Utilizing Knowledge-Base Semantics in Graph-Based Algorithms

Adnan Darwiche
Rockwell Science Center
1049 Camino Dos Rios
Thousand Oaks, Ca 91362
darwiche@risc.rockwell.com

Abstract

Graph-based algorithms convert a knowledge base with a graph structure into one with a tree structure (a join-tree) and then apply tree-inference on the result. Nodes in the join-tree are cliques of variables and tree-inference is exponential in w^*, the size of the maximal clique in the join-tree. A central property of join-trees that validates tree-inference is the running-intersection property: the intersection of any two cliques must belong to every clique on the path between them. We present two key results in connection to graph-based algorithms. First, we show that the running-intersection property, although sufficient, is not necessary for validating tree-inference. We present a weaker property for this purpose, called running-interaction, that depends on non-structural (semantical) properties of a knowledge base. We also present a linear algorithm that may reduce w^* of a join-tree, possibly destroying its running-intersection property, while maintaining its running-interaction property and, hence, its validity for tree-inference. Second, we develop a simple algorithm for generating trees satisfying the running-interaction property. The algorithm bypasses triangulation (the standard technique for constructing join-trees) and does not construct a join-tree first. We show that the proposed algorithm may in some cases generate trees that are more efficient than those generated by modifying a join-tree.

Introduction

Graph-based algorithms have become the standard approach for reasoning in a number of fields, including constraint satisfaction and probabilistic reasoning (Dechter 1992; Mackworth & Freuder 1985; Jensen, Lauritzen, & Olesen 1990; Pearl 1988), and have been introduced to other fields such as truth maintenance and diagnosis (Geffner & Pearl 1987; Dechter & Dechter 1994; Darwiche 1995; Darwiche & Pearl 1994). In graph-based algorithms, a knowledge base has two parts: (a) pieces of information about domain atoms/variables, such as conditional probabilities, clauses, constraints, component descriptions, and operators; and (b) a graphical depiction of the structure of these pieces of information, that is, the way they relate variables. The graphical structure may be provided by the user (as in probabilistic networks) or constructed by the reasoner (as in constraint satisfaction).

In either case, state-of-the-art graph-based algorithms use the following technique (Jensen, Lauritzen, & Olesen 1990; Pearl 1988; Dechter 1992; Dechter & Pearl 1989; Jensen & Jensen 1994). First, they convert a knowledge base with a graph structure into an equivalent knowledge base with a tree structure. Second, they perform linear inference on the tree structure. The tree structure is typically called a join-tree. Each of its nodes represents a set of variables (a cluster) in the original graph. The size of the maximal cluster in the join-tree is called the *tree-width* and is denoted by w^*. Inference on a join-tree is known to be exponential in w^*.

Inference on a join-tree is simple conceptually. When looked at procedurally, each cluster in the tree performs a local inference involving the set of variables it contains, and the results of these local inferences are combined to reach a result for the global inference involving all variables. The combination of local inferences is typically implemented using a message passing mechanism where messages are exchanged between clusters to communicate local results. The difference between various versions of graph-based algorithms seems to be (a) the type of local inferences performed at each cluster (examples are deciding entailment, computing partial diagnoses, and solving constraints); and (b) the operators used to combine local results into global ones (examples are addition and multiplication in probability, conjoin and disjoin in deciding entailment). For simplicity of exposition, we shall focus in this paper on graph-based algorithms for deciding entailment. The results will extend easily to other logic-based reasoning tasks.

A key property of join-trees that validates tree-inference — in particular, the ability to compose a global result by combining local results — is the running-intersection property: The intersection of any two clusters must be contained in all clusters on the path between them. This property is characteristic of join-trees and delimits the space of trees that are used by graph-based algorithms. Our key result in this paper is that this property is too strong in certain cases, if one is allowed to examine the logical content (semantics) of a knowledge base in addition to its structure. We also provide a number of companion results that capitalize on this observation for improving the complexity of graph-based inference.

This paper is structured as follows. First, we briefly introduce graph-based reasoning for propositional entailment and explain why the running-intersection property is sufficient to validate tree-inference. Second, we show that although the running-intersection property is sufficient for this validation, it is not necessary. We state a weaker property, called running-interaction, that is sufficient for this validity. We also present a simple algorithm that modifies a join-tree for the purpose of reducing its w^*, possibly destroying its running-intersection property, while maintaining its running-interaction property. Third, we define the class of I-trees which satisfy the running-interaction property and show that some I-trees can never be generated by modifying a join-tree. We then present an algorithm for generating I-trees that does not require triangulation, which is the standard technique for constructing join-trees. We finally close with some concluding remarks on the presented results.

Inference using a Join-Tree

The inference problem we focus on in this paper is propositional entailment. A graph-based algorithm for entailment takes the following input: (1) a structured set of propositional sentences (structured database) and (2) a clause. The algorithm decides whether the database entails the clause. Figure 1(a) depicts a structured database, which has two parts: a directed graph $\mathcal{G}$ with atomic propositions as its nodes, and a set of propositional sentences Δ that are distributed among the nodes of $\mathcal{G}$. The sentences associated with a node can only mention that node and its parents in the graph $\mathcal{G}$.[1]

State-of-the-art graph-based algorithms convert a structured database into a join-tree. Figure 1(b) contains an example join-tree for the structured database in Figure 1(a). The details of constructing a join-tree are not important in this section. What is important, however, are the following properties of a join-tree:

1. Each node in a join-tree represents a cluster of atomic propositions;
2. The sentences in a structured database (Figure 1(a)) are distributed among the clusters of a join-tree (Figure 1(b)) so that atoms appearing in a sentence also appear in the cluster it is associated with.

We use Ψ_i to denote the sentences associated with cluster $\mathcal{C}_i$ and refer to Ψ_i as the *local database* of $\mathcal{C}_i$.

[1]The graph $\mathcal{G}$ need not be directed in general.

An entailment test of the form $\Psi_i \models \alpha$, where α is a clause that mentions only atoms in cluster $\mathcal{C}_i$, is called a *local test*. If Δ is the union of all local databases in a join-tree, then an entailment test wrt Δ can be composed from local entailment tests wrt databases Ψ_i. In fact, tree-inference, the name we use to refer to the join-tree algorithm, can be viewed as a recipe for this composition.

Before we present tree-inference, show why it works, and why it could still work under weaker conditions, we need a couple of theorems about logical entailment. These theorems are called *decomposition* and *case analysis* and they represent the basis of tree-inference.

Theorem 1 (Decomposition) *Let Δ and Γ be two sets of propositional sentences (databases) and let S be all atomic propositions that are common to both Δ and Γ. Let α be a clause over S, that is, α contains one literal for each atom in S. Then:*

$$\Delta \cup \Gamma \models \alpha \vee \beta \quad \textit{iff} \quad \Delta \models \alpha \vee \beta(\Delta) \textit{ or } \Gamma \models \alpha \vee \beta(\Gamma).$$

Here, β is a clause, $\beta(\Delta)$ is the subset clause of β with atoms occurring in Δ, and $\beta(\Gamma)$ is the subset clause of β with atoms occurring in Γ.

Since $\Delta \cup \Gamma \models \alpha \vee \beta$ is equivalent to $\Delta \cup \Gamma \cup \{\neg\alpha\} \models \beta$, the theorem is intuitively saying that we can decompose an entailment test involving two databases Δ and Γ into two simpler tests, one involving Δ and the other involving Γ, as long as the truth of atoms that are common to Δ and Γ is fixed (by $\neg\alpha$). The Decomposition theorem is most valuable when used with case analysis:

Theorem 2 (Case-Analysis) *Let Δ be a database and let S be a set of atomic propositions. If $\alpha_1, \ldots, \alpha_n$ are all possible clauses over S, then*

$$\Delta \models \beta \quad \textit{iff} \quad \Delta \models \alpha_1 \vee \beta \textit{ and } \ldots \textit{ and } \Delta \models \alpha_n \vee \beta.$$

Case analysis is typically used to set the stage for using Decomposition. That is, if we want to decompose a test $\Delta \cup \Gamma \models \gamma$ and the clause γ does not mention all atoms that are common to Δ and Γ, then we can perform a case analysis on these atoms:

Corollary 1 (Intersection) *Let Δ and Γ be two databases and let S be all atomic propositions that are common to both Δ and Γ. If $\alpha_1, \ldots, \alpha_n$ are all possible clauses over S, then $\Delta \cup \Gamma \models \beta$ iff*

$$\begin{aligned} &\Delta \models \alpha_1 \vee \beta(\Delta) \textit{ or } \Gamma \models \alpha_1 \vee \beta(\Gamma) \textit{ and } \ldots \textit{ and} \\ &\Delta \models \alpha_n \vee \beta(\Delta) \textit{ or } \Gamma \models \alpha_n \vee \beta(\Gamma). \end{aligned}$$

This corollary is the basis of tree-inference, which is stated next. First, the following notation. For a join-tree $\mathcal{T}$ and arc $\mathcal{C}_i$–$\mathcal{C}_j$, we use $\mathcal{T}_{ij}$ to denote the subtree that continues to include cluster $\mathcal{C}_i$ after deleting arc $\mathcal{C}_i$–$\mathcal{C}_j$. For example, in Figure 1(b), the subtree $\mathcal{T}_{12}$ contains one cluster $\mathcal{C}_1$, while the subtree $\mathcal{T}_{21}$ contains two clusters $\mathcal{C}_2$ and $\mathcal{C}_3$. We also Ψ_{ij} to denote all sentences that appear in the subtree $\mathcal{T}_{ij}$. For example,

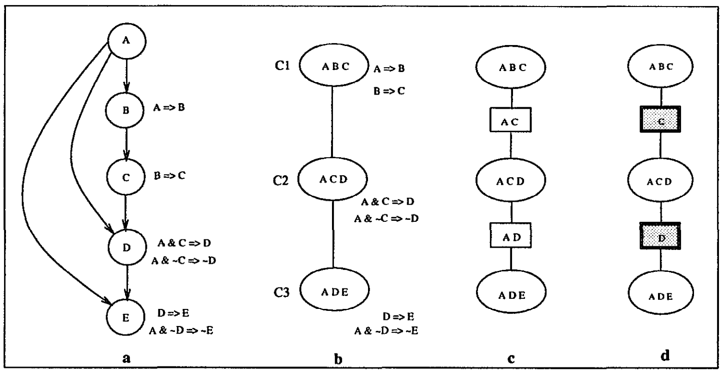


Figure 1: (a) A structured database: a set of propositional sentences that are structured using a directed graph of propositional atoms; (b) a join-tree; (c) the sepsets of the join-tree — intersections of adjacent clusters; (d) reducing sepsets in a join-tree — we cannot remove atom A from any of the clusters, because it appears in the sentences associated with the clusters.

in Figure 1(b), $\Psi_{12} = \{A \supset B, B \supset C\}$, while $\Psi_{21} = \{A \wedge C \supset D, A \wedge \neg C \supset \neg D, D \supset E, A \wedge \neg D \supset \neg E\}$. Theorem 3 below shows how the result of a global entailment test (wrt a join-tree) can be composed from local entailment tests (wrt its clusters).

Theorem 3 (Tree-Inference) *Let $\mathcal{T}$ be a join-tree, Δ be the set of sentences associated with clusters in $\mathcal{T}$, and let β be a clause. The entailment test $\Delta \models \beta$ is equivalent to $Test[\mathcal{T}, \beta]$, where $Test[\mathcal{T}, \beta]$ iff $\Psi_i \models \beta$ when $\mathcal{T}$ contains only one cluster $\mathcal{C}_i$; and $Test[\mathcal{T}, \beta]$ iff*

$$AND_{\alpha} \quad Test[\mathcal{T}_{ij}, \alpha \vee \beta_{ij}] \quad OR \quad Test[\mathcal{T}_{ji}, \alpha \vee \beta_{ji}],$$

otherwise. Here, α ranges over all clauses of atoms $\mathcal{C}_i \cap \mathcal{C}_j$ for some adjacent clusters $\mathcal{C}_i$ and $\mathcal{C}_j$ in $\mathcal{T}$; and β_{ij} is the subset clause of β with atoms occurring in subtree $\mathcal{T}_{ij}$.

Algorithms for deciding entailment using a join-tree can be viewed as implementations of the recursion in Theorem 3. When looked at procedurally, the theorem suggests that we can decompose an entailment test wrt to a join-tree $\mathcal{T}$ into a set of entailment tests wrt subtrees $\mathcal{T}_{ij}$ and $\mathcal{T}_{ji}$ by performing a case analysis on the set of atoms $\mathcal{C}_i \cap \mathcal{C}_j$. This set is denoted by $\mathcal{S}_{ij}$ and is known as the *sepset* of arc $\mathcal{C}_i$–$\mathcal{C}_j$. This process can then be applied recursively on each of the resulting subtrees, until we reach boundary conditions. A boundary condition is a join-tree that has a single cluster. An entailment test wrt such a tree is local by definition.

Clearly, this process will terminate since the number of clusters in each subtree is getting smaller each time we decompose across an arc. The recursion, however, does not provide a control strategy for choosing an arc to decompose a join-tree across. Moreover, many of the recursive calls will be redundant. Implementations of join-tree inference take care of this by (a) implementing the recursion bottom-up starting from clusters with a single neighbor and (b) keeping tables with clusters and sepsets to record intermediate results. These details, however, are not relevant to our current discussion. Suffice it to say that these implementations are linear in the size of the constructed tables, and the tables are exponential in the size of their corresponding clusters and sepsets.[2]

The Interaction Theorem

Tree-inference is a simple, recursive application of the Intersection Corollary. The specific connection is as follows. In this corollary: set S to the sepset $\mathcal{S}_{ij}$ of some arc $\mathcal{C}_i$–$\mathcal{C}_j$ in the join-tree $\mathcal{T}$; set Δ to the database Ψ_{ij}; and set Γ to the database Ψ_{ji}. If we make these settings, we can use the Intersection Corollary to decompose a computation wrt a join-tree $\mathcal{T}$ into a set of computations wrt the subtrees $\mathcal{T}_{ij}$ and $\mathcal{T}_{ji}$. For this decomposition to be valid, however, S must contain all atoms that are common to both Δ and Γ. This is exactly what the running-intersection property of tree $\mathcal{T}$ guarantees.

Specifically, the running-intersection property guarantees that the sepset $\mathcal{S}_{ij}$ contains all atoms that are common to databases Ψ_{ij} and Ψ_{ji}. For example, in Figure 1(b), the atoms in Ψ_{12} are $\{A, B, C\}$ and the atoms in Ψ_{21} are $\{A, C, D, E\}$. The intersection between the two sets is $\{A, C\}$, which is also the sepset $\mathcal{S}_{12}$ shown in Figure 1(c).

In other words, tree-inference is valid because the join-tree satisfies the running-intersection property, which in turn guarantees the condition of the Intersection Corollary (the Decomposition Theorem, to be more specific). As we show next, the condition of this corollary (Theorem) is not necessary for validating the corollary (Theorem). This means that the running-

[2]Applying the recursion without caching results in tables will *not* lead to this linear complexity.

intersection property is also not necessary for validating tree-inference.

Definition 1 *An atom is positive (negative) in a database Δ if all its literals in the clausal form of Δ are positive (negative).*

Theorem 4 (Interaction) *Let Δ and Γ be two databases and let S be all atomic propositions p such that p is positive in one of Δ or Γ and negative in the other. If $\alpha_1, \ldots, \alpha_n$ are all possible clauses over S, then $\Delta \cup \Gamma \models \beta$ iff*

$$[\Delta \models \alpha_1 \vee \beta(\Delta) \textit{ or } \Gamma \models \alpha_1 \vee \beta(\Gamma)] \textit{ and } \ldots \textit{ and }$$
$$[\Delta \models \alpha_n \vee \beta(\Delta) \textit{ or } \Gamma \models \alpha_n \vee \beta(\Gamma)].$$

Let us consider an example. Let $\Delta =_{def} \{A \supset B\}$ $\Gamma =_{def} \{B \supset C\}$, $\alpha =_{def} A \supset C$. Then $\Delta \cup \Gamma \models \alpha$ cannot be decomposed into $\Delta \models \alpha$ or $\Gamma \models \alpha$ simply because α does not follow from either Δ or Γ alone, yet it follows from their combination. Now let us define Δ differently, $\Delta =_{def} \{A \supset \neg B\}$. Then the decomposition is valid for $\alpha =_{def} A \supset C$. That is, although atom B is common to databases Δ and Γ, and although atom B is not mentioned in clause α, the decomposition is valid because atom B has the same (negative) sign in both databases. In general, in the Intersection Corollary (and Decomposition Theorem), if some atom is only positive or only negative in both Δ and Γ, the atom can be excluded from S while still maintaining the validity of the corollary.

The implications of this on tree-inference is as follows. If an atom appears in the sepset $\mathcal{S}_{ij}$ and has the same sign in both databases Ψ_{ij} and Ψ_{ji}, then it can be removed from the sepset without compromising the validity of tree-inference. Consider Figure 1 for an example. Atom A appears in the sepset $\mathcal{S}_{12}$. However, atom A has the same sign in databases Ψ_{12} and Ψ_{21}. Therefore, the atom can be removed from the sepset while preserving the validity of tree-inference. Similarly, A appears in sepset $\mathcal{S}_{23}$ and has the same sign in both databases Ψ_{23} and Ψ_{32}. Therefore, it can also be removed from this sepset.

Therefore, if we have access to the local databases associated with clusters in a join-tree, we can possibly eliminate some of the atoms appearing in sepsets, which reduces the complexity of tree-inference, while still preserving its validity. This is not contradictory, however, with complexity results of graph-based algorithms which say that these algorithms are exponential in w^*. These complexity results hold for all possible local databases that may be associated with clusters, therefore depending only on structural properties of a database. What the Interaction Theorem is saying, however, is that we can improve on the w^* bound if we are allowed to use the non-structural (semantical) properties of a database in addition to its structural ones. This improvement is implemented by the following algorithm:

Algorithm 1 (Modify-Join-Tree) *Perform the following steps in order before applying tree-inference to a join-tree:*

1. *For each arc $\mathcal{C}_i$-$\mathcal{C}_j$ in the join-tree, and each atom A in the sepset $\mathcal{S}_{ij}$, eliminate A from the sepset if A has the same sign in both Ψ_{ij} and Ψ_{ji}.*
2. *For each cluster $\mathcal{C}_i$ in the join-tree, and each atom A in the cluster, eliminate A from the cluster if it does not appear in the local database Ψ_i and does not belong to any of the sepsets $\mathcal{S}_{ij}$ for all j.*

This procedure is clearly polynomial in all important parameters; atoms, clauses, and cluster sizes. Moreover, it may lead to reducing w^* of the given join-tree. When applying this procedure to the join-tree in Figure 1(c), we obtain the one in Figure 1(d). Note that although both sepsets were reduced in size, the clusters were not affected. This should not be surprising, however, since a cluster must include, at least, all atoms that appear in its associated sentences. The three clusters in Figure 1 include exactly those atoms.

The tree resulting from Algorithm 1 is said to satisfy the *running-interaction property* because for each arc $\mathcal{C}_i$-$\mathcal{C}_j$ in the tree, the reduced sepset of the arc summarizes the interaction between the sentences in databases Ψ_{ij} and Ψ_{ji}. It should be clear, given the Interaction Theorem, that a tree satisfying this property will produce valid results under tree-inference. We stress, however, that although the running-interaction property is sufficient to validate tree-inference, it may also not be necessary. Therefore, it is possible that one can weaken this property even further while retaining the validity of tree-inference. But we do not pursue this issue in the paper.

I-Trees

In light of the Interaction Theorem, the running-intersection property and the notion of a join-tree are too strong. The goal of this section is to define a new class of trees that is characterized by the weaker running-interaction property. This class of trees is useful for developing algorithms that transform a graph-based database into a tree-based one.

Definition 2 (I-Tree) *Let Δ be a set of propositional sentences. An I-tree for Δ is a tree $\mathcal{T}$ of clusters satisfying the following properties:*

- *associated with each cluster $\mathcal{C}_i$ in $\mathcal{T}$ is a local database Ψ_i where $\Psi_i \subseteq \Delta$ and $\Delta = \bigcup_j \Psi_j$;*
- *associated with each arc $\mathcal{C}_i$-$\mathcal{C}_j$ in $\mathcal{T}$ is a set $\mathcal{I}_{ij}$ (called an inset) containing all atoms that appear with different signs in Ψ_{ij} and Ψ_{ji};*
- *the atoms in a cluster $\mathcal{C}_i$ are those appearing in its local database Ψ_i and insets $\mathcal{I}_{ij}$.*

It is important to note the following difference between I-trees and join-trees. First, local databases are part of the definition of an I-trees. Second, we cannot compute insets of an I-tree from the clusters although we can compute sepsets from clusters in a join-tree.

There are two key reasons for defining the class of I-Trees explicitly:

1. Some I-trees can never be generated by modifying a join-tree as suggested by Algorithm 1.
2. There is a simple algorithm for generating I-trees that does not use triangulation, which is the standard technique for computing join-trees.

Join-trees are constructed through a set of graph operations, which include moralization, triangulation, clique identification, and clique ordering. These steps are meant to guarantee the running-intersection property. Since we may destroy this property later, one wonders whether it is necessary to guarantee it in the first place. In fact, the I-tree definition suggests that one may be able to construct I-trees directly, without having to construct join-trees first. This is indeed true as we show next.

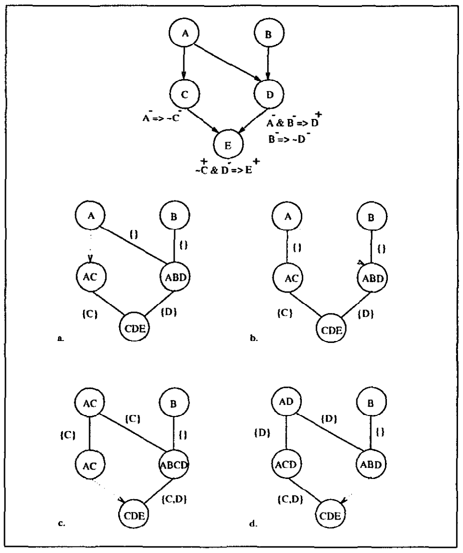
Without loss of generality, we shall assume that our database is structured around a directed graph as shown in Figure 1(a). That is, we have a directed graph $\mathcal{G}$ where each of its nodes is an atomic proposition; we have a set of propositional sentences attached to each node; and these sentences mention only that node and its parents in $\mathcal{G}$. We now show how to construct an I-tree for such a database:

Algorithm 2 (I-Tree-Construction) *To construct an I-tree for a structured database with graph $\mathcal{G}$:*

1. *eliminate enough arcs from the graph $\mathcal{G}$ to transform it into a connected tree $\mathcal{T}$;*
2. *For each node N_i in tree $\mathcal{T}$, define Ψ_i as the set of sentences associated with N_i in graph $\mathcal{G}$;*
3. *For each arc N_i–N_j in tree $\mathcal{T}$, define the inset $\mathcal{I}_{ij}$ as the set of atoms having different signs in databases Ψ_{ij} and Ψ_{ji};*
4. *Convert each node N_i in tree $\mathcal{T}$ into a cluster C_i containing the atoms appearing in database Ψ_i and insets $\mathcal{I}_{ij}$ for all j;*
5. *Eliminate clusters that are contained by their neighbors (transferring their associated sentences to the containing neighbor).*

Note that arc elimination is not deterministic. Depending on which arcs are eliminated, a better or worse I-tree can be generated. Figure 2 depicts four possible I-trees for the same structured database. The I-trees in Figures (a) and (b) are more efficient than those in Figures (c) and (d), assuming that efficiency is measured by the total size of tables that an implementation will construct.

We now show a structured database with the following properties. First, a triangulation-based method would generate a unique join-tree for this database, with $w^* = 4$. Second, when reducing the size of separators and clusters of the resulting join-tree, as suggested by Algorithm 1, only the sepset sizes are affected. The size of clusters remains the same. However, when constructing an I-tree using Algorithm 2, we obtain a tree with $w^* = 3$. The structured database is shown in Figure 4(a). Its join-tree together with the reduction is shown in Figure 3. The constructed I-tree is shown in Figure 4. Note, however, that the tables constructed by an implementation for the I-tree in Figure 3 have a slightly smaller size than those constructed for the I-tree in Figure 4.


A
B
C
D
E
$A^- \Rightarrow \neg C^-$
$A^- \& B^- \Rightarrow D^+$
$B^- \Rightarrow \neg D^-$
$\neg C^+ \& D^- \Rightarrow E^+$
AC
ABD
CDE
{}
{C}
{D}
ABCD
AD
ACD
{C,D}
a.
b.
c.
d.
Figure 2: Generating I-trees. Clusters that are contained by their neighbors have not been eliminated. The sign of each atom is also shown, which is useful for computing arc labels (insets).

One reason why some I-trees cannot be generated by modifying join-trees using Algorithm 1 is that the standard method for constructing join-trees does not generate all trees that satisfy the running-intersection property to start with. The standard method is based on the following steps:

1. moralize a graph by (a) connecting every pair of parents for each node and (b) dropping the directionality of arcs;
2. triangulate the graph by adding enough arcs so that every cycle of length four or more has a chord;
3. identify cliques of the resulting arc; and
4. connect the identified cliques into a tree satisfying the running intersection property.

Figure 5(a) depicts a graph that has a unique join-tree, when using this standard method, which is shown in Figure 5(b). A standard implementation of tree-inference with respect to this join-tree would require four tables to be constructed: one table of size 2^{n+1} for each clique and two tables of size 2^n for the sepset. The

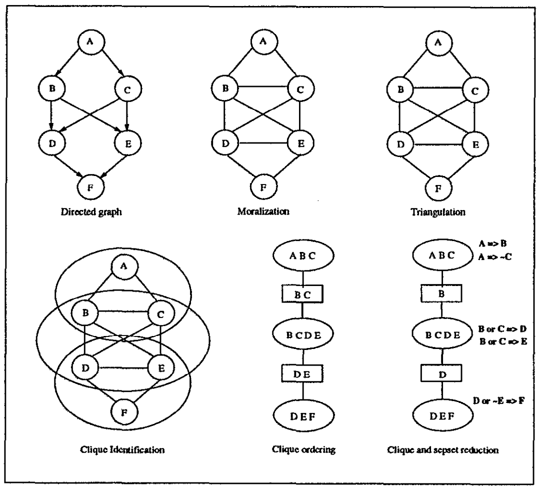


Figure 3: A unique join-tree construction using the standard method.

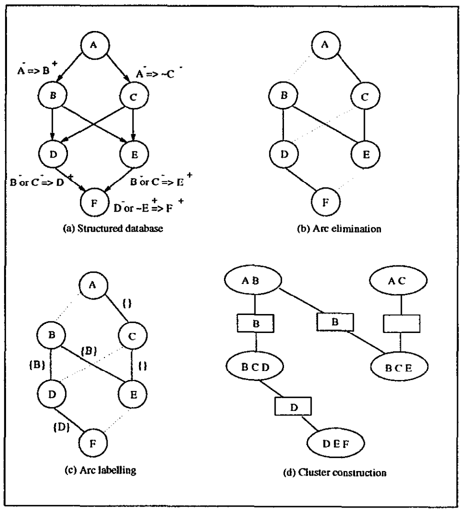


Figure 4: Constructing an I-tree using Algorithm 2.

total size is then $2^{n+2} + 2^{n+1}$. However, Figure 5(c) shows another join-tree that is constructed by a procedure we shall give next. The tree consists of one cluster and, trivially, satisfies the running-intersection property. An implementation would construct only one table of size 2^{n+2} with respect to this tree.

The difference between the two methods shows up more dramatically when using them to construct an I-tree. If we apply Algorithm 1 to the join-tree in Figure 5(b), we obtain the same join-tree; no reduction takes place. However, if we apply Algorithm 2 to the the structured database in Figure 5(a), we obtain the the I-tree in Figure 5(d). An implementation would construct one table of size 2^{n+1} for cluster $\{B_1, \ldots, B_n, C\}$, one table of size 4 for each cluster $\{A, B_i\}$; and two tables of size 2 for each sepset $\{B_1\}$, therefore, totaling $2^{n+1} + 8n$. This is better than $2^{n+2} + 2^{n+1}$ which is the total size when using Algorithm 1 on the join-tree in Figure 5(b).

Algorithm 2 can be modified slightly to construct join-trees:

Algorithm 3 (Join-Tree-Construction) *To construct a join-tree for graph $\mathcal{G}$:*

1. *eliminate enough arcs from the graph $\mathcal{G}$ to transform it into a connected tree $\mathcal{T}$;*
2. *for each arc N_i-N_j in tree $\mathcal{T}$, define sepset $\mathcal{S}_{ij}$ as the intersection of (a) atoms in $\mathcal{T}_{ij}$ and their parents in $\mathcal{G}$ and (b) atoms in $\mathcal{T}_{ji}$ and their parents in $\mathcal{G}$;*
3. *convert each node N_i in $\mathcal{T}$ into a cluster C_i that includes N_i, its parents in $\mathcal{G}$, and the atoms in sepset $\mathcal{S}_{ij}$ for all j;*
4. *eliminate the clusters contained by their neighbors.*

An example of this procedure is shown in Figure 5(c) and another is shown in Figure 6.

What is interesting about join-trees is that although running-intersection is the property characterizes them, they are typically not defined using this property but by the way they are typically constructed: moralize, triangulate, etc. This is probably due to the dominance of this standard method of construction. But other methods exist for generating join-trees — see (Draper 1995) for example — which are based on applying transformations to a graph of clusters in order to generate a tree of clusters that satisfies the running-intersection property. Six transformation are defined for this purpose in (Draper 1995) and some constraints are formulated so algorithms using these transformations can terminate. Algorithm 3 can be viewed as applying transformations to the original directed graph, and is closely related to the Collapse transformation proposed in (Draper 1995).

Concluding Remarks

This paper makes several contributions to graph-based algorithms. Most importantly is the observation that knowledge base content (its non-structural properties) can be useful in reducing the complexity of graph-based algorithms. In fact, we argue that knowledge base structure alone, although very useful, cannot lead to completely satisfactory results. A Horn database, for example, may have a very complicated structure, yet it is computationally well behaved. Similarly, however, looking at knowledge base content may not suffice in certain cases: a database may not be Horn and yet have a tree structure that permits linear-time inference. Therefore, neither structure, nor content are enough on their own and we need methods that utilize both. We have presented one such method in this paper.

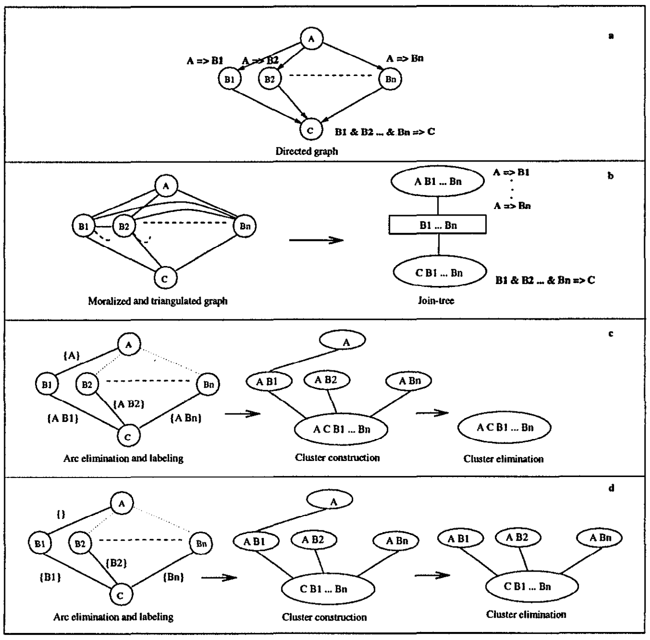


Figure 5: Constructing a join-tree using both the standard method and Algorithm 3: (a) a structured database; (b) a unique join-tree constructed using the standard method; (c) a join-tree constructed using Algorithm 3; (d) an I-tree constructed using Algorithm 2.

We have also presented a number of results that help us in gaining more insights into graph-based algorithms for logical reasoning, such as the Intersection Corollary and Interaction Theorem. Moreover, our proposed method for generating join-trees is also a contribution that seems to stand on its own, independently of the main theme of this paper. We did not, however, provide an analysis of the relative merits of this new method as compared to the standard one based on triangulation. But this is beyond the scope of this paper.

We close by observing that although we have restricted our discussion to propositional entailment, the results we have presented can be extended easily to a number of logic-based inferences such as computing diagnoses using graph-based methods (Geffner & Pearl 1987; Dechter & Dechter 1994; Darwiche 1995; Darwiche & Pearl 1994).

Acknowledgement

The work reported in this paper was partially supported by ARPA contract F30602-95-C-0251.

References

Darwiche, A., and Pearl, J. 1994. Symbolic causal networks for reasoning about actions and plans. Working notes: AAAI Spring Symposium on Decision–Theoretic Planning.

Darwiche, A. 1995. Model-based diagnosis using causal networks. In *Proceedings of International Joint Conference on Artifical Intelligence (IJCAI)*, 211–217.

Dechter, R., and Dechter, A. 1994. Structure-driven algorithms for truth maintenance. *Artificial Intelligence.* To appear.

Dechter, R., and Pearl, J. 1989. Tree clustering for constraint networks. *Artificial Intelligence* 353–366.

Dechter, R. 1992. Constraint networks. *Encyclopedia of Artificial Intelligence* 276–285. S. Shapiro, editor.

Draper, D. L. 1995. Clustering without (thinking about) triangulation. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence (UAI)*, 125–133.

Geffner, H., and Pearl, J. 1987. An improved constraint-propagation algorithm for diagnosis. In *Proceedings of IJCAI*, 1105–1111.

Jensen, F. V., and Jensen, F. 1994. Optimal Junction Trees. In *Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence*, 360–366.

Jensen, F. V.; Lauritzen, S.; and Olesen, K. 1990. Bayesian updating in recursive graphical models by local computation. *Computational Statistics Quarterly* 4:269–282.

Mackworth, A. K., and Freuder, E. C. 1985. The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence* 25(1).

Pearl, J. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann Publishers, Inc., San Mateo, California.

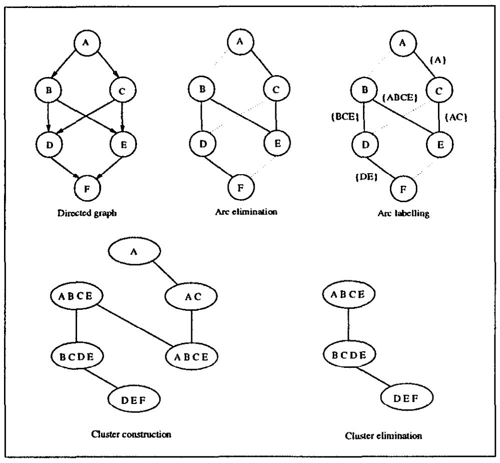


Figure 6: Constructing a join-tree using Algorithm 3.