Dynamic Multiagent Probabilistic Inference

Xiangdong An [a,*], Yang Xiang [b], Nick Cercone [c]

[a] *Faculty of Computer Science, Dalhousie University*
Halifax, Nova Scotia B3H 1W5, Canada

[b]*Department of Computing and Information Science, University of Guelph*
Guelph, Ontario N1G 2W1, Canada

[c]*Department of Computer Science and Engineering, York University*
Toronto, Ontario M3J 1P3, Canada

Abstract

Cooperative multiagent probabilistic inference can be applied in areas such as building surveillance and complex system diagnosis to reason about the states of the distributed uncertain domains. In the static cases, multiply sectioned Bayesian networks (MSBNs) have provided a solution when interactions within each agent are structured and those among agents are limited. However, in the dynamic cases, the agents' inference will not guarantee exact posterior probabilities if each agent evolves separately using a single agent dynamic Bayesian network (DBN). Nevertheless, due to the discount of the past, we may not have to use the whole history of a domain to reason about its current state. In this paper, we propose to reason about the state of a distributed dynamic domain period by period using an MSBN. To reduce the influence of the ignored history on the posterior probabilities to a minimum, we propose to observe as many observable variables as possible in the modeled history. Due to the limitations of the problem domains, it could be very costly to observe all observable variables. We present a distributed algorithm to compute all observable variables that are relevant to our concerns. Experimental results on the relationship between the computational complexity and the length of the represented history, and effectiveness of the approach are presented.

Key words: Multiagent uncertain reasoning, Reasoning in dynamic systems, (Dynamic) Bayesian networks, Agent privacy, Exact and approximate reasoning

1 Introduction

For cooperative multiagent systems, one of the tasks we need to study is how multiple agents can collectively reason about the state of the problem domain based

* Corresponding author. Tel.:+1-902-420-5003; fax:+1-902-496-8101.
Email address: `xan@cs.dal.ca` (Xiangdong An).

on their local knowledge, local observation (evidence), and limited communication with each other. This task is referred to by some authors as *distributed interpretation* [1], which arises in many areas such as inventory control, power network grids, equipment monitoring, smart house, cooperative design, battlefield assessment, and surveillance.

Multiply sectioned Bayesian networks (MSBNs) [2] provide a coherent framework for probabilistic reasoning in distributed interpretation systems with uncertainties, which have been applied in many areas such as medical diagnosis [3], equipment monitoring and diagnosis [4], and distributed network intrusion detection [5]. However, problem domains such as medical diagnosis and equipment monitoring and diagnosis are dynamic in general. MSBNs do not provide facilities to properly manage and absorb historical information in distributed dynamic domains [6]. Actually in distributed dynamic domains, if we allow the historical probabilistic messages from different subdomains to be passed over time separately, the dependencies among these separated messages would be lost. This indicates that, at least for probabilistic inference, the spatial distribution of multiagent systems conflicts with the temporal message passing requirement of dynamic domains. We have difficulty to perform probabilistic inference in dynamic multiagent systems ideally on all of the following aspects: exactness, distribution and effectiveness.

We cannot sacrifice the distribution since multiagent inference has to be done distributedly. We could make some tradeoffs between the ideallness and either the exactness or the effectiveness or both.

It has been widely recognized that the recent past is more relevant to the current state of domain than the distant past [7–9], which is called the discount of the past. In the probabilistic framework, weak influence means small effects on the posterior probabilities. In a stochastic world, the influence of the historical probabilistic knowledge would be weakened over distance (the length of influence chains), time and quantity (as many influences combine) [10,11]. In this paper, we propose to reason about the state of a dynamic multiagent domain based on its recent history, instead of its whole history. To reduce the impacts of the ignored history on the inference results, we propose to observe (gather information from) the modeled history as much as possible. In dynamic domains, some aspects of the remote past are referenced in the recent history. The more variables in the modeled history we observe, the less influence the ignored history would have on the posterior probabilities. Observing everything that is observable would reduce the impacts of the ignored remote past to a minimum. However, each observation would have a cost. It could be very expensive to observe everything. For example, although many medical laboratory tests may help improve the accuracy of a patient's diagnosis, the patient may not want to take all of them due to the cost and the potential side effects involved. To fully and efficiently take advantage of the information provided by the modeled history, we propose to observe all observable variables that are relevant to the concerned variables.

A suite of algorithms is presented to distributedly compute all observable variables that are relevant to the concerned variables, where agents' privacy is preserved. The correctness and the complexity of the set of algorithms is analyzed. To facilitate the understanding of the set of algorithms, a single agent version of the set of algorithms is presented before the multiagent version. An algorithm called Bayes-Ball [12] was once presented to compute the *requisite observations* for a set of concerned variables in Bayesian networks (BNs). The requisite observations for a set S of variables are those observed variables in the domain, which are relevant to the state of S. Bayes-Ball solves the same problem as our single agent version algorithm in the same computational complexity. Nevertheless, Bayes-Ball is different from our single agent version algorithm in that our method computes the requisite observations based on the explicit observable descendant information of nodes in the BNs. Observable descendant information can be reused once obtained. This makes our method cheaper in both time and space complexities when requisite observations for multiple sets of concerned variables need to be computed.

Experimental results on the relationship between the computational complexity and the length of the represented period, and the effectiveness of the approach are presented.

The rest of the paper is organized as follows. In Section 2, we review the related work. The necessary background knowledge is introduced in Section 3. In Section 4, the issues involved in the dynamic multiagent probabilistic inference and the proposed solution are presented. In Section 5, the set of algorithms for computing the observable relevant variables in MSBNs is presented and analyzed. Experimental results are provided in Section 6. In Section 7, the conclusion is made.

2 Related Work

Both the Markov decision processes (MDPs) [13] and the partially observable Markov decision processes (POMDPs) [14] are probabilistic models for probabilistic reasoning and acting in stochastic systems. They have been extended and applied to probabilistic reasoning and acting in dynamic multiagent systems recently.

Extended from MDPs, the multiagent MDPs (MMDPs) [15] assume a full view of the global state by each agent, whereas the decentralized MDPs (Dec-MDPs) assume a different partial view of the global state by each agent [16]. In either MMDPs or Dec-MDPs, an agent can fully observe the state of the world in its view. In Dec-MDPs, agents may be allowed to communicate about their deterministic local states with costs. Though solving an MDP is P-complete, solving a Dec-MDP is NEXP-complete [16].

The decentralized POMDPs (Dec-POMDPs) [17], extended from POMDPs, are more related with our work than Dec-MDPs, where each agent only has an incomplete information about its subdomain. In Dec-POMDPs, though agents execute their local policies distributedly based on their local observations, planning is generally centralized [18]. Agents may be allowed to communicate about their observations to improve the policy computation. In Dec-POMDPs, no probabilistic messages are passed among agents. Hence, there does not exist the divided temporal probabilistic message passing problem. Solving a POMDP is EXP-complete (PSPACE-complete if the transitions are deterministic) [19], whereas solving a Dec-POMDP is NEXP-complete [16].

In POMDPs, the state is encoded in a single random variable, which is not efficient in modeling large state spaces with structures. Dynamic decision networks (DDNs) [20] have been proposed to model and solve sequential decision problems with large structured state spaces. DDNs were extended from dynamic Bayesian networks (DBNs) with decision nodes and utility nodes. Factored POMDPs [21] were proposed to represent large structured POMDPs compactly. In a factored POMDP, a DBN is used to compactly represent the transition models and observations models. In this paper, we investigate how to properly use MSBNs to compactly represent and reason about the states of the distributed partially observable dynamic domains.

In [22], an approximate inference method for DBNs, which we call Boyen-Koller (BK), was investigated. The method works on stochastic processes that are composed of weakly interacting subprocesses. In the method, the joint belief on the DBN interface is approximated by the product of marginals that correspond to respective individual subprocesses. Message passing over time is done approximately through these marginal products, whereas belief updating at each time instant is done exactly. It was shown that the approximation error remains bounded over time. Motivated by BK, a more aggressive DBN approximate inference approach, called Factored Frontier (FF), has been presented [23]. FF is very similar to BK algorithm, but instead of doing an exact belief updating at each time instant, it always works with the factored distributions.

Though the approximation error of BK is bounded over time, it is still unclear how tight the bound is [24]. The tightness of the bound is related with the strength of the interactions among subprocesses (agents). There is no guarantee on the boundness of approximation error from FF. It has been shown [24] both BK and FF are special cases of loopy belief propagation (LBP), which is not guaranteed to converge [25].

3 Background Knowledge

3.1 Notations and Terminology

Let X, Y and Z be disjoint subsets of variables in V. We use the notation $I(X, Z, Y)_P$ to denote the conditional independence of X and Y given Z; thus,

$$I(X, Z, Y)_P \text{ iff } P(\mathbf{x}|\mathbf{y}, \mathbf{z}) = P(\mathbf{x}|\mathbf{z})$$

for any configuration **x** of X, and any configurations **y** and **z** of Y and Z such that $P(\mathbf{y}, \mathbf{z}) > 0$.

In a directed graph, when two arcs meet in a path, the shared node can be described as a node of *tail-to-tail*, *head-to-tail* or *head-to-head*.

Definition 1 *Let X, Y and Z be disjoint subsets of nodes in a DAG G. A path ρ between nodes $x \in X$ and $y \in Y$ is **closed** by Z whenever one of the following two conditions is true: (1) there exists $z \in Z$ that is a node of either tail-to-tail or head-to-tail on ρ; (2) there exists a node w that is a node of head-to-head on ρ and neither w nor any descendant of w is in Z. If both conditions are false, then ρ is rendered **open** by Z. Nodes x and y are **d-separated** by Z if every path between x and y is closed by Z; X and Y are **d-separated** by Z if for every $x \in X$ and $y \in Y$, x and y are d-separated by Z.*

We use the notation $< X|Z|Y >_G$ to denote that X and Y are separated (d-separated) by Z in graph G.

A **dependency model** M over a set U of variables is a model that can determine whether $I(X, Z, Y)_M$ is true, for all possible triplets of disjoint subsets X, Y and Z. A **probabilistic model**, which is a complete specification of a joint probability distribution (JPD), is a dependency model.

A graph G is an **I-map** of a dependency model M over a set V of variables, if there is a one-to-one correspondence between nodes in G and variables in V and for every disjoint subsets X, Y and Z, we have $< X|Z|Y >_G \Rightarrow I(X, Z, Y)_M$. A graph is a **minimal I-map** if all edges in it are necessary for it to remain an I-map.

3.2 Bayesian Networks

Definition 2 *A **Bayesian network (BN)** is a triplet (V, G, P), where V is a set of variables, G is a connected DAG, and there is a one-to-one correspondence between nodes in G and variables in V. P is a set of probability distributions: $P = \{P(v|\pi(v)) \mid v \in V\}$, where $\pi(v)$ denotes the set of parents of v in G. G is a minimal I-map of $P(V)$.*

3.3 Dynamic Bayesian Networks

Definition 3 *A **dynamic Bayesian network (DBN)** is a quadruplet*

$$G = (\bigcup_{t=0} V_t, \bigcup_{t=0} E_t, \bigcup_{t=0} E_t^{\rightarrow}, \bigcup_{t=0} P_t). \quad (1)$$

Each V_t is a set of nodes labeled by variables, which represents the dynamic domain at time instant t ($0 \leq t < k$). Collectively, $V = \bigcup_{t=0}^{k} V_t$ represents the dynamic domain over k instants. Each E_t is a set of arcs among nodes in V_t, which represents *dependencies among domain variables at time t. Each $E_t^{\rightarrow}$ is a set of temporal arcs each of which is directed from a node in V_{t-1} to a node in V_t ($0 < t < k$). The subset of V_t ($0 \leq t < k$) $FI_t = \{x | x \in V_t \ \& \ \exists y < x, y > \in E_{t+1}^{\rightarrow}\}$ is called the* ***forward interface*** *of V_t where $< x, y >$ is a temporal arc directed from x to y. The subsets of V_t ($0 < t < k$) $BI_t = I_t \cup \{z | z \in V_t \ \& \ \exists y (y \in I_t \ \& \ z \in \pi(y))\}$ is called the* ***backward interface*** *of V_t, where $I_t = \{y | y \in V_t \ \& \ \exists x (< x, y > \in E_t^{\rightarrow})\}$. Each $D_t = (V_t \cup FI_{t-1}, E_t \cup E_t^{\rightarrow})$ or $(V_t \cup BI_{t+1}, E_t \cup E_{t+1}^{\rightarrow})$ is a DAG and each P_t is a set of probability distributions*

$$P_t = \begin{cases} P(V_0), & t=0 \\ P(V_t | FI_{t-1}) or P(BI_{t+1} | V_t), & t > 0. \end{cases} \tag{2}$$

The pair $S_t = (D_t, P_t)$ is called a ***slice*** *of the DBN.*

Figure 1 shows the structure of a DBN of n slices, where $V_1 = \{a_1, b_1, c_1, d_1\}$, $E_1 = \{(a_1, b_1), (b_1, c_1), (b_1, d_1), (d_1, c_1)\}$, $E_1^{\rightarrow} = \{(a_0, b_1), (d_0, c_1)\}$, $FI_1 = \{a_1, d_1\}$ and $BI_1 = \{a_1, b_1, c_1, d_1\}$. The slice of DBN at time $t = 1$ is $D_1 = \{V_1 \cup FI_0, E_1 \cup E_1^{\rightarrow}\}$ where $FI_0 = \{a_0, d_0\}$. Each slice of a DBN is a BN. At any time $t = j \leq n - 1$, the slices $S_0, S_1, ..., S_{j-1}$ represent the domain history and $S_{j+1}, ..., S_{n-1}$ predict the future. Evidences may be entered into $S_0, ..., S_j$.

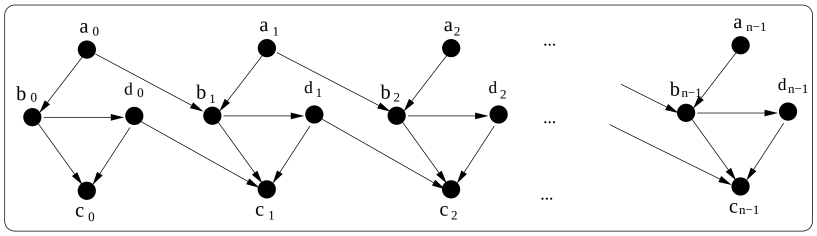

Fig. 1. A simple sample dynamic Bayesian network.

3.4 Overview of MSBNs

In an MSBN, a set of $n > 1$ agents A_0, A_1, ..., A_{n-1} populates a total universe V of variables. Each A_i has knowledge over a subdomain $V_i \subset V$ encoded as a *Bayesian subnet* (V_i, G_i, P_i). The collection $\{G_0, G_1, ..., G_{n-1}\}$ of local DAGs encodes agents' knowledge of domain dependencies. Local DAGs of an MSBN should overlap and be organized into a *hypertree*.

Definition 4 *Let $G = (V, E)$ be a connected graph sectioned into subgraphs $\{G_i = (V_i, E_i)\}$. Let these subgraphs be organized into a tree Ψ where each node, called a* ***hypernode****, is labeled by G_i and each link between G_i and G_j, called a* ***hyperlink****, is labeled by the interface $V_i \cap V_j$ such that for each pair of nodes G_l and G_m, $V_l \cap V_m$ is contained in each subgraph on the path between G_l and G_m. The tree Ψ is called a* ***hypertree*** *over G.*

Each hyperlink serves as the information channel between agents connected and is referred to as an *agent interface*. We say all variables in agent interfaces are *public* or *shared* among agents involved, and all others are *private*. To allow efficient and exact inference, each hyperlink should render the subdomains connected conditionally independent. It has been shown that this implies the following structural condition [6].

Definition 5 *Let G be directed graph such that a hypertree over G exists. A node x contained in more than one subgraph with its parents $\pi(x)$ in G is a* ***d-sepnode*** *if there exists a subgraph that contains $\pi(x)$. An interface I is a* ***d-sepset*** *if every $x \in I$ is a d-sepnode.*

Theorem 1 *Let Ψ be a hypertree over a directed graph $G = (V, E)$. For each hyperlink I which splits Ψ into two subtrees over $U \subset V$ and $W \subset V$ respectively, $U \setminus I$ and $W \setminus I$ are d-separated by I if and only if each hyperlink in Ψ is a d-sepset.*

The overall structure of an MSBN is a hypertree MSDAG.

Definition 6 *A* ***hypertree MSDAG*** *$G = \bigcup_i G_i$, where each $G_i = (V_i, E_i)$ is a DAG, is a connected DAG such that there exist a hypertree over G and each hyperlink is a d-sepset.*

An MSBN is composed of a hypertree MSDAG and the corresponding numerical probability distributions.

Definition 7 *An* ***MSBN*** *M is a triplet (V, G, P). $V = \bigcup_i V_i$ is the* ***total universe*** *where each V_i is a set of variables, called a* ***subdomain***. *$G = \bigcup_i G_i$ is a hypertree MSDAG where nodes of each subgraph G_i are labeled by elements of V_i. Let x be a variable and $\pi(x)$ be all parents of x in G. For each x, exactly one of its occurrences (in a G_i containing $\{x\} \cup \pi(x)$) is assigned $P(x|\pi(x))$, and each occurrence in other subgraphs is assigned a unit constant potential. $P = \prod_i P_i$ is the JPD where each P_i is the product of the potentials associated with nodes in G_i. Each triplet $S_i = (V_i, G_i, P_i)$ is called a* ***subnet*** *of M. Two subnets S_i and S_j are said to be* ***adjacent*** *if G_i and G_j are adjacent in the hypertree.*

4 Issues and Solution

4.1 Dynamic MSBNs

We first look at how MSBNs can be extended and applied to dynamic multiagent probabilistic inference. We propose to use an MSBN to model one time instant of a dynamic multiagent domain. The MSBNs over all time instants are called a *dynamic MSBN* (dMSBN).

A dMSBN is defined as in Definition 8.

Definition 8 *A* ***dynamic MSBN*** *is a quadruplet*

$$M = (\bigcup_{t=0} V_t, \bigcup_{t=0} E_t, \bigcup_{t=0} E_t^{\rightarrow}, \bigcup_{t=0} P_t).$$

Each $V_t = \bigcup_i V_{t,i}$ ($0 \leq i < n$) is the total universe at time t, where $V_{t,i}$ is a set of variables, called subdomain i at time t. Each $E_t = \bigcup_i E_{t,i}$ is a set of arcs among nodes in V_t, where $E_{t,i}$ is a set of arcs among nodes in $V_{t,i}$. Each $E_t^{\rightarrow} = \bigcup_i E_{t,i}^{\rightarrow}$ is a set of temporal arcs directed from nodes in V_{t-1} to nodes in V_t, where $E_{t,i}^{\rightarrow}$ is the set of temporal arcs directed from nodes in $V_{t-1,i}$ to nodes in $V_{t,i}$. The subset of $V_{t,i}$

$$FI_{t,i} = \{x \in V_{t,i} \mid (\exists y)(< x, y > \in E_{t+1,i}^{\rightarrow})\}$$

is called the ***forward interface*** *of $V_{t,i}$, whereas $FI_t = \bigcup_i FI_{t,i}$ is called the forward interface of V_t. The subset of $V_{t,i}$*

$$BI_{t,i} = \{y \in V_{t,i} \mid (\exists x)(< x, y > \in E_{t,i}^{\rightarrow})\} \cup \{z \in V_{t,i} \mid (\exists y)(z \in \pi(y) \& \\ (\exists x)(< x, y > \in E_{t,i}^{\rightarrow}))\}$$

is called the ***backward interface*** *of $V_{t,i}$, whereas $BI_t = \bigcup_i BI_{t,i}$ is called the backward interface of V_t. Each $G_t = (V_t \cup FI_{t-1}, E_t \cup E_t^{\rightarrow})$ is a MSDAG and each P_t is a probability distribution*

$$P_t = \begin{cases} P(V_0), & \text{t=0} \\ P(V_t|FI_{t-1}), & t > 0. \end{cases} \tag{3}$$

The triplet M_t=($V_t \cup FI_{t-1}, E_t \cup E_t^{\rightarrow}, P_t$) is an MSBN, called a ***slice*** *of the dynamic MSBN at time t.*

Temporal dependencies in a dMSBN only happen within the same subdomains. Hence, there exists a DBN corresponding to each subdomain (agent), where the uncertain knowledge at each time instant is represented by a BN.

Let D_0 be the MSDAG over (V_0, E_0). A (stationary) dynamic MSBN can be represented by a pair ($S_0, S_{\rightarrow}$), where S_0 is an MSBN (V_0, D_0, P_0) and $S_{\rightarrow}$ represents how the dynamic MSBN evolves over time. S_0 and $S_{\rightarrow}$ together define

$$P(V_t \mid V_{t-1}) = P(V_t \mid FI_{t-1}) = \prod_{v \in V_t} P(v|\pi(v)), 0 < t,$$

where $P(v \mid \pi(v))$ is defined by $P(V_0)$ for those $v \in V_t$ with $\pi(v) \subseteq V_t$, and by $S_{\rightarrow}$ for those with $\pi(v) \subseteq FI_{t-1}$.

For a forward or a backward interface in a dMSBN, we have Lemma 1.

Lemma 1 *In a dMSBN, for a forward interface FI_t, we have $< V_{0:t}|FI_t|V_{t+1:T} >$; for a backward interface BI_t, we have $< V_{0:t-1}|BI_t|V_{t:T} >$.*

That is, a forward or backward interface in a dMSBN separates the past from the future.

Proof: By definition of FI_t, any path ρ between a node in the future f and a node in the past p should be via a node $n \in FI_t$. Since the node $f' \in V_{t+1}$ that is adjacent to n on ρ should be a child of n, no matter how the node $p' \in V_t$ that is adjacent to n on ρ is connected with n, ρ should be closed by n. Hence, FI_t d-separates $V_{0:t}$ and $V_{t+1:T}$.

By definition of BI_t, any path ρ between a node in the future f and a node in the past p should be via a node $n \in BI_t$. Node n should be the child of a node $p' \in V_{t-1}$ that is adjacent to n on ρ. If the node $f' \in V_t$ that is adjacent to n on ρ is a child of n, ρ is closed by n; otherwise, $f' \in BI_t$. Then, no matter how the node $f'' \in V_{t:T}$ that is adjacent to f' on ρ is connected with f', f' should block ρ. Hence, BI_t d-separates $V_{0,t-1}$ and $V_{t:T}$. $\square$

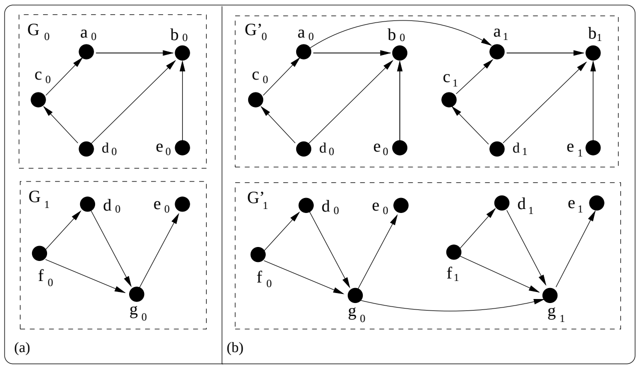

Fig. 2. A dynamic MSBN extended from an MSBN: (a) An MSBN over two subdomains G_0 and G_1; (b) The first two slices of the dynamic MSBN.

Figure 2 shows a two agent MSBN extended over a distributed dynamic domain. The structure of a slice of the dynamic MSBN is shown as in (a), and the first two consecutive slices of the dynamic MSBN are shown as in (b), where each dotted box represents a subdomain. A DBN is formed in each subdomain. The temporal dependencies are signified by the arcs (a_0, a_1) and (g_0, g_1) respectively as shown in (b). Either the forward interface $FI_0 = \{a_0, g_0\}$ or the backward interface $BI_1 = \{a_1, c_1, g_1, d_1, f_1\}$ separates the two parts it connects.

Therefore, we can construct a model to represent the distributed uncertain knowledge in a dynamic multiagent system. Next, we discuss the difficulties we face when using dMSBNs to perform inference.

4.2 Issues

4.2.1 Decomposition Issue

The decomposition issue exists in DBNs. However, it becomes a fatal problem for probabilistic reasoning using dynamic MSBNs.

In DBNs, between any two consecutive slices, there exists an interface — a forward interface or a backward interface — that d-separates the two slices. For messages to be properly passed forward via such interfaces, the elimination should be done by eliminating nodes in the previous slice (except interface nodes) first. A junction tree (JT) obtained based on such triangulation would make interfaces complete (when forward or backward interfaces are optimal) [26]. This makes the size of the slice interface the lower bound of computational complexity of reasoning using DBNs.

This problem becomes fatal for dynamic MSBNs since it requires that all messages passed from the preceding slice to the current slice be in the form of a single JPD. This not only makes the reasoning using dMSBNs expensive, but also results in the difficulty of the distribution of multiagent inference.

4.2.2 Distribution Issue

Ideally, we would like each agent to be able to maintain its own belief on its own subdomain and all agents to be able to benefit from each others' knowledge up to the relevant history. However, agents have difficulty to propagate their beliefs from one time instant to next time instant individually. The message passing separately in each subdomain would constitute loopy belief propagation. The slice interfaces at each subdomain and agent interfaces at each time instant won't d-separate the corresponding instants or subdomains.

For example, in the MSBN as shown in Figure 2 (a), the agent interface $\{d_0, e_0\}$ separates the two subdomains G_0 and G_1 it connects. Once the MSBN evolves, the corresponding interface at each new instant won't separate the corresponding subdomains any more. For example, in Figure 2 (b), the interface $\{d_1, e_1\}$ at instant 1 does not separate the two subdomains it connects because of the paths $< a_0, a_1 >$ and $< g_0, g_1 >$. For similar reasons, slice interfaces in each subdomain do not separate the two consecutive instants of the subdomain.

4.3 Local Inference

The two issues discussed above strongly imply that exact multiagent probabilistic reasoning over unbounded time periods could not be achieved by maintaining agents' belief over a finite time.

In this paper, we propose to model a dynamic multiagent domain over a *period* of time into an MSBN, and then reason about the state of the domain period by period exactly. For each new period, the initial prior belief of the domain is assumed. For example, the dynamic MSBN over time instants 0 and 1 as shown in Figure 2 (b) is actually an MSBN over a period of two time instants. The corresponding subdomains are G'_0 and G'_1 respectively. The interface between the two subdomains is $\{d_0, e_0, d_1, e_1\}$, which is the union of the corresponding agent interfaces over all instants of the period. The period could be much longer. The two consecutive periods could overlap on some instants. Using the MSBN, the state of domain could be reasoned about period by period exactly.

Within each period, a message does not have to be passed instant by instant. There are no conflicts between the distribution of multiagent systems and the centralization of temporal message passing. The extended agent interfaces separate the two subdomains they connect. The remote historical knowledge is ignored. However, the influence of the probabilistic knowledge would become weakened over distance, time and quantity. More recent history contains more relevant information about the current state of a domain. The history of a reasonable length could contain sufficient relevant information for reasoning about the state of the domain.

In particular, by a denser observation of the modeled history, the influence of the ignored history would be reduced. We propose to observe all relevant observable variables in the modeled period to reduce the influence. A notion called the graphical observable Markov boundary (GOMB) is proposed to capture all relevant and observable variables regarding the state of a set of concerned variables.

5 Graphical Observable Markov Boundaries

In this section, we define and discuss how to compute the GOMB of a set of variables in an MSBN.

5.1 Markov Boundaries

Due to limitation of domains, not all variables are observable. Due to limitation of bandwidth, it may be very costly to observe all observable variables. It would be ideal if we could find and only observe the relevant observable variables. The concept of Markov boundary gives us a hint.

Definition 9 [27] *Let M be a dependency model over a set V of variables. Let v be a variable such that $v \in V$. A* ***Markov blanket*** *$L(v)$ of v is any subset $S \subset V$ of variables for which*

$$I(\{v\}, S, V \setminus S \setminus \{v\})_M \text{ and } v \notin S. \tag{4}$$

A Markov blanket is called a ***Markov boundary*** *$B(v)$ of v if it is a minimal Markov blanket of v.*

That is, a Markov boundary of a variable provides us with a set of variables that is relevant to the state of the variable. [1]

Nevertheless, finding a Markov boundary of a variable in a probability distribution may not be tractable, since the verification of conditional independencies in

[1] In a not strictly positive probability distribution, the Markov boundary of a variable may not be unique [28].

probabilistic models is generally infeasible. Also, in probabilistic graphical models, the graphical structures may not capture all conditional independencies in the corresponding probabilistic models. For a node in a graphical model, the union of its parents, its children and the parents of its children is generally not a Markov boundary of the node because a node may have different neighbors in different minimal I-map DAGs [27]. In particular, the concepts of Markov blanket and Markov boundary are defined only for a single variable *without* the observabilities of their members considered.

On the other hand, when we do inference with graphical models, we generally only take advantage of the independencies expressed by the graphical structures. In particular, conditional independencies in Bayesian network structures can be identified in polynomial time [27]. We would revise the definition of Markov boundary based on d-separation in graphical structures. The revised Markov boundary of a variable can be efficiently and uniquely obtained, which we call the *graphical Markov boundary* (GMB) of the corresponding variable. We then further extend it to a set of variables, and eventually introduce the *graphical observable Markov boundary* (GOMB) of a set of variables, which only include observable variables.

5.2 Graphical Markov Boundaries

5.2.1 Graphical Markov Boundaries of a Single Node

Definition 10 emphasizes that we are interested in the Markov blanket and the Markov boundary defined based on d-separations on G, instead of independencies in P.

Definition 10 *Let N=(V, G, P) be a BN where DAG G is a minimal I-map of P. Let v be a variable such that $v \in V$. A* ***graphical Markov blanket*** *$L(v)$ of v is any subset $S \subset V$ of variables for which*

$$< \{v\}|S|V \setminus S \setminus \{v\} >_G \text{ and } v \notin S. \tag{5}$$

A graphical Markov blanket is called the ***graphical Markov boundary*** *$B(v)$ of v if it is a minimal graphical Markov blanket of v.*

For the graphical Markov boundary (GMB), we have Proposition 1.

Proposition 1 *For a node v in a BN N=(V, G, P), the union A of v's parents, v's children and the parents of v's children forms a graphical Markov blanket $L(v)$ of v in N. The graphical Markov blanket is a graphical Markov boundary $B(v)$ of v. The graphical Markov boundary is unique.*

Proof: Straightforward. $\square$

5.2.2 Graphical Markov Boundaries of a Set of Nodes

Since most times we are interested in the state of a set of variables, we further extend the two concepts to a set of variables.

Definition 11 *Let N=(V, G, P) be a BN. Let R be a set of variables such that $R \subset V$ and $R \neq \emptyset$. A **graphical Markov blanket** $L(R)$ of R is any subset $S \subset V$ of variables for which*

$$< R|S|V \setminus R \setminus S >_G \text{ and } R \cap S = \emptyset. \tag{6}$$

*A graphical Markov blanket $L(R)$ is called the **graphical Markov boundary** $B(R)$ of R if it is a minimal graphical Markov blanket of R.*

To facilitate the description of their members, we first define the *parents* and *children* of a set of variables.

Definition 12 *Let N=(V, G, P) be a BN. Let R be a set of variables such that $R \subset V$ and $R \neq \emptyset$. We say any node in $V \setminus R$ that is a parent of a node in R is a **parent** of R, and any node in $V \setminus R$ that is a child of a node in R is a **child** of R.*

We have Proposition 2 regarding the members of GMB of a set of variables.

Proposition 2 *Let N=(V, G, P) be a BN. Let R be a set of variables such that $R \subset V$ and $R \neq \emptyset$. The union A of R's parents, R's children and the parents of R's children forms a graphical Markov blanket $B(R)$ of R in the BN. The graphical Markov blanket $L(R)$ of R is the graphical Markov boundary $B(R)$ of R. The graphical Markov boundary is unique.*

Proof: Straightforward as the proof for Proposition 1. □

Graphical Markov boundaries can only be used for relevant observation in fully observable problem domains. In partially observable problem domains, members of a GMB may not be observable.

5.3 Graphical Observable Markov Boundaries

In this subsection, we introduce the graphical observable Markov boundary (GOMB) of a set S of variables to capture all observable relevant variables regarding the state of S.

5.3.1 Definition

In the following definition, we use $\Omega_{obs}(X)$ to denote all observable variables in a set X.

Definition 13 *Let N = (V, G, P) be a BN where G is a minimal I-map of P. Let R*

be a set of unobservable variables such that $R \subset V$. The GOMB $B(R)$ of R is a minimal subset $S(\subset V)$ of observable variables such that

$$< R|S|\Omega_{obs}((V \setminus S) \setminus R) >_G . \tag{7}$$

In the following discussion, we may also call the graphical Markov boundaries defined by Definitions 10 and 11 the *immediate* graphical Markov boundary. Regarding GOMB, we have Proposition 3.

Proposition 3 *For a set R of unobservable nodes in a BN N=(V, G, P), its GOMB B always exists and is unique. Given B, R is independent of all observable nodes in $V \setminus B$, but may not be independent of all other nodes in $V \setminus B \setminus R$.*

Proof: The GOMB B of R always exists because $I(R, S, \emptyset)$ guarantees that the set $S = \Omega_{obs}(V)$ satisfies Equation (7).

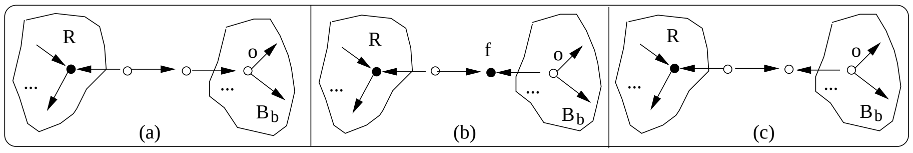

Fig. 3. A path between R and o is not closed by $B_b \setminus \{o\}$, where the white nodes are observable and the black nodes are unobservable: (a) No observable head-to-tail or tail-tail nodes on the path are in B_b. (b) There do not exist any unobservable head-to-head nodes on the path. (c) Any observable head-to-head nodes should be in B_b.

We prove the uniqueness by contradiction. Suppose there exist at least two GOMBs B_a and B_b for R such that $B_a \neq B_b$, and $|B_a| \leq |B_b|$. Hence, there exists at least one observable node $o \in B_b$ such that $o \notin B_a$. That is, R should be d-separated from o by B_a but not by $B_b \setminus \{o\}$. Therefore, there should be a path between a node in R and o not closed by $B_b \setminus \{o\}$. As shown in Figure 3, since the path is not closed by $B_b \setminus \{o\}$, no head-to-tail or tail-to-tail nodes on the path are in $B_b \setminus \{o\}$ or B_b. There also should not exist any head-to-head node f on the path that is not in $B_b \setminus \{o\}$ since otherwise the path is closed by f. If there is any head-to-head node on the path that is in $B_b \setminus \{o\}$, o won't be d-separated from R by B_b since we have known there does not exist any head-to-tail or tail-to-tail node on the path that is in $B_b \setminus \{o\}$. Hence, there should not exist any head-to-head node on the path. Since B_a d-separates o from R, there should exist an observable node g on the path that is in B_a, but not in B_b. Nevertheless, the observable node g won't be d-separated from R by B_b. This is in contradiction with the assumption that $< R|B_b|\Omega_{obs}(V \setminus B_b \setminus R) >_G$ holds.

Since R contains no observable nodes, from Equation 7 and the assumption that G is a minimal I-map of P, we have $< R|B|\Omega_{obs}(V \setminus B) >_G \Rightarrow I(R, B, \Omega_{obs}(V \setminus B))_P$. However, $I(R, B, V \setminus B \setminus R)_P$ may not hold. For example, in the BN as shown in Figure 4 (a), given the GOMB of node a $B(a) = \{d\}$, node a is independent of node b, but not independent of node c. □

5.3.2 Computation Illustration

By computing the GOMB $B(a)$ of a in Figure 4 (b), we illustrate how to compute the GOMB of a node in a BN. We then present a set of general algorithms for this computation.

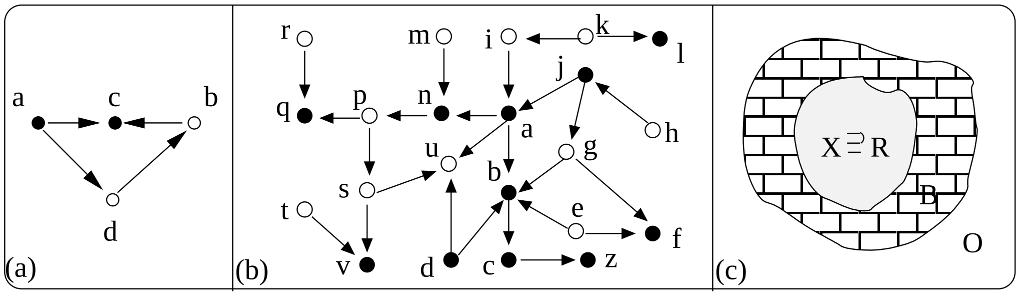

Fig. 4. (a) Given the GOMB $B(a) = \{d\}$ of node a, a may not be independent of all other nodes; (b) The computation of the GOMB of node a in a BN. The white nodes are observable and the black nodes are unobservable. (c) The division of a domain V by the graphical observable Markov boundary B of a set R of nodes.

In the BN as shown in Figure 4 (b), we first check the immediate graphical Markov boundary of a. Its observable child u and its observable parent i would be members of $B(a)$. Hence, $B(a) = \{i, u\}$. Since paths to node a via unobservable parent j or unobservable children b and n are still open, we put them in a set T (initially $T = \emptyset$) for further processing later. Hence, $T = \{j, b, n\}$. The processing of the parents of a child node α of node a depends on whether α or any of its descendants is observable.

(1) If α is observable (e.g. u), we process the other of its parents (s, d) immediately. Since node a is a processed parent of u, we need mark a to prevent it form being processed repeatedly. Actually this is the case for all nodes put in $B(a)$ or T. We mark and put observable s in $B(a)$ and unobservable d in T. Hence, $B(a) = \{i, u, s\}$ and $T = \{j, b, n, d\}$. We process parents of observable child u of node a immediately because u would be put in $B(a)$ (instead of T), which would not be further processed.

(2) If α is unobservable but has observable descendants (e.g. n), it should be put in T. Both its parents and children should be further examined.

(3) If neither α nor any of its descendants are observable (e.g. b), its parents and descendants need not be further processed. For example, the other parents (d, e, g) of b need not be examined. This is because the path from node d, e or g to a via b is closed by the absence of b and any of its descendants from $B(a)$ (they are unobservable). It should be noted that, although g is not examined because of b, it

would be examined and put in $B(a)$ as a child of j. This is because otherwise one other path to node a via g would be open.

By similarly further processing nodes in T until T becomes $\emptyset$, we should have the final graphical observable Markov boundary $B(a) = \{i, u, s, g, h, m, p\}$.

5.3.3 Division of Domain

We say the graphical observable Markov boundary B of a set R of nodes separates the whole domain V into 3 parts as shown in Figure 4 (c): the nodes *in* B (the area in brick pattern), the set X ($\supseteq R$, the grey shaded area) of nodes *inside* B, and the set O ($O = (V \setminus B) \setminus X$) of nodes *outside* B. The set X of nodes inside B contains all nodes in R, and a set K of unobservable nodes which are not d-separated from R by B. In the example above, $B(a) = \{i, u, s, g, h, m, p\}$, $X(a)$=$\{j, b, n, d, a, c, z\}$, and $O(a)$=$\{r, q, t, v, k, e, f\}$. Given B, R is independent of O, observable or not, but X may not necessarily be independent of O. For example, in the BN as shown in Figure 4 (b), given $B(a)$, node a is independent of all nodes in $O(a)$. However, $b \in X(a)$ is not independent of $O(a)$ because at least node b has a direct path with node e.

5.3.4 Algorithms for Computing the Observable Descendants

In the computation of GOMBs, we need to know if an unobservable node has any observable descendants. The information can be obtained by a recursive process which recursively checks descendants of an unobservable node until an observable descendant is discovered or all descendants are checked. This process is presented as Algorithms 1 and 2.

In the two algorithms, we associate each node v in a BN N with a variable O_v. If v or any of its descendants is observable in N, $O_v = 1$; otherwise $O_v = 0$. Initially we set $O_v = -1$. We say the observability of v or its descendants is *unknown* if $O_v = -1$. If a node v is unobservable, we need check the observabilities of its descendants to determine the value of O_v.

Algorithm 1 initializes $O_v = 1$ for each observable node v and $O_v = -1$ for each unobservable node v. Then Algorithm 2 is called on each node v where $O_v = -1$. Algorithm 2 is a recursive algorithm which determines if an unobservable node v has any observable descendants. It does so by searching all possible descendant branches. It backtracks from a node y with *determined* O_y or a leaf node. When it returns, any node x visited should have a determined O_x. In this algorithm, "$\vee$" on line 4 is a boolean "or" operator.

Note this algorithm won't return even when $O_v = 1$ is determined from one of its descendant branch. Since we want O_v of each node v in the BN, *checkOD*, once called, would return only after all descendant branches have been properly searched. Hence, Algorithm 2 will be called on a node in the BN at most once. The complexity of the computation is $O(|V|)$ in the number $|V|$ of nodes in the BN.

Algorithm 1 (computeOD)

```
Input: a BN $N = (V, G, P)$.
Output: the BN $N$ where $O_v$ of each node $v$ in $N$ is known.
begin
1    for each node $v$ in $N$, do
2        if $v$ is observable, set $O_v = 1$;
3        otherwise set $O_v = -1$;
4    for each node $v$ where $O_v = -1$, do
5        call checkOD($N, v$);
end
```

Algorithm 2 (checkOD)

```
Procedure checkOD($N, v$)
Input: a BN $N = (V, G, P)$ and a node $v$.
Output: Any node $x$ visited before its return has a known $O_x$.
begin
1    set $O_v = 0$;
2    for each child $y$ of $v$, do
3        if $O_y = -1$, set $O_y$ =checkOD($N, y$);
4        $O_v = O_v \vee O_y$;
5    return $O_v$;
end
```

5.3.5 Algorithms for Computing GOMBs in BNs

Once we know if each unobservable node has any observable descendants, we can compute the GOMB B of a set R of nodes in a BN. Since nodes in R are concerned, they will for sure not be in their GOMB B. Summarized from the computation illustration of GOMB above, the other nodes are processed as follows (T is initialized with R):

(1) For a parent node p of any nodes in T: if observable, put it into B; otherwise put it into T for further processing.
(2) For an observable child node c of any nodes in T: put it into B, and for each g of its parents, put it into B if observable or put it into T for further processing otherwise.
(3) For an unobservable child node c of any nodes in T: if it has observable descendants, put it into T for further processing.
(4) For an unobservable child node c of any nodes in T: if it has no observable descendants, nothing needs to be done from this node.

All nodes put into B or T should be marked so that they won't be further processed a second time. Since the number of nodes in a BN is limited, T will become $\emptyset$.

When T is $\emptyset$, the nodes in B form the GOMB of R. We present the idea into Algorithm 3, where "elif" represents "else if".

In Algorithm 3, lines 7, 8, 9 and 10 correspond to situation 1. Lines 11, 12 and 13 correspond to part of situation 2. Lines 14, 15, 16 and 17 correspond to the other part of situation 2. Lines 18 and 19 correspond to situation 3. Lines 5 and 6 ensure T will eventually become $\emptyset$. Lines 4, 8, 13, 15 and 19 mark visited nodes so that they won't be processed a second time. Since nothing needs to be done for situation 4, no lines of code correspond to it.

Algorithm 3 (ComputeGOMBinBN) *Let R be a set of unobservable nodes in a BN $N = (V, G, P)$. The GOMB $B(R)$ of R in N is returned.*

```
1   set $B = \emptyset, T = R$;
2   for each node $v \in V$, do
3     associate $v$ with a variable $b_v$;
4     if $v \in R$, set $b_v = 1$; else set $b_v = 0$;
5   while $T \neq \emptyset$, do
6     pick $v \in T$ and set $T = T \setminus \{v\}$;
7     for each parent $p$ of $v$ where $b_p = 0$, do
8       set $b_p = 1$;
9       if $p$ is observable, set $B = B \cup \{p\}$;
10      else, set $T = T \cup \{p\}$;
11    for each child $c$ of $v$ where $b_c = 0$, do
12      if $c$ is observable, do
13        set $b_c = 1$, and $B = B \cup \{c\}$;
14        for each parent $g$ of $c$ where $b_g = 0$, do
15          set $b_g = 1$;
16          if $g$ is observable, set $B = B \cup \{g\}$;
17          else, set $T = T \cup \{g\}$;
18      elif any descendant of $c$ is observable, do
19        set $b_c = 1$ and $T = T \cup \{c\}$;
```

Though each node is processed at most once, its neighbors may be checked for their membership of B or T. Therefore, the GOMB of a set of nodes in a BN can be returned in a time of $O(|V| + |E|)$, where $|V|$ is the number of nodes in the BN, and $|E|$ is the number of arcs in the BN. Regarding Algorithm 3, we have Proposition 4.

Proposition 4 *Let R be a set of unobservable nodes in a BN N. Let B be the set of nodes returned by Algorithm 3. Then B is the GOMB of R in N.*

Proof: Initially $T = R$. Whenever a node is removed from T, the members of its immediate graphical Markov boundary, not processed before, are processed depending on the situations they belong to.

Once T becomes $\emptyset$, all paths from any nodes in R to any observable nodes in $V \setminus R \setminus B$ should have been closed by nodes in B. Suppose there exists a path ρ between a node $r \in R$ and an observable node $x \in V \setminus R \setminus B$ not closed by B. That is, on the path there exist no observable head-to-tail or tail-to-tail nodes that are in B, there exist no unobservable head-to-head nodes, and if there exist any observable head-to-head nodes, they should be in B. We consider it for different cases: (1) if there exist no observable head-to-head nodes on that path, then all nodes on the path should be of head-to-tail or tail-to-tail. If any of them are observable, as shown in Figure 5 (a), at least one of them should be reached by r and put in B via lines 7, 8, 9 and 10; and/or lines 11, 12, 13, 14, 15, 16 and 17; and /or lines 18 and 19. This is in contradiction with the assumption that the path is not closed. If none of them is observable, x should be reached by r and put in B. This is in contradiction with the assumption that $x \in V \setminus R \setminus B$. (2) if there exist any observable head-to-head nodes that are in B, as shown in Figure 5 (b), from these head-to-head nodes, x should be reached and put in B via lines 7, 8, 9 and 10; and/or lines 11, 12, 13, 14, 15, 16 and 17; and /or lines 18 and 19 because no observable head-to-tail or tail-to-tail nodes on the path will be in B. This is in contradiction with the assumption that $x \in V \setminus R \setminus B$. Hence, all paths from any observable nodes in $V \setminus R$ to R should have been closed.

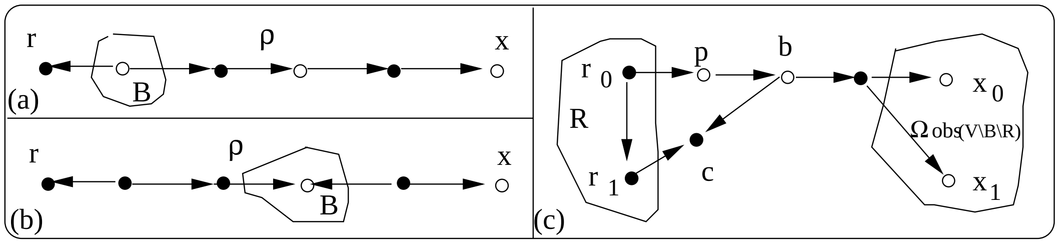

Fig. 5. (a) There should not exist any observable head-to-tail or tail-tail nodes between r and x; (b) If any head-to-head node is in B, x can also be reached by r; (c) Paths separated by b are also separated by some other ways. The white nodes are observable and the black nodes are unobservable.

Next we show B is minimal to satisfy $< R|B|\Omega_{obs}(V \setminus B \setminus R) >_G$. Suppose there exists a node $b \in B$ that could be removed, i.e. $< R|(B \setminus \{b\})|\Omega_{obs}(V \setminus B \setminus R) \cup \{b\} >_G$ holds. Therefore, $< R|(B \setminus \{b\})|b >_G$ holds, and all paths between R and b should be closed by some nodes in B other than b. As shown in Figure 5 (c), these paths can be closed either by an unobservable head-to-head node c, or by a head-to-tail or tail-to-tail node $p \in B \setminus \{b\}$ on the path. However, if that is the case, b cannot be reached by r from these paths via lines 7, 8, 9 and 10; and/or lines 11, 12, 13, 14, 15, 16 and 17; and /or lines 18 and 19 and hence b should have never been in B. $\square$

5.4 *Multiagent Cooperative Computation of GOMBs*

In this subsection, we provide a set of algorithms for distributed computation of GOMBs in MSBNs.

5.4.1 GOMB in MSBNs

The GOMB of a set of nodes in an MSBN could appear across all Bayesian subnets. For example, in the MSBN as shown in Figure 6, the members of GOMB of $f_0 \in G_1$ could appear in both G_0 and G_1. Its immediate GMB is $\{d_0, g_0\}$. If we assume d_0 is unobservable, $\{d_0, g_0\}$ is not a GOMB. We therefore need to further consider the immediate GMB of d_0. If c_0, b_0, a_0, and e_0 are all observable, the GOMB of f_0 should be $\{g_0, c_0, b_0, a_0, e_0\}$. If we further assume a_0, a_1, c_1 and d_1 are all unobservable, the computation would return back to G_1 and the GOMB of f_0 would further include f_1 and g_1. Hence, the computation of the GOMB of a set of nodes in an MSBN could occur in a subnet many times.

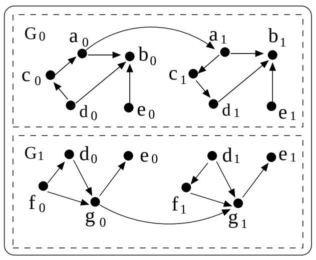

Fig. 6. A trivial MSBN over two subdomains G_0 and G_1.

In the following 3 Subsubsections, we propose a suite of algorithms to compute the GOMB $B(R)$ of a set R of nodes in an MSBN M of n subnets. Let $N_i(0 \leq i < n)$ be the n subnets over subdomains $V_i(0 \leq i < n)$, respectively. Let $A_i(0 \leq i < n)$ be the corresponding agents on $V_i(0 \leq i < n)$. For each adjacent agent A_k of A_0, we denote $V_k \cap V_0$ by I_k.

5.4.2 Cooperative Computation of the Observable Descendants

We first compute the observable descendants information distributedly. Algorithm 4 is started by the system coordinator to activate the computation. It first makes some initialization by calling Algorithm 5 (line 1), where each unobservable node is assigned a value “-1”, which indicates that it is still unknown if the corresponding node has any observable descendants or not. Then it calls Algorithm 6 (line 4) on each of such nodes to figure out the answer.

Algorithm 4 (multiComputeOD) *An MSBN M of n subnets is populated by multiple agents with one at each subnet. The system coordinator does the following:*

```
1 call multiInitialOD;
2 for each agent $A_i$ ($i = 0, 1, ..., n-1$), do
3     for each node $v$ in $V_i$ where $O_v = -1$, do
4         $O_v$=localOD($A_i, v$);
```

In an MSBN, we call a node a *global leaf node* if it does not have any children in any subnet DAGs. Algorithm 6 recursively checks all descendants (could be in adjacent subnets) of an unobservable node v to figure out if v has any observable descendants. For computational efficiency, the backtracking happens only when either observable nodes or global leaf nodes have been reached. A node needs to call *localOD* at most once to figure out if it has any observable descendants. Hence, the computational complexity is $O(mn)$, where n is the number of subnets in the corresponding MSBN, and m is the maximum number of nodes a subnet can have. For Algorithms 4, 5 and 6, we have Proposition 5.

Algorithm 5 (multiInitialOD) *Each agent A_i ($i = 0, 1, ..., n-1$) does the following:*

```
1  for each node $v$ in $V_i$, do
2    if $v$ is observable, set $O_v = 1$;
3    otherwise, set $O_v = -1$;
```

Algorithm 6 (localOD(A_{k0}, v)) *Let A_{k0} be an agent with a local subnet N_{k0}. A caller is either an adjacent agent A_s or the system coordinator. When A_{k0} is called by caller on v, it does the following:*

```
1   if $O_v = -1$, set $O_v = 0$; else return $O_v$;
2   for each child $c$ of $v$, do
3     if $O_c = -1$, do
4       if $c$ is a shared local leaf node in $N_{k0}$, do
5         for each adjacent $A_j$ containing $c$, except $A_s$, do
6           pass descendant info on $V_{k0} \cap I_j$ to $A_j$;
7           $O_c$ = localOD($A_j, c$);
8           $O_v = O_v \vee O_c$;
9       else $O_c$ = localOD($A_{k0}, c$);
10    $O_v = O_v \vee O_c$;
```

Proposition 5 *Algorithms 4, 5 and 6 determine O_a for each node a in an MSBN. All messages passed among agents are through public nodes.*

Proof: We have shown that Algorithms 1 and 2 can figure out such information for every node in a single agent BN. Algorithms 4, 5 and 6 modify Algorithms 1 and 2 by adding some multiagent cooperation mechanisms. Hence, it is sufficient if we can show such mechanisms work when computation is across different subdomains.

Algorithm 5 initializes every node in every subnet when called by Algorithm 4. Then Algorithm 6 is called by Algorithm 4 on each agent to figure out the observable descendant information for every node in the respective subdomain. Since an agent can only be activated by the system coordinator or a caller agent, the computation is performed agent by agent. That is, at one time, there is only one active agent.

In Algorithm 6, line 1 ensures that the required information will be returned properly if it has been available. Lines 4, 5, 6, 7 and 8 process cooperation operations with adjacent agents. For a shared local leaf node c, the computation would be extended to the corresponding adjacent agents. In the corresponding agents, O_c could have been available. If so, it is immediately returned by line 1. All the observable descendant information on shared nodes is passed to the adjacent agents (line 6) because the observable descendant information may have been available for some of these nodes and could be required by the corresponding adjacent agents. Line 8 absorbs the observable descendant information obtained from the corresponding adjacent agents by boolean "or" ($\vee$) operation. $\Box$

5.4.3 Cooperative Computation of GOMBs

Based on the observable descendant information obtained by Algorithms 4, 5 and 6, we can compute the GOMB $B(R)$ of a set R of nodes in an MSBN M distributedly.

In a distributed problem domain, a variable could be *logically* shared by different agents, but the entity represented by the variable can only *physically* locate in one subdomain. We call the subdomain where an entity physically exists the *host subdomain* of the entity and the corresponding agent its *host agent*. We assume a variable can only be observed by its *host agent*. Therefore, each agent A_i should keep the part $B_i(R)$ of $B(R)$ it can observe, called the *partial graphical observable Markov boundary* (PGOMB) of R in the corresponding subdomain, for observation.

We present a suite of algorithms to cooperatively compute GOMB in an MSBN. The system coordinator activates the computation by executing Algorithm 7, which first distributes R to the corresponding agents and then call each involved agent to do some initialization by running Algorithm 8 (lines 1, 2, and 3). After that, the cooperative computation of $B(R)$ is performed by running Algorithm 9 (line 4).

Algorithm 7 (ComputeMB) *An MSBN M of n subnets is populated by multiple agents with one at each subnet. Let R be a set of nodes in M. The system coordinator does the following for multiple agents to figure out the GOMB $B(R)$ of R, which is the union of $B_i(R)(0 \leq i < n)$, in M.*

```
1   for each agent $A_i$, do
2       send $R \cap V_i$ to $A_i$;
3       call $A_i$ to run InitializeMB;
4   call ExpandMB;
```

By Algorithm 8, each agent receives a set T_i of the corresponding concerned nodes from the system coordinator for the cooperative computation of the GOMB $B(R)$ (line 1) and makes some initialization (lines 2, 3 and 4). Lines 2 and 3 associate each node v with a variable b_v. If $b_v = 1$, the node v should have been processed. Since a DAG is generally multiply connected, such a marking is necessary to prevent infinite loops. Line 4 initializes the corresponding PGOMB in each subdomain.

Algorithm 8 (InitializeMB) *Let A_i be the agent over a subnet $N_i = (V_i, G_i, Pi)$. When called by the system coordinator, it does the following:*

```
1   receive $T_i = R \cap V_i$;
2   for each node $v$ in $V_i$, do
3       if $v \in R$, set $b_v$=1; else, set $b_v$=0;
4   set $B_i = \emptyset$, $Q_i = \emptyset$;
```

Algorithm 9 (ExpandMB) *Let Q_k ($k = 0, 1, ..., n-1$) be the set of unobservable nodes collected by agent $A_k (k = 0, 1, ..., n-1)$ for extended computation of GOMB. The system coordinator does the following:*

```
1   for each agent $A_i$, do
2       if $T_i = \emptyset$, continue;
3       call $A_i$ to run ComputePMB;
4   for each agent $A_i$, do
5       call $A_i$ to run collectNodes;
6   if all $Q_i$'s ($0 \leq i < n$) are $\emptyset$'s, return;
7   else, set $T_i' s = Q_i' s$, restart the algorithm;
```

Algorithm 9 calls Algorithm 10 to compute the PGOMB of R in each corresponding subnet (lines 1, 2 and 3). This is called one *pass* of the computation of the GOMB $B(R)$. Several passes may be needed to reach the final GOMB $B(R)$. A pass of computation of PGOMB by Algorithm 10 could be extended across different subnets. Agent A_i ($0 \leq i < n$) uses Q_i ($0 \leq i < n$) to receive the shared unobservable nodes from the adjacent agents when extension happens. [2] In Algorithm 9, Algorithm 11 is called by each agent to collect such nodes from the the corresponding agents (lines 4 and 5). Then all Q_i's are checked (lines 4 and 5). If all Q_i's are $\emptyset$'s, the computation is finished; otherwise, a new pass of computation has to be started in the corresponding agents.

Algorithm 10, which is very similar to Algorithm 3 except for some mechanisms for message passing among agents, performs one pass of the computation of the PGOMB $B(R)$ in a subnet. The set H_{k0} contains all unobservable nodes assigned initially and reached in computation (lines 1, 8, 16, and 19). The public nodes in H_{k0} are passed to the corresponding adjacent agents for extending the computation of $B(R)$ in other subnets (lines 20 and 21). The Algorithm 10 computes the PGOMB in an iterative way. A recursive version can recursively check the immediate GMB of each of T_{k0} members.

Algorithm 11 collects unobservable public nodes from the adjacent agents for the extended computation of the GOMB $B(R)$. Since this may not be the first pass of computation, all nodes evaluated (and hence *marked*) by A_i are removed from X (line 4). The set Q_{k0} are set to be $\emptyset$ in the beginning (line 1). If it remains $\emptyset$ after

[2] Q_i ($0 \leq i < n$) is initialized at line 4 of Algorithm 8.

the collection, no computation is necessary in subnet N_{k0} at this time.

Algorithm 10 (ComputePMB) *Let T_{k0} be a set of unobservable nodes in N_{k0}. Let H_{k0} be the set used to collect the corresponding unobservable nodes involved in the pass of computation. Denote the adjacent agent of A_{k0} by $A_{k1}, A_{k2}, ..., A_{km}$. When called by the system coordinator, the agent A_{k0} does the following to figure out the GOMB members in V_{k0} corresponding to T_{k0}:*

```
1   set $H_{k0} = T_{k0}$;
2   while $T_{k0} \neq \emptyset$, do
3     pick $v \in T_{k0}$ and set $T_{k0} = T_{k0} \setminus \{v\}$;
4     for each parent $p$ of $v$ where $b_p = 0$, do
5       set $b_p = 1$;
6       if $p$ is observable, set $B_{k0} = B_{k0} \cup \{p\}$;
7       else, do
8           set $T_{k0} = T_{k0} \cup \{p\}$, $H_{k0} = H_{k0} \cup \{p\}$;
9     for each child $c$ of $v$ where $b_c = 0$, do
10      if $c$ is observable, do
11        set $b_c = 1$, and $B_{k0} = B_{k0} \cup \{c\}$;
12        for each parent $g$ of $c$ where $b_g = 0$, do
13          set $b_g = 1$;
14          if $g$ is observable, set $B_{k0} = B_{k0} \cup \{g\}$;
15          else, do
16            set $T_{k0} = T_{k0} \cup \{g\}$, $H_{k0} = H_{k0} \cup \{g\}$;
17      elif any descendant of $c$ is observable, do
18        set $b_c = 1$;
19        set $T_{k0} = T_{k0} \cup \{c\}$, $H_{k0} = H_{k0} \cup \{c\}$;
20  for each adjacent agent $A_j (j = k1, k2, ..., km)$ of $A_{k0}$, do
21    pass $H_{k0} \cap I_j$ to $A_j$ over $I_j$;
```

Algorithm 11 (collectNodes) *Let A_{k0} be an agent over a subnet N_{k0}. Denote the adjacent agents of A_{k0} by $A_{k1}, A_{k2}, ..., A_{km}$. When called by the system coordinator, A_{k0} does the following:*

```
1   set $Q_{k0} = \emptyset$;
2   for each adjacent agent $A_j$ ($j = k1, k2, ..., km$) of $A_{k0}$, do
3     receive a set $X$ of nodes over $I_j$ from $A_j$;
4     remove any nodes marked by $A_{k0}$ from $X$;
5     set $Q_{k0} = Q_{k0} \cup X$;
```

Since the computation of $B(R)$ will finish when all nodes are visited, the computational complexity of the suite of algorithms is $O(n + m)$, where n is the number of nodes in the MSBN, and m is the number of arcs in the MSBN. For the suite of algorithms, we have Proposition 6.

Proposition 6 *Algorithms 7, 8, 9, 10 and 11 compute and return the GOMB $B(R)$*

of a set R of nodes in an MSBN. All messages passed among agents are through public nodes.

Proof: The set of algorithms is different from Algorithm 3 in that they need to process message passing among agents for extended computation of the GOMB $B(R)$. To reach $B(R)$, several passes of computation could occur in one subdomain. Since we have proved the Algorithm 3 for single agent case in Proposition 4, we here need to show that the message passing and multiple passes of computation are properly done.

Only public unobservable nodes collected in the computation in a subnet could be involved in the extended computation in the adjacent agents. Such nodes are passed to the corresponding agents over their interfaces (lines 20 and 21 of *ComputePMB*). If any of them are already evaluated in the corresponding agents, they will be removed from the extended computation (line 4 of *collectNodes*). Hence, the number of nodes in a subnet that need to be evaluated will become less and less until none. The computation in each subnet will be eventually finished (line 6 of *ExpandMB*). $\square$

5.4.4 Cooperative Distribution of GOMB

Until now, all nodes that belong to $B(R)$ should have been reached. However, they may not have been properly distributed to PGOMB $B_i(R)$ ($0 \leq i < n$).

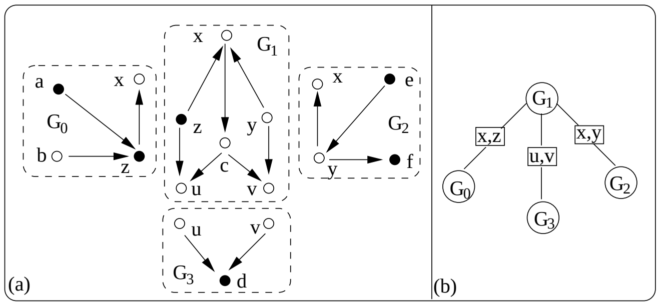

Fig. 7. The GOMB of a set of nodes in an MSBN may not be properly distributed. (a) The 4 subnets of an MSBN. (b) The hypertree MSDAG over the DAGs in (a). The white nodes are observable and the black nodes are unobservable.

Figure 7 shows a trivial MSBN of 4 subnets G_0, G_1, G_2 and G_3, where each dotted box represent one subnet. The white nodes are observable and the black nodes are unobservable. We assume the 4 subnets are controlled by 4 agents A_0, A_1, A_2 and A_3 respectively. Suppose A_0, A_1, A_2 and A_3 are the host agents of $\{a, b, z\}$, $\{c, v, x\}$, $\{e, f, y\}$ and $\{d, u\}$, respectively. Assume that we compute the GOMB of node c in the MSBN. Since c is a private node in G_1, in the first pass, only agent A_1 run *ComputePMB* on G_1. All other agents do not join the pass of computation. After the first pass of the computation, we get $B_1 = \{x, u, v, y\}$, and $H_1 = \{z, c\}$. Then the public node z in H_1 is passed to the corresponding adjacent agent A_0.

A_0 uses Q_0 to hold z. Since $Q_0 = \{z\}$ is not empty, *ExpandMB* is restarted. We get $B_0 = \{b, x\}$, and $H_0 = \{z, a\}$ in G_0 in the second pass of the computation. Then the public node z in H_0 is passed back to the corresponding adjacent agent A_1. However, Q_1 will become $\emptyset$ since A_1 has evaluated and marked node z. Hence, Q_0, Q_1, Q_2 and Q_3 are all empty at this time and *ComputeMB* finishes. Though all members of $B(c)$ have been identified in the MSBN, they may not have been distributed to the proper $B_i(c)'s$ properly. For example, y and u can only be observed by A_2 and A_3 respectively, but both $B_2(c)$ and $B_3(c)$ are empty. Hence, we need properly distribute the available GOMB members to the corresponding host agents.

We present a set of 3 algorithms to properly distribute GOMB $B(R)$ to PGOMB $B_i(R)'s$ ($0 \leq i < n$). Algorithm 12 is called to start the set of algorithms, where *CollectMB* and *DistributeMB* are called one after another by an arbitrary agent. Both *CollectMB* and *DistributeMB* are recursive algorithms, which recursively collect or distribute related GOMB members from or to the corresponding agents, through agents' interfaces.

Algorithm 12 (UnifyMarkovBoundary) *Let M be an MSBN of n subnets. The system coordinator selects an arbitrary agent A_r to run* CollectMB *in G_r. After it finishes, A_r runs* DistributeMB *in G_r. Finally, each agent A_i removes nodes it cannot observe from B_i ($0 \leq i < n$).*

Algorithm 13 (CollectMB) *Let A_{k0} be an agent over the subnet N_{k0} of an MSBN M. A caller can be the system coordinator or an agent A_s. Denote the additional adjacent agents of A_{k0} by $A_{k1}, A_{k2}, ..., A_{km}$. Agent A_{k0} does the following when called by a caller:*

```
for each agent $A_i (i = k1, k2, ..., km)$, do
   call $A_i$ to run CollectMB;
   receive a set $X = B_i \cap I_i$ of nodes over $I_i$;
   set $B_{k0} = B_{k0} \cup X$;

if caller is an agent $A_s$, do
   send $A_s$ the GOMB members $B_{k0} \cap I_s$ over $I_s$;
```

For the set of 3 algorithms for PGOMB distribution, we have Proposition 7.

Proposition 7 *Let $N_i = (V_i, G_i, P_i)(0 \leq i < 0)$ be the subnets of an MSBN M. Let R be a set of unobservable nodes in M. Let $B(R)$ be the union of all the GOMB members reached by Algorithms 7, 8, 9, 10 and 11. Algorithms 12, 13 and 14 properly distribute the members of the GOMB $B(R)$ to the corresponding PGOMB $B_i(R)(0 \leq i < n)$. All messages passed among agents are through public nodes.*

Proof: In a hypertree MSDAG, any nodes shared by two hypernodes B and C also

Algorithm 14 (DistributeMB) *Let A_{k0} be an agent over the subnet N_{k0} of an MSBN M. A caller can be the system coordinator or an agent A_s. Denote the additional adjacent agents of A_{k0} by $A_{k1}, A_{k2}, ..., A_{km}$. Agent A_{k0} does the following when called by a caller:*

```
if caller is an agent $A_s$, do
   receive a set $X$ of GOMB members over $I_s$ from $A_s$;
   add $X$ to $B_{k0}$;

for each agent $A_i (i = k1, k2, ..., km)$, do
   send $B_{k0} \cap I_i$ to $B_i$ of $A_i$ over $I_i$;
   call $A_i$ to run DistributeMB;
```

appear in every hypernode on the path between them. By recursive collection, the shared $B(R)$ members between a some agent A_x and agent A_r will be collected from A_x to agent A_r and any agents on the path between them. By recursive distribution, the shared $B(R)$ members between agent A_r and a some agent A_x will be distributed from A_r to A_x and any agents on the path between them. Hence, by recursive collection and distribution, a shared node $u \in B(R)$ should have reached every $B_i(R)$ where $u \in V_i$.

After all nodes for which A_i ($0 \leq i < n$) is not a host agent are removed from $B_i(R)$ ($0 \leq i < n$), we have a set of PGOMBs that can be observed properly by each agent. $\square$

By applying Algorithms 12, 13 and 14 to the above example, we have $B_0 = \{b\}$, $B_1 = \{v, x\}$, $B_2 = \{y\}$, and $B_3 = \{u\}$.

Since both *CollectMB* and *DistributeMB* are called once at each subnet, and at each call, only a finite number of intersection operations are made, the computation should finish in $O(mn)$ time, where n is the number of agents in the corresponding MSBN, and m is the maximum number of neighbors an agent can have in the hypertree.

6 Experiments

In this section, we show the effectiveness of the proposed dynamic multiagent probabilistic inference method on the simulated sequential digital circuits. We also give the relationship between the computational complexity and the length of the represented period.

6.1 The Sequential Digital Circuits

In sequential digital circuits, some devices may become faulty in the run time. We use the proposed dynamic multiagent inference method to detect such faulty devices.

The synchronous sequential digital circuit as shown in Figure 8 is composed of 5 components. It has a total of 62 devices including 20 *inverters*, 21 *and* gates, 13 *or* gates, 1 *xor* gate, 3 *D flip-flops*, and 4 *J-K flip-flops*. Each component can be associated with a computational agent responsible for monitoring and troubleshooting the component. The agents can acquire local observations from sensors and reason about the values of unobservable variables within the component. Components are interfaced with each other, and observations obtained by one agent could be valuable to another agent. When modeling these components, one agent should have some variables shared with some other agents.

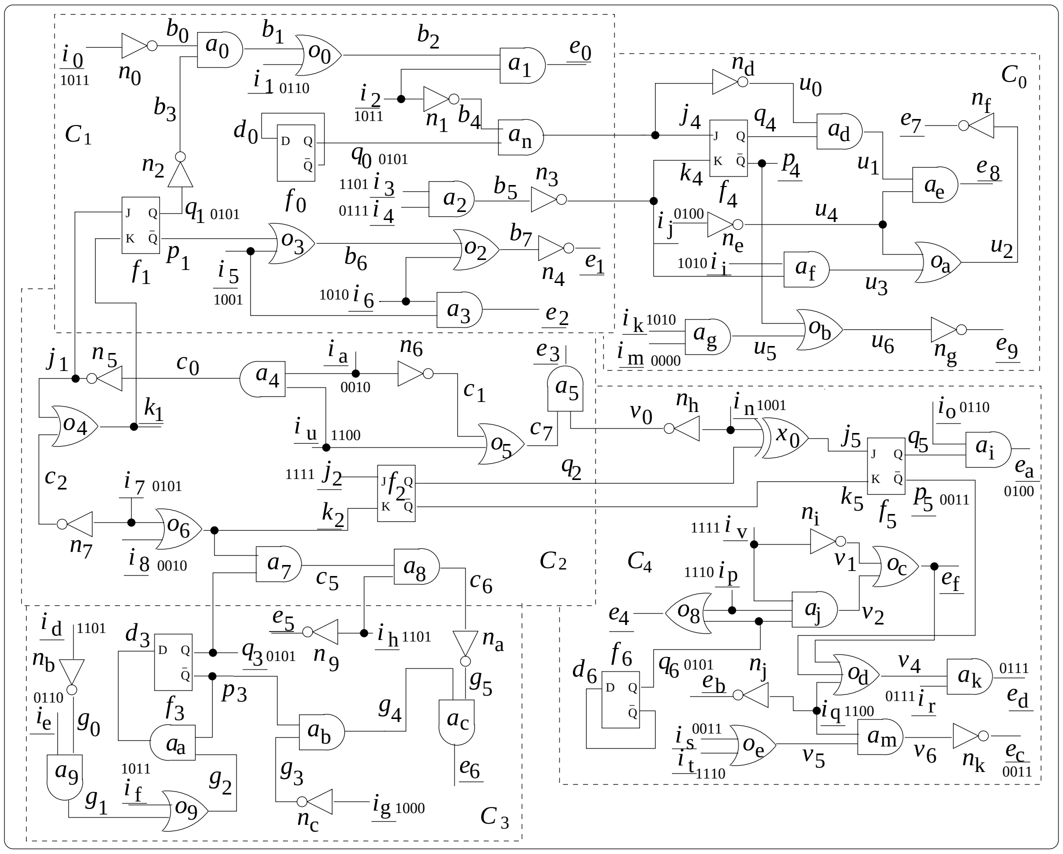

Fig. 8. A synchronous sequential digital circuit. Each dashed box represents one component.

6.2 The MSBN Model

We monitor and diagnose the simulated circuit with a five agent MSBN over a period of four time instants (clocks). Figures 9 and 10 shows the two Bayesian subnets corresponding to component 0 and 1, respectively, where each variable is labeled by its variable name followed by the variable's index (which readers may ignore). Each variable name is composed of a string indicating the corresponding device or signal and a digit indicating the respective time instant. For example, in Figure 9, $f4\_3$ denotes the state of J-K flip flop $f4$ in subdomain U_0 at relative time instant 3. The five Bayesian subnets of the MSBN are organized into a hypertree as shown in Figure 11.

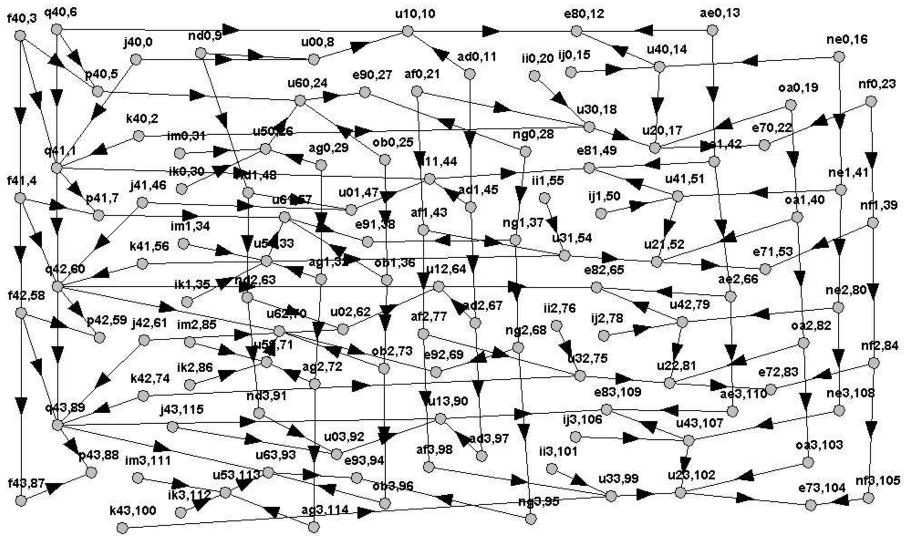
Fig. 9. The subnet G_0 for component U_0.

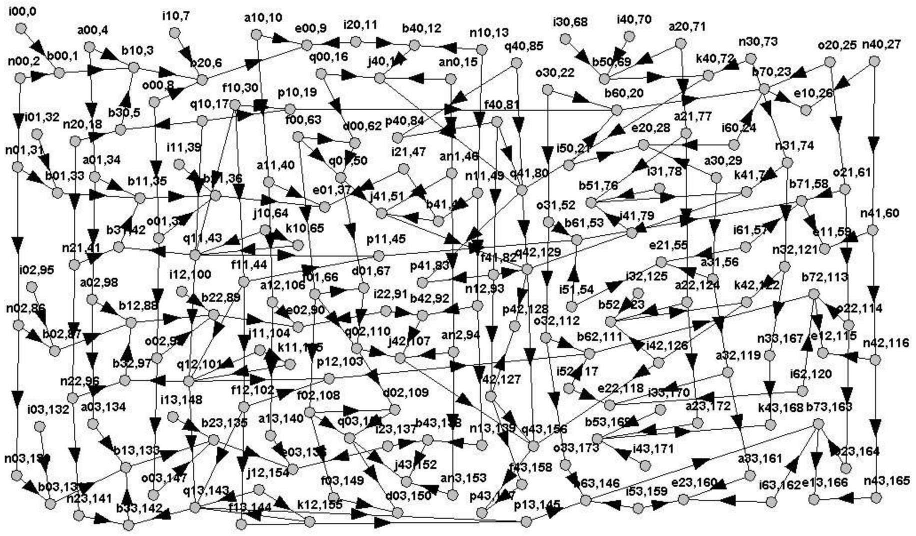
Fig. 10. The subnet G_1 for component U_1.

In addition to the dependency structures, we have the following prior beliefs on representational parameters. The state of a device (flip-flops or logic gates) at a time instant is represented by a boolean variable and is either *normal* or *abnormal*. A device could become faulty (abnormal) at a probability of 1%. If a device is normal at time $t = i$, it may become abnormal at $t = i + 1$ with a probability of 1%. If it is abnormal at time $t = i$, it will stay abnormal. A faulty device may produce correct output(s): a faulty *not* gate outputs correctly with a probability of

50%; a faulty *and* gate outputs correctly with a probability of 20%; a faulty *or* gate outputs correctly with a probability of 70%; a faulty *xor* gate outputs correctly with a probability of 30%; and either outputs ($Q, \bar{Q}$) of a faulty flip-flop could be correct with a probability of 30%.

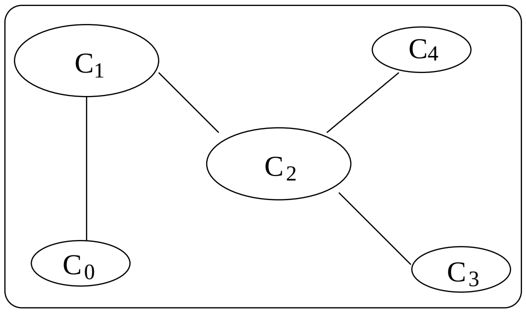

Fig. 11. The hypertree MSDAG of the five agent MSBN.

We assume that the state of a device is not observable. We also assume that the observation of an input or output has a cost. Therefore, observing all inputs and outputs is not an option. To make the situation more challenging, we assume that not all inputs or outputs of a device are observable.

6.3 Lower Bound on the Length of the Modeled Period

To detect the problems of a flip-flop in a sequential digital circuit, we need to model at least two instants so that we have the chance to observe the outputs of the flip-flop. When many flip-flops are chained together and signals between them are not observable, the length of the modeled period is lower bounded by one more than the number of flip-flops such connected. For example, as shown in Figure 12, a sequence of n J-K flip-flops are connected one after another, where each irregular box represents a combinational circuit receiving outputs from the preceding flip-flop and providing inputs to the posterior flip-flop except the first one only providing inputs to flip-flop jk_0 and the last one only receiving outputs from flip-flop jk_{n-1}. If we can only observe the variables in irregular boxes I and O_{n-1}, we may need to model the domain over a period of at least $n + 1$ instants to properly reason about the state of the domain. This is because the output O_{n-1} won't be affected by the possibly problematic outputs from flip-flop J-K jk_0 within n instants. That is, there exists a *delay* between the time when the problem happens and the time when the affected outputs are observed.

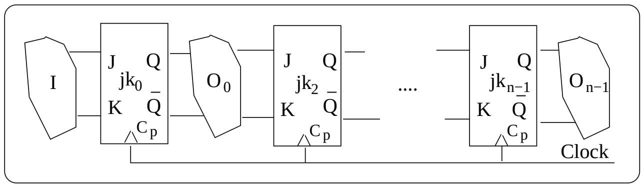

Fig. 12. A sequence of J-K flip-flops.

Therefore, the number of flip-flops that are chained together and the observability of variables between them could tell us the minimal period we need to model.

For the digital circuit as shown in Figure 8, a period of at least three instants needs to be modeled to reason about the state of the domain because the longest sequences of flip-flops where variables between them are unobservable are the sequences of two flip-flops. However, a modeled period of three instants may not give us very confident inference results for this problem domain. We show this by randomly picking and observing a period of three consecutive instants of the problem domain, where flip-flop f_2 is assumed to be faulty. Figure 8 shows all input signals and some of output signals over a period of four instants. The outputs provided here are those that could be affected by the outputs of faulty f_2 and cannot be properly predicted based on given inputs. The signal values are provided as strings of four digits. For example, "0101" beside i_7 indicates that i_7 takes values '0', '1', '0' and '1' chronologically in the period. We test on the first 3 instants.

We assume one input (k_2) and all outputs (q_2, k_5) of the faulty J-K flip-flop f_2 are unobservable. In particular, its outputs are direct or indirect inputs of another J-K flip-flop f_5, whose inputs are not observable. However, one of f_5's outputs p_5 is observable, and its other output q_5 is an input of an *and* gate a_i, whose output e_a is observable. We underline all observable variables in Figure 8.

For a model over a period of three instants, we have the graphical observable Markov boundary $B(R_0^3)$ of R_0^3=$\{f_{20}, f_{21}, f_{22}\}$:

$$B(R_0^3) = \{i_{70}, i_{71}, i_{72}, i_{80}, i_{81}, i_{82}, j_{20}, q_{30}, q_{31}, q_{32}, i_{g0}, i_{g1}, i_{g2}, e_{60}, e_{61}, e_{62}, i_{h0}, i_{h1}, i_{h2}, i_{d0}, i_{d1}, i_{e0}, i_{e1}, i_{f0}, i_{f1}, i_{n0}, i_{n1}, p_{50}, p_{51}, p_{52}, e_{a0}, e_{a1}, e_{a2}, i_{o0}, i_{o1}, i_{o2}\}.$$

For simplicity, we do not differentiate host agents here. The graphical observable Markov boundary contains 36 variables. It is interesting that not all observable direct inputs of J-K flip-flop f_2 are contained in $B(R_0^3)$. Only j_{20} at the first instant (0) are contained in $B(R_0^3)$. This is because the inputs to f_2 at instant 2 have not gone through f_2, and no any effects have been produced. Although the inputs to f_2 at instant 1 have gone through f_2, the effects of f_2's outputs won't be observed at e_a or p_5 at instant 2 before these outputs go through another J-K flip-flop f_5 (i.e. any information resulted from inputs to f_2 at instant 1 are currently contained somewhere between the two J-K flip-flops, where no variables are observable). Also, although i_{n0} and i_{n1} are contained in $B(R_0^3)$, i_{n2} and i_{n3} are not. This is because no effects of i_{n2} and i_{n3} would be observed at e_a or p_5 within the period.

By inference based on the observation of $B(R_0^3)$ and communication among agents, J-K flip-flop f_2 is believed to be faulty with a belief of 72.65%. All other devices in component C_2 are believed to be normal with beliefs over 96% except *or* gate o_6 77.86%. All devices in components C_0, C_1 and C_3 are believed to be normal with beliefs over 96.06%. All devices in component C_4 are believed to be normal with beliefs over 96.06% except flip-flop f_5 89.81%. Though the inference indicates that

f_2 could be faulty, we may not be very confident about the result since the posterior (72.65%) is not very positive.

6.4 More Than Minimal Instants

When we monitor the circuit as shown in Figure 8 over periods of four instants, we may get better results. Based on the similar assumptions as the example above, we have the graphical observable Markov boundary $B(R_0^4)$ of R_0^4=$\{f_{20}, f_{21}, f_{22}, f_{23}\}$:

$$\begin{aligned} B(R_0^4) = \{ & i_{70}, i_{71}, i_{72}, i_{73}, i_{80}, i_{81}, i_{82}, i_{83}, j_{20}, j_{21}, q_{30}, q_{31}, q_{32}, q_{33}, i_{g0}, i_{g1}, i_{g2}, \\ & i_{g3}, e_{60}, e_{61}, e_{62}, e_{63}, i_{h0}, i_{h1}, i_{h2}, i_{h3}, i_{d0}, i_{d1}, i_{d2}, i_{e0}, i_{e1}, i_{e2}, i_{f0}, i_{f1}, i_{f2}, i_{n0}, \\ & i_{n1}, i_{n2}, p_{50}, p_{51}, p_{52}, p_{53}, e_{a0}, e_{a1}, e_{a2}, e_{a3}, i_{o0}, i_{o1}, i_{o2}, i_{o3}\}. \end{aligned}$$

The graphical observable Markov boundary contains 50 variables. Just like in $B(R_0^3)$, not all observable direct inputs of J-K flip-flop f_2 over the period are contained in $B(R_0^4)$. Only j_{20} and j_{21} at first two instants (0 and 1) are contained in $B(R_0^4)$. This is because the inputs to f_2 at instant 3 have not gone through f_2, and no effects could be observed. Although the inputs to f_2 at instant 2 have gone through f_2, the effects of f_2's outputs have not been observed at e_a or p_5 before they go through another J-K flip-flop f_5 (i.e. any information resulted from inputs to f_2 at instant 2 are contained somewhere between the two J-K flip-flops, where no variables are observable). Also though i_{n0}, i_{n1}, and i_{n2} are contained in $B(R_0^4)$, but i_{n3} are not. This is because no effects of i_{n3} would be observed at e_a or p_5 within the period.

After $B(R_0^4)$ are observed and agents communicate with each other, the inference indicates that f_2 could be faulty with a belief of 84.21% (f_{23}). All other devices in component C_2 are believed to be normal with beliefs over 94%. All devices in components C_0, C_1 and C_3 are believed to be normal with beliefs over 96%. All devices in component C_4 are believed to be normal with beliefs over 93.95%. The inference results are significantly improved with one more instant modeled. This example also indicates that the potential improvement space will become less and less over more and more instants. This, from another perspective, shows a long history may not be so necessary in reasoning about the state of a dynamic domain.

6.5 Multiple Faults

To make the situation more complex, we next assume, besides J-K flip-flop f_2 in component C_2, J-K flip-flop f_4 in component C_0 is also abnormal. We randomly pick six consecutive instants as shown in Figure 13, where input signals, and those output signals that could be affected by the outputs of f_2 and f_4 are provided and presented as strings of six digits. The six instants will be divided into two consecutive periods each of which contains 4 instants. The two periods overlap over two instants. In Figure 13, the observable variables are underlined.

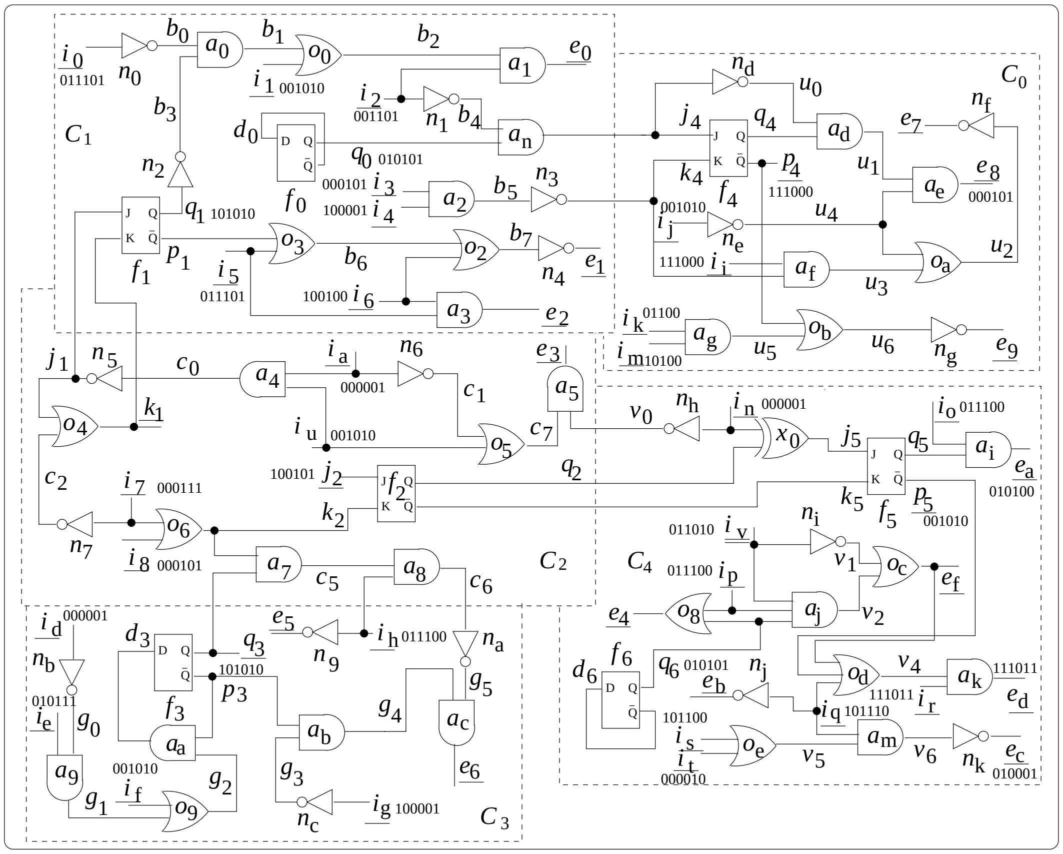
Fig. 13. The signal values of a sequential digital circuit over 6 instants.

The graphical observable Markov boundary $B(R_1^4)$ of R_1=$\{f_{40}, f_{41}, f_{42}, f_{43}\}$ is as shown as follows, and $B(R_0^4)$ is the same as given in Subsection 6.4. There are a total of 33 variables in $B(R_1^4)$.

$$B(R_1^4) = \{i_{i0}, i_{i1}, i_{i2}, i_{i3}, i_{j0}, i_{j1}, i_{j2}, i_{j3}, e_{80}, e_{81}, e_{82}, e_{83}, e_{70}, e_{71}, e_{72}, e_{73}, p_{40}, p_{41}, p_{42}, p_{43}, i_{20}, i_{21}, i_{22}, p_{40}, p_{41}, p_{42}, p_{43}, i_{30}, i_{31}, i_{32}, i_{40}, i_{41}, i_{42}\}.$$

Based on the observation of both $B(R_0^4)$ and $B(R_1^4)$ in the first period and communication among agents, the inference indicates that J-K flip-flop f_2 is believed to be faulty with a belief over 94.89%. This belief is higher than the one obtained in Subsection 6.4. This is because stronger evidence, that indicates f_2 could be faulty, appears in $B(R_0^4)$ in this period (after observing $B(R_0^4)$, (R_1^4) is irrelevant to the state of f_2). The strength of the evidence we observe could be different from period to period for the same problem. When a system is monitored period by period, the system problems could be detected in different periods. Once a device is believed to be faulty in some period, the device should be fixed or replaced (though it may not be indicated to be faulty at some other periods).

In component C_2, all other devices are believed to be normal with beliefs over 96.06% except *or* gate o_6 with a belief over 94.30% in the first period.

The problem in J-K flip-flop f_4, however, is not properly detected in this period, which is believed to be abnormal with a belief of 16.21%. All other devices in components C_0 are believed to be normal with beliefs over 96.06%. All devices in component C_1 are believed to be normal with beliefs over 96.06% except *inverter* n_1 with a belief of 66.74%, *and* gate a_n with a belief of 45.79%. That is, the system seems to attribute inconsistencies observed to n_1 and a_n instead of f_4. Anyway, neither belief is high enough to conclude that either n_1 or a_n or both are abnormal. All devices in component C_3 and C_4 are believed to be normal with beliefs over 96.06% except *exclusive or* gate x_0 with a belief of 93.48% and J-K flip-flop f_5 with a belief of 95.65%.

Next, we continue to monitor the circuit over a new period of 4 instants, which has two instants overlapped with the previous one. The inference based on the observation of both $B(R_0^4)$ and $B(R_1^4)$ and communication among agents indicates that J-K flip-flop f_2 is believed to be faulty with a belief of 47.25%. This is because that confusing evidence is observed in this period corresponding to the problem in f_2. Nevertheless, J-K flip-flop f_4 is believed to be faulty with a belief of 84.54%. All other devices in component C_0 are believed to be normal with beliefs over 96.06% except *and* gate a_d with a belief over 86.74%. All devices in component C_1 are believed to be normal with beliefs over 95.64% except *and* gate a_2 with a belief of 86.71%, *inverter* n_3 91.69%, and *inverter* n_1 82.98%. All other devices in component C_2 are believed to be normal with beliefs over 96.06% except *or* gate o_6 with a belief of 86.28%. All devices in components C_3 are believed to be normal with beliefs over 95.99%. All devices in components C_4 are believed to be normal with beliefs over 96.06% except *exclusive or* gate x_0 85.00%, and flip-flop f_5 73.14%. This time, we are confident that f_4 is faulty.

In the two consecutive periods, the posterior belief over the same entity may fluctuate significantly. For example, in the first period, f_2 is believed to be faulty with a belief over 94.89% and f_4 16.21%, and in the next period, f_2 is believed to be faulty with a belief of 47.25% and f_4 84.54%. The fluctuation is considered rational, which is caused by the variation in the strength of the evidence observed corresponding to the problems in the two devices in the two periods. In the first period, the observed evidence strongly indicates that f_2 is faulty and weakly indicates f_4 is faulty, and in the second period, the observed evidence strongly indicates that f_4 is faulty but weakly indicates that f_2 is faulty.

With dynamic systems being monitored period by period, the problems in the systems would be detected whenever proper evidence appears.

6.6 Complexity Growth

In general, the longer the period of a dynamic domain we can model, the better the inference results we can reach would be. However, the period of a dynamic domain we can model into an MSBN should be limited by the computational complexity. The computational complexity of inference using MSBNs is dominated by the largest clique size in the corresponding LJFs. In this subsection, we experimentally show the relationship between the computational complexity and the length of the modeled period of a dynamic domain.

The experiment is done on 3 different sequential digital circuits with different levels of complexity. Each circuit is divided into five components, and is modeled with a 5 agent MSBN over a range of periods. Depending on the circuit complexity, the period modeled could be up to 10 instant long. Beside the sequential digital circuit as shown in Figure 8, the other two sequential digital circuits are as shown in Figures 14 and 15, respectively.

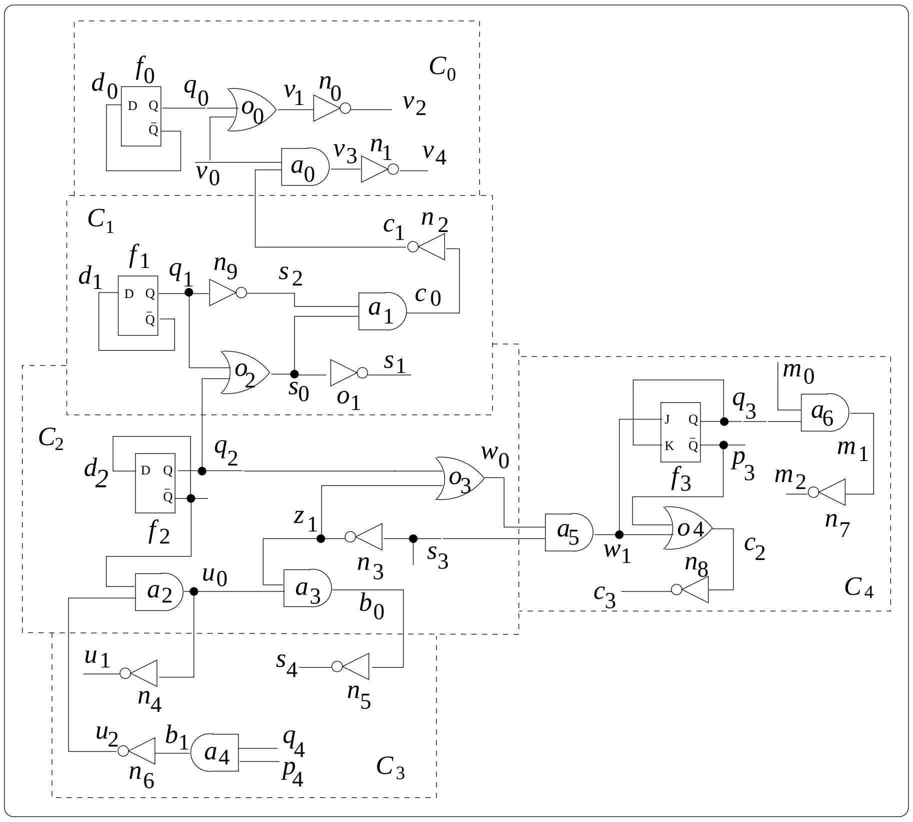

Fig. 14. A simple sequential digital circuit of five components.

The circuit as shown in Figure 14 has the simplest topological structure. The interfaces between any two adjacent components are as shown in Table 1. When modeled into an MSBN, a new set of the corresponding interface variables will be added to the corresponding interfaces for each new instant added.

The sequential digital circuit as shown in Figure 15 has a more complex topological structure. The interfaces between any two components are as shown in Table 1. Compared to the two sequential digital circuits, the one as shown in Figure 8 has the most devices and signals.

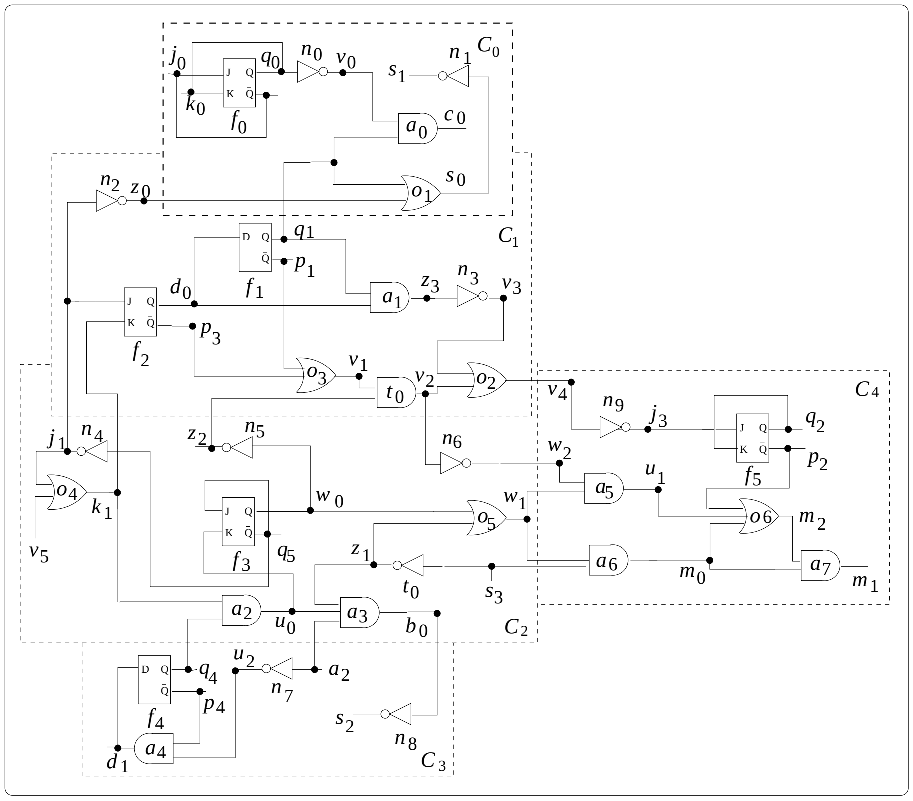

Fig. 15. A more complex sequential digital circuit of five components.

Table 1
Interfaces of the circuits in Figure 14 and 15.

Components	Interfaces of circuit in Fig. 14	Interfaces of circuit in Fig. 15
$C_0\&C_1$	c_1, v_3, v_4	z_0, q_1
$C_1\&C_2$	q_2, s_0, s_1	j_1, k_1, z_2, v_2
$C_2\&C_3$	u_2, u_0, b_0	u_0, b_0, d_2, q_4
$C_2\&C_4$	w_0, s_3	w_2, w_1, s_3, j_3

For the digital circuit as shown in Figure 8, we produce 4 MSBNs corresponding to the modeled periods from length 2 to length 5. For the digital circuit as shown in Figure 14, we produce 9 MSBNs corresponding to the modeled periods from length 2 to length 10. For the digital circuit as shown in Figure 15, we produce 4 MSBNs corresponding to the modeled periods from length 2 to length 5.

Since the computational complexity of the inference using MSBNs is dominated by the size of the largest cliques in the corresponding LJFs (linked junction forests), we show how the size of the largest cliques grows over the length of the represented period. Tables 2, 3 and 4 show the respective experimental results on different circuits.

Table 2
Maximum clique size vs the length of the modeled period for the circuit in Figure 8.

# of instants	Size of the largest cliques	Total # of cliques
2	8	246
3	13	395
4	17	527
5	22	645

Table 3
Maximum clique size vs the length of the modeled period for the circuit in Figure 14.

# of instants	Size of the largest cliques	Total # of cliques
2	8	114
3	11	148
4	14	211
5	19	252
6	21	287
7	24	330
8	27	371
9	29	416
10	31	458

Table 4
Maximum clique size vs the length of the modeled period for the circuit in Figure 15.

# of instants	Size of the largest cliques	Total # of cliques
2	11	108
3	19	164
4	30	219
5	39	267

From the Tables 2, 3 and 4, the size of the largest cliques and the number of cliques in the corresponding linked junction forests grow linearly in the number of instants modeled. For example, in the column “Size of the largest clique” of Table 2, the difference between any two consecutive numbers are approximately the same (5). Note the size of the largest cliques is not only affected by the complexity and the size of the circuits, but is also affected by how you model the circuits (e.g. the choices of interface variables and the interface size, etc.).

7 Conclusion

In dynamic multiagent domains, individual agents generally cannot evolve separately using DBNs because the temporal probabilistic messages from different subdomains are dependent of each other. This results in the decomposition issue and the distribution issue, which make it hard for the dynamic multiagent probabilistic inference to be performed both exactly and effectively. Nevertheless, in dynamic systems, the influence from the past could be weakened very quickly. We propose to represent and reason about the state of a dynamic multiagent domain period by period. By denser and relevant observation, the influence of the ignored history on the inference is reduced to a minimum. For relevant observation, we introduce graphical observable Markov boundary (GOMB) to capture all relevant observable variables. GOMB may also help us locate the relevant agents in the inference. For example, as shown in Subsection 6.4, the GOMB members in $B(R_0^4)$ (corresponding to flip-flop f_2 in the circuit as shown in Figure 8) only appear in subdomain C_2, C_3 and C_4. Hence, it is possible that, after proper initial processing, the 5 agent MSBN can be simplified to a 3 agent MSBN for cheaper inference.

Experiments show that the proposed method can successfully work on the simulated cases. The size of the largest cliques in the corresponding LJFs grows linearly in the length of the represented period. Hence, the period length affects the computational complexity in an exponential way. The length of the modeled period cannot be too long.

In the future, more experiments on more problem domains will be performed to further investigate the proposed approach.

References

[1] V. R. Lesser, L. D. Erman, Distributed interpretation: a model and experiment, IEEE Transactions on Computers C-29 (12) (1980) 1144–1163.

[2] Y. Xiang, B. Pant, A. Eisen, M. P. Beddoes, D. Poole, Painulm: a neuromuscular diagnostic aid using multiply sectioned Bayesian networks, in: D. I. Hudson (Ed.), Proceedings of ISMM (International Society for Mini and Microcomputers) International Conference on Mini and Microcomputers in Medicine and Healthcare, Long Beach, CA, 1991, pp. 64–69.

[3] Y. Xiang, B. Pant, A. Eisen, M. P. Beddoes, D. Poole, Multiply sectioned Bayesian networks for neuromuscular diagnosis, Artificial Intelligence in Medicine 5 (1993) 293–314.

[4] Y. Xiang, H. Geng, Distributed monitoring and diagnosis with multiply sectioned Bayesian networks, in: AAAI Spring Symposium on AI in Equipment Service Maintenance and Support, AAAI Press, Stanford, CA, 1999, pp. 18–25.

[5] A. Ghosh, S. Sen, Agent-based distributed intrusion alert system, in: Proceedings of the 6th International Workshop on Distributed Computing (IWDC'04), Kolkata, India, 2004, pp. 240–251.

[6] Y. Xiang, Probabilistic Reasoning in Multiagent Systems: A Graphical Models Approach, Cambridge University Press, 2002.

[7] S. Buchegger, J.-Y. L. Boudec, A robust reputation system for P2P and mobile ad-hoc networks, in: Proceedings of the 2nd Workshop on the Economics of Peer-to- Peer Systems, Cambridge, MA, 2004.

[8] B. Skyrms, R. Pemantle, A dynamic model of social network formation, Proceedings of National Academy of Sciences of the United States of America (PNAS) 97 (16) (2000) 9340–9346.

[9] H. Li, S. Majumdar, Dynamic decisions with short-term memories, Tech. rep., Department of Economics, University of Toronto (2005).

[10] D. Koller, Representation, reasoning, learning, Keynote talk at the 17th International Joint Conference on Artificial Intelligence (IJCAI-2001), Seattle, WA.

[11] L. M. de Campos, J. M. Fernandez-Luna, Reducing propagation effort in large polytrees: an application to information retrieval, in: Proceedings of the First European Workshop on Probabilistic Graphical Models (PGM'02), Cuenca, Spain, 2002, pp. 35–44.

[12] R. D. Shachter, Bayes-ball: the rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams), in: G. F. Cooper, S. Moral (Eds.), Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-1998), Morgan Kaufmann Publishers, Madison, WI, 1998, pp. 480–487.

[13] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming, John Wiley & Sons, Inc., New York, NY, 1994.

[14] L. P. Kaelbling, M. Littman, A. R. Cassandra, Planning and acting in partially observable stochastic domains, Artificial Intelligence 101 (1998) 99–134.

[15] C. Boutilier, Multiagent systems: challenges and opportunities for decision-theoretic planning, AI Magazine 20 (4) (1999) 35–43.

[16] D. S. Bernstein, S. Zilberstein, N. Immerman, The complexity of decentralized control of Markov decision processes, in: C. Boutilier, M. Goldszmidt (Eds.), Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence (UAI-2000), Morgan Kaufmann Publishers, Stanford, CA, 2000, pp. 32–37.

[17] C. V. Goldman, S. Zilberstein, Optimizing information exchange in cooperative multi-agent systems, in: Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2003), ACM Press, Melbourne, Australia, 2003, pp. 137–144.

[18] R. Nair, M. Tambe, M. Yokoo, D. Pynadath, S. Marsella, Taming decentralized POMDPs: towards efficient policy computation for multiagent settings, in: Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI'03), Morgan Kaufmann, Acapulco, Mexico, 2003, pp. 705–711.

[19] M. Littman, Algorithms for sequential decision making, Ph.D. thesis, Department of Computer Science, Brown University, Providence, Rhode Island (1996).

[20] J. Forbes, T. Huang, K. Kanazawa, S. Russell, The BATmobile: towards a Bayesian automated taxi, in: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI-1995), Morgan Kaufmann Publishers, Montréal, Québec, Canada, 1995, pp. 1878–1885.

[21] B. Sallans, Learning factored representations for partially observable Markov decision processes, in: S. Solla, T. Leen, K. R. Muller (Eds.), Proceedings of the Advances in Neural Information Processing Systems 12 (NIPS-1999), MIT Press, Denver, CO, 1999, pp. 1050–1056.

[22] X. Boyen, Inference and learning in complex stochastic processes, Ph.D. thesis, Computer Science Department, Stanford University, Stanford, CA (2002).

[23] K. Murphy, Y. Weiss, The factored frontier algorithm for approximate inference in DBNs, in: J. S. Breese, D. Koller (Eds.), Proceedings of the 17th Conference on Uncertainty in Artificial Intelligence (UAI-2001), Morgan Kaufmann Publishers, Seattle, WA, 2001, pp. 378–385.

[24] K. Murphy, Dynamic Bayesian networks: representation, inference and learning, Ph.D. thesis, CS Division, UC Berkeley, Berkeley, CA (July 2002).

[25] A. T. Ihler, J. W. F. III, A. S. Willsky, Loopy belief propagation: convergence and effects of message errors, Journal of Machine Learning Research 6 (May) (2005) 905–936.

[26] Y. Xiang, Temporally invariant junction tree for inference in dynamic Bayesian networks, in: R. E. Mercer, E. Neufeld (Eds.), Advances in Artificial Intelligence: Proceedings of the 12th Biennial Conference of the Canadian Society for Computational Studies of Intelligence, LNAI 1418, Springer, Vancouver, BC, Canada, 1998, pp. 363–377.

[27] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers, San Franciso, CA, 1988.

[28] J. Pearl, A. Paz, Graphoids: a graph-based logic for reasoning about relevance relations, in: B. D. Boulay, D. Hogg., L. Steels (Eds.), Advances in Artificial Intelligence 2, Amsterdam: North Holland, 1985, pp. 357–363.