Optimization of Inter-agent Belief Updating in Multiply Sectioned Bayesian Networks

Yang Xiang
Department of Computer Science, University of Regina
Regina, Saskatchewan, Canada S4S 0A2, yxiang@cs.uregina.ca

November 30, 1994

Abstract

Recent developments show that Multiply Sectioned Bayesian Networks (MSBNs) can be used for diagnosis of natural systems as well as for model-based diagnosis of artificial systems. They can be applied to single-agent oriented reasoning systems as well as multi-agent distributed probabilistic reasoning systems.

Belief propagation between a pair of subnets in a MSBN plays a central role in maintenance of global consistency. This paper studies the operation **UpdateBelief** for inter-subnet propagation originally presented with MSBNs. We analyze how the operation achieves its functionality, which provides hints as for how its efficiency can be improved.

We then define two new implementations of **UpdateBelief** that reduce the computational time for inter-subnet propagation. One of them is optimal in the sense that the minimal amount of computation for coordinating multi-linkage belief propagation is required. The optimization problem is solved through the solution of a graph-theoretic problem: the minimal weight open tour in a tree.

Keywords: Belief propagation, probabilistic reasoning, Bayesian networks, graph theory, multi-agent reasoning

1 Introduction

Multiply sectioned Bayesian networks (MSBNs) [9] is an extension of Bayesian networks (BNs) [4, 3, 1], originally developed for modular knowledge representation and more efficient inference computation in large application domains [7]. The basic assumption of MSBNs is *localization* [9, 8, 7]: Subdomains of the target domain are loosely coupled such that evidence and queries focus on one subdomain for a period of time before shifting to a different subdomain. Based on localization, a MSBN represents a large domain by a set

of interrelated Bayesian subnets, such that inference computation can be confined within one subnet at a time.

Two recent developments in probabilistic reasoning using BNs widened the scope of applicability of the MSBN representation and inference formalism:

Srinivas [5] proposed a hierarchical approach for model-based diagnosis, which can be viewed as a special case of MSBNs. For example, the set of input node I, output node O, mode node M, and dummy node D [5], which collectively form an interface between a higher level and a lower level in the hierarchy, is a d-sepset [9]. The 'composite joint tree' [5] corresponds to the 'hypertree' [9]. The way in which inference is performed in the composite join tree corresponds to the operation *ShiftAttention* [9]. His work showed that MSBNs can be applied to diagnosis of both natural systems (e.g., human body [7]) and artificial systems (e.g., electronic circuits [5]).

Instead of viewing a MSBN as representing multiple perspectives of a single reasoning agent, a MSBN can be viewed as representing multiple agents in a domain each of which holds one perspective of the domain. Following this new semantics, Xiang [6] extended MSBNs to distributed multi-agent probabilistic reasoning, where multiple agents collect local evidence asynchronously in parallel and exchange information infrequently to achieve a common goal. Due to this new view, we shall use the terms 'subnet' and 'agent' interchangeably in the paper.

Given the widened applicability of the MSBN formalism, this paper reexamines the key inference operation **UpdateBelief** [9] of MSBNs used in propagating belief from one agent to another. We propose two new versions of **UpdateBelief** to improve its efficiency. We compare the two improvements and indicate their trade-offs.

2 Multiply Sectioned Bayesian Networks

This section first extends the definition of d-sepset, a set of interfacing variables between agents, and then briefly reviews the theory of MSBNs. We shall use freely the formal results in [9, 6] for concepts that the rest of the paper depends on.

2.1 An extension to the definition of d-sepset

A MSBN consists of a set of interrelated Bayesian subnets. Each subnet shares a non-empty set of variables with at least one other subnet. The interfacing set between each pair of subnets must satisfy a *d-sepset* condition such that, when the pair is isolated from the MSBN, the interfacing set renders the two subnets conditionally independent.

We augment the d-sepset definition [9] to cover more cases. Given graphs $G^1 = (N^1, E^1)$ and $G^2 = (N^2, E^2)$, where N is the set of nodes and E is the set of links, we shall denote their *union* graph by $G^1 \sqcup G^2 = (N^1 \cup N^2, E^1 \cup E^2)$.

Definition 1 (The d-sepset) *Let $D = D^1 \sqcup \ldots \sqcup D^\beta$ be a DAG. Let N^i be the set of*

nodes of D^i. The set of nodes $I = N^i \cap N^j$ is a **d-sepset** *between two distinct subDAGs D^i and D^j if, for every $A \in I$ with its parents π in D, either (1) $\pi \subseteq N^i$ or (2) $\pi \subseteq N^j$ or (3) $\pi \subseteq N^k$ where $k \neq i$ and $k \neq j$.*

An element of a d-sepset is called a **d-sepnode**. *When the above condition holds, D is said to be* **multiply sectioned** *or simply* **sectioned** *into $\{D^1, \ldots, D^\beta\}$.*

The above case (3) is not covered in the original definition. Such case arises in MSBNs like the one in Figure 1 [6]. The intersection of N^1 and N^2 is $\{\tau, \iota\}$. The parents of τ is neither contained in D^2 nor D^3, but contained in D^1.

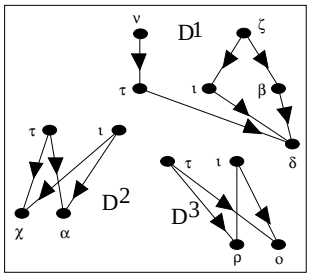

Figure 1: A MSBN with three subnets: D^1, D^2, and D^3.

It can be easily shown that the conditional independence property held with the original definition of d-sepset is still valid with the new definition.

2.2 Overview of MSBNs

The overall structure with which the subnets of a MSBN are organized is subject to a constraint called *soundness of sectioning*. Without the condition, a MSBN is subject to loss of information when later transformed into its secondary representation: a linked junction forest. A sufficient condition for sound sectioning is to construct a MSBN with a *hypertree* structure. The hypernodes in the hypertree are subnets of the MSBN. The hyperlinks are d-sepsets between subnets. Conceptually, the hypertree structured MSBN is built by adding one subnet (hypernode) to existing ones at a time. Only one d-sepset (hyperlink) to an existing subnet is explicitly stored. A hypertree structured MSBN guarantees that each hyperlink renders the two parts of the MSBN that it connects conditionally independent. Figure 2 depicts a hypertree structured MSBN. Each box represents a subnet. Boundaries between boxes represent interfacing sets. The superscripts of subDAGs indicate a possible order of construction.

Once a hypertree structured MSBN is constructed, it is then converted into a *linked junction forest* (LJF) of the identical hypertree structure. Each hypernode in the hypertree is a *junction tree* (clique tree, join tree) converted from a subnet in the hypertree structured MSBN. Each hyperlink of the hypertree is a set of *linkages* converted from the corresponding d-sepset in the hypertree structured MSBN in the following way:

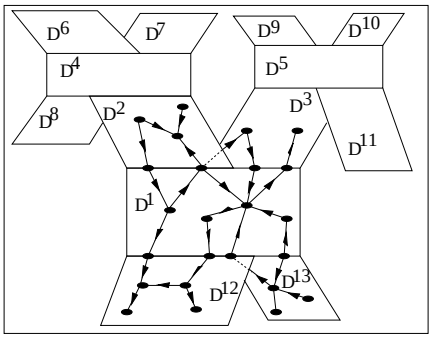

Figure 2: A MSBN with a hypertree structure.

Definition 2 (linkage [9]) *Let I be the d-sepset between two JTs T^a and T^b. A* **linkage** *of T^a relative to T^b is a set L of nodes such that: (1) There exists a clique $C \in T^a$ with $L = C \cap I$. C is called the* **host clique** *of L. (2) There is no subset of L that is also a linkage.*

In general, there are multiple linkages between two JTs, which are then indexed:

Algorithm 3 (indexing linkages [9]) *Let linkages between two JTs be defined.*

1. *Pick one JT, say T. Create a tree G with nodes labeled by the linkages. Connect two nodes by a link if their linkage hosts in T are connected by a path on which every intermediate clique is a non-linkage host. Call G a* **linkage tree**.
2. *Index the nodes of G into $L_1, L_2, \ldots$ in any order consistent with G, i.e., for every $i > j$ there is a unique predecessor $j(i) < i$ such that $L_{j(i)}$ is adjacent to L_i.*

The redundancy sets associated with the set of linkages are then defined:

Definition 4 (redundancy set [9]) *Let a set of linkages $\mathbf{L} = \{L_1, \ldots, L_g\}$ be indexed by Algorithm 3. A* **redundancy set** *R_i for index i is defined as*

$$R_i = \begin{cases} \phi & \text{if } i = 1, \\ L_i \cap L_{j(i)} & i > 1;\ j(i) < i;\ L_{j(i)} \text{ is adjacent to } L_i \text{ in the linkage tree } G. \end{cases}$$

The linkage tree so constructed is itself a junction tree, and the redundancy sets are sepsets of the linkage tree [9]. This allows a belief table $B(\underline{I})$ on d-sepset be constructed by belief tables of linkages and redundancy sets in the following form: $B(\underline{I}) = \prod B(L_j) / \prod B(R_k)$ where $B(L_j)$ is the belief table of a linkage, and $B(R_k)$ is the belief table of a redundancy set.

The above assembled $B(\underline{I})$ may not be a correct marginalization of the belief on the correspond JT, which we denote by $B(I)$. A LJF is said to be *separable* if $B(\underline{I}) \propto B(I) = \sum_{N \setminus I} B(T)$ for every JT T and the related d-sepset I, where $\propto$ reads 'proportional to', N is the set of variables contained in T, and $\sum$ denotes marginalization.

The topological constraint for separability is based on the concept of host tree:

Definition 5 (host tree [9]) *Let T be a junction tree and* **L** *be the set of linkages between T and a neighbour JT. A* **host tree** *of T relative to* **L** *is the clique tree resultant from recursively removing from T every leaf clique that is a non-linkage host.*

When every host tree in a LJF satisfies a *host composition* condition [9], the separability is guaranteed.

Figure 3 shows a JT taken from [7]. The d-sepset is

$$I = \{A, B, C, D, E, F, G, H, I, J, K, L, M\}.$$

The linkages for this case happen to be the same as the linkage hosts, i.e., $L_i = C_i$ $(i = 1, \ldots, 4)$. But this is not the general case. The linkage tree has the set of nodes $\{C_1, C_2, C_3, C_4\}$, and the set of links $\{(C_1, C_2), (C_2, C_3), (C_3, C_4)\}$. The set of linkages L_i $(i = 1, \ldots, 4)$, as indexed, determine the set of redundancy sets: $\{L_1 = \phi, L_2 = \{A, G, I, J, L, M\}, L_3 = \{B, D, E, F, I, J, L, M\}, L_4 = \{B, D, F, H, J, L, M\}\}$. The host tree is the subtree induced by C_1, C_2, C_3, C_4 and the clique on the path from C_1 to C_2.

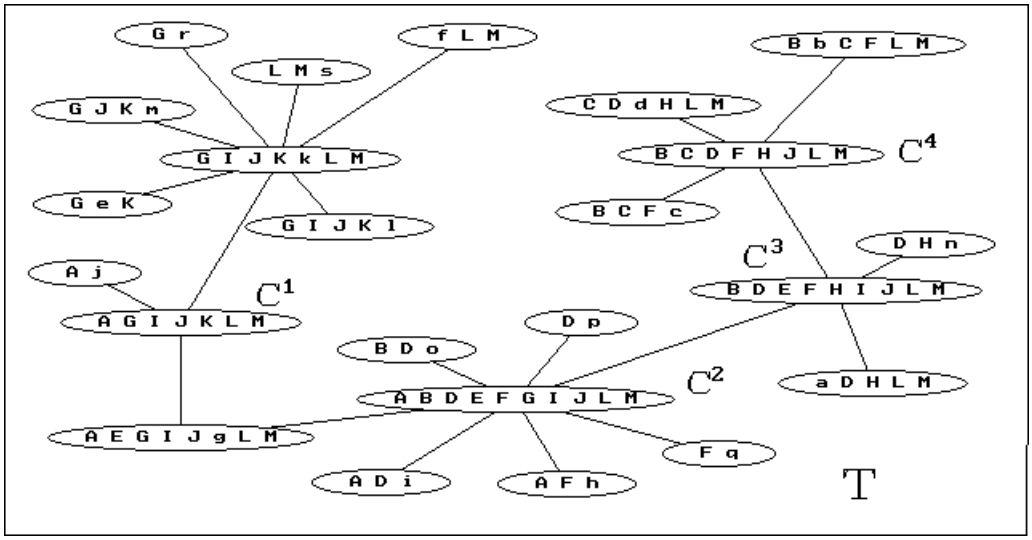

Figure 3: A junction tree taken from the LJF of PAINULIM with variable names revised to simplify presentation. Upper case letters represent d-sepnodes and lower case letters represent non-d-sepnodes. The cliques C_1, C_2, C_3, C_4 are linkage hosts.

Parallel to the transformation of the graphical structure, there is a corresponding transformation of the probability tables in the MSBN to *belief tables* in the junction forest. The overall conversion guarantees a *joint system belief* of the LJF can be assembled from belief tables distributed in the forest, which is equivalent to the joint probability distribution of the MSBN.

To answer queries by efficient local computation in a LJF, it must be made consistent. A LJF is *locally consistent* if all JTs are *internally consistent*, i.e., when marginalized onto the same set of variables different belief tables in a junction tree yield the same marginal distribution. A LJF is *boundary consistent* if each pair of linked JTs are consistent with respect to their interfacing set. A LJF is *globally consistent* iff it is both locally consistent and boundary consistent.

A set of operations [9, 6] are developed to achieve consistency in a LJF during evidential reasoning: **BeliefInitialization** establishes initial global consistency. **DistributeEvidence** causes an outward belief propagation within a JT, and brings the JT internally consistent after evidence on variables in a single clique has been entered. **UnifyBelief** brings a JT internally consistent after evidence on variables in multiple cliques has been entered. **EnterEvidence** updates belief in a JT in light of new evidence, and brings the JT internally consistent again by calling either **DistributeEvidence** or **UnifyBelief**. **AbsorbThroughLinkage** brings two linkage hosts in different JTs into consistency. **UpdateBelief** updates the belief of a JT T relative to an adjacent JT, and brings T internally consistent. In a single agent MSBN, **ShiftAttention** allows the user to enter multiple pieces of evidence into a JT of current attention, and, when the user shifts attention to a target JT, maintains consistency along the hyperpath in the hypertree structured forest from the current JT to the target. In a multi-agent MSBN, **CommunicateBelief** regains global consistency after multiple agents have obtained evidence asynchronously in parallel. Both **ShiftAttention** and **CommunicateBelief** rely on **UpdateBelief** for inter-subnet belief propagation.

3 Insight into the UpdateBelief Operation

The operation **UpdateBelief** plays a central role in inter-agent communication. It is defined as follows:

Operation 6 (UpdateBelief [9]) *Let $\{L_1, \ldots, L_m\}$ be the set of linkages between JTs T^a and T^b. Let U_i^a and U_i^b be the linkage hosts of L_i ($i = 1, \ldots, m$) in T^a and T^b, respectively. When* **UpdateBelief** *is initiated by T^a relative to T^b, the following is performed:*

AbsorbThroughLinkage *is called in U_i^a for each i (in ascending order) to absorb from U_i^b through L_i, followed by a* **DistributeEvidence** *called in U_i^a.*

In order to bring two neighbor JTs into consistency, we need to propagate the belief table $B(I)$ on the d-sepset I from T^a to T^b. Since the size of $B(I)$ is exponential to the size of I, the propagation can be expensive. In multi-agent MSBNs with remotely located agents, it would be necessary to pass $B(I)$ through communication channels. **UpdateBelief** makes use of conditional independence among members of the d-sepset, and propagates $B(I)$ by only passing the belief tables $B(L_i)$, which are collectively smaller in size when $B(I)$ is large. For example, if the d-sepset I contains ten binary variables,

$B(I)$ has a size of 1024. If $I = L_1 \cup L_2 \cup L_3$ with each L_i containing 5 variables, then the three belief tables on linkages have a total size 96.

However, this savings in communication bandwidth has a price to pay. As analyzed in [9], each **AbsorbThroughLinkage** must be followed by a **DistributeEvidence** in T^a. When T^a is large, this repeated **DistributeEvidence** can be expensive. Based on the argument "communication is slower than computation" [2], the tradeoff is justified [6] by observing that **DistributeEvidence** involves only local computation. The objective of this paper is, under the guideline that communication savings should be preferred over computation savings, to improve the efficiency of the local computation as much as possible.

In the original presentation of **UpdateBelief** [9], how the operation achieves its goal is not analyzed. The proof of the following theorem gains insight into this operation, and provides hints for improvement of its efficiency.

Theorem 7 *Let I be the d-sepset between JTs T^a and T^b in a separable LJF. Let $\{L_1, \ldots, L_m\}$ be the set of linkages. Let the two JTs be internally consistent. Let $B(I^a)$ ($B(I^b)$) be the belief table on I defined by marginalization of $B(T^a)$ ($B(T^b)$).*

After **UpdateBelief**, *$B'(T^a) = B(T^a) * B(I^b)/B(I^a)$, and T^a is internally consistent.*

Proof:

We prove by induction on the index of linkages. **AbsorbThroughLinkage** in **UpdateBelief** is performed by T^a in the order $L_1, \ldots, L_m$. After **AbsorbThroughLinkage** is performed through L_1, we have

$$B_1(T^a) = B(T^a) * B(L_1^b)/B(L_1^a).$$

After **AbsorbThroughLinkage** is performed through L_2, we have

$$B_2(T^a) = B_1(T^a) * B(L_2^b)/B_1(L_2^a).$$

Note that the two $B()$ in the right-hand side of the previous equation are now replaced by $B_1()$. This is due to the first **DistributeEvidence** that follows the first **AbsorbThroughLinkage**. Since $B_1(L_2^a)$ is the marginalization of $B_1(T^a)$, the following equation holds:

$$B_1(L_2^a) = B(L_2^a) * B(L_1 \cap^b L_2)/B(L_1 \cap^a L_2),$$

where '$\cap^a$' signifies that the intersection is defined in T^a.

Substituting $B_1(T^a)$ and $B_1(L_2^a)$, we obtain

$$B_2(T^a) = B(T^a) * \frac{B(L_1^b) * B(L_2^b)/B(L_1 \cap^b L_2)}{B(L_1^a) * B(L_2^a)/B(L_1 \cap^a L_2)} = B(T^a) * \frac{B(L_1 \cup^b L_2)}{B(L_1 \cup^a L_2)},$$

where the second equality holds because of the way in which the linkage tree is constructed and the separability (Section 2.2).

Assume that, after **AbsorbThroughLinkage** is performed through L_i followed by **DistributeEvidence**, we have

$$B_i(T^a) = B(T^a) * \frac{B(\bigcup_{j\leq i}^b L_j)}{B(\bigcup_{k\leq i}^a L_k)},$$

and

$$B_i(L_{i+1}^a) = B(L_{i+1}^a) * \frac{B((\bigcup_{j\leq i}^b L_j) \cap^b L_{i+1})}{B((\bigcup_{k\leq i}^a L_k) \cap^a L_{i+1})}.$$

After **AbsorbThroughLinkage** is performed through L_{i+1}, we have

$$B_{i+1}(T^a) = B_i(T^a) * B(L_{i+1}^b)/B_i(L_{i+1}^a).$$

By substituting $B_i(T^a)$ and $B_i(L_{i+1}^a)$, it yields

$$\begin{aligned} B_{i+1}(T^a) &= B(T^a) * \frac{B(\bigcup_{j\leq i}^b L_j) * B(L_{i+1}^b)/B((\bigcup_{j\leq i}^b L_j) \cap^b L_{i+1})}{B(\bigcup_{k\leq i}^a L_k) * B(L_{i+1}^a)/B((\bigcup_{k\leq i}^a L_k) \cap^a L_{i+1})} \\ &= B(T^a) * \frac{B(\bigcup_{j\leq i+1}^b L_j))}{B(\bigcup_{k\leq i+1}^a L_k))}, \end{aligned}$$

where the second equality holds because of the way in which the linkage tree is constructed and the separability (Section 2.2).

After the **DistributeEvidence** is performed, T^a is internally consistent, and it follows that

$$B_{i+1}(L_{i+2}^a) = B(L_{i+2}^a) * \frac{B((\bigcup_{j\leq i+1}^b L_j) \cap^b L_{i+2})}{B((\bigcup_{k\leq i+1}^a L_k) \cap^a L_{i+2})}.$$

Therefore, after **AbsorbThroughLinkage** is performed through the last linkage L_m, we obtain the updated belief

$$B_m(T^a) = B(T^a) * \frac{B(\bigcup_{j\leq m}^b L_j))}{B(\bigcup_{k\leq m}^a L_k))} = B(T^a) * B(\underline{I^b})/B(\underline{I^a}) = B(T^a) * B(I^b)/B(I^a),$$

where $B(\underline{I^a})$ ($B(\underline{I^b})$) is the belief table on I assembled from belief tables of linkages and redundancy sets defined in T^a (T^b), and the last equality holds due to the separability.

T^a is internally consistent after the last **DistributeEvidence**. □

4 Efficiency Improvement of UpdateBelief

In the proof of Theorem 7, it is observed that the inductive assumption on $B_i(L_{i+1}^a)$ is valid as long as the host tree (Section 2.2) is made consistent after the **AbsorbThrough-Linkage** through L_i. The consistency of the entire JT is not necessary. Hence belief propagation beyond the boundary of the host tree, as performed by **DistributeEvidence**, is not necessary.

We therefore define a new operation **DistributeEvidenceOnHostTree**. It is the same as **DistributeEvidence** except that it terminates at the leaves of the host tree. For example, if **DistributeEvidenceOnHostTree** is called in C_2 in the JT of Figure 3, the belief propagation will proceed along two directions: One goes from C_2 to C_1 along the unique path between them and terminates at C_1, and another goes from C_2 to C_4 and terminates at C_4.

Replacing **DistributeEvidence** by the new operation, we can define a new version of **UpdateBelief**:

Operation 8 (UpdateBelief2) *Let $\{L_1, \ldots, L_m\}$ be the set of linkages between JTs T^a and T^b. Let U_i^a and U_i^b be the linkage hosts of L_i ($i = 1, \ldots, m$) in T^a and T^b, respectively. When* **UpdateBelief2** *is initiated by T^a relative to T^b, the following is performed:*

AbsorbThroughLinkage *is called in U_i^a for each i (in ascending order) to absorb from U_i^b through L_i. For $i = 1, \ldots, m-1$, it is followed by* **DistributeEvidenceOnHostTree** *called in U_i^a. For $i = m$, it is followed by* **DistributeEvidence** *called in U_m^a.*

Corollary 9 *Let I be the d-sepset between JTs T^a and T^b in a separable LJF. Let $\{L_1, \ldots, L_m\}$ be the set of linkages. Let the two JTs be internally consistent. Let $B(I^a)$ ($B(I^b)$) be the belief table on I defined by marginalization of $B(T^a)$ ($B(T^b)$).*

After **UpdateBelief2**, *$B'(T^a) = B(T^a) * B(I^b)/B(I^a)$, and T^a is internally consistent.*

Proof:

After each **DistributeEvidenceOnHostTree**, the host tree is internally consistent. Hence, all intermediate results on $B_i(T^a)$ and $B_i(L_{i+1}^a)$ ($i = 1, \ldots, m$) in the proof of Theorem 7 are still valid. The difference is that T^a as a whole is not internally consistent.

The **DistributeEvidence** at the end restores the internal consistency of T^a. □

UpdateBelief2 performs repeatedly **DistributeEvidenceOnHostTree** instead of **DeistributeEvidence**. Computational savings are obtained by not having to propagate belief beyond the host tree for a number of times proportional to the number of linkages. When the host tree is significantly smaller than the JT, the savings can also be significant.

5 Further Efficiency Improvement of UpdateBelief

Examination of the proof of Theorem 7 shows that propagation of belief to the entire host tree is still unnecessary. For the result of the theorem to be valid, it is sufficient to update $B_{i-1}(L_{i+1}^a)$ to $B_i(L_{i+1}^a)$, after **AbsorbThroughLinkage** through L_i. This implies that, between two successive **AbsorbThroughLinkage**, it is sufficient to propagate the new belief to the next host clique. Based on this idea, a more efficient **UpdateBelief** can be defined. We illustrate the new operation with an example:

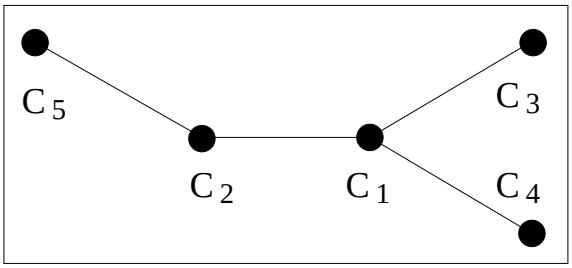

Figure 4: A host tree with five linkage hosts.

Consider the host tree in Figure 4. We assume each clique is a host, and each clique is indexed by the index of the corresponding linkage. Suppose **AbsorbThroughLinkage** is performed in the order $i = 1, \ldots, 5$. After **AbsorbThroughLinkage** through L_1, we propagate the new belief in C_1 to C_2 (one inter-clique propagation). After **AbsorbThroughLinkage** through L_2, we propagate the new belief in C_2 to C_1 and then to C_3 (two inter-clique propagations). Propagating new belief this fashion, we need to perform $1 + 2 + 2 + 3 = 8$ inter-clique belief propagations, before the **AbsorbThroughLinkage** through L_5. Compared to $4 + 4 + 4 + 4 = 16$ inter-clique propagations needed in **UpdateBelief2**, there is an about 50% computational savings.

Additional savings can be obtained by optimizing the new operation. We note that **AbsorbThroughLinkage** needs not be performed in the ascending order of linkage indexes, since linkages can be indexed by any order consistent with the host tree (Section 2.2). The same set of redundancy sets will be constructed. We therefore have the freedom to perform **AbsorbThroughLinkage** in any order consistent with the host tree.

If we choose the order $i = 5, 2, 1, 3, 4$ for the host tree in Figure 4, we only need to perform $1 + 1 + 1 + 2 = 5$ inter-clique belief propagations: Three propagations less than the previous order.

In the next section, we present the result on how to determine the optimal order for performing **AbsorbThroughLinkage**, given a host tree.

6 Optimization of UpdateBelief

The problem of finding optimal order for the performance of **AbsorbThroughLinkage** in **UpdateBelief** can be abstracted into the following:

Problem Statement 10 *Given a weighted tree of n nodes, order the nodes from 1 to n such that $\sum_{i=1}^{n-1} w(i, i+1)$ is minimized, where $w(i, i+1)$ is the path weight from node i to node $i + 1$.*

The tree in this model corresponds to the given host tree. The link weight corresponds to the amount of computation to propagate belief across cliques in the host tree. The model assumes that every node is a host. If this is not the case, the propagation through non-host nodes can be modeled into the link weights such that the resultant tree has no non-host node.

The problem can be restated equivalently in a simpler form using the concept *walk* in graph theory.

Problem Statement 11 *Given a weighted tree of n nodes, find a walk with the minimal weight that visits every node at least once.*

We define the concept *tour* to be used frequently in solving the above problem:

Definition 12 (tour) *A* **tour** *of a graph is a walk that visits each node at least once. A* **closed tour** *is a tour that starts and ends with the identical node. Otherwise, it is an* **open tour**.

The problem can now be equivalently expressed as follows:

Problem Statement 13 *Given a weighted tree of n nodes, find an open tour with the minimal weight.*

In order to develop an algorithm that solve the above problem, we first study a closed tour, since the minimal weight open tour has a simple relationship with the minimal weight closed tour. To simplify the intermediate derivations, we assume that all link weights are identical. We then simply deal with a minimal length tour with the length of each link being one. We remove this assumption at the end.

Lemma 14 *A closed tour of a tree with the minimal length traverses each link exactly twice.*

Proof:

We prove by induction: The statement is trivially true for a tree with only one link. Assume that it is true for a tree with k link(s).

Consider a tree with $k+1$ links. Select a leaf x with its internal neighbor y. If we remove x and the link (x, y), the resultant subgraph is still a tree with k link(s). By assumption, it has a minimal length closed tour that traverses each link exactly twice. To include x and (x, y) in the closed tour, one must at least travel from y to x and then come back. This completes a closed tour of the original tree with the minimal additional link traversal. □

For example, a closed tour of the minimal length for the tree in Figure 5 is

$$(C_5, C_2, C_6, C_7, C_8, C_7, C_6, C_2, C_1, C_9, C_{10}, C_9, C_1, C_3, C_1, C_4, C_1, C_2, C_5).$$

The length of the tour is 18, which is twice of the number of links of the tree.

We define a *terminal chain* used in the following discussion.

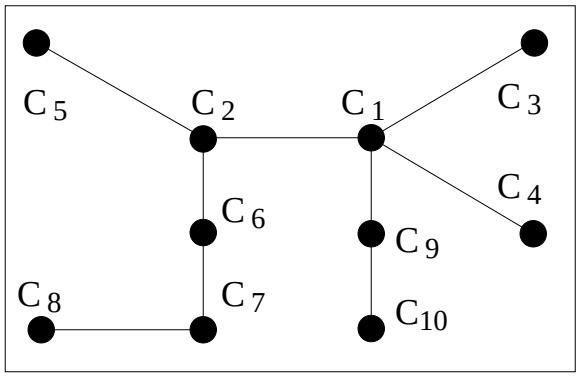

Figure 5: A tree with identical length for links

Definition 15 (terminal chain) *A* **terminal chain** *in a graph is a simple open path that terminates at both ends by a pair of leaf nodes.*

For instance, one terminal chain in Figure 5 is $(C_5, C_2, C_6, C_7, C_8)$. The simple open path (C_{10}, C_9, C_1) is not a terminal chain, since C_1 is not a leaf.

We establish the relation between a minimal length closed tour and a minimal length open tour through a terminal chain:

Lemma 16 *An open tour of a tree can be constructed such that its length is the length of the minimal length closed tour minus the length of a terminal chain.*

Proof:

It is sufficient to show that an open tour can be constructed by reconnecting a minimal length closed tour without traversing a terminal chain twice.

Choose an arbitrary terminal chain from the tree. Start from one leaf of the terminal chain and travel along the chain. At each internal node x of degree 3 or more, for each subtree other than the chain, traverse the subtree as in the original tour. After each subtree at x other than the chain has been toured, continue to travel along the chain. The open tour terminates when the other leaf of the chain is reached. The open tour travels the same set of links as a minimal length open tour except that the terminal chain is traveled only once. □

We illustrate the constructive proof using Figure 5. Suppose we are given the previous minimal length (18) closed tour

$$(C_5, C_2, C_6, C_7, C_8, C_7, C_6, C_2, C_1, C_9, C_{10}, C_9, C_1, C_3, C_1, C_4, C_1, C_2, C_5),$$

and a terminal chain $(C_5, C_2, C_6, C_7, C_8)$ of length 4. By reconnecting the tour and traversing the chain only once, we obtain the open tour

$$(C_5, C_2, C_1, C_9, C_{10}, C_9, C_1, C_3, C_1, C_4, C_1, C_2, C_6, C_7, C_8).$$

It has the length $18 - 4 = 14$.

Lemma 17 extends the result of Lemma 16 to a minimal length open tour:

Lemma 17 *An open tour of a tree with the minimal length has the length of the minimal length closed tour minus the length of the longest terminal chain.*

Proof:

Assume an open tour is constructed as in the proof of Lemma 16, which is based on a terminal chain of the longest length of all terminal chains. It is sufficient to show that no single link traversal can be removed from this tour such that it remains to be an open tour.

Each link along the terminal chain is traversed only once. If any one of these link traversals is removed, the tour will be disconnected. Each link in a subtree other than the terminal chain is traversed twice, once travels away from the terminal chain, and the other time travels towards the chain. If any of these link traversals is removed, the tour of the subtree will be disconnected. □

We now remove the assumption of identical link weight. The result for the tour problem follows:

Corollary 18 *An open tour of a tree of the minimal weight can be constructed by modifying a closed tour of the minimal weight such that the terminal chain of the maximal weight is traversed only once.*

Proof:

This is a direct extension of Lemma 17 to weighted trees. The replacement of length by weight is valid because (1) whether link weights are identical or not is irrelevant to the validity of Lemma 14 and Lemma 16; and (2) Lemma 17 will still be valid if different link weights are used. □

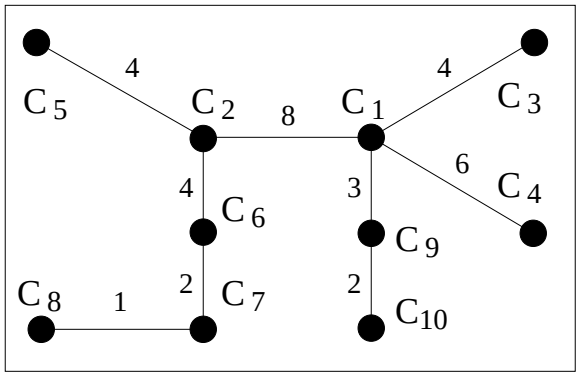

Figure 6: A weighted tree for the problem of the minimal weight tour

In Figure 6, the terminal chain of the maximal weight $(1 + 2 + 4 + 8 + 6 = 21)$ is $(C_8, C_7, C_6, C_2, C_1, C_4)$. Therefore, the open tour of the minimal weight is

$$(C_8, C_7, C_6, C_2, C_5, C_2, C_1, C_3, C_1, C_9, C_{10}, C_9, C_1, C_4),$$

and the minimal weight is $2 * (1 + 2 + 4 + 4 + 8 + 3 + 2 + 6 + 4) - 21 = 47$.

The following algorithm orders the nodes in a given tree such that a minimal weight open tour can be constructed by following the order:

Algorithm 19

Input: *A weighted tree of a set N of n nodes with m leaves $v_1, \ldots, v_m$ $(m < n)$.*
Var: *m functions f_i $(i = 1, \ldots, m)$ from elements of N to the set of real numbers*
Output: *An open tour of the tree.*
begin

1 for $i = 1$ to m, do
$f_i(v_i) := 0$
2 for $i = 1$ to m, do
for each node z with $f_i(z)$ undefined
if z is accessible from a node t and $f_i(t)$ is defined, do
$f_i(z) = f_i(t) + w(z, t)$
3 for each leaf z, do
$M(z) := \max_{i=1}^{m} f_i(z)$
4 Find the leaf x such that $M(x) = \max_{i=1}^{m} M(v_i)$
5 Retrieve j such that $f_j(x) = M(x)$
6 Retrieve leaf y such that $f_j(y) = 0$
7 Visit nodes in N along the terminal chain from x to y traversing each subtree in a depth-first fashion, and order a node when it is visited the first time

end

The steps 1 through 6 of the algorithm finds the terminal chain with the maximal weight. It can be viewed as a variation of Dijkstra's shortest-path algorithm in a tree. It differs from the latter in that it finds the longest (heavest) path between a non-predetermined pair of leaves. The step 7 of the algorithm produces the order as stated in Problem Statement 10, which can be used easily to construct a minimal weight open tour.

The algorithm has a complexity of $O(n^2)$ for both time and space.

Figure 7 illustrates the Algorithm 19. The weighted tree is identical to that in Figure 6 up to a renaming of nodes. The renaming is performed such that the five leaves are indexed from 1 to 5, satisfying the input description of Algorithm 19. Note that the indexing of nodes in Figure 6 is consistent with the tree structure, but the indexing in Figure 7 is

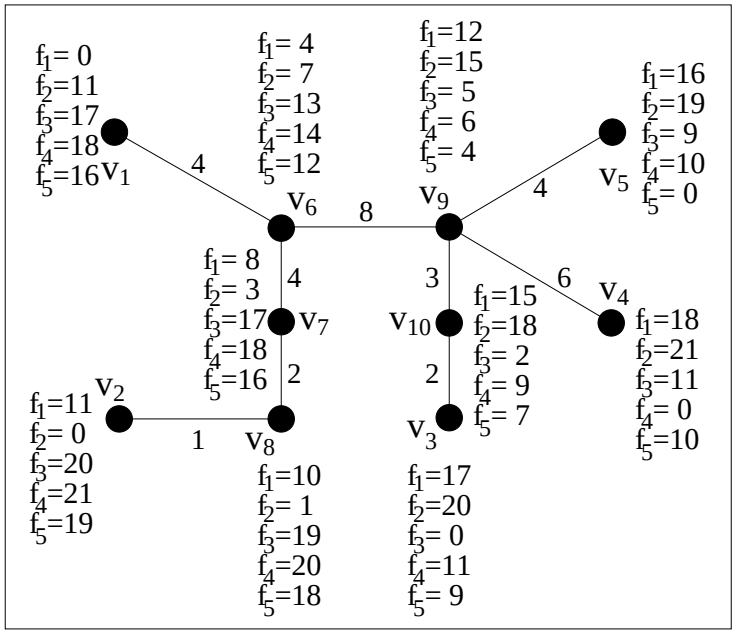

Figure 7: A weighted tree to illustrate Algorithm 19.

not. To simplify the presentation, arguments for functions are omitted (i.e., $f_1(v_2) = 11$ is depicted as $f_1 = 11$).

Following the Algorithm 19, we obtain $M(v_1) = 18$, $M(v_2) = 21$, $M(v_3) = 20$, $M(v_4) = 21$, and $M(v_5) = 19$. Therefore, $x = v_2$, $j = 4$, and $y = v_4$. The heavest terminal chain is from v_2 to v_4. This is the same as we obtained from Figure 6. The last step produces the order: $(v_2, v_8, v_7, v_6, v_9, v_4)$ from which the same minimal weight open tour as we obtained from Figure 6 can be constructed.

Before we can use the order in the new version of **UpdateBelief**, we need to guarantee one more property that the order is consistent with the tree structure, which is required by Theorem 7. This is certainly true since the next node to order (according to step 7 of Algorithm 19) is always adjacent to the subtree traversed so far.

We summarize the above discussion on Algorithm 19 by the following theorem. The proof is trivial given Corollary 18, and the equivalence of Problem Statement 10 and 13.

Theorem 20 *Let T be a weighted tree. The node order generated by Algorithm 19 is consistent with T, and the open tour constructed according to the order has the minimal weight.*

We are ready to define another improved version of **UpdateBelief**: Since propagation to the host tree is not necessary, we define a new operation **DistributeEvidenceOn-**

Chain. It is the same as **DistributeEvidence** except belief only propagate along a chain in the host tree to a given clique. The new **UpdateBelief** follows:

Operation 21 (UpdateBelief3) *Let $\{L_1, \ldots, L_m\}$ be the set of linkages between JTs T^a and T^b. Let U_i^a and U_i^b be the linkage hosts of L_i ($i = 1, \ldots, m$) in T^a and T^b, respectively. Let the indexes $i = 1, \ldots, m$ be consistent with the order produced by Algorithm 19. When* **UpdateBelief3** *is initiated by T^a relative to T^b, the following is performed:*

AbsorbThroughLinkage *is called in U_i^a for each i (in ascending order) to absorb from U_i^b through L_i. For $i = 1, \ldots, m-1$, it is followed by* **DistributeEvidenceOnChain** *called in U_i^a along the unique path from U_i^a to U_{i+1}^a. For $i = m$, it is followed by* **DistributeEvidence** *called in U_m^a.*

Given a JT and a set of linkages to a neighbor JT, **UpdateBelief3** is optimal in the sense that the minimal amount of computation for coordinating multi-linkage belief propagation is required.

7 Comparison of Different Implementations of UpdateBelief

This paper addresses the computation efficiency of belief propagation between a pair of Bayesian subnets in a MSBN. Inter-subnet belief propagation involves the passage of the probability distribution on d-sepset I from one subnet to a neighbor.

A brute force method will form a large clique that contains I in each subnet involved, and pass the belief table $B(I)$ directly. It is computationally the most expensive, both locally and inter-subnet-wise.

The first improvement is the original **UpdateBelief**: Only belief tables on linkages are passed, which are collectively smaller than $B(I)$.

The second improvement is **UpdateBelief2**: Multiple performance of **DistributeEvidence** are replaced by **DistributeEvidenceOnHostTree**. Distribution beyond the host tree is saved.

The next improvement is **UpdateBelief3**: Multiple performance of **DistributeEvidenceOnHostTree** are replaced by **DistributeEvidenceOnChain**. Distribution to the entire host tree is reduced to a chain leading to the next host.

We indicate that the savings in computation time are obtained by increased sophistication in control mechanisms: Replacement of the brute force method (equivalent to a single linkage propagation) by **UpdateBelief** requires the coordination of multiple linkage propagation. Replacement of **UpdateBelief** by **UpdateBelief2** requires additional control: Distribution is to be terminated at the leaves of the host tree. Finally, replacement of **UpdateBelief2** by **UpdateBelief3** requires more specific control: Distribution must proceed along predetermined chains.

Acknowledgement

This work is supported by the Dean's Research Funding from Faculty of Science, University of Regina, the General NSERC Grant from University of Regina, and Research Grant OGP0155425 from NSERC.

References

[1] E. Charniak. Bayesian networks without tears. *AI Magazine*, 12(4):50–63, 1991.

[2] R. Davis and R.G. Smith. Negotiation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1):63–109, 1983.

[3] R.E. Neapolitan. *Probabilistic Reasoning in Expert Systems.* John Wiley and Sons, 1990.

[4] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, 1988.

[5] S. Srinivas. A probabilistic approach to hierarchical model-based diagnosis. In *Proc. Tenth Conf. Uncertainty in Artificial Intelligence*, pages 538–545, Seattle, Washington, 1994.

[6] Y. Xiang. Distributed multi-agent probabilistic reasoning with bayesian networks. In *To appear in Proc. Eighth International Symposium on Methodologies for Intelligent Systems*, Charlotte, North Carolina, Oct. 1994.

[7] Y. Xiang, B. Pant, A. Eisen, M. P. Beddoes, and D. Poole. Multiply sectioned bayesian networks for neuromuscular diagnosis. *Artificial Intelligence in Medicine*, 5:293–314, 1993.

[8] Y. Xiang, D. Poole, and M. P. Beddoes. Exploring locality in bayesian networks for large expert systems. In *Proc. Eighth Conference on Uncertainty in Artificial Intelligence*, pages 344–351, Stanford, CA, 1992.

[9] Y. Xiang, D. Poole, and M. P. Beddoes. Multiply sectioned bayesian networks and junction forests for large knowledge based systems. *Computational Intelligence*, 9(2):171–220, 1993.