yet they uniquely define the agent's subjective probability distribution over its subdomain [33]. Furthermore, the tree topology allows local inference to be performed by passing messages (probability distributions) over separators along the tree structure (we describe the inference operation in the next section). When the size of the largest cluster is bounded, the inference is efficient.

During communication, an agent needs to send its subjective probability distribution over an agent interface to the neighboring agent. As we discussed earlier, sending the message as a single distribution over the agent interface has the exponential complexity, which motivated agent interface enhancement. However, interface enhancement only ensures that there exists conditional independence relations within the interface. It does not create an explicit data structure to utilize these independence relations. The data structure that serves this purpose is called *linkage tree*.

Essentially, the linkage tree is also a cluster tree. The cluster tree compiled from the agent's subnet is composed of all variables in the agent's subdomain. On the other hand, the linkage tree is composed of only variables in an agent interface and is used only for computation of message to the corresponding adjacent agent. A linkage tree is derived from the local cluster tree and it inherits all the conditional independence relations among interface variables that are explicitly encoded in the local cluster tree. Details on how to derive linkage tree from local cluster tree can be found from reference [36]. Figure 16 shows the linkage tree of agent A_0 for computing messages to A_1. Each cluster in the linkage tree is called a *linkage*. Each linkage has a corresponding cluster in the cluster tree, called its *host*, that contains the linkage.

Each linkage is associated with a probability distribution that is derived from the distribution associated with its host. From these linkage distributions, the agent's subjective probability distribution over the agent interface can be constructed through factorization. Although the distribution over the

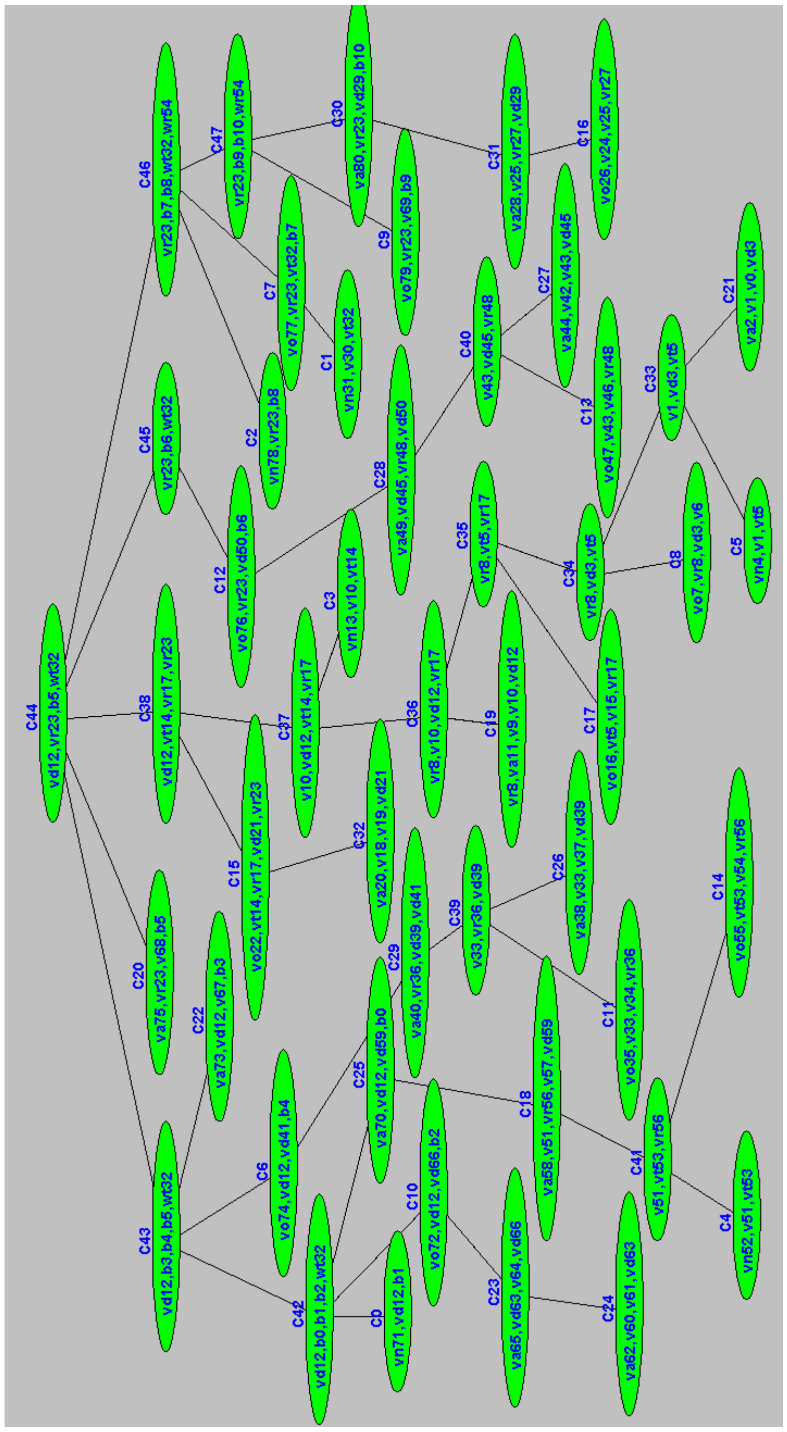


Fig. 15. The cluster tree for agent A_0.

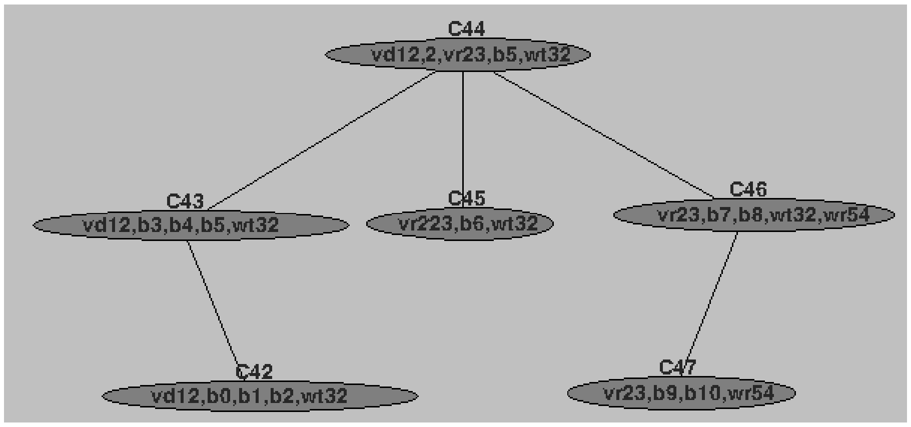


Fig. 16. The linkage tree for computing messages between A_0 and A_1.

interface has a size of $2^{15} = 32768$, the inter-agent message made of linkage distributions has a total size of $3 * 2^5 + 2 * 2^4 + 1 * 2^3 = 136$.

Note that the possibility of efficient message representation using linkage trees is a direct consequence of exploring conditional independence within the agent interface. The compilation operation automatically identifies such independence relations if they exist. If these relations do not yet exist in the natural agent interface, they must be brought into the interface through interface enhancement (Section 6) .

8 Multiagent Inference

The above compilation effectively converts the collective knowledge of multiple agents, originally represented as an MSBN, into a set of linked cluster trees. Using the local cluster tree, each agent can perform inference autonomously without cooperation from other agents. For instance, if x is a variable representing a sensor output and an agent observes the value x being logic 1, then the observation can be entered into the cluster tree as follows: First, a cluster that contains x is selected. As mentioned above, the cluster has an associated probability distribution, which specifies the probability for each combination of the values of variables in the cluster. If a combination has the value of x being logic 0, then the probability of the combination is set to 0, meaning that this combination is now impossible given the observation $x = \mathit{logic}\ 1$. The remaining combinations will have their probability values scaled up so that they sum to one, while maintaining their original relative magnitudes. This operation is termed *entering observation*.

After each sensor observation has been entered into the corresponding cluster, the change in these clusters must be propagated to other clusters in order to achieve their impact on other variables that depend on them. This is done through message passing along the local cluster tree. Each cluster receives a message from each neighbor cluster. It sends one message to each neighbor after it has received messages from all other neighbor clusters. Each message is simply a probability distribution over the corresponding separator and it is computed from the distribution associated with the sending cluster and messages received by the cluster. Over each separator, exactly two messages are sent, one in each direction.

Since there are no more clusters than variables in the subdomain and there are less separators than clusters, it can be seen that if the clusters are small in size, the message propagation is efficient. It has also been shown [33, 3] that the cluster probability distribution obtained through the propagation is exact relative to the probability theory, the background knowledge of the subnet, and sensor observations.

Since subnets in our case study represents components that are interconnected and therefore mutually constrained, sometimes communication with other agents allows an agent to better ascertain the current situation of its component than what is achievable by the agent's autonomous inference only. When such is the case, agents engage in a communication operation so that they can benefit from each other's local sensor observations. The communication operates in a similar fashion as the message passing in a cluster tree, but at the agent level and along the hypertree:

Each agent receives a message from each neighbor agent according to the hypertree agent organization. It sends one message to each neighbor agent after it has received messages from all other neighbor agents. A message is a set of probability distributions each of which is over a linkage with the receiving agent and is derived from the probability distribution of the linkage host cluster. Collectively, these distributions define the sending agent's belief over the agent interface. Between each pair of adjacent agents on the hypertree, exactly two messages are sent, one in each direction.

Since there are less linkages than clusters in the local cluster tree, there are as many hypernodes in the hypertree as the number of agents, and there are less hyperlinks than hypernodes, it can be seen that if the clusters are small in size, the agent communication is also efficient. Furthermore, mathematical analysis [36, 7] shows that the agents' beliefs after communication are exact, relative to the probability theory, the collective background knowledge of all agents encoded in their subnets, and sensor observations of all agents. In the next section, we demonstrate how these operations can be used to answer the normality and culprit queries.

9 Sensor Net Monitoring and Fault Isolation

To monitor the digital system domain, each agent collects sensor outputs and reason about the state of its subdomain autonomously. Less frequently, agents may choose to communicate in order to benefit from information in other agents. Through interleaving local inference and communication, agents can collectively answer the normality and culprit queries.

The tool DMasMsbn in WebWeavr supports agent sensing, inference and communication. We demonstrate digital system monitoring through the following scenario: AND gate wa_{130} in U_1 and OR gate $y0_{49}$ in U_3 are faulty and produce incorrect output signals. The incorrect outputs propagate through other gates and produce more incorrect signals throughout the system. Agents' task is to detect that the system is abnormal (answering the normality query) and to isolate the faulty gates (answering the culprit query).

To demonstrate the operation of the MAS while avoiding the cost of implementing the digital hardware physically, the tool Scenario Simulator from WebWeavr is used to simulate the digital system and associated sensor network. The simulator accepts a set of externally specified input signals to the digital system, simulates the behavior of all digital gates including the faulty gates, and generates output signals of all gates. It responds to agents' request for observations and enforces the assumption that the state of a gate is not observable. When a valid request is received from an agent, the value of the corresponding signal as would be perceived by the sensor will be sent to the agent.

To monitor the domain, each agent is assumed to have the bandwidth to observe at one time as many sensors as about 5% of variables in its subdomain. We assume that all signals are observable except the outputs of the two faulty gates wa_{130} and $y0_{49}$. As there are more observable signals than what are permitted by the bandwidth, some strategy must be utilized to chose what to observe. If gates differ in their prior probabilities of being faulty, those gates with high fault probabilities may be observed with priority. Signals corresponding to their input and output are likely to detect their faults soon after they occur. In the case study, we have assumed the same prior fault probability for all gates. Hence, a random set of signals is observed initially. The first round of observations is shown in Table 2. After entering the

Table 2. Sensor observations in round 1.

A_0	$v_9, v_{30}, vr_{23}, vd_{45}, vd_{12}$
A_1	$w_{37}, wd_{50}, wt_{24}, e_3, w_{89}, w_{136}, w_{121}, w_{120}, w_{119}, w_{107}$
A_2	$x_{27}, x_9, x_{12}, xd_{33}$
A_3	$y_{45}, y_{48}, y_{104}, y_{69}, y_{97}, y_{111}, y_{12}, yr_{27}$
A_4	$z_{60}, z_{58}, z_{55}, z_{15}, z_1, z_{12}$

observations, each agent updates its belief autonomously. Since these local observations are not sufficient to detect any abnormality within each subdomain, and automonous reasoning at individual agents cannot take into account the constraints between components, none of the agents detects any problem.

However, after one round of communication among agents, during which one message is passed from each agent to each adjacent agent, the pooling of information allows agents to detect abnormality. A_0 has $P(va_{44} = bad|obs) = 0.025$. Note that this is two and half times higher than the prior fault probability value 0.01. A_1 has a number of gates suspected,

$$wn_{132}, wo_{124}, wo_{163}, wa_{126}, wa_{122}, wa_{139}, wa_{141}, wa_{130},$$

for instance, $P(wa_{130} = bad|obs) = 0.131$. Similarly, A_2 has $P(xa_{32} = bad|obs) = 0.132$, A_3 suspected

$$yn_{39}, yo_{43}, yo_{49}, yo_{15}, yo_{102}, yo_{121}, yo_{95}, ya_{105}, ya_{46},$$

and A_4 suspected $zn_{20}, zn_6, zo_{18}, zo_{61}, za_{59}, za_{13}, za_{56}$. Therefore, agents have collectively answered the normality query negatively.

The large number of candidate faulty devices is a consequence of propagation of incorrect outputs of the two faulty gates to other devices which causes their outputs to be incorrect. Note that the set of candidates includes the two faulty gates wa_{130} and $y0_{49}$. Therefore, if these devices are replaced, the system will be back to normal. However, that would be too costly. The large number of candidates and low faulty probability value for each tell the agents that further investigation is needed.

Alarmed, each agent makes more observations, subject to the bandwidth restriction. Since the agents now have some candidate gates suspected to be faulty, the observations can be focused on the input and output signals of these gates. A_0 observes signals associated with the suspected gate va_{44}. Its output vd_{45} has been observed. Hence, its inputs v_{42} and v_{43} are observed. After entering observation and autonomous inference, A_0 no longer suspects va_{44}.

A_1 observes the output of each suspected gate, as listed in Table 3, except that of wa_{130} (as has been deliberately forbidden to make the decision process more interesting). After entering observation and autonomous inference, A_1 reduces its uncertainty on the original eight candidate faulty gates and now suspects only three:

$$wn_{128}, wn_{132}, wa_{130}.$$

Note that wn_{128} is not one of the gates suspected earlier.

Table 3. Sensor observations in round 2.

A_0	v_{42}, v_{43}
A_1	$wd_{140}, wd_{142}, wt_{133}, wd_{123}, wr_{125}, wd_{127}, c_5$
A_2	xd_{16}, c_5
A_3	$yd_{47}, yr_{96}, yr_{103}, yd_{106}, yr_{16}, yt_{40}, yr_{44}, e_3$
A_4	$h_6, zd_{57}, zt_{21}, zr_{19}, zt_7, zd_{14}$

A_2 observes two signals and no longer suspects xa_{32} after inference. A_3 observes 8 signals and decides that $P(yo_{49} = bad|obs) = 0.504$ and $P(yo_{95} = bad|obs) = 0.504$. Given that the signal between the two, yr_{50}, is not observable, this is the best that anyone can achieve. A_4 observes 6 signals. After inference, it decides that its subdomain is normal and does not have any fault.

As A_1 suspects three gates, it makes one more observation related to them: wt_{129}. After inference, it reduces the suspected gates to only wn_{132} (with $P(wn_{132} = bad|obs) = 0.387$) and wa_{130} (with $P(wa_{130} = bad|obs) = 0.617$), which is the best that anyone can achieve given the unobservability of the signal between the two gates.

As the result of the above multiagent inference, A_1 correctly isolates faulty devices to wn_{132} and wa_{130}, and A_3 correctly isolates to yo_{49} and yo_{95} (note that wa_{130} and yo_{49} are the true faulty devices). The probability of each of these four devices being faulty is at least 0.387, while all other gates suspected earlier have their probabilities of being faulty dropped to almost zero in all agents. Given that we have forbidden observability between wn_{132} and wa_{130} and between yo_{49} and yo_{95}, the agents have answered the culprit query well. That is, they isolated faulty devices to the smallest possible set given the information available from the sensor network. Replacement of the four devices according to the answer to the query will return the system to its normal state.

What would happen if some agents in the MAS fail? To ensure exactness of inference/communication as well as efficiency, the MSBN-based MAS uses the hypertree agent organization. Because each hyperlink separates the MAS into two agent communities, if the communication link between two adjacent agents fails, the two resultant communities will no long be able to cooperate as we demonstrated above. Furthermore, if an agent of k neighbors fails, the MAS will be broken into k separate communities.

On the other hand, agents in each community can still cooperate within themselves. Theoretical analysis [7] shows that after they communicate, each agent's belief is exact relative to the probability theory, the knowledge encoded in all agents within the community, sensor observations in the entire MAS up to the last communication before the breaking of the MAS, and sensor observations made by all agents in the community since the breaking. Therefore, the MSBN framework allows the MAS to fail gracefully, rather than to function as all or nothing.

10 Summary

MSBNs extend BNs to provide a rigorous computational framework for intelligent sensor network applications. The key advantages of the framework are the following:

1. Agents' beliefs regarding the interpretation of the sensor observations are exact according to Bayesian probability theory.

2. Inference at each agent is autonomous and no centralized control is needed.
3. Communication within the agent society can be initiated by any agent and no fixed controller is needed.
4. As long as the dependence structure of agents are sparse, the inference and communication are efficient.
5. Operations for model construction, model compilation, inference, and communication protect agent privacy. Agent interface enhancement is the only step where information about preferably private variables may be disclosed. This step is not necessary for exact inference using the MSBN-based MAS and should be regarded as an option for trading privacy with efficiency.

The performance guarantees (on autonomy, exactness, efficiency and privacy) offered by the MSBN framework require careful model construction, model compilation and inference-communication operations. Most of the compilation, inference and communication operations can be fully automated as demonstrated by tools in WebWeavr toolkit. The model construction is the step that demands particular effort from sensor network practitioners even with the aid of tools. The modeling task can be broken down into the following:

1. The integrator of the MAS needs to partition the domain into subdomains over which individual agents will be developed, to specify agent interfaces, and to define the hypertree agent organization. For many problems, there exists some natural partition. Care must be taken so that all requirements on agent interfaces are satisfied.
2. The developer of each agent must specify the agent's subnet. This includes the dependence structure over the agent's subdomain in terms of a DAG and the CPT for each node in the DAG.
3. Once the agent organization and subnets are specified, they should be subject to verification. If global acyclicity and d-sepnode conditions are violated, the subnets must be revised. Negotiation among agent developers and integrator is needed to determine alternative modifications to subnets and who will make the changes.
4. The modeling task may not end yet after subnets pass verification. After they are compiled into linked cluster trees, resultant linkage trees may not support efficient communication. In such case, interface enhancement is needed. To enable enhancement, for each subnet, the agent developer needs to specify a subset of variables as preferably private so that they may be added to the agent interface.
5. Once agent interface is sufficiently efficient, measured by the size of the largest linkage, the model construction is complete.

As any model of a complex domain, it only reflects the best knowledge available at the time. As new information becomes available, the model may be refined. The functional units of an MSBN-based MAS are the agents. Therefore, the modeling units are the subnets embodied by agents. A subnet consists of its graphical structure and its numerical CPTs. When the problem domain is an artifact, such as a piece of equipment, the subnet structure is constructed from the structure of the designed artifact. Hence, unless the artifact is modified, it is unlikely that the subnet structure needs refinement. On the other hand, CPTs encodes information on the artifact's faulty behavior, which is not designed. Therefore, refinement of CPTs is not only possible but also desirable. It can be easily accomplished by utilizing the fault frequency data accumulated. As subnets are thus refined, the MAS will perform more effectively.

Acknowledgements

The financial support from National Sciences and Engineering Research Council (NSERC) of Canada through Discovery Grant is acknowledged.

References

1. J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, 1988.
2. R.E. Neapolitan. *Probabilistic Reasoning in Expert Systems*. John Wiley and Sons, 1990.
3. G. Shafer. *Probabilistic Expert Systems*. Society for Industrial and Applied Mathematics, Philadelphia, 1996.
4. E. Castillo, J. Gutierrez, and A. Hadi. *Expert Systems and Probabilistic Network Models*. Springer, 1997.
5. R.G. Cowell, A.P. Dawid, S.L. Lauritzen, and D.J. Spiegelhalter. *Probabilistic Networks and Expert Systems*. Springer, 1999.
6. F.V. Jensen. *Bayesian Networks and Decision Graphs*. Springer-Verlag, New York, 2001.
7. Y. Xiang. *Probabilistic Reasoning in Multi-Agent Systems: A Graphical Models Approach*. Cambridge University Press, Cambridge, UK, 2002.
8. Y. Xiang. WebWeavr-IV Research Toolkit. www.cis.uoguelph.ca/˜yxiang/.
9. H.P. Nii. Blackboard systems: the blackboard model of problem solving and the evolution of blackboard architectures. *AI Magazine*, 7(2):38–53, 1986.
10. A. Rao and M. Georgeff. Deliberation and its role in the formation of intentions. In B. D'Ambrosio, P. Smets, and P.P. Bonissone, editors, *Proc. 7th Conf. on Uncertainty in Artificial Intelligence*, pages 300–307. Morgan Kaufmann, 1991.
11. M. Wooldridge. *An Introduction to Multiagent Systems*. John Wiley & Sons, 2002.
12. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.
13. R. Reiter. A logic for default reasoning. *Artificial Intelligence*, 13.

14. J. McCarthy. Circumscription - a form of non-monotonic reasoning. *Artificial Intelligence*, 13:27–39, 1980.
15. D.V. McDermott and J. Doyle. Non-monotontc logic i. *Artificial Intelligence*, 13:41–72, 1980.
16. J. Doyle. A truth maintenance system. *Artificial Intelligence*, 3(12):231–271, 1979.
17. C.L. Mason and R.R. Johnson. DATMS: a framework for distributed assumption based reasoning. In L. Gasser and M.N. Huhns, editors, *Distributed Artificial Intelligence II*, pages 293–317. Pitman, 1989.
18. M.N. Huhns and D.M. Bridgeland. Multiagent truth maintenance. *IEEE Trans. Sys., Man, and Cybernetics*, 21(6):1437–1445, 1991.
19. N. Roos, A.T. Teije, and C. Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. In *Proc. 2nd Inter. Joint Conf. on Autonomous Agents and Multiagent Systems*, pages 655–661, Melbourne, 2003.
20. C. Guestrin, P. Bodi, R. Thibau, M. Paski, and S. Madde. Distributed regression: an efficient framework for modeling sensor network data. In *Proc. 3rd inter. symposium on Information processing in sensor networks*, pages 1–10, Berkeley, CA, 2004.
21. C. Boutilier. Multiagent systems: Challenges and opportunities for decision-theoretic planning. *AI Magazine*, pages 35–43, Winter, 1999.
22. P. Xuan, V. Lesser, and S. Zilberstein. Communication in multi-agent Markov decision processes. In *Proc. 6th Inter. Conf. on Multi-agent Systems*, pages 467–468, 2000.
23. R. Nair, P. Varakantham, M. Tambe, and M. Yokoo. Networked distributed pomdps: A synthesis of distributed constraint optimization and pomdps. In *Proc. 20th National Conference on Artificial Intelligence*, pages 133–139, 2005.
24. D.S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *Proc. 16th Conf. on Uncertainty in Artificial Intelligence*, pages 32–37, Stanford, 2000.
25. S. Moral, R. Rumi, and A. Salmeron. Mixtures of truncated exponentials in hybrid Bayesian networks. In *Lecture Notes in Artificial Intelligent*, volume 2143, pages 135–143. Springer-Verlag, 2001.
26. R.T. Cox. Probability, frequency and reasonable expectation. *American J. of Physics*, 14(1):1–13, 1946.
27. de Finetti. Foresight: its logical laws, its subjective sources. *Ann. Inst. H. Poincare*, 7:1–68, 1937. Reprinted in 1980 in Studies in Subjective Probability (H.E. Kyburg and H.E. Smokler, Eds.), 93-158.
28. Y. Xiang and V. Lesser. On the role of multiply sectioned Bayesian networks to cooperative multiagent systems. *IEEE Trans. Systems, Man, and Cybernetics-Part A*, 33(4):489–501, 2003.
29. Y. Xiang. Verification of dag structures in cooperative belief network based multi-agent systems. *Networks*, 31:183–191, 1998.
30. Y. Xiang and X. Chen. Interface verification for multagent probabilistic inference. In J.A. Gamez, S. Moral, and A. Salmeron, editors, *Advances in Bayesian Networks*, pages 19–38. Springer, Berlin, 2004.
31. Y. Xiang and K. Zhang. Agent interface enhancement: Making multiagent graphical models accessible. In *Proc. 5th Inter. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS'06)*, pages 19–26, 2006.

32. S.L. Lauritzen and D.J. Spiegelhalter. Local computation with probabilities on graphical structures and their application to expert systems. *J. Royal Statistical Society, Series B*, (50):157–244, 1988.
33. F.V. Jensen, S.L. Lauritzen, and K.G. Olesen. Bayesian updating in causal probabilistic networks by local computations. *Computational Statistics Quarterly*, (4):269–282, 1990.
34. F.V. Jensen. *An Introduction To Bayesian Networks.* UCL Press, 1996.
35. Y. Xiang. Cooperative triangulation in MSBNs without revealing subnet structures. *Networks*, 37(1):53–65, 2001.
36. Y. Xiang. A probabilistic framework for cooperative multi-agent distributed interpretation and optimization of communication. *Artificial Intelligence*, 87(1-2):295–342, 1996.