NORTH-HOLLAND

Bounded Recursive Decomposition: A Search-Based Method for Belief-Network Inference Under Limited Resources

Stefano Monti and Gregory F. Cooper

Intelligent Systems Program and Section of Medical Informatics, University of Pittsburgh, Pennsylvania

ABSTRACT

This paper presents a new inference algorithm for belief networks that combines a search-based algorithm with a simulation-based algorithm. The former is an extension of the recursive decomposition (RD) algorithm proposed by Cooper, which is here modified to compute interval bounds on marginal probabilities. We call the algorithm bounded-RD. The latter is a stochastic simulation method known as Pearl's Markov blanket algorithm. Markov simulation is used to generate highly probable instantiations of the network nodes to be used by bounded-RD in the computation of probability bounds. Bounded-RD has the anytime property, and produces successively narrower interval bounds, which converge in the limit to the exact value.

KEYWORDS: ***Bayesian belief networks, belief updating, incremental bounding algorithms, simulation***

1. INTRODUCTION

Belief networks [3] and influence diagrams [4] are powerful formalisms for modeling and reasoning with uncertainty. They have been extensively used in a large variety of applications ranging from medical diagnosis to fault diagnosis in

Address correspondence to Stefano Monti, Intelligent Systems Program, University of Pittsburgh, B50A Lothrop Hall, 190 Lothrop St., Pittsburgh, PA 15261. E-mail: smonti@isp.pitt.edu.

Received October 1995; accepted December 1995

S0888-613X/96/$15.00
PII S0888-613X(96)00012-0

complex machinery [5–10], stimulating the development of efficient techniques for probabilistic inference in belief networks.

Several algorithms have been devised for probabilistic inference with belief networks, and when applied to specific network topologies, these algorithms can be computationally tractable [11, 12]. Among the tractable methods are those based on assumptions about the nature of the probability distribution to be modeled, such as noisy-OR gates [3], similarity networks [13, 14], and the additive belief-network model [15]. However, in its general formulation, the problem of inference in belief networks, both exact and approximate, is NP-hard [16, 17].

The worst-case intractability, while not invalidating the research on exact techniques for minimizing inference time, makes the research on approximate inference methods particularly important. Besides the class of exact algorithms [1, 12, 18–20], there are two main classes of approximate methods[1]: (1) stochastic simulation methods, which compute an estimate of the exact probabilities by sampling the space of possible instantiations of the network [2, 22, 23], and (2) search-based approximate methods, which search for the most probable instantiations in this space [24–26]. Most of the algorithms belonging to the last class can also be characterized as incremental bounding algorithms, since they compute successively narrower upper and lower bounds on the probability of interest [27–29]. The rate at which these bounds are narrowed is generally contingent upon the existence of a relatively small number of probable instantiations that cover a large portion of the probability space, which in turn depends on the asymmetry of the individual prior and conditional probability distribution in the network [30].

The main motivation for our work stems from the consideration that simulation-based algorithms and search-based algorithms reflect two approaches to inference that are complementary. On the one hand, simulation-based methods yield a point-value estimate of the probability of interest. On the other hand, search-based methods provide us with correct interval bounds on the exact probability, and the width of the interval is a clear gauge of how useful these bounds can be to the reasoning process. When facing a decision problem, a probability interval may be sufficient to select the optimal decision. If not, the estimate may be used to make the decision in a time-critical situation. Assuming we have access to interval bounds on the probability of interest, a first and immediate way of controlling the accuracy of a point-value estimation is to test whether it falls within the given bounds. At the same time, assuming the estimation does fall within the computed bound, it represents the best first guess on the probability of interest.

In this paper, we present a new inference algorithm for belief networks that combines a search-based algorithm with a simulation-based algorithm. The former is an extension of the recursive decomposition (RD) algorithm proposed by Cooper

[1] An alternative and less explored approach, which we do not consider in this paper, is based on the manipulation of the network structure, by removing selected edges so as to reduce the connectivity of the network [21].

in [1], which is here modified to compute interval bounds on marginal probabilities. We call the algorithm *bounded-RD*. The latter is a stochastic simulation method, usually referred to as *straight simulation*, which is also known as Pearl's Markov blanket algorithm [2]. Markov simulation is used to generate highly probable instantiations of the network nodes to be used by bounded-RD in the computation of probability bounds. Bounded-RD has the anytime property, and produces successively narrower interval bounds, which converge in the limit to the exact value.

The remainder of this paper is organized as follows: In Section 2 we briefly review the belief-network formalism, and illustrate the recursive decomposition algorithm, which is described in more detail in [1]. In Section 3, we describe bounded-RD, and explain how the algorithm makes use of stochastic simulation to try to maximize the tightness of the interval bounds to be computed. In Section 4 we present the results of some experiments we conducted with a test-bed implementation of the algorithm, and in Section 5 we briefly discuss some relevant related work. Finally, in Section 6 we conclude the paper with a short summary and some suggestions for further work.

2. BACKGROUND

2.1. Basic Concepts and Notation

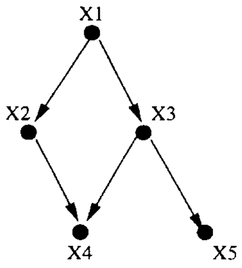
A belief network is defined by a triple (G, Ω, P), where $G = (\mathcal{X}, E)$ is a directed acyclic graph with a set of nodes $\mathcal{X} = \{x_1, \ldots, x_n\}$ representing domain variables,[2] and with a set of arcs E representing dependencies among domain variables; Ω is the space of possible instantiations of the domain variables[3]; and P is a probability distribution over the instantiations in Ω. Given a node $x \in \mathcal{X}$, we use π_x to denote the set of parents of x in $\mathcal{X}$. The family θ_x of a node x is defined as the set $\{x\} \cup \pi_x$. In general, the family of a set $X \subseteq \mathcal{X}$, denoted by θ_X, is the union of the families of the nodes in X, i.e., $\theta_X = \bigcup_{x \in X} \theta_x$.

In Figure 1, we give an example of a simple network structure, derived from [31], which we use throughout the paper to illustrate basic concepts. By "reading" the network structure, and by giving a causal interpretation to the links displayed, we see that *metastatic cancer* (x_1) is a cause of *brain tumor* (x_3), and that it can also cause an increase in *total serum calcium* (x_2). Furthermore, *brain tumor* can cause *papilledema* (x_5), and both *brain tumor* and an increase in *total serum calcium* can cause a patient to lapse into a *coma* (x_4).

The key feature of belief networks is their explicit representation of conditional independence among events (domain variables). In particular, each variable is

[2] In this paper, we make no distinction between the network nodes and the variables they represent.

[3] An instantiation ω of all n variables in $\mathcal{X}$ is an n-uple of values $\{X_1, \ldots, X_n\}$ such that $x_i = X_i$ for $i = 1, \ldots, n$.



X1:	*metastatic cancer*
X2:	*total serum calcium*
X3:	*brain tumor*
X4:	*coma*
X5:	*papilledema*

$$\begin{aligned}
P(x_1) &= 0.2\\
P(x_2 \mid x_1) &= 0.7\\
P(x_2 \mid \overline{x}_1) &= 0.1\\
P(x_3 \mid x_1) &= 0.6\\
P(x_3 \mid \overline{x}_1) &= 0.2\\
P(x_4 \mid x_2, x_3) &= 0.8\\
P(x_4 \mid x_2, \overline{x}_3) &= 0.3\\
P(x_4 \mid \overline{x}_2, x_3) &= 0.4\\
P(x_4 \mid \overline{x}_2, \overline{x}_3) &= 0.1\\
P(x_5 \mid x_3) &= 0.4\\
P(x_5 \mid \overline{x}_3) &= 0.1
\end{aligned}$$

Figure 1. A simple belief network, with set of nodes $\mathcal{X} = \{x_1, x_2, x_3, x_4, x_5\}$, parent sets $\pi_{x_1} = \emptyset$, $\pi_{x_2} = \pi_{x_3} = \{x_1\}$, $\pi_{x_4} = \{x_2, x_3\}$, $\pi_{x_5} = \{x_3\}$, and families $\theta_{x_1} = \{x_1\}$, $\theta_{x_2} = \{x_1, x_2\}$, $\theta_{x_3} = \{x_1, x_3\}$, $\theta_{x_4} = \{x_2, x_3, x_4\}$, and $\theta_{x_5} = \{x_3, x_5\}$. All the nodes represent binary variables, taking values from the domain {*True, False*}. We use the notation $\overline{x}_i$ to denote (x_i = *False*). The probability tables give the values of $p(x_i \mid \pi_{x_i})$ only, since $p(\overline{x}_i \mid \pi_{x_i}) = 1 - p(x_i \mid \pi_{x_i})$.

independent of its nondescendants given its parents. This property is usually referred as the *Markov condition*, and it allows us to express the probability distribution P by means of probability tables associated with the domain variables. That is, each node x_i in $\mathcal{X}$ is augmented with a probability table containing the probabilities of the node's values conditioned on its parents (i.e., the table associated to the node x_i stores the probability distribution $P(x_i \mid \pi_{x_i})$, where π_{x_i} is empty if x_i is a node without parents). Figure 1 shows the probability table for each node in the network.

The probability of any instantiation in Ω can then be computed from the probabilities in the belief network. In fact, it can be shown [3, 32] that the joint probability of any particular instantiation of all n variables in a belief network can be calculated as follows:

$$P(X_1, \ldots, X_n) = \prod_{i=1}^{n} P\big(X_i \mid \pi_{x_i}\big). \tag{1}$$

The complete set of conditional independence assertions implied by a network structure can be determined by means of the concept of d-separation, a graphical characterization introduced by Pearl in [3]:

If A, B, and D are three disjoint subsets of nodes in the directed acyclic graph G, the set D is said to d-separate A from B, if for every path between a node in A and a node in B one of the following conditions holds: i) the path contains a node e with converging arrows and neither e nor any of its descendants belong to D; or ii) the path contains a node e that does not have converging arrows, and e belongs to D.

It can be proved that d-separation actually characterizes all and only the conditional independence assertions that follow from satisfying the Markov condition in a belief network [3]. The concept of d-separation is important because through the identification of small d-separators we can decompose the network into probabilistically independent subnetworks, and their reduced size makes them more manageable and easier to understand. In the next section we will see how the identification of proper d-separators is essential to the effective application of our inference algorithm.

The phrase *probabilistic inference using belief network* usually refers to the calculation of conditional probabilities of the form $P(H' \mid E')$, where H' and E' are instantiations of the subsets H and E of $\mathcal{X}$. The calculation of $P(H' \mid E')$ is also called a *query*. When the conditioning set E is empty, the problem reduces to the computation of the marginal probability $P(H')$. By applying Equation (1), $P(H')$ can be calculated as follows:

$$P(H') = \sum_{\mathcal{X}-H} P(X_1, \ldots, X_n) = \sum_{\mathcal{X}-H} \prod_{i=1}^{n} P(X_i \mid \pi_{x_i}), \tag{2}$$

where $\mathcal{X} - H$ denotes the set difference (i.e., $\mathcal{X} - H = \{x \mid x \in \mathcal{X},\ x \notin H\}$). When the conditioning set E is not empty, we can still apply Equation (2) to the calculation of $P(H' \mid E')$, since $P(H' \mid E') = P(H', E')/P(E')$, and $P(H', E')$ and $P(E')$ can both be computed by Equation (2). The calculation of Equation (2) by exhaustive enumeration can be performed for trivial networks only, since the number of instantiations to be enumerated is exponential in the number of nodes in the network.

2.2. Belief Network Inference by Recursive Decomposition

Belief network inference by recursive decomposition is a *divide-and-conquer* technique that performs the calculation of Equation (2) by recursively decomposing the network, and by mapping the resulting decomposition into a corresponding factorization of the summation in the equation. The decomposition is aimed at reducing the number of operations needed (multiplications and additions) by eliminating the redundancies inherent in Equation (2).

EXAMPLE 1 We introduce the technique with a simple example using the belief network of Figure 1. Suppose we wish to calculate $P(x_5 = T)$ [for brevity, $P(x_5')$]. The application of the brute-force approach of Equation (2) to compute $P(x_5')$ yields

$$P(x_5') = \sum_{x_1,\ldots,x_4} P(x_5' \mid x_3) P(x_4 \mid x_3, x_2) P(x_3 \mid x_1) P(x_2 \mid x_1) P(x_1). \tag{3}$$

Note that variable x_3 is a d-separator for the network, because according to the Markov condition, given the value of x_3, the variable x_5 is probabilistically independent of the variables x_1, x_2, x_3, x_4. We could thus select x_3 as a separator variable, to perform the same calculation as in Equation (3) by decomposing the summation as follows:

$$P(x_5') = \sum_{x_3} \left(P(x_5' \mid x_3) \sum_{x_1,x_2,x_4} P(x_4 \mid x_3, x_2) P(x_3 \mid x_1) P(x_2 \mid x_1) P(x_1) \right). \quad (4)$$

Instantiating the variable x_3 in the outer sum renders the factor $P(x_5' \mid x_3)$ independent of the second inner sum. The evaluations in Equation (4) are performed from the inside outward, and the complete evaluation of Equation (4) requires 15 additions and 50 multiplications, while Equation (3) requires 15 additions and 64 multiplications. Carrying the example further, we see that the variable x_2, together with the variable x_3 already instantiated, is a d-separator for the subnetwork $\{x_1, \ldots, x_4\}$, which is the set of variables composing the second sum of Equation (4). Selecting x_2 as the new separator variable results in the decomposition

$$P(x_5') = \sum_{x_3} \left[P(x_5' \mid x_3) \sum_{x_2} \left(\sum_{x_1} P(x_3 \mid x_1) P(x_2 \mid x_1) P(x_1) \right.\right.$$
$$\left.\left. \times \sum_{x_4} P(x_4 \mid x_3, x_2) \right) \right]. \quad (5)$$

The complete evaluation of Equation (5) requires 11 additions and 22 multiplications, representing a considerable reduction from the number of operations needed when applying the brute-force approach of Equation (3). Assume we cache the values of the summation $\sum_{x_2}[\cdots]$ indexed by the value assigned to x_3, and let us refer to these values with $\sigma(x_3)$. If we now wish to calculate $P(x_5'')$, for some $x_5'' \neq x_5'$, we can use the cached $\sigma(x_3)$, since their value is independent of the particular value assigned to x_5.[4] We can then use Equation (6) to calculate the probability of interest:

$$P(x_5'') = \sum_{x_3} P(x_5'' \mid x_3)\, \sigma(x_3), \quad (6)$$

which requires 1 addition and 2 multiplications, representing a 15-fold reduction in the number of additions and a 32-fold reduction in the number of multiplications from the numbers required by the brute-force approach given by Equation (3).

Belief-network inference by recursive decomposition is based on a systematic application of the decomposition method informally introduced in the example above. By looking at Example 1, we can see that every step in the decomposition corresponds to the partition of a set of nodes, and to the determination of a d-separator for the partition components. In fact, the decomposition that leads to Equation (4) from Equation (3) is obtained by partitioning the set $\{x_1, \ldots, x_5\}$ into the two sets $\{x_5\}$ and $\{x_1, \ldots, x_4\}$, and by selecting x_3 as the d-separator of these two sets. Likewise, the decomposition that leads to Equation (5) from Equation

[4]For the sake of the explanation, assume that x_5 has more than two values; otherwise the probability of interest could simply be calculated as $P(x_5'') = 1 - P(x_5')$.

(4) is obtained by decomposing the set $\{x_1, \ldots, x_4\}$ into the two sets $\{x_1, x_2, x_3\}$ and $\{x_4\}$, and by selecting the set $\{x_2, x_3\}$ as their d-separator. Notice also that x_5 is already instantiated when performing the first step of the decomposition, and both x_3 and x_5 are already instantiated when performing the second step of the decomposition.

The decomposition process just described can be effectively formalized by means of the function f, which we define below together with the main theorem that establishes how f can be recursively decomposed. To this purpose, we need to introduce some additional terminology. For any set $X \subseteq \mathcal{X}$, we denote with $\Omega(X)$ the set of all possible instantiations of X, and with X' and X^i arbitrary instantiations of X, where the latter notation is used when we need to distinguish between different instantiations. Given X and Y as two subsets of $\mathcal{X}$, let $X/_{Y'}$ denote the partial instantiation of set X obtained by instantiating the nodes in $X \cap Y$ as specified by Y'. For example, if $X = \{x_1, x_2\}$, $Y = \{x_2, x_3\}$, and $Y' = \{(x_2 = T, x_3 = F)\}$, then $X/_{Y'} = \{(x_1 = T, x_2 = T), (x_1 = F, x_2 = T)\}$. That is, $X/_{Y'}$ represents the set of possible instantiations of X where x_2 is clamped to T.

DEFINITION 1 *Let $\mathcal{X}$ be the set of all variables in a belief network, let X be a subset of $\mathcal{X}$, and let θ_X denote the family set of X (i.e., $\theta_X = \bigcup_{x \in X} \theta_x$). Given a subset $H_X \subseteq \mathcal{X}$ such that $(H_X \cup X) = \theta_X$, and given an arbitrary instantiation H'_X of H_X, we define the function f as follows:*

$$f(X, H'_X) = \sum_{X/_{H'_X}} \prod_{x \in X} P(x \mid \pi_x) , \tag{7}$$

where the summation is taken over all the possible instantiations of $X - H_X$, since the variables in H_X are instantiated as specified by H'_X.

From Equation (7), we can see that when $X = \mathcal{X}$, the invocation of $f(X, H'_X)$ corresponds to the calculation of the marginal probability $P(H'_X)$. The following theorem establishes the way function f can be recursively decomposed.

THEOREM 1 *Given the invocation of $f(X, H'_X)$ for any H_X and X satisfying Definition 1, and given any partition (Y, Z) of X, the set $S = (\theta_Y \cap \theta_Z) - H_X$ renders the following equation valid:*

$$f\big(X, H'_X\big) = \sum_{S} f\big(Y, H_Y/_{S' \cup H'_X}\big)\ f\big(Z, H_Z/_{S' \cup H'_X}\big), \tag{8}$$

where $H_Y = \theta_Y \cap (S \cup H_X)$, $H_Z = \theta_Z \cap (S \cup H_X)$.

A proof of the theorem can be found in [1]. In Equation (8), the summation is over all the instantiations S' of S. Notice that $H_Y/_{S' \cup H'_X}$ is a full instantiation of H_Y, since by construction $H_Y \subseteq S \cup H_X$, and is consistent, since $H_X \cap S = \emptyset$. Likewise, $H_Z/_{S' \cup H'_X}$ is a consistent full instantiation of H_Z. Basically, the theorem establishes that it is possible to find a set $(S \cup H_X)$ that renders the components of the partition (Y and Z) conditionally independent of the remaining belief-network variables. That is, given S and H_X as defined in Theorem 1, the variables in the set Y are conditionally independent of the variables in the set $\mathcal{X} - (S \cup H_X \cup Y)$. Analogously, the variables in the set Z are conditionally independent of the variables in the set $\mathcal{X} - (S \cup H_X \cup Z)$. This comes as no surprise, since it can be proved that the set $S \cup H_X$ d-separates each of Y and Z from the remaining belief-network variables [1]. Notice that the theorem does not say how the partition (Y, Z) should be chosen, and in general, the number of possible partitions is large. The selection of an appropriate partition is the most delicate issue in the application of RD, and the efficacy of the method largely depends on it.

As we previously mentioned, the theorem actually provides a method for calculating the marginal probability $P(Q')$ of any subset Q of $\mathcal{X}$. The value of $P(Q')$ is obtained by invoking $f(X, H'_X)$, with $X = \mathcal{X}$, $H_X = Q$, and $H'_X = Q'$. Theorem 1 provides a way of decomposing f, by partitioning the set X into two subsets Y and Z, and by recursively applying f to the partition components. The recursion terminates at each function invocation $f(X, H'_X)$ in which $|X| = 1$, and the variable x in X and its parents are instantiated. At this point, $f(X, H'_X)$ evaluates directly to $P(x' \mid \pi'_x)$. As long as we require that the set Y and the set Z in the partition (Y, Z) each contain at least one node, the first argument of function f becomes smaller at progressively deeper levels of the recursion. Thus, we must eventually reach invocations of f of the form $f(X, H'_X)$ in which $|X| = 1$. Let x designate the sole element in X. We know that the parents of x must be instantiated, because it follows from Definition 1 that $H_X \supseteq \theta_X - X = \theta_x - \{x\} = \pi_x$. If x is instantiated, we return $P(x \mid \pi_x)$ as discussed. If x is not instantiated, we set $Y = \{x\}$ and apply Equation (8) once more, with $Z = H_Z = \emptyset$ and $S = \{x\}$. At this point, $f(Y, H'_Y)$ returns $P(x \mid \pi_x)$, and $f(Z, H'_Z) = f(\emptyset, \emptyset)$ evaluates to 1 by definition.

EXAMPLE 2 We can apply the method just described to the calculation of $P(x_5 = T)$ of Example 1, by invoking $f(X, H'_X)$ with $X = \mathcal{X}$ and $H'_X = x'_5$. Provided we first partition $\mathcal{X}$ into $(Y, Z) = (\{x_1, \ldots, x_4\}, \{x_5\})$, and we further partition Y into the two components $\{x_1, \ldots, x_3\}$ and $\{x_4\}$, we obtain the same decomposition of Equation (5). The recursive application of Theorem 1 is illustrated in

$$f(\mathcal{X}, x'_5) = \sum_{x_3} f(\{x_5\}, \{x_3, x'_5\})\ f(\{x_1, x_2, x_3, x_4\}, \{x_3\}) \tag{9}$$

$$= \sum_{x_3} f(\{x_5\}, \{x_3, x'_5\}) \sum_{x_2} f(\{x_1, x_2, x_3\}\{x_2, x_3\})\ f(\{x_4\}, \{x_2, x_3\}) \tag{10}$$

$$= \sum_{x_3} \left[P(x'_5 \mid x_3) \sum_{x_2} \left(\sum_{x_1} P(x_3 \mid x_1) P(x_2 \mid x_1) P(x_1) \right.\right.$$

$$\left.\left. \times \sum_{x_4} P(x_4 \mid x_3, x_2) \right) \right]. \tag{11}$$

The initial partition of $\mathcal{X}$ into $(Y, Z) = (\{x_1, \ldots, x_4\}, \{x_5\})$ yields the separator set $S = \theta_Y \cap \theta_Z - H_X = \{x_3\}$, which results in the decomposition of Equation (9). In the right-hand invocation of f in Equation (9), we have $X = \{x_1, \ldots, x_4\}$ and $H_X = \{x_3\}$, and the partition of X into $(Y, Z) = (\{x_1, \ldots, x_3\}, \{x_4\})$ yields the separator set $S = \theta_Y \cap \theta_Z - H_X = \{x_2\}$ and produces the decomposition illustrated in Equations (10). Applying now the function f as given by Definition 1 results in Equation (11).

2.3. An Implementation of Recursive Decomposition

To facilitate the description of the algorithm that implements the function f, we introduce some additional terminology. We call the set S the *summation set.* We call H_X the *instantiation set,* because it contains all the instantiated variables in a given invocation $f(X, H'_X)$. We use *instantiation cache* to denote a table that stores the value of $f(X, H'_X)$ indexed by the instantiated variables in the set H_X. We call $X - H_X$ the *variable set,* because it represents the variables that are uninstantiated when $f(X, H'_X)$ is invoked. Finally, we call X the *total set.*

Suppose that decompositions of a network are performed recursively, beginning with $H_X = \emptyset$ and with $X = \mathcal{X}$, until single terms of the form $P(x \mid \pi_x)$ are encountered. Call this a *complete decomposition* of a given belief network.[5] By starting with $H_X = \emptyset$ at the top level of the decomposition, we create a complete decomposition that can be used to calculate $P(Q')$ for any $Q \subseteq \mathcal{X}$. The complete decomposition of a network can be represented as a binary tree of records, where each record contains a summation, evaluation, instantiation, and variable set. We call this tree the *decomposition tree* or *d-tree.* Figure 2(a) shows a complete d-tree for the network of Figure 1, corresponding to the decomposition of Equation (11). Notice that the d-tree also includes the summation over x_5, since the decomposition corresponds to the invocation of $f(\mathcal{X}, \emptyset)$. Each node of the tree corresponds to a record containing four sets of variables and two pointers to the children records. Also, associated with every record is the instantiation cache storing the values of $f(X, H'_X)$, indexed by the values of the variables in H_X. In other words, for a given instantiation H'_X, the corresponding cache entry stores the value returned by f given H'_X. For example, the $\sigma(x_3)$ in Equation (6) corresponds to the cache of the record R_3, with x_3 the instantiation set.

In Figure 3 we give a schematic description of the algorithm implementing the function f by making use of the tree structure just described. As already mentioned, summation, evaluation, and instantiation sets are variables local to the currently accessed record, and do not need to be passed as arguments. The argument to the function is the pointer i to the current record. The global variable ω keeps track of the instantiated nodes in $\mathcal{X}$, and the global variable Q' stores the query of which the algorithm is computing the probability. In any

[5] In general, there are many complete decompositions of a belief network.

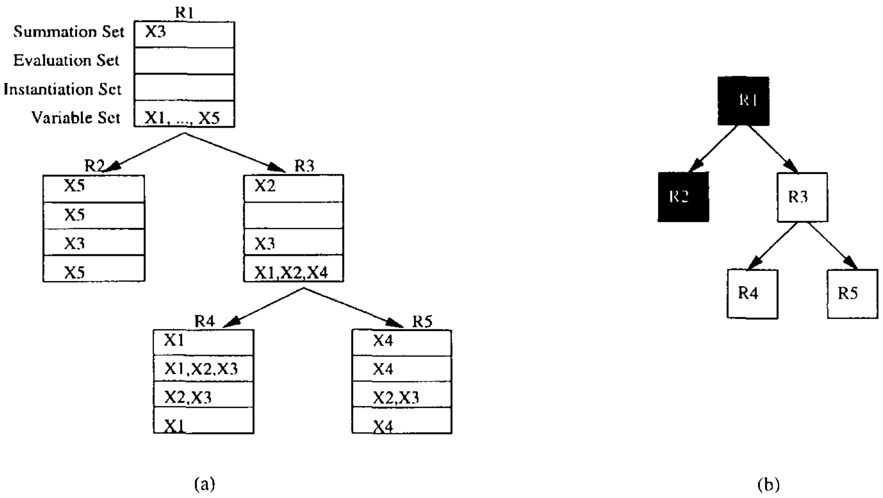


Figure 2. The decomposition tree for the simple network of Figure 1. (a) Each node in the d-tree is called a *record* and contains four sets and two pointers to the children. (b) The shaded records correspond to the *active subtree* for the calculation of $P(x'_5)$.

```
function $f(i)$
    input:         $i$            the pointer to the current record
    local vars:    $H_X$, $S$     instantiation and summation set
                   cache          the cache for the current record
                   $i_Y$, $i_Z$   the pointers to the two children of record $i$
    global vars:   $\omega$       the current instantiation of $\mathcal{X}$
                   $Q'$           the query set, i.e., we are computing $P(Q')$
begin
    if $i = 0$ return 1                      /* end of the recursion, leaf reached */
    sum $\leftarrow$ cache[$\omega[H_X]$]
    if sum $= -1$ then                       /* -1 denotes a reset cache entry */
        sum $\leftarrow 0$
        for each instantiation $S'$ of $S$ consistent with $Q'$ do
            $\omega[S] \leftarrow S'$
            sum $\leftarrow$ sum $+ f(i_Y) \times f(i_Z)$
        end /* for */
        cache[$\omega[H_X]$] $\leftarrow$ sum
    end /* then */
    return sum
end /* function $f$ */
```

Figure 3. The algorithm that implements the function f.

recursive invocation of f, the cache entry for the current instantiation of its associated H_X is accessed to check if the needed value of f has already been computed. The current instantiation of H_X is determined by the values of the corresponding nodes in ω, given by $\omega[H_X]$, and the corresponding cache entry is denoted by $cache[\omega[H_X]]$. If the cache entry contains the desired value, that value is returned; otherwise it must be computed. The *for* loop in Figure 3 corresponds to the summation over the instantiations of the summation set S of Equation (8). Notice that the loop condition has the additional constraint that the instantiation of S must be consistent with the instantiation of Q. That is, if S contains nodes that are also in Q (i.e., $S \cap Q \neq \emptyset$), those nodes need to be instantiated as specified by Q'.[6] Once the loop is completed, the result of the computation is stored in the appropriate cache entry, and the result is returned.

An efficient way of using the algorithm of Figure 3 is the following. The tree is first *initialized*, i.e., the function $f(\mathcal{X}, \emptyset)$ is called, corresponding to the summation over the whole joint probability space. The function f should clearly evaluate to 1. As a side effect, all the cache entries of each record are initialized to their initial (more general) values. When a query $P(Q')$ is submitted, we do not need to recompute f for all the records of the tree, but only for those records for which Q' may affect the value returned by f. In fact, it is possible to determine this set of records before actually carrying out the computation. A record has its associated invocation of f affected by Q' if its variable set $V = X - H_X$ contains some of the nodes in Q. We call the set of affected records the *active subtree*, denoted with $\mathcal{A}(Q)$. Formally, if we denote the variable set of a record r with V_r, the active subtree for a given query $P(Q')$ is defined as $\mathcal{A}(Q) = \{r \mid V_r \cap Q \neq \emptyset\}$. For all the records in $\mathcal{A}(Q)$, the function f needs to be actually computed. For all the other records, the values stored into their cache can be retrieved. Figure 2(b) shows the active subtree $\mathcal{A}(\{x_5\})$ for the calculation of $P(x_5')$.

We can illustrate the rationale behind this technique with an example. Consider again the decomposition corresponding to Figure 2(a), which we recall below (notice that we now include the summation over x_5, since this equation allows for the initialization of the decomposition tree):

$$f(\mathcal{X}, \emptyset) = \sum_{x_3} \left[\sum_{x_5} P(x_5 \mid x_3) \sum_{x_2} \left(\sum_{x_1} P(x_3 \mid x_1) P(x_2 \mid x_1) P(x_1) \times \sum_{x_4} P(x_4 \mid x_3, x_2) \right) \right]. \tag{12}$$

Consider now the calculation of $P(x_5')$ from Example 1. Using the decomposition

[6]We need the additional constraint because the instantiation set H_X associated to a given record is determined at initialization time, when invoking $f(\mathcal{X}, \emptyset)$, and does not allow for the clamped nodes in Q'.

tree of Figure 2(a), results in the following equation:

$$f(\mathcal{X}, \{x_5'\}) = \sum_{x_3} \left[P(x_5' \mid x_3) \sum_{x_2} \left(\sum_{x_1} P(x_3 \mid x_1) P(x_2 \mid x_1) P(x_1) \right.\right.$$
$$\left.\left. \times \sum_{x_4} P(x_4 \mid x_3, x_2) \right) \right]. \qquad (13)$$

Note that the only difference between Equation (13) and Equation (12) is in the summation over x_5, which is missing in Equation (13). It corresponds to the record R_2 in Figure 2(a). The summation over x_2 corresponding to the record R_3 remains the same.

3. BOUNDED RECURSIVE DECOMPOSITION

In the previous section, we illustrated an algorithm for belief-network inference by recursive decomposition, and we presented an implementation of the algorithm. As we pointed out in the introduction, exact probabilistic inference with belief networks is NP-hard, which means that there are instances in which inference by recursive decomposition still results computationally intractable.

In this section, we discuss *bounded RD*, an inference algorithm based on RD which, in order to reduce the computational cost of inference, computes interval bounds on the marginal probability of a specified set of nodes. The method assumes that it is possible to perform the complete decomposition and initialization of the network, as illustrated in the previous section. If the network is such that not even a complete decomposition and initialization is computationally feasible, the method is not applicable.[7]

In what follows, we first present an example that illustrates the main idea on which the method is based. We then give a formal description of the method, and finally illustrate the use of Markov simulation in tightening probability bounds.

EXAMPLE 3 Consider again the network of Figure 1, and the calculation of $P(x_5 = T)$ [for brevity, $P(x_5')$] as described in Example 1, where we introduced the use of the cached values in $\sigma(x_3)$ to avoid redundant computation when calculating $P(x_5'')$. As already mentioned, $\sigma(x_3)$ corresponds to the cache associated to record R_3, and the values it stores correspond to the results of the summation in the

[7]Remember that a complete decomposition is a decomposition resulting from the invocation of f starting with $H_X = \emptyset$, and $X = \mathcal{X}$ until single terms of the form $P(x \mid \pi_x)$ are encountered. The initialization of the d-tree consists of initializing the cache entries of each record in the decomposition tree. There might be cases when the connectivity and the size of the network are such that even the initialization of the decomposition tree is computationally infeasible. For example, for a fully connected network containing 100 binary nodes, a cache of size 2^{99} is needed, thus making the initialization of the cache entries not feasible given the current resource constraints.

x_3	$\sigma(x_3)$
T	0.628
F	0.372

x_3	$\sigma^*(x_3)$
T	0.403
F	0.085

Figure 4. The values of $\sigma(x_3)$ and $\sigma^*(x_3)$ as computed in Example 3, Equations (14) and (15) respectively.

following equation for the different values assigned to x_3:

$$\sigma(x_3) = \sum_{x_2}\left(\sum_{x_4} P(x_4 \mid x_3, x_2) \sum_{x_1} P(x_3 \mid x_1) P(x_2 \mid x_1) P(x_1)\right). \tag{14}$$

Consider now the calculation of $P(x_5 = T, x_4 = T)$ [for brevity, $P(x_5', x_4')$]. The values stored in $\sigma(x_3)$ need to be recomputed, since they do not allow for the fact that x_4 is now clamped. If we denote the newly computed values with $\sigma^*(x_3)$, the probability of interest can be computed as $P(x_5', x_4') = \sum_{x_3} P(x_5' \mid x_3)\sigma^*(x_3)$, where $\sigma^*(x_3)$ is given by the following summation:

$$\sigma^*(x_3) = \sum_{x_2}\left(P(x_4' \mid x_3, x_2) \sum_{x_1} P(x_3 \mid x_1) P(x_2 \mid x_1) P(x_1)\right). \tag{15}$$

Figure 4 gives the values of $\sigma(x_3)$ and $\sigma^*(x_3)$, and we can see that $0 \le \sigma^*(x_3) \le \sigma(x_3)$ holds for all x_3, since $\sigma^*(x_3)$ is obtained from $\sigma(x_3)$ by simply removing the summation over x_4. This suggests that we could use the values stored in $\sigma(x_3)$ as upper bounds on the values stored in $\sigma^*(x_3)$, to compute upper and lower bounds on the probability $P(x_5', x_4')$. An application of this strategy is illustrated in the following equations, where we show the computation of the upper bound P_U and lower bound P_L on $P(x_5', x_4')$ obtained by computing $\sigma^*(x_3 = T)$ only, and by using the cached value of $\sigma(x_3 = F)$ as an upper bound, and 0 as a lower bound, on the value of $\sigma^*(x_3 = F)$ (the numerical probabilities used in the computation are taken from Figure 1):

$$\begin{aligned} P_U(x_5', x_4') &= P(x_5' \mid x_3 = T)\sigma^*(x_3 = T) \\ &\quad + P(x_5' \mid x_3 = F)\sigma(x_3 = F) = 0.198, \end{aligned} \tag{16}$$

$$P_L(x_5', x_4') = P(x_5' \mid x_3 = T)\sigma^*(x_3 = T) + 0 = 0.161. \tag{17}$$

The application of this method allows for a reduction in computation of about 50%. Notice that the choice of the instantiation of x_3 for which to compute $\sigma^*(x_3)$ is very important. Had we chosen to compute $\sigma^*(x_3 = F)$ instead, and to use the cached value of $\sigma(x_3 = T)$ as an upper bound on the value of $\sigma^*(x_3 = T)$, the resulting bounds on $P(x_5', x_4')$ would be $[0.008, 0.26]$. This example illustrates how critical the selection of the proper instantiations can be, which is a point to which we will return.

3.1. Computation of Probability Bounds

As explained in Section 2.3, the strategy for answering a query of the form $P(Q')$ is to reset all the records of the decomposition tree in the active subtree $\mathcal{A}(Q)$, and to recompute the function f for those records, while retrieving the values stored in the cache for all the other records. The strategy, which we call d-tree-based recursive decomposition, is summarized:

$$f(X, H'_X) = \begin{cases} cache[H'_X] & \text{if not reset,} \\ \sum\limits_{\substack{\omega_j \in \Omega(S) \\ \omega_j \models q'_X}} f(Y, H_Y/_{H'_X \cup \omega_j}) f(Z, H_Z/_{H'_X \cup \omega_j}) & \text{otherwise,} \end{cases} \tag{18}$$

where S is the current summation set, q is the set $S \cap Q$, and q' denotes its instantiation consistent with Q', i.e., $q' = (S \cap Q)/_{Q'}$. Notice that the source of inefficiency in the computation of the function f is in the possibly large number of instantiations of the summation set S. The idea, informally introduced in Example 3 (where $S = \{x_3\}$), is to select a subset of the instantiations of S ($x_3 = T$ in the example) on which to perform exact computation, and to retrieve the values of f calculated at initialization time for the remaining instantiations ($x_3 = F$ in the example). These initialization values represent an upper bound on the actual values that would be returned if the function f were actually computed. In fact, the following lemma is straightforward to prove:

LEMMA 1 *Consider an invocation of $f(X, H'_X)$ for any H_X and X satisfying Definition 1. For any $K_X \subseteq \mathcal{X}$ such that $K_X \supseteq H_X$, and for any instantiation K'_X of K_X such that $K'_X \supseteq H'_X$, the following relation holds:*

$$0 \le f(X, K'_X) \le f(X, H'_X). \tag{19}$$

Proof According to Definition 1, $f(X, K'_X) = \sum_{X/_{K'_X}} P(X)$, and $f(X, H'_X) = \sum_{X/_{H'_X}} P(X)$. Since $K'_X \supseteq H'_X$, it follows that the set of instantiations $X/_{K'_X}$ is a subset of $X/_{H'_X}$. We can thus write $f(X, H'_X) = \sum_{X/_{K'_X}} P(X) + \sum_\Delta P(X) = f(X, K'_X) + \sum_\Delta P(X)$, where Δ is the set of instantiations in $X/_{H'_X} - X/_{K'_X}$. Since $\sum_\Delta P(X) \ge 0$, this concludes the proof. ■

Let us denote with $\overline{f}(X \mid H^i_X)$ the value stored at initialization time into the ith cache entry of the record corresponding to the total set X. Lemma 1 tells us that, when computing $P(Q')$ for some $Q \subseteq \mathcal{X}$, the exact calculation of any $f(X, H^i_X)$ in the active subtree $\mathcal{A}(Q)$ is upper-bounded by the corresponding $\overline{f}(X \mid H^i_X)$, the value stored at initialization time. In fact, by looking at Equation (18), we see that when $Q \neq \emptyset$, the calculation of $f(X, H^i_X)$ corresponds to the calculation of $f(X, H^i_X \cup q')$. We can then apply Lemma 1 by setting $K_X = H_X \cup q$ and $K'_X = K_X/_{H^i_X \cup q'}$.

Let us apply this result to the calculation of f for a single record in the decomposition tree, where the relevant sets of the record are its total set X, its instantiation set H_X, and its summation set S. With (Ω_1, Ω_2) we denote an arbitrary partition of the set $\Omega(S)$ of all instantiations of S. The upper bound f_U and the lower bound f_L on the value of $f(X, H_X^i)$, for any $H_X{}^i$, can be computed as follows:

$$f_U(X, H_X^i) = \sum_{\substack{\omega_j \in \Omega_1 \\ \omega_j \models q'_X}} f(Y, H_Y^{ij})\, f(Z, H_Z^{ij}) + \sum_{\substack{\omega_j \in \Omega_2 \\ \omega_j \models q'_X}} \overline{f}(Y \mid H_Y^{ij})\, \overline{f}(Z \mid H_Z^{ij}), \tag{20}$$

$$f_L(X, H_X^i) = \sum_{\substack{\omega_j \in \Omega_1 \\ \omega_j \models q'_X}} f(Y, H_Y^{ij})\, f(Z, H_Z^{ij}) + 0, \tag{21}$$

where we have used the simplifying notation $H_Y^{ij} = H_Y/_{\omega_i \cup H_X^j}$ and $H_Z^{ij} = H_Z/_{\omega_i \cup H_X^j}$. Notice that in Equations (20) and (21), we assume that the recursive invocations of f in the summation over Ω_1 return exact values. However, if we apply the above idea to any recursive invocation of f, the two equations for the upper and lower bounds need to be rewritten as follows:

$$f_U(X, H_X^i) = \sum_{\substack{\omega_j \in \Omega_1 \\ \omega_j \models q'_X}} f_U(Y, H_Y^{ij})\, f_U(Z, H_Z^{ij}) + \sum_{\substack{\omega_j \in \Omega_2 \\ \omega_j \models q'_X}} \overline{f}(Y \mid H_Y^{ij})\, \overline{f}(Z \mid H_Z^{ij}), \tag{22}$$

$$f_L(X, H_X^i) = \sum_{\substack{\omega_j \in \Omega_1 \\ \omega_j \models q'_X}} f_L(Y, H_Y^{ij})\, f_L(Z, H_Z^{ij}) + 0. \tag{23}$$

We can then apply Equations (22) and (23) to every record in $\mathcal{A}(Q)$ to compute interval bounds on the point-value of $P(Q')$ for any $Q \subseteq \mathcal{X}$. To compute interval bounds on the conditional probability $P(Q^i \mid R')$ of any Q and R subsets of $\mathcal{X}$, we first need to compute the marginals $P(Q^i, R')$ for every $Q^i \in \Omega(Q)$. Let us denote with $P_U(Q^i, R')$ and $P_L(Q^i, R')$ the upper and lower bounds on the exact value of $P(Q^i, R')$ respectively. Bounds on the conditional probability $P(Q^i \mid R')$ can be computed as follows:

$$P_U(Q^i \mid R') = \frac{P_U(Q^i, R')}{P_U(Q^i, R') + \sum_{k \neq i} P_L(Q^k, R')}, \tag{24}$$

$$P_L(Q^i \mid R') = \frac{P_L(Q^i, R')}{P_L(Q^i, R') + \sum_{k \neq i} P_U(Q^k, R')}. \tag{25}$$

3.2. Partition of $\Omega(S)$ by Markov Simulation

So far, we have not specified how to determine the partition (Ω_1, Ω_2) of a given $\Omega(S)$, that is, how to determine which instantiations of a given summation-set to consider for exact computation. The partition of the summation set instantiations needs to be computed for each of the records in the active subtree for the current query. The choice of the right partitions is crucial, and it strongly affects the tightness of the interval bounds. In this section we present a method that makes use of Markov simulation to generate highly probable instantiations. However, other methods can be readily adopted.

The general algorithm to perform this task is illustrated in Figure 5. The algorithm iterates through a loop. At each iteration, a complete instantiation of the network is generated, that is, every node in the network is instantiated. Let us call one of these complete instantiations a *sample*. Every sample identifies a unique instantiation of the summation-set of each record in $\mathcal{A}(Q)$. If $w \in \Omega(\mathcal{X})$ is the sample generated, the corresponding instantiation of a certain summation set S is $S' = S/_{\omega}$. If S' does not already belong to the partition component Ω_1 of the corresponding record, it is added to it, and it is removed from Ω_2. This process is repeated for each record in $\mathcal{A}(Q)$, and it terminates when the desired number k of summation set instantiations has been generated. The input parameter k specifies the total number of instantiations to be included in the Ω_1 of the summation sets in the active subtree $\mathcal{A}(Q)$. The total number N of possible instantiations of the summation sets in $\mathcal{A}(Q)$ is given by $N = \sum_{r \in \mathcal{A}(Q)} |\Omega(S_r)|$, where S_r denotes the summation set of record r.

Therefore the number of instantiations included in Ω_1 is given by $\sum_{r \in \mathcal{A}(Q)} |\Omega_{1,r}|$ (where $\Omega_{1,r}$ denotes the partition component Ω_1 for record r), and the algorithm terminates when this number is greater than k. The other input to the algorithm is the query set Q [e.g., if we want to compute $P(A' \mid B')$, the query set will be $Q = A \cup B$].

The algorithm of Figure 5 provides a general schema for partitioning the summation sets in $\mathcal{A}(Q)$. The crucial step is the generation of the samples of the network. The most straightforward solution would be to generate samples randomly. While having the advantage of requiring no overhead (i.e., no time is spent looking for a particular instantiation), this solution may result in poor convergence of the interval bounds (experimental results supporting this conclusion are presented in the next section). The approach we have adopted makes use of Markov simulation. Markov simulation tends to generate samples that are highly probable given the evidence [e.g., if we wish to compute $P(A' \mid B')$, the set B is the evidence set].

We constrain the simulation process so as to avoid the generation of redundant samples, by which we mean instantiations that do not add any element to the Ω_1 partition component of any record in the active subtree. According to this definition, if $\mathcal{S} = \bigcup_{r \in \mathcal{A}(Q)} S_r$ is the set of nodes contained in summation sets of the active subtree, all the samples differing only in the value assigned to nodes in

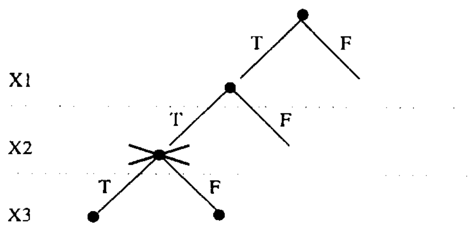
```
procedure partitionS(k, Q)
    input:  k       the desired number of instantiations
            Q       the set of query nodes
    counter ← 0
    while counter < k do
        $\omega$ ← generate_sample($\mathcal{X}, \Omega, P$)
        for each $r \in \mathcal{A}(Q)$ do
            let $(\Omega_1, \Omega_2)$ be the current partition of $\Omega(S_r)$
            $S' \leftarrow S_r/_\omega$
            if $S' \notin \Omega_1$ then
                $\Omega_1 \leftarrow \Omega_1 \cup \{S'\}$
                $\Omega_2 \leftarrow \Omega_2 - \{S'\}$
                counter ← counter +1
                if counter ≥ k then exit from partitionS
            end /* if */
        end /* for */
    end /* while */
end /* partitionS */
```

Figure 5. The algorithm to compute the partition (Ω_1, Ω_2) of $\Omega(S_r)$ for each record r in the current active subtree $\mathcal{A}(Q)$.

$\mathcal{X} - S$ are redundant. The first constraint is thus to generate only samples that differ in the value of at least one node in S. However, even when enforcing this constraint, the possible number of redundant samples is still high.[8] The concept is better explained with an example. Consider an active subtree of two records only, r_1 and r_2, and suppose that the summation set of each record contains a single binary node, x_1 or x_2, respectively. The two samples $(x_1 = 0, x_2 = 0)$ and $(x_1 = 1, x_2 = 1)$ already exhaust the space of possible instantiations of the two summation sets [i.e., in both records we have $\Omega_1 = \Omega(S_X)$], and every other of the 2^2 possible instantiations of (x_1, x_2) is redundant. In general, given an active subtree of n records, each with a summation set containing a single binary node, the minimum possible number of samples necessary to exhaust the space of summation-set instantiations is always 2 (e.g., the samples $00\cdots0$ and $11\cdots1$), while the number of distinct samples of n binary variables is 2^n.

The generation of redundant samples is an unnecessary overhead that we want to avoid. Therefore, we associate with every summation set in $\mathcal{A}(Q)$ a tree structure that keeps track of the instantiations of S_X already in the corresponding Ω_1, and we rely on these structures to force the simulation process to generate nonredundant samples only. An example of a tree for a generic summation set containing three binary nodes is depicted in Figure 6. The tree keeps track of the summation-set instantiations already generated, and prevents Markov simulation from generating

[8]The exact number of reduntant samples, assuming binary variables only, is $2^{|S|} - N$, where N is the total number of possible instantiations of summation sets in $\mathcal{A}(Q)$, defined previously.



samples generated to produce the tree to the left:

$x_1 = T,\ x_2 = T,\ x_3 = T$

$x_1 = T,\ x_2 = T,\ x_3 = F$

$x_1 = T,\ x_2 = F,\ x_3 = F$

Figure 6. An example of a tree structure accounting for the summation-set instantiations generated. It corresponds to the hypothetical summation set $S = \{x_1, x_2, x_3\}$, where all the nodes are binary.

them again. If, for example, given the tree in Figure 6, in the next sample we set $x_1 = T$, then the assignment $x_2 = T$ is not considered, since the corresponding path in the tree is blocked (which means that all the possible samples containing the sequence $x_1 = T, x_2 = T$ have already been generated). The overhead due to such bookkeeping is negligible, and it is largely offset by the cost we would incur should we not constrain the simulation process.

4. EVALUATION: EXPERIMENTAL RESULTS

In this section we describe the results of some experiments we conducted with a prototype implementation of bounded-RD. The prototype is written in C++ and runs on a Sun Sparc20 running Solaris 2.3. All the experiments are performed on the belief network ALARM, a multiply connected network originally developed to model anesthesiology problems that may occur during surgery [6]. It contains 37 nodes/variables and 46 arcs. The nodes take between two and four distinct values. Of these nodes, 16 are evidence nodes, 8 are disease nodes, and 13 are intermediate pathophysiological nodes. Notice that, given the network size, the use of approximate methods is superfluous, since exact methods can perform inference in this network efficiently (in a time in the order of milliseconds). The results we present here must thus be interpreted in relative terms, keeping in mind that the actual application of the algorithm here proposed would become advantageous in larger or more highly connected networks.

The experiments are aimed at evaluating the performance of the algorithm in terms of response time and in terms of convergence of the bounds. To this purpose, a large number of queries are submitted, and the average performance over this set of queries is measured. Figure 7 shows the general structure of the algorithm used for the simulation. Evidence sets of different cardinalities are selected randomly, and their instantiation is generated by logic sampling. Given a set of evidence nodes E, queries of the form $P(x_i \mid E')$ are submitted, for each of the eight disease nodes x_i and for all the possible instantiations of each disease node. This process is repeated for every possible cardinality of the evidence set, and for several cycles for

```
input:
  $\mathcal{D}$   the set of the disease nodes of the network
  $\mathcal{E}$   the set of the evidence nodes of the network
begin
  for num-evidence = min-evidence to max-evidence do
    repeat num_cycles times
      $E \leftarrow$ select num-evidence evidence nodes randomly from $\mathcal{E}$
      $E' \leftarrow$ instantiate $E$ by logic sampling
      for each disease-value of each disease in $\mathcal{D}$ do
        compute $P(disease{=}disease\text{-}value \mid E')$
      end
    end
  end
end
```

Figure 7. The basic algorithm used in the experiments.

each selected cardinality. In particular, the parameter *num_cycles* is set to 10 in all the experiments, while *min-evidence* and *max-evidence* vary as described below. The cardinality of the evidence set is important in that it affects the convergence of the bounds; therefore we give the performance of the algorithm for different cardinality ranges.

In Figure 8 we show the average convergence of the bounds (measured as *upper-bound − lower-bound*) as a function of the parameter k/N, where both k and N are formally defined in Section 3. The ratio k/N measures the proportion of summation-set instantiations considered for exact computation (i.e., the proportion of instantiations included in the Ω_1 partition components of the summation sets in the active subtree). When $k = N$, all the possible instantiations are included in the Ω_1 partition components, and both upper and lower bound converge to the exact point-value probability. Figure 8 shows four plots, corresponding to different evidence-set cardinality ranges. The curve labeled "1 to 5" plots the average convergence of the bounds for queries having an evidence-set containing between one and five nodes. The three other plots are for different-size evidence sets. Figure 8 shows that the convergence of the bounds decreases as larger evidence sets are considered. However, when a large number of nodes are instantiated, the exact algorithm usually outperforms the approximate algorithms in terms of inference time. For this reason, in the remaining experiments we limit the maximum evidence-set size to eight nodes, which represents 50% of the total number of evidence nodes in the ALARM network.

Figure 9 compares the convergence of the bounds of bounded RD using Markov simulation and bounded RD with random instantiation of the summation sets. The convergence is here plotted as a function of time, and the average time of exact computation is also plotted (the straight vertical line). We can see that the bounds produced by bounded RD with Markov simulation converge faster than the ones produced by random instantiation of the summation sets. Notice that bounded RD

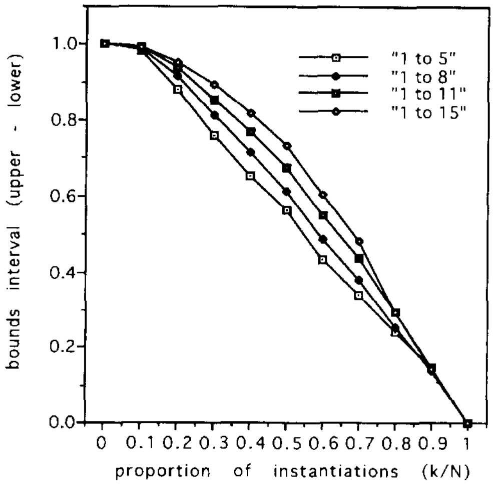


Figure 8. Plot of the average bounds interval (i.e., *upper-bound – lower-bound*) as a function of the proportion of summation-set instantiations included in Ω_1. The label "n to m" indicates that the corresponding curve measures the average bounds interval over queries with evidence set containing a number of nodes between n and m.

with Markov simulation on the average computes the exact probability in less time than bounded RD with random instantiation (the time for exact inference of the two algorithms corresponds to the intersection of the respective curves with the x-axis).

Figure 10 shows the time requirements of the different algorithm components (namely, Markov generation of samples, and computation of interval bounds), and compares them with the time requirements of the basic RD algorithm. As explained in Section 3, Markov simulation is constrained so as to avoid generating redundant instantiations. Besides "constrained" Markov simulation we plot "unconstrained" Markov simulation (i.e., Markov simulation also generating redundant samples), and you can see that the overhead due to constraining the simulation is negligible.

Figures 11 and 12 show the absolute and relative error of Markov simulation as a function of time. The absolute error is defined as $|e - p|$, where e is the probability estimate and p is the exact point-value probability. The relative error is defined as $|e - p|/p$. The time range is the same as the time range of bounded RD, that is, we are interested in the accuracy of Markov simulation within the time boundaries determined by bounded RD.

Finally, Figure 13 shows a plausible scenario where bounded RD and Markov simulation might be profitably used together. In the example, the probability of interest is $P(x_{16} = 0 \mid x_2 = 1, x_{11} = 1, x_{13} = 1, x_{14} = 0)$. Figure 13 shows the upper and lower bound of this probability as computed by bounded RD, together

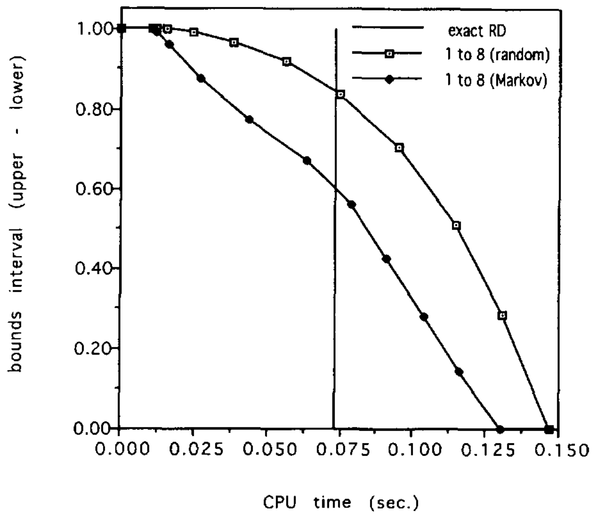


Figure 9. Average convergence of the bounds (measured as *upper-bound – lower-bound*) as a function of CPU time, for queries with evidence set containing between one and eight nodes. The convergence of bounded RD with instantiation of the summation sets by Markov simulation is compared with the convergence of bounded RD with random instantiation of the summation sets. The vertical bar indicates the average time for exact computation.

with the estimation produced by Markov simulation. Notice that the estimation initially falls within the probability bounds, and crosses the upper bound after approximately 40 ms. At that point the bound is still wide, but the point-value estimation suggests that the upper bound is the closest to the exact probability. Consider a scenario in which a decision based on the probability computed above needs to be taken, and the decision threshold is close to 0. The exact probability for this specific case is computed in about 70 ms, and the lower bound computed by bounded RD is about 0.10 after only 30 ms. Assuming that the threshold for the decision at stake is below 0.10, it means that the optimal decision can still be made by saving about 60% of the computation time.

5. RELATED WORK

In this section we briefly review some relevant approximate algorithms. They all qualify as incremental-bounding algorithms in that they all compute successively narrower interval bounds as more resources are allocated to the inference task. Algorithms similar in some respects to ours are localized partial evaluation (LPE)

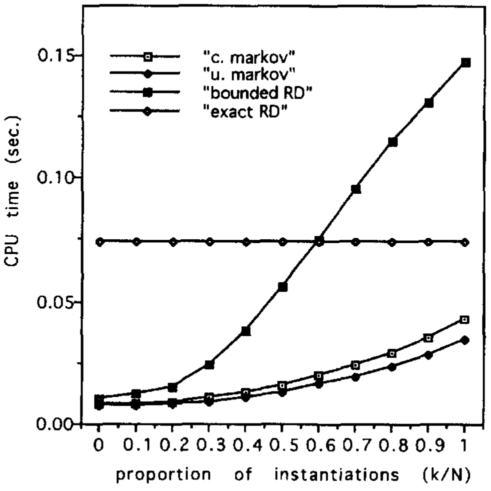


Figure 10. Average CPU time of the different algorithm components as a function of k/N, the proportion of instantiations included in Ω_1. The label "c. markov" stands for "constrained" Markov simulation, and the label "u. markov" stands for "unconstrained" Markov simulation. The horizontal bar indicates the average time for exact computation.

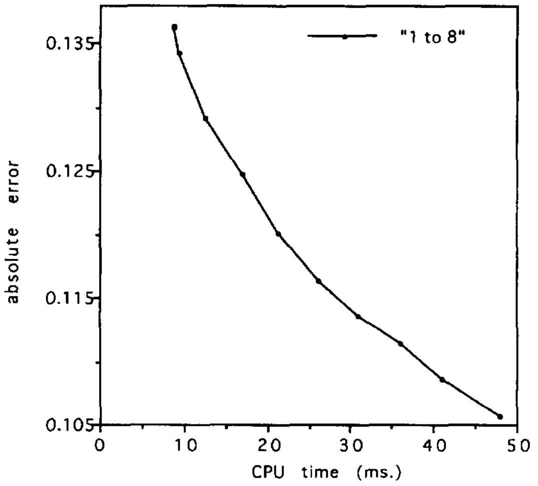


Figure 11. Average absolute error over conditional probabilities of the form $P(x_i \mid E)$ for all disease nodes x_i, and for evidence sets E containing between one and eight nodes, as a function of CPU time.

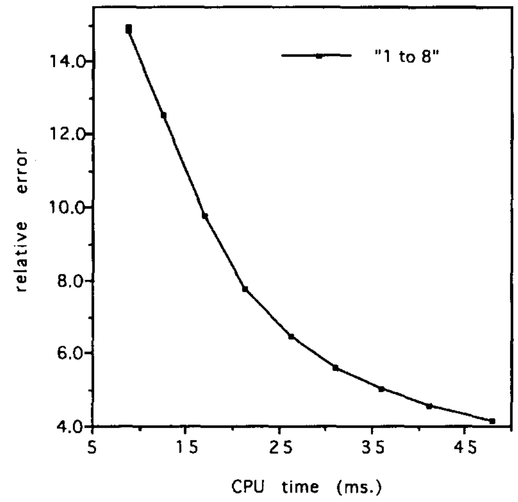


Figure 12. Average relative error over conditional probabilities of the form $P(x_i \mid E)$ for all disease nodes x_i, and for evidence sets E containing between one and eight nodes, as a function of CPU time.

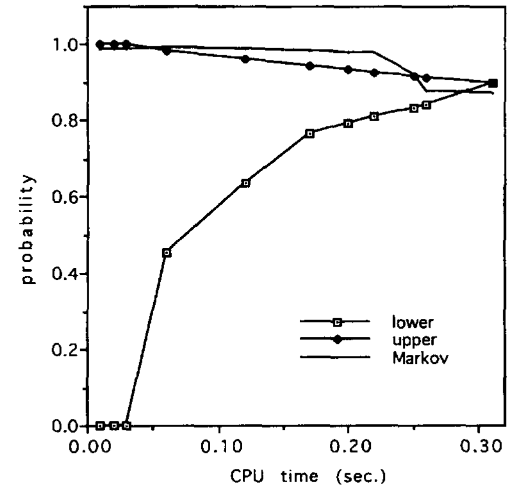


Figure 13. An example of how bounded RD and Markov simulation can complement each other in the estimation of the probability of a given event.

[28], the SPI algorithm [27], bounded cutset conditioning [29], and Poole's search-based algorithm [26].

LPE [28] is based on a standard message propagation technique. It generates bounds by considering only a subset of the nodes in the network, and produces successively narrower bounds by iterating over successively larger subsets of nodes. The messages (the λ and π in Pearl's notation) from outside the selected subset are expressed as [0, 1] bounds, and in order to reduce their adverse effect on the width of the bounds to be computed, a technique is used that takes advantage of the dependency between these messages.

The SPI algorithm [27] is similar to our algorithm in the use of an evaluation polytree to factorize the calculation of the probability of interest (similar to our decomposition tree), and in the caching of the intermediate terms to avoid redundant computations. On the other hand, the SPI search strategy is markedly different in that it is based on assumptions about the nature of the distribution so as to estimate the mass of the yet-to-be-computed terms.

Bounded conditioning is a version of cutset conditioning which computes probability bounds by considering only a subset of the possible assignments to the cutset nodes [29]. Like bounded-RD, it is modular in that the search for the most probable assignments to consider is clearly separated from the update algorithm, thus allowing for the adoption of different search techniques [33].

Poole's algorithm computes probability estimates by enumerating only the most likely complete instances (i.e., assignments to all nodes) [26]. Critical to the method is the search algorithm for finding the most likely instances. Poole proposes an efficient top-down search strategy which works well when applied to extreme distributions, though it becomes very inefficient for less extreme distributions.

6. CONCLUSIONS AND FUTURE WORK

In this paper we present a new algorithm for inference in belief networks that combines an incremental-bounding algorithm with a simulation-based algorithm to compute interval bounds on conditional probabilities. The algorithm inherits all the desirable properties of exact recursive decomposition which are illustrated in detail in [1, 34]. Among them are:

(1) *Flexibility.* The algorithm can be easily combined with other exact and approximation techniques to develop hybrid algorithms. For example, decomposition could be performed until a singly connected network is encountered. At that point, other inference methods, such as Pearl's message passing [3], could be used. As another example, we might combine the recursive decomposition method with clique-propagation techniques, such as Jensen's algorithm [18].

(2) *Modularity.* In RD the inference algorithm is clearly separated from the decomposition algorithm needed for the construction of the decomposition tree.

As researchers develop more efficient algorithms for finding small d-separators (a problem which can be easily translated into the problem of finding small vertex separators), we can immediately apply these results towards decreasing the run-time complexity of belief-network inference. In bounded-RD we introduce a new dimension into the modularity of the technique, in that the computation of bounds is clearly separated from the search strategy for the generation of summation-set instantiations. In this framework, it becomes easy to adopt alternative search strategies. One such strategy is the application of the polynomial algorithm proposed by Goldszmidt in [33] to generate the most probable instantiations. Another technique is backward simulation as defined in [35], which is reported to perform well when the evidence "dominates" the priors.

(3) *Efficiency.* Shachter et al. in [36] conjecture the polynomial-time equivalence of RD and the clustering algorithms [18, 19], which are generally considered the fastest exact inference algorithms currently available for belief-network inference. Our experience in applying RD and clustering algorithms to perform inference on ALARM indicate that the two algorithms are comparable in inference speed.

In the evaluation illustrated in Section 4, we presented the results of our experiments. We would like to stress again that the issue of whether the bounds are narrowed enough to allow a significant reduction in computation time depends on the decision problem that is being solved. For some problems, the bounds could be wide, and one could still make an optimal decision, because the decision threshold is near one extreme (0 or 1). The RD-bounded algorithm provides for the possibility of taking advantage of such situations, when they exist.

The evaluation of bounded RD that is reported in this paper is preliminary, and a more extensive set of experiments over a large variety of network structures and probability distributions need to be performed in order to be able to draw any conclusions on the general behavior of the algorithm, and on its suitability to specific network topologies and probability distributions. Nontheless, the current results suggest that the approach has promise as a practical tool for belief-network inference.

ACKNOWLEDGMENTS

We would like to thank Mark Peot, Marek Druzdzel, and the anonymous reviewers for providing useful comments on a preliminary version of this manuscript.

References

1. Cooper, G. F., Bayesian belief-network inference using recursive decomposition, Tech. Report KSL-90-05, Section of Medical Informatics, Stanford Univ., 1990.

2. Pearl, J., Evidential reasoning using stochastic simulation of causal models, *Artificial Intelligence* 32, 247–257, 1987.

3. Pearl, J., *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Interference*, Morgan Kaufmann, 1988.

4. Howard, R. A., and Matheson, J. E., Influence diagrams, in *Readings in Decision Analysis*, Chapter 38 (R. A. Howard and J. E. Matheson, Eds.), 763–771, 1984.

5. Andreassen, S., Woldbye, M., Falk, B., and Andersen, S. K., Munin—a causal probabilistic network for interpretation of electromyographyc findings, *Proceedings of 10th International Joint Conference on AI*, Milan, Italy, 1987.

6. Beinlich, I., Suermondt, H., Chavez, H., and Cooper, G. F., The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks, *2nd Conference of AI in Medicine, Europe*, London, England, 247–256, 1989.

7. Carenini, G., Monti, S., and Banks, G., An information-based Bayesian approach to history-taking, *5th Conference of AI in Medicine, Europe*, Pavia, Italy, 129–138, 1995.

8. Cooper, G. F., Current research on the development of expert systems based on belief networks, *Appl. Stochastic Models Data Anal.* 5, 39–52, 1989.

9. Heckerman, D. E, Horwitz, E. J., and Nathwani, B. N., Toward normative expert systems: Part I. The Pathfinder Project, *Methods Inform. Med.* 31(2), 90–105, 1992.

10. Matzkevich, I., and Abramson, B., Decision analytic networks in artificial intelligence, *Management Sci.* 41(1), 1–22, Jan. 1995; `http://www.auai.org/dnai-bda.ps.Z`.

11. Kim, J. H., and Pearl, J., A computational model for causal and diagnostic reasoning in inference engines, *Proceedings of 8th IJCAI*, Karlsruhe, West Germany, 190–193, 1983.

12. Pearl, J., Fusion, propagation, and structuring in belief networks, *Artificial Intelligence*, 29, 241–248, 1986.

13. Heckerman, D. E., A tractable algorithm for diagnosing multiple diseases, in *Uncertainty in Artificial Intelligence* 5, 163–171, 1990.

14. Heckerman, D. E., Probabilistic similarity networks, *Networks* 20, 607–636, 1991.

15. Dagum, P., and Galper, A., Additive belief network models, *Proceedings of 9th Conference of Uncertainty in AI*, 91–98, 1993.

16. Cooper, G. F., The computational complexity of probabilistic inference using Bayesian belief networks, *Artificial Intelligence* 42, 1990.

17. Dagum, P., and Luby, M., Approximating probabilistic inference in Bayesian belief networks in NP-hard, *Artificial Intelligence* 60, 141–153, 1993.

18. Jensen, F. V., Lauritzen, S. L., and Olesen, K. G., Bayesian updating in causal probabilistic networks by local computation, *Comput. Statist.* 4, 269–282, 1990.

19. Lauritzen, S. L., and Spiegelhalter, D. J., Local computations with probabilities on graphical structures and their application to expert systems, *J. Roy. Statist. Soc.* 50, 157–224, 1990.

20. Peot, M. A., and Shachter, R. D., Fusion and propagation with multiple observation in belief networks, *Artificial Intelligence* 48, 299–318, 1991.

21. Kjærulff, U., Reduction of computational complexity in Bayesian networks through removal of weak dependencies, *Proceedings of the 10th Conference of Uncertainty in Artificial Intelligence*, San Francisco, CA, 374–382, 1994.

22. Chavez, R. M., and Cooper, G. F., A randomized approximation algorithm for probabilistic inference on Bayesian belief networks, *Networks* 20, 661–685, 1990.

23. Henrion, M., Propagating uncertainty by logic sampling in Bayes' networks, *Proceedings of the AAAI Workshop on Uncertainty in Artificial Intelligence*, Philadelphia, 1986.

24. Henrion, M., Search-based methods to bound diagnostic probabilities in very large belief nets, *Proceedings of the 7th Conference of Uncertainty in AI*, 142–150, 1991.

25. Santos, E., and Shimony, S. E., Belief updating by enumerating high-probability independence-based assignments, *Proceedings of the 10th Conference of Uncertainty in AI*, 506–513, 1994.

26. Poole, D., The use of conflicts in searching Bayesian networks, *Proceedings of the 9th Conference of Uncertainty in AI*, 359–367, 1993.

27. D'Ambrosio, B., Incremental probabilistic inference, *Proceedings of the 9th Conference of Uncertainty in AI*, 301–308, 1993.

28. Draper, D. L., and Hanks, S., Localized partial evaluation of belief networks, *Proceedings of the 10th Conference of Uncertainty in AI*, 170–177, 1994.

29. Horvitz, E. J., Suermondt, H. J., and Cooper, G. F., Bounded conditioning: Flexible inference for decision under scarce resources, *Proceedings of 5th Workshop of Uncertainty in AI*, Windsor, Ontario, 182–193, 1989.

30. Druzdzel, M., Some properties of joint probability distributions, *Proceedings of the 10th Conference of Uncertainty in Artificial Intelligence*, 187–194, 1994.

31. Cooper, G. F., NESTOR: A computer-based medical diagnostic that integrates causal and probabilistic knowledge, Tech. Report HPP-84-48, Stanford Univ., Palo Alto, CA, 1984.

32. Shachter, R. D., Intelligent probabilistic inference, in *Uncertainty in Artificial Intelligence 1* (L. N. Kanal and J. F. Lemmer, Eds.), Amsterdam, North-Holland, 371–382, 1986.

33. Goldszmidt, M., Fast belief update using order-of-magnitude probabilities, *Proceedings of the 11th Conference of Uncertainty in AI*, 1995.

34. Cooper, G. F., A recursive-decomposition method for solving AI graph problems, Tech. Report SMI-93-1, Dept. of Medicine, Univ. of Pittsburgh, 1993.

35. Fung, R., and Del Favero, B., Backward simulation in Bayesian networks, *Proceedings of the 10th Conference of Uncertainty in AI* (R. Lopez de Mantras and D. Poole, Eds.), San Francisco, CA, Morgan Kaufmann, 227–234, 1994.

36. Shachter, R. D., Andersen, S. K., and Szolovits, P., Global conditioning for probabilistic inference in belief networks, *Proceedings of the 10th Conference of Uncertainty in AI*, 514–522, 1994.