An anytime scheme for bounding posterior beliefs

Bozhena Bidyuk and Rina Dechter
Donald Bren School of Information and Computer Science
University Of California Irvine
bbidyuk@ics.uci.edu
dechter@ics.uci.edu

Abstract

This report presents an any-time scheme for computing lower and upper bounds on posterior marginals in Bayesian networks. The scheme draws from two previously proposed methods, bounded conditioning [9] and bounds propagation algorithm [16]. Following the principles of cutset conditioning [18], our method enumerates a subset of cutset tuples and applies exact reasoning in the network instances conditioned on those tuples. The probability mass of the remaining tuples is bounded using a variant of bound propagation. We show that our new scheme improves on the earlier schemes.

1 Introduction

Computing bounds on posterior marginals is a special case of approximating posterior marginals with the desired degree of precision which is NP-hard. Our bounding framework is based on two previously proposed bounds computation schemes, bounded conditioning and bound propagation.

Bounded conditioning [9] is founded on the principles of the cutset-conditioning method [18]. Given a Bayesian network and a subset of variables C=$\{C_1, ..., C_k\}$ (e.g., a loop-cutset), we can obtain exact posterior marginals by enumerating over all cutset tuples $c^i \in D(C)$ over all cutset variables using the formula:

$$P(x|e) = \frac{\sum_{i=1}^{M} P(x, c^i, e)}{\sum_{i=1}^{M} P(c^i, e)} \quad (1)$$

The computation of quantities $P(x, c^i, e)$ and $P(c^i, e)$ for any assignment $c = c^i$ is linear in the network size if C is a loop-cutset and exponential in w if C is a w-cutset. The limitation of the cutset-conditioning method is that the number of cutset tuples, M, grows exponentially with the cutset size. Namely, $M = \prod_{i=1}^{k} |D_i|$ where D_i is the domain of node $C_i \in C$.

In [9], the authors observed that often a small number of tuples $h << M$ contains most of the probability mass of $P(e) = \sum_{i=1}^{M} P(c^i, e)$. Thus, they proposed the **bounded conditioning** method which computes the probabilities $P(x, c^i, e)$ and $P(c^i, e)$ exactly only for the h tuples, $1 \leq i \leq h$, while bounding the rest by their priors. The first h tuples were selected based on their prior probability $P(c^i)$.

Bounded conditioning was the first method to offer any-time properties and to guarantee convergence to the exact marginals with time as $h \rightarrow M$. Its effectiveness was demonstrated in [9] on the example of an Alarm network with 37 nodes and a loop-cutset of size 5 (M=108). The empirical results demonstrate convergence of the algorithm with h but also indicate that the width of the bounds interval for a fixed h decreases as more evidence is added.

Bound propagation scheme, proposed recently in [16], obtains bounds by iteratively solving a linear optimization problem for each variable such that the minimum and maximum of the objective function correspond to lower and upper bounds on the posterior marginals. The performance of the scheme was demonstrated in [16] on the example of well-known Alarm network, Ising grid network, and regular bipartite graphs.

In our work here, we propose a framework, which we term Any Time Bounds (ATB), that also builds upon the principles of conditioning. Like bounded conditioning, it explores fully h cutset tuples, and bounds the rest of the probability mass spread over the unexplored tuples. The scheme improves over bounded conditioning in several ways. First, it bound smore accurately the mass of the unexplored tuples in polynomial time. Second, it uses cutset sampling [5, 6] for finding high-probability cutset tuples. The any-time framework allows to plugin any scheme for bounding joint probabilities to bound the probability mass over unexplored tuples. Finally, utilizing an improved variant of bound propagation algorithm as a plugin within our any-time framework yields a scheme that achieves greater accuracy than either bounded conditioning or bound propagation.

Section 3 provides background on the previously proposed methods of bounded conditioning and bound propagation. Section 4 defines our ATB framework. We first demonstrate how we can bound the probability mass over unexplored tuples in polynomial time and then derive lower and upper bounds on the posterior marginals. We tie the computation of bounds on posterior marginals to bounding a polynomial number of probabilities $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$ where $c_{1:q} = \{c_1, ..., c_q\}$ is an instantiation of a subset of cutset variables. In Section 7, we explore special cases where we can improve the lower and/or upper bound on posterior marginals for select variables over ATB bounds. Section 5 discusses the implementation issues concerning bounding of probabilities $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$ with bound propagation plug-in and searching for h high-probability cutset tuples. We perform empirical evaluation in Section 8 and draw final conclusions in Section 9.

2 Background

DEFINITION **2.1 (belief networks)** *Let $X = \{X_1, ..., X_n\}$ be a set of random variables over multi-valued domains $D(X_1), ..., D(X_n)$. A* belief network (BN) *is a pair (G, P) where G is a directed acyclic graph on X and $P = \{P(X_i|pa_i)|i = 1, ..., n\}$ is the set of conditional probability matrices associated with each X_i. An evidence e is an instantiated subset of variables E.*

DEFINITION **2.2 (Irrelevant Node and Relevant Subnetwork)** *An* irrelevant node *of a node X is a child node Y that is not observed and does not have observed descendants. The* relevant subnetwork *of X is a subnetwork obtained by removing all irrelevant nodes in the network.*

3 Background

3.1 Bounded Conditioning

Bounded conditioning is an any-time scheme for computing posterior bounds in Bayesian networks [9] originating from the loop-cutset conditioning method (see Eq.(1)). Given a loop-cutset C, evidence E, and some node X in a Bayesian network $\mathcal{B}$, the method computes exactly $P(c^i, e)$ and $P(x, c^i, e)$ for h cutset tuples and bounds the rest using prior distribution. The h tuples are selected based on their prior weight $P(c^i)$. It is implied in the paper that the tuples without evidence are enumerated and sorted. In general, it can be accomplished without enumerating all cutset tuples, with or without evidence, using greedy heuristic search.

Let M denote the total number of cutset tuples. In [9], the authors derive bounds from the following formula:

$$P(x|e) = \sum_{i=1}^{M} P(x|c^i, e)P(c^i|e) = \sum_{i=1}^{h} P(x|c^i, e)P(c^i|e) + \sum_{i=h+1}^{M} P(x|c^i, e)P(c^i|e) \quad (2)$$

Setting $\forall i > h,\ P(x|c^i, e) = 0$ in the above expression yields a lower bound on $P(x|e)$:

$$P(x|e) \geq \sum_{i=1}^{h} P(x|c^i, e)P(c^i|e) \quad (3)$$

The conditional probability $P(c^i|e)$ can be obtained by normalizing the joint probabilities $P(c^i, e)$:

$$P(c^i|e) = \frac{P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e) + \sum_{j=h+1}^{M} P(c^j, e)} \quad (4)$$

We can obtain a lower bound on $P(c^i|e)$ by replacing $P(c^j, e)$, $j > h$, in the denominator of the fraction above with the upper bound value $P(c^j)$:

$$P(c^i|e) \geq \frac{P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e) + \sum_{j=h+1}^{M} P(c^j)}$$

Replacing $P(c^i|e)$ in Eq.(3) with a lower bound yields a lower bound on $P(x|e)$:

$$P^L(x|e) = \sum_{i=1}^{h} P(x|c^i, e) \frac{P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e) + \sum_{j=h+1}^{M} P(c^j)} \quad (5)$$

Multiplying $P(x|c^i, e)$ by the numerator, we have:

$$P^L(x|e) = \frac{\sum_{i=1}^{h} P(x|c^i, e)P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e) + \sum_{j=h+1}^{M} P(c^j)} \quad (6)$$

Finally, since $P(x|c^i, e)P(c^i, e) = P(x, c^i, e)$ and since we no longer need to differentiate between indexes i and j, we can transform the numerator in Eq.(6) to obtain a more compact lower bound representation:

$$P^L(x|e) = \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i)} \quad (7)$$

The upper bound is obtained by setting $P(x|c^i, e) = 1$ for $i > h$ and replacing $P(c^i|e)$ with an upper bound:

$$P(x|e) \leq \sum_{i=1}^{h} P(x|c^i, e)P^U(c^i|e) + \sum_{i=h+1}^{M} P^U(c^i|e) \tag{8}$$

For $i \in [1, h]$, we obtain upper bound $P^U(c^i|e)$ from Eq.(4) by dropping the $\sum_{i=h+1}^{M} P(c^i, e)$ from denominator. Substituting the resulting upper bound in Eq.(8) yields:

$$P^U(x|e) \leq \sum_{i=1}^{h} P(x|c^i, e)\frac{P(c^i, e)}{\sum_{j=1}^{h} P(c^j, e) + \sum_{j=h+1}^{M} P(c^j, e)} + \sum_{i=h+1}^{M} P^U(c^i|e) \tag{9}$$

Factoring $P(x|c^i, e)$ into numerator and replacing $P(x|c^i, e)P(c^i, e)$ with $P(x, c^i, e)$ as we did in the derivation of lower bound, we transform Eq.(9) into:

$$P(x|e) \leq \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M} P(c^i, e)} + \sum_{i=h+1}^{M} P^U(c^i|e) \tag{10}$$

The upper bound $P^U(c^i|e)$ for $i \in [h+1, M]$ can be obtained through a series of transformations which we detail in Appendix A yielding the final upper bound on $P(x|e)$:

$$P^U(x|e) \quad = \quad \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} + \sum_{i=h+1}^{M} P(c^i) + \frac{[\sum_{i=h+1}^{M} P(c^i)]^2}{\sum_{i=1}^{h} P(c^i, e)} \tag{11}$$

It should be noted, that in upper bound derivation in [9], the authors separate the h tuples into two groups. The first group contains tuples where bounded conditioning computes exactly both $P(c^i, e)$ and $P(x|c^i, e)$. The second group contains tuples for which bounded conditioning only computes exactly $P(c^i, e)$ and sets $P(x|c^i, e) = 1$. If k, $k < h$, denotes the number of tuples were algorithm computes $P(x|c^i, e)$ exactly while setting $P^L(x|c^i, e) = 0$ and $P^U(x|c^i, e) = 1$ for the rest of $h - k + 1$ tuples, the lower and upper bounds derived in Eq.(7) and Eq.(11) become:

$$P^{L'}(x|e) \quad = \quad \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i)} \tag{12}$$

$$P^{U'}(x|e) \quad = \quad \frac{\sum_{i=1}^{k} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} + \frac{\sum_{i=k+1}^{h} P(c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} + \sum_{i=h+1}^{M} P(c^i) + \frac{[\sum_{i=h+1}^{M} P(c^i)]^2}{\sum_{i=1}^{h} P(c^i, e)} \tag{13}$$

Clearly, $P^L(x|e) \geq P^{L'}$ and $P^U(x|e) \leq P^{U'}$. In the future, we use $P^L(x|e)$ in Eq.(7) $P^U(x|e)$ in Eq.(11) as the basis for comparison with ATB bounds.

The bounds expressed in Eq.(7) and Eq.(11) converge to the exact posterior marginals as $h \to M$. The convergence rate depends on the form of the distribution $P(C|e)$. The scheme was validated in [9] on the example of Alarm network with 37 nodes. Its loop-cutset contains 5 nodes and number of cutset tuples equals $M = 108$. Applied to an instance of the network without evidence, bounded conditioning algorithm produced a small bounds interval, on the order of 0.01 or less, after generating 40 out of 108 cutset instances. However, when processing the same 40 cutset tuples in each instance, the bounds interval length increased as evidence was added. With 3 and 4 nodes assigned, the bounds interval length rose to ≈ 0.15. Hence, while the

empirical results demonstrated the convergence of bounded conditioning, they also showed the deterioration in the quality of the bounds as more nodes are observed.

We can add to the analysis of bounded condtioning that the upper bound in Eq.(11) can become > 1. Dropping the first two summants from Eq.(11), we obtain:

$$P^U(x|e) \geq \frac{[\sum_{i=h+1}^{M} P(c^i)]^2}{\sum_{i=1}^{h} P(c^i, e)}$$

And since $\sum_{i=1}^{h} P(c^i, e) \leq \sum_{i=1}^{M} P(c^i, e) = P(e)$, we get that:

$$P^U(x|e) \geq \frac{[\sum_{i=h+1}^{M} P(c^i)]^2}{P(e)}$$

This shows that $P^U(x|e)$, as defined in [9], can become arbitrarily large when $P(e)$ is small compared to $P(c^i)$. For example, if $\exists i > h, P(c^i) > 0.01$ while $P(e) = 1E{-}15$, then $P^U(x|e) \geq 1E + 11$.

Another limitation of bounded is that prior distribution often offers poor heuristics for finding the tuples with high probability mass which has been observed in importance sampling.

3.2 Bound Propagation

Bound propagation (BdP) [16] is an iterative algorithm that utilizes the local network structure to formulate a linear optimization problem for each node $X_i \in X$ such that the minimum and maximum of the objective function correspond to the upper and lower bounds on posterior marginals $P(x_i|e)$. Let Y denote Markov blanket of node X_i $markov_i = Y = \{Y_1, ..., Y_k\}$. The idea is to compute posterior marginals via:

$$P(x|e) = \sum_{y_1,...,y_k} P(x_i|y_1, ..., y_k)P(y_1, ..., y_k|e) \tag{14}$$

where $P(x|y_1, ..., y_k)$ is an entry in the probability table of X conditioned on the instantiation of variables in its Markov blanket $y_1, ..., y_k$. The joint probabilities $P(y_1, ..., y_k|e)$ over the Markov blanket are unknown, but we know that the sum of all probabilities equals 1:

$$\sum_{y_1,...,y_k} P(y_1, ..., y_k|e) = 1 \tag{15}$$

Further, $\forall Y_j \in Y$, $\forall y_j \in D(Y_j)$, $\sum_{y \setminus Y_j, Y_j = y_j} P(y_1, ..., y_k|e) = P(y_j|e)$. Denoting arbitrary lower and upper bounds on $P(y_i|e)$ by $P^L(y_i|e)$ and $P^U(y_j|e)$ respectively, we can write:

$$P^L(y_j|e) \leq \sum_{y \setminus y_j, Y_j = y_j} P(y_1, ..., y_k|e) \leq P^U(y_j|e) \tag{16}$$

Hence, for each variable X_i, we have a linear optimization problem with the objective function $P(x_i|e)$, defined in Eq.(14), that is minimized and maximized with respect to all $P(y_1, ..., y_k|e)$. For each instance of Markov variables $y = \{y_1, ..., y_k\}$, the $P(y_1, ..., y_k|e)$ is a variable and $P(x_i|y_1, ..., y_k)$ is the corresponding objective function coefficient. The number of variables is exponential in the size of the Markov blanket. The constraints are defined in Eq.(15) (sum-to-1 constraint) and Eq.(16). For each variable Y_j in the Markov blanket of X_i, there will be $|D(Y_j)|$ constraints of type Eq.(16). The total number of constraints equals $1 + \sum_j |D(Y_j)|$.

To clarify that $P(y_1, ..., y_k|e)$ are the variables of the optimization problem, we can denote $z_{y_1,...,y_k} = P(y_1, ..., y_k|e)$. Replacing $P(y_1, ..., y_k|e)$ with $z_{y_1,...,y_k}$ in Eq.(14), we obtain an objective function:

$$P(x|e) = \sum_{y_1,...,y_k} P(x|y_1, ..., y_k) z_{y_1,...,y_k}$$

over exponential number of variables $z_{y_1,...,y_k}$. Then, the sum-to-1 constraint in Eq.(15) tranforms into:

$$\sum_{y_1,...,y_k} z_{y_1,...,y_k} = 1$$

and $\forall Y_j \in Y$, $\forall y_j \in \mathcal{D}(Y_j)$, we will a constraint of the form:

$$P^L(y_j|e) \leq \sum_{y \setminus y_j, Y_j = y_j} z_(y_1, ..., y_k) \leq P^U(y_j|e)$$

Example 1 *Let $markov_i = \{A, B\}$. Let $\mathcal{D}(A) = \{0, 1\}$ and $\mathcal{D}(B) = \{0, 1, 2\}$. Let $P(x_i|A, B)$ be defined as follows:*

$$\begin{aligned} P(x_i|0,0) &= 0.1 \\ P(x_i|0,1) &= 0.2 \\ P(x_i|0,2) &= 0.3 \\ P(x_i|1,0) &= 0.4 \\ P(x_i|1,1) &= 0.5 \\ P(x_i|1,2) &= 0.6 \end{aligned}$$

Then the objective functiontion of the linear optimization problem can be defined as follows:

$$\begin{aligned} P(x|e) &= 0.1P(a=0,b=0|e) + 0.2P(a=0,b=1|e) + 0.3P(a=0,b=2|e) \\ &+ 0.4P(a=1,b=0|e) + 0.5P(a=1,b=1|e) + 0.5P(a=1,b=2|e) \end{aligned}$$

s.t.

$$\begin{aligned} P(a=0,b=0|e) &+ P(a=0,b=1|e) + P(a=0,b=2|e) \\ &+ P(a=1,b=0|e) + P(a=1,b=1|e) + P(a=1,b=2|e) = 1 \end{aligned}$$

$$P^L(a=0|e) \leq P(a=0,b=0|e) + P(a=0,b=1|e) + P(a=0,b=2|e) \leq P^U(a=0|e)$$

$$P^L(a=1|e) \leq P(a=1,b=0|e) + P(a=1,b=1|e) + P(a=1,b=2|e) \leq P^U(a=1|e)$$

$$P^L(b=0|e) \leq P(a=0,b=0|e) + P(a=1,b=0|e) \leq P^U(b=0|e)$$

$$P^L(b=1|e) \leq P(a=0,b=1|e) + P(a=1,b=1|e) \leq P^U(b=1|e)$$

$$P^L(b=2|e) \leq P(a=0,b=2|e) + P(a=1,b=2|e) \leq P^U(b=2|e)$$

The minimum and maximum of the objective function correspond to the lower and upper bounds on $P(x_i|e)$. Initializing bounds $P^L(x_i|e)$ and $P^U(x_i|e)$ to 0 and 1 for all variables, the algorithm solves linear minimization and maximization problems for each variable and updates the bounds. The process is iterated until convergence (namely, until the bounds no longer change). Convergence is guaranteed since with every iteration, the bounds get closer to the posterior marginals or do not change. We summarize algorithm in Figure 1.

The inputs to the algorithm are a Bayesian network $\mathcal{B}$ over $X = \{X_1, ..., X_n\}$ and initial values of lower and upper bounds $P^L(x_i|e)$ and $P^U(x_i|e)$ for each variable X.

The output of the algorithm are the revised lower and upper bounds $P^L(x_i|e)$ and $P^U(x_i|e)$ for each variable $X_i \in X$. In each iteration, inside the repeat loop, for each $X_i inX$ the algorithm first computes a conditional probability table $P(X_i|markov_i)$ and then solves the linear optimization problem for each value $x_i \in \mathcal{D}(X_i)$. After computing min and max of the objective function, the lower and upper bounds $P^L(x_i|e)$ and $P^U(x_i|e)$ are updated.

Bound Propagation
Input: A belief network ($\mathcal{B}$) over variables $X = \{X_1, ..., X_n\}$, evidence $E \subset X$.
Input/Output: lower bounds array LB, upper bounds array UB.
Repeat:
$\forall X_i \in X \backslash E$, $\forall x_i \in \mathcal{D}(X_i)$ do:
1. Compute conditional probability table over Markov blanket $\mathbf{Y} = \{Y_1, ..., Y_k\}$ of X_i:
   $\forall \{y_1, ..., y_k\} \in \mathcal{D}(\mathbf{Y})$ do:
   $P(x_i|y_1, ..., y_k) \leftarrow \alpha P(x_i|pa_i) \prod_j P(ch_j|pa_j)$
2. Define constraints of a linear optimization problem:

$$\sum_Y P(y_1, ..., y_k|e) = 1$$

$\forall Y_j \in \mathbf{Y}$, $\forall y_j \in \mathcal{D}(Y_j)$,

$$P^L(y_j|e) \leq \sum_{y \backslash y_j, Y_j = y_j} P(y_1, ..., y_k|e) \leq P^U(y_j|e)$$

3. Solve the problem using a standard LP simplex algorithm:

$$P^L(x_i|e) \leftarrow \min \sum_{y_1,...,y_k} P(x_i|y_1, ..., y_k, e) P(y_1, ..., y_k|e) \quad (17)$$

$$P^U(x_i|e) \leftarrow \max \sum_{y_1,...,y_k} P(x_i|y_1, ..., y_k, e) P(y_1, ..., y_k|e) \quad (18)$$

Until Converged

Figure 1: Bound Propagation (BdP) Algorithm

The paper showed that BdP performed quite well on Ising grid and regular two-layer networks. In both classes of benchmaks, the size of the variable's Markov blanket was fixed due to the default network structure in case of Ising grid and due to the enforced maximum number of children and maximum number of parents per node. The algorithm was also tested with the Alarm network without evidence. In case of Alarm network, BdP obtained small bounds interval for several nodes but could not obtain good bounds for root nodes 12, 13, 14, 11 although the relative subnetwork of each of these nodes consists of only the node itself and, hence, the posteriors equal the priors. The latter is demonstrative of how BdP exploits local network structure to its advantage while ingoring the global network properties.

In practice, algorithm BdP as presented in [16] is feasible only for networks having bounded Markov blanket size since the number of variables in the optimization problem in Figure 1 grows exponentially with the size of the Markov blanket.

4 Architecture for Any-Time Bounds

In this section, we outline our any-time bounding scheme. It builds on the same principles as bounded conditioning. Namely, given a cutset C and some method

for generating cutset tuples in some order, the probabilities of the first h tuples are evaluated exactly and the rest are upper and lower bounded.

First, we define our notation. Given a subset of variables $C \subset X$, let $o = \{c_1, ..., c_{|C|}\}$ denote an ordering of the cutset variables. Let lower-case $c = \{c_1, ..., c_{|C|}\}$ denote an instantiation of cutset C. Let $M = |\mathcal{D}(C)|$ denote the number of different cutset tuples, namely, the size of the state-space $D(C)$ of cutset C. Indexing tuples 1 through M, we denote c^i, $1 \leq i \leq M$, a particular tuple in that ordering. The tuples c and c^i will always denote an assignment to all variable in the cutset. We will use $c_{1:q}$ and $c^i_{1:q}$ to denote a partial instantiation of cutset variables. Namely, $c_{1:q} = \{c_1, ..., c_q\}$, $q < |C|$, denotes some assignment to the first q variables in cutset C. The superindex i in $c^i_{1:q}$ indicates a partular assignment to the specified subset of cutset variables.

The algorithm computes exactly quantities $P(x, c^i, e)$ and $P(c^i, e)$ for $1 \leq i \leq h$ and bounds the sums $\sum_{i=h+1}^{M} P(x, c^i, e)$ and $\sum_{i=h+1}^{M} P(c^i, e)$ for $i > h$. We will refer to our bounds computation framework as ATB for Any-Time Bounds.

The ATB architecture is founded on two principles. First, given a contant h, it replaces the sums over the tuples c^{h+1},...,c^M with a sum over a polynomial number of partially-instantiated cutset tuples, by grouping together tuples sharing variable value assignments. In particular, we will show how to compute lower and upper bounds on $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$ whose number is bounded polynomially. The detailes are provided in Section 4.1.

Second, we develop new exressions for lower and upper bounds on posterior marginals as a function of the lower and upper bounds on the joint probabilities $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$. We assume in our derivation that there is an algorithm $\mathcal{A}$ that can compute probabilities $P_{\mathcal{A}}(x, c_{1:q}, e)$ and $P_{\mathcal{A}}(c_{1:q}, e)$. We maintain this assumption throughout unless stated otherwise. To simplify notation, we usually omit subindex $\mathcal{A}$.

ATB yilelds an upper bound that is guaranteed to be less than 1. The bounding interval generated is smaller than that generated by bounded conditioning. Thus, as we will show, we obtain a tighter upper bound formulation than Bounded Conditioning that allows to incorporate any bounds on $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$. We defer the problem of selecting high probability cutset tuples to Section 5 and the bounding scheme of $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$ to Section 5.1.

4.1 Polynomial Processing Time

We obtain the any-time bounding scheme using the cutset conditioning formula as a starting formula, similar to the wasy bounded conditioning was developed. Given a Bayesian network $\mathcal{B}$, a cutset C, let $M = \prod_{C_i \in C} |\mathcal{D}(C_i)|$ be the total number of cutset tuples and let h be the number of generated cutset tuples, $0 < h < M$. We can assume without loss of generality that the generated h tuples are the first h cutset tuples. Then, for a node X with $x' \in \mathcal{D}(X)$, we can re-write Eq.(1) separating the summation over the generated tuples 1 through h and the rest as:

$$P(x'|e) = \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{i=h+1}^{M} P(x', c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i, e)} \tag{19}$$

If C is a loop-cutset, $P(x', c^i, e)$ and $P(c^i, e)$, $0 \leq i \leq h$, can be computed in polynomial time. The question is how to compute or bound $\sum_{i=h+1}^{M} P(x', c^i, e)$ and $\sum_{i=h+1}^{M} P(c^i, e)$ without enumerating all tuples c^i, $i > h$.

Consider a fully-expanded search tree of depth $|C|$ over the cutset search space expanded in the order C_1,...,C_l. A path from the root to the leaf at depth $|C|$ corresponds to a cutset tuple. Hence, there is a one-to-one mapping between each leaf at depth $|C|$ and a fully-instantiated cutset tuple. Assume that we mark all the tree edges on paths that correspond to the first generated h cutset tuples. Then the unexpanded tuples c^i, $h+1 \leq i \leq M$, correspond to the unmarked leaves. We can recursively trim all unmarked leaves until only leaves branching out of marked nodes remain, thus producing a truncated search tree.

DEFINITION **4.1 (Truncated Search Tree)** *Given a search tree T covering the search space $\mathcal{H}$ over variables X_1,...,X_n, a* **truncated search tree** *relative to a subset $S \subset \mathcal{D}(X_1) \times ... \times \mathcal{D}(X_n)$ of full assignments, $S = \{x^1, ..., x^t\}$ where $x^j = \{x_1^j, ..., x_n^j\}$, obtained by marking the edges and nodes associated with S and then removing all unmarked edges and nodes except those branching out from marked nodes.*

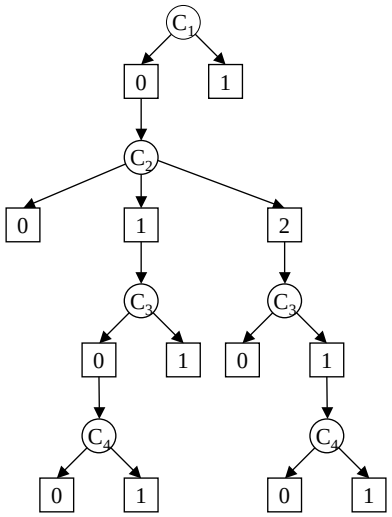

Figure 2: A search tree for cutset $C = \{C_1, ..., C_4\}$.

Therefore, the leaves at depth $< |C|$ in the truncated tree correspond to the partially-instantiated cutset tuples. A path from the root C_1 to a leaf C_q at depth q is a tuple denoted $c_{1:q} = \{c_1, ..., c_q\}$ over those cutset variables which we will refer to as truncated or partially-instantiated cutset tuples, since only variables C_1 through C_q are instantiated while the rest of the variables C_{q+1} through $C_{|C|}$ are not. Full cutset is denoted $c_{1:l}$.

An example of a truncated search tree is shown in Figure 2. Given a cutset of size 4, $C = \{C_1, ..., C_4\}$, $\mathcal{D}(C_1) = \{0, 1\}$, $\mathcal{D}(C_2) = \{0, 1\}$, $\mathcal{D}(C_3) = \{0, 1, 2\}$, $\mathcal{D}(C_4) = \{0, 1\}$ and four fully-instantiated tuples $\{0, 1, 0, 0\}$, $\{0, 1, 0, 1\}$, $\{0, 2, 1, 0\}$, and $\{0, 2, 1, 1\}$, the remaining partially instantiated tuples are $\{c_1 = 0, c_2 = 0\}$, $\{c_1 = 0, c_2 = 1, c_3 = 1\}$, $\{c_1 = 0, c_2 = 2, c_3 = 0\}$, and $\{c_1 = 1\}$.

It is easy to see that the number of truncated tuples, denoted M', is bounded by $O(h \cdot (d-1) \cdot |C|)$, where d is the maximum domain size, since every node C_j in the path from root C_1 to leaf C_l can have no more than $(d-1)$ emanating leaves.

Proposition 1 *If C is a cutset, d bounds the domain size, and h is the number of generated cutset tuples, the number of partially-instantiated cutset tuples in the truncated search three is bounded by $O(h \cdot (d-1) \cdot |C|)$.*

We index the partially instantiated tuples from 1 to M' and denote the j-th tuple $c^j_{1:q_j}$, $1 \leq j \leq M'$, where q_j denotes the tuple's length. Clearly, the probability mass over the cutset tuples c^{h+1}, ..., c^M can be captured via the sum of the truncated tuples. Namely:

Proposition 2

$$\sum_{h+1}^{M} P(c^i, e) = \sum_{j=1}^{M'} P(c^j_{1:q_j}, e) \tag{20}$$

$$\sum_{h+1}^{M} P(x', c^i, e) = \sum_{j=1}^{M'} P(x', c^j_{1:q_j}, e) \tag{21}$$

Therefore, we can bound the ungenerated tuples in Eq.(19) by bounding a polynomial number of summands, over the partially-instantiated tuples.

4.2 Bounds

Now that we have addressed the problem of enumerating the rest of the cutset tuples in polynomial time, we will derive the expressions for bounding the summands in Eq.(20) and Eq.(21). Replacing the summation over tuples $h+1$ through M with summation over the partially-instantiated tuples 1 through M' in Eq.(19), we get:

$$P(x'|e) = \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P(x', c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j}, e)} \tag{22}$$

Assume that we have an algorithm $\mathcal{A}$ that, for any partial assignment $c_{1:q}$, can generate lower and upper bounds $P^L_A(c_{1:q}, e)$ and $P^U_A(c_{1:q}, e)$ and $P^L(x, c^i_{1:q_i}, e)$ and $P^U(x, c^i_{1:q_i}, e)$ s.t. $P^L_A(c_{1:q}, e) \leq P(c_{1:q}, e) \leq P^U_A(c_{1:q}, e)$ and $P^L_A(x, c_{1:q}, e) \leq P(x, c_{1:q}, e) \leq P^U_A(x, c_{1:q}, e)$. In the future derivations, we drop the algorithm's name when there is no confusion.

A brute force lower bound expression using Eq.(22) can be obtained by replacing each $P(x', c^j_{1:q_j}, e)$ with its lower bound (reducing numerator) and each $P(c^j_{1:q_j}, e)$ with its upper bound (increasing denominator) yielding:

$$P(x'|e) \geq \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x', c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)} \triangleq P^{L_1}_{\mathcal{A}}(x'|e) \tag{23}$$

However, a tighter boundd can be obtained if we apply additional transformations to Eq. (22) and prove a helpful lemma. First, we decompose $P(c^j_{1:q_j}, e)$, $0 \leq j \leq M'$, as follows:

$$P(c^j_{1:q_j}, e) = \sum_x P(x, c^j_{1:q_j}, e) = P(x', c^j_{1:q_j}, e) + \sum_{x \neq x'} P(x, c^j_{1:q_j}, e) \tag{24}$$

Replacing $P(c^j_{1:q_j}, e)$ in Eq.(22) with the right-hand size expression in Eq. (24), we obtain:

$$P(x'|e) = \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P(x', c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(x', c^j_{1:q_j}, e) + \sum_{x \neq x'} \sum_{j=1}^{M'} P(x, c^j_{1:q_j}, e)} \quad (25)$$

We will use the following two lemmas:

LEMMA **4.1** *Given positive numbers $a > 0$, $b > 0$, $\delta \geq 0$, if $a < b$, then: $\frac{a}{b} \leq \frac{a+\delta}{b+\delta}$.*
■

LEMMA **4.2** *Given positive numbers a, b, c, c^L, c^U, if $a < b$ and $c^L \leq c \leq c^U$, then:*

$$\frac{a + c^L}{b + c^L} \leq \frac{a + c}{b + c} \leq \frac{a + c^U}{b + c^U}$$

■

The proof of both lemmas is straight forward. Lemma 4.1 merely states a simple mathematical fact that is easily obtained by hand by computing the difference between two values. Lemma 4.2 applies the result of Lemma 4.1 to a particular type of fraction which, as we will see, occurs often in our bounds derivation. Namely, Lemma 4.2 says that if the sums in numerator and denominator have some component c in common, then replacing c with a larger value in both numerator and denominator yields a larger fraction. Replacing c with a smaller value in both places yields a smaller fraction.

Observe now that in Eq.(25) the sums in both numerator and denominator contain component $P(x', c^j_{1:q_j}, e)$. Hence, we can apply Lemma 4.2.

We will obtain a lower bound by replacing summant $P(x', c^i_{1:q_i}, e)$ in Eq.(22) with a lower bound in both numerator and denominator:

$$P(x'|e) \geq \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x', c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x', c^j_{1:q_j}, e) + \sum_{x \neq x'} \sum_{j=1}^{M'} P(x, c^j_{1:q_j}, e)} \quad (26)$$

and replacing $P(x, c^j_{1:q_j}, e)$, $x \neq x'$, with its upper bound:

$$P(x'|e) \geq \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x', c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x', c^j_{1:q_j}, e) + \sum_{x \neq x'} \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x, c^j_{1:q_j}, e)} \triangleq P^{L_2}_{\mathcal{A}}(x'|e) \quad (27)$$

Hence, we have obtained two expressions for lower bound on $P(x'|e)$, P^{L_1} defined in Eq. (23) and P^{L_2} defined in Eq. (23). In general case, neither bound dominates the other. In Section 4.3, we will define conditions when we can predict that the lower bound P^{L_2} is greater than P^{L_1}. In particular, P^{L_2} always dominates P^{L_1} if $|\mathcal{D}(X)| = 2$.

The upper bound formulation can be obtained in a similar manner. Again, we note that both numerator and denominator in Eq.(25) contain summants $P(x', c^j_{1:q_j}, e)$. Subsequently, from Lemma 4.2, replacing each $P(x', c^j_{1:q_j}, e)$ with a corresponding upper bound $P^U_A(x', c^j_{1:q_j}, e)$ in Eq.(25) yields an upper bound on $P(x'|e)$:

$$P(x'|e) \leq \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x', c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x', c^j_{1:q_j}, e) + \sum_{x \neq x'} \sum_{j=1}^{M'} P(x, c^j_{1:q_j}, e)} \quad (28)$$

The expression above still contains unknowns $P(x, c^j_{1:q_j}, e)$, $x \neq x'$. Replacing each $P(x, c^j_{1:q_j}, e)$, $x \neq x'$, with a lower bound (reducing denominator), we obtain an upper bound P^{U_1} on $P(x'|e)$:

$$P(x'|e) \leq \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x', c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(x', c^j_{1:q_j}, e) + \sum_{x \neq x'} \sum_{j=1}^{M'} P^L_{\mathcal{A}}(x, c^j_{1:q_j}, e)} \triangleq P^{U_1}_{\mathcal{A}}(x'|e) \tag{29}$$

Unlike the derivation in bounded conditioning, the upper bound above is guaranteed to be ≤ 1 for any lower and upper bounds on $P(x, c^j_{1:q_j}, e)$.

The derivation of bounds for cutset nodes is similar to the above. The main difference in the formulation of the bounds for a cutset node C_k is that the enumeration is over a subspace of cutset space $Z = C \backslash C_k$. And since the number of generated cutset tuples with different values of C_k maybe different, the number of remaining partially-instantiated tuples for different values of C_k maybe different as well. Hence, we use subindex c_k in the number of fully-instantiated tuples h_{c_k} and the number of partially-instantiated tuples M'_{c_k} to indicate enumeraiton over the tuples where $C_k = c_k$. We provide details in Appendix B. Here, we only summarize results. For any node $C_k \in C$, an upper bound on $P(c'_k|e)$ can be expressed as follows:

$$P^{U_1}_{\mathcal{A}}(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c_k}} P^U_{\mathcal{A}}(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c_k}} P^U_{\mathcal{A}}(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P^L_{\mathcal{A}}(c_k, z^j_{1:q_j}, e)} \tag{30}$$

The two expressions for lower bounds, corresponding to $P^{L_1}(x|e)$ and $P^{L_2}(x|e)$ are:

$$P^{L_1}_{\mathcal{A}}(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L_{\mathcal{A}}(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)} \tag{31}$$

$$P^{L_2}_{\mathcal{A}}(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L_{\mathcal{A}}(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P^U_{\mathcal{A}}(c_k, z^j_{1:q_j}, e)} \tag{32}$$

Next, we demonstrate the computation of ATB bounds on a simple example.

Example 2 *Assume we have a Bayesian network $\mathcal{B}$ with cutset $C = \{C_1, ..., C_4\}$. Assume all cutset nodes have domains of size 2 except for node C_3 which has a domain of size 3. So, $\mathcal{D}(C_1) = \mathcal{D}(C_2) = \mathcal{D}(C_4) = \{0, 1\}$ and $\mathcal{D}(C_3) = \{0, 1, 2\}$. The resulting total number of cutset tuples $M = 24$. Let X be some node in $\mathcal{B}$. We will demonstrate here the computation of the simple lower and upper bounds from ATB framework on $P(x|e)$ for some value $x' \in \mathcal{D}(X)$. Assume we generate four, $h = 4$, cutset tuples: $c^1 = \{0, 1, 0, 0\}$, $c^2 = \{0, 1, 0, 1\}$, $c^3 = \{0, 2, 1, 0\}$, $c^4 = \{0, 2, 1, 1\}$.*

The corresponding truncated search tree is shown in Figure 2. For tuple $\{c^1_1, c^2_2, c^1_3, c^1_4\}$, we compute exactly probabilities $P(x, 0, 1, 0, 0, e)$ and $P(c^1_1, c^2_2, c^1_3, c^1_4)$. Similarly, we obtain exact probabilities $P(x, c^1_1, c^2_2, c^1_3, c^2_4)$ and $P(c^1_1, c^2_2, c^1_3, c^2_4)$ for the second cutset instance $\{c^1_1, c^2_2, c^1_3, c^2_4\}$. Since $h = 4$, the $\sum_{i=1}^{h} P(x, c^i, e)$ and $\sum_{i=1}^{h} P(c^i, e)$ used in lower and upper bounds computation are instantiated as follows:

$$\sum_{i=1}^{4} P(x', c^i, e) = P(x', c^1, e) + P(x', c^2, e) + P(x', c^3, e) + P(x', c^4, e)$$

Any-Time Bounds Architecture

Input: A belief network ($\mathcal{B}$), variables X, evidence $E \subset X$, cutset $C \subset X \backslash E$, truncated search tree T.

Output: lower bound $LB1$, lower bound $LB2$, upper bound UB.

Let $S_{1:h}$ denote $\sum_{i=1}^{h} P(c^i, e)$ and $S_{x,1:h}(x_k)$ denote $\sum_{i=1}^{h} P(x_k, c^i, e)$.

Initialize: $S_{1:h} \leftarrow 0$, $\forall X_k \in X \backslash C, E$, $\forall x_k \in \mathcal{D}(X_k)$, $S_{x,1:h}(x_k) \leftarrow 0$.

1. Generate cutset tuples:

For $i = 1$ to h do:

$c^i \leftarrow GetNextFullyInstantiatedTuple(T)$

$S_{1:h} \leftarrow S_{1:h} + P(c^i, e)$

$\forall c_k \in C$: $S_{x,1:h}(c_k^i) \leftarrow S_{x,1:h}(c_k^i) + P(c^i, e)$

$\forall X_k \in X \backslash C, E$, $\forall x_k \in \mathcal{D}(X_k)$: $S_{x,1:h}(x_k) \leftarrow S_{x,1:h}(x_k) + P(x_k, c^i, e)$

End For

2. Traverse partially-instantiated tuples:

Let $S_{1,M'}$ denote $\sum_{j=1}^{M'} P^U(c^i, e)$, $LB(x_k)$ and $UB(x_k)$ denote $\sum_{j=1}^{M'} P^L(x_k, c^i, e)$, and $\sum_{j=1}^{M'} P^U(x_k, c^i, e)$.

Initialize: $S_{1,M'} \leftarrow 0$; $\forall X_k \in X \backslash C, E$, $\forall x_k \in \mathcal{D}(X_k)$, $LB(x_k) \leftarrow 0$, $UB(x_k) \leftarrow 0$.

While ($c_{1:q_j}^j \leftarrow GetNextPartiallyInstantiatedTuple(T)$) do:

$S_{1,M'} \leftarrow S_{1,M'} + P^U(c_{1:q_j}^j, e)$

$\forall C_k \in C$, $k \leq q_j$ do:

$LB(c_k^j) \leftarrow LB(c_k^j) + P^L(c_{1:q_j}^j, e)$

$UB(c_k^j) \leftarrow UB(c_k^j) + P^U(c_{1:q_j}^j, e)$

$\forall C_k \in C$, $k > q_j$, $\forall c_k \in \mathcal{D}(C_k)$ do:

$LB(c_k) \leftarrow LB(c_k) + P^L(c_k, c_{1:q_j}^j, e)$

$UB(c_k) \leftarrow UB(c_k) + P^U(c_k, c_{1:q_j}^j, e)$

$\forall X_k \in X \backslash C, E$, $\forall x_k \in \mathcal{D}(X_k)$ do:

$LB(x_k) \leftarrow LB(x_k) + P^L(x_k, c_{1:q_j}^j, e)$

$UB(x_k) \leftarrow UB(x_k) + P^U(x_k, c_{1:q_j}^j, e)$

End While

3. Compute bounds:

$$LB[k][d] = \max \left\{ \begin{array}{l} \frac{S_{x,1:h} + LB(x_k^d)}{S_{1:h} + S_{1:M'}} \\ \frac{S_{x,1:h} + LB(x_k^d)}{S_{1:h} + LB(x_k^d) + \sum_{r \neq d} UB(x_k^d)} \end{array} \right.$$

$$UB[k][d] = \frac{S_{x,1:h} + UB(x_k^d)}{S_{1:h} + UB(x_k^d) + \sum_{r \neq d} LB(x_k^d)}$$

Figure 3: Any-Time Bounds Architecture

$$\sum_{i=1}^{4} P(c^i, e) \quad = \quad P(c^1, e) + P(c^2, e) + P(c^3, e) + P(c^4, e)$$

The remaining partial tree pathes are: $c^1_{1:2} = \{0, 0\}$, $c^2_{1:3} = \{0, 1, 1\}$, $c^3_{1:3} = \{0, 2, 1\}$, *and* $c^4_{1:1} = \{1\}$. *Since these four tuples are partial instantiations of cutset nodes, we compute bounds on joint probabilities instead of exact values. For example, for the tuple* $\{0, 0\}$, *we compute bounds on* $P(x, 0, 0, e)$ *and* $P(0, 0, e)$. *Hence the sums over the partially instantiated tuples will have the form:*

$$S^U(x) = P^U(x, c^1_{1:2}, e) + P^U(x, c^2_{1:3}, e) + P^U(x, c^3_{1:3}, e) + P^U(x, c^4_{1:1}, e)$$

$$S^L(x) = P^L(x, c^1_{1:2}, e) + P^L(x, c^2_{1:3}, e) + P^L(x, c^3_{1:3}, e) + P^L(x, c^4_{1:1}, e)$$

From Eq.(29) we get:

$$P^{U_1}(x'|e) = \frac{\sum_{i=1}^{4} P(x', c^i, e) + S^U(x')}{\sum_{i=1}^{4} P(c^i, e) + S^U(x') + \sum_{x \neq x'} S^L(x)}$$

From Eq.(27) we get:

$$P^{L_2}(x'|e) = \frac{\sum_{i=1}^{4} P(x, c^i, e) + S^L(x')}{\sum_{i=1}^{4} P(c^i, e) + S^L(x') + \sum_{x \neq x'} S^U(x)}$$

Thus, we generated 4 cutset tuples and 4 partial cutset tuples. The total number of tuples processed is $M' = 4 + 4 = 8 << M = 24$.

In summary, we have described a framework for computing bounds in an any-time fashion which we call ATB for Any-Time Bounds. We provide the complete algorithm for ATB in Figure 3. The upper bound is defined via Eq.(29). The equivalent upper bound for cutset nodes is defined in Eq.(30). The lower bounds are defined via Eq.(23) and Eq.(27) since neither of those two bounds is dominant in general case. The equivalent lower bounds for cutset nodes are defined in Eq.(31) and Eq.(32). The idea is to generate h cutset tuples and compute exactly the corresponding $P(x, c, e)$ and $P(c, e)$. If C is a loop-cutset, those quantities can be obtained in linear time by an inference algorithm (e.g. belief propagation). We use an off-the-shelf algorithm $\mathcal{A}$ to bound $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$ in all partially instantiated cutset tuples in the truncated search tree. Since the number of partially observed cutset tuples grows polynomially with h, the ATB framework has a polynomial time complexity:

THEOREM **4.1 (Time Complexity)** *Given an algorithm deriving lower and upper bounds to any collection of variable instantiation and assuming it takes at most* T *time to bound* $P(c_{1:q_i}, e)$ *and* $P(x, c_{1:q_i}, e)$, *the ATB takes* $O(T \cdot h \cdot (d-1) \cdot |C|)$ *time to bound partially-instantiated cutset tuples. If* N *is the problem input size, the total complexity of ATB is the time of ATB is* $O(h \cdot N + T \cdot h \cdot (d-1) \cdot |C|)$ *where* d *is the maximum domain size.*

Theorem 4.1 follows immediately from Proposition 1.

In the next Section 4.3, we compare expressions for lower bounds in Eq.(23) and Eq.(27) and define the conditions when lower bound $P^{L_2}_{\mathcal{A}}$ cominates.

4.3 Properties of Lower Bounds

We can show that the lower bound $P^{L_2}_{\mathcal{A}}$ value in Eq.(27) is larger than the brute force lower bound $P^{L_1}_{\mathcal{A}}$ expression in Eq.(23) under certain conditions. In particular, $P^{L_2}_{\mathcal{A}}$ is guaranteed to dominate $P^{L_1}_{\mathcal{A}}$ when node X has domain of size 2.

THEOREM **4.2 (Lower Bound Dominance)** *If* $P^L(x, c^j_{1:q_j}, e)$ *and* $P^U(x, c^j_{1:q_j}, e)$ *are defined as follows:*

$$\forall x \in \mathcal{D}(X), P^L(x, c^j_{1:q_j}, e) = P^L(x|c^j_{1:q_j}, e)P^L(c^j_{1:q_j}, e)$$

$$\forall x \in \mathcal{D}(X), P^U(x, c^j_{1:q_j}, e) = P^U(x|c^j_{1:q_j}, e)P^U(c^j_{1:q_j}, e)$$

and

$$P^L(x'|c^j_{1:q_j}, e) = 1 - \sum_{x \neq x'} P^U(x|c^j_{1:q_i}, e) \tag{33}$$

then $P^{L_1}(x'|e) \geq P^{L_2}(x'|e)$ *where* $P^{L_1}(x'|e)$ *and* $P^{L_2}(x'|e)$ *are defined in Eq.(23) and Eq.(27) respectively.*

The proof is straight forward and is provided in Appendix C.

Corollary 1 (Lower Bound Dominance) *If* $\mathcal{D}(X) = 2$ *and* $P^L(x, c^j_{1:q_j}, e)$ *and* $P^U(x, c^j_{1:q_j}, e)$ *are defined as follows:*

$$\forall x \in \mathcal{D}(X), P^L(x, c^j_{1:q_j}, e) = P^L(x|c^j_{1:q_j}, e)P^L(c^j_{1:q_j}, e)$$

$$\forall x \in \mathcal{D}(X), P^U(x, c^j_{1:q_j}, e) = P^U(x|c^j_{1:q_j}, e)P^U(c^j_{1:q_j}, e)$$

then $P^{L_1}(x'|e) \geq P^{L_2}(x'|e)$ *where* $P^{L_1}(x'|e)$ *and* $P^{L_2}(x'|e)$ *are defined in Eq.(23) and Eq.(27) respectively.*

The proof is straight forward and is provided in Appendix C. The corollary follows from observation that the equality condition in Eq.(33) can be always enforced in nodes with domains of size 2. If $P^L(x'|c^i_{1:q_i}, e) < 1 - \sum_{x \neq x'} P^U(x|c^i_{1:q_i}, e)$, then we can safely increase lower bound so that the equality stands. If $P^L(x'|c^i_{1:q_i}, e) > 1 - \sum_{x \neq x'} P^U(x|c^i_{1:q_i}, e)$, then we can adjust upper bound value.

However, the lower bound P^{L_1} can dominate lower bound P^{L_2} when node's domain size is greater than 2. If $P^L(x'|c^i_{1:q_i}, e) < 1 - \sum_{x \neq x'} P^U(x|c^i_{1:q_i}, e)$, we can increase the lower bound and set $P^L(x'|c^i_{1:q_i}, e) = 1 - \sum_{x \neq x'} P^U(x|c^i_{1:q_i}, e)$ same as in case of bi-valued variables. However, if $P^L(x'|c^i_{1:q_i}, e) > 1 - \sum_{x \neq x'} P^U(x|c^j_{1:q_j}, e)$, the P^{L_1} could yield a higher value. Hence, in practice, we should compute both bounds for nodes with domains of size > 2 and pick the highest value.

Next, we compare the ATB bounds and the bounds obtained in the bounded conditioning [9].

4.4 Comparing ATB bounds and Bounded Conditioning Bounds

In the ATB framework, the lower bounds are defined by expressions in Eq.(23) and Eq.(27). Let us denote by BF a brute-force algorithm that trivially instantiates $P^L(x', c^j_{1:q_j}, e) = 0$, $P^U(x', c^j_{1:q_j}, e) = 1$, and $P^U(c^j_{1:q_j}, e) = P(c^j_{1:q_j})$. Then, from Eq.(23), we get:

$$P^{L_1}_{BF}(x'|e) = \frac{\sum_{i=1}^{h} P(x', c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j})} = \frac{\sum_{i=1}^{h} P(x', c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M} P(c^j)} \tag{34}$$

Using algorithm BF with lower bound P^{L_2} expressed in Eq.(27), we get:

$$P_{BF}^{L_2}(x'|e) = \frac{\sum_{i=1}^{h} P(x', c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j})} = \frac{\sum_{i=1}^{h} P(x', c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M} P(c^j)} \tag{35}$$

The right hand side in the equations (34) and (35) equals the expression for the bounded conditioning lower bound in Eq.(7). Namely, $P_{BF}^{L_1}(x'|e) = P_{BF}^{L_1}(x'|e) = P^L(x'|e)$ where $P^L(x'|e)$ is obtained via Eq.(7). Clearly, both bounds $P_{BF}^{L_1}(x'|e)$ and $P_{BF}^{L_2}(x'|e)$ are superior when compared to bounded conditioning as long as we can compute at least some non-zero lower bounds on $P^L(x', c^i_{1:q_i}, e)$ and/or better than prior $P(c_{1:q})$ upper bounds on $P^U(x, c^j_{1:q_j}, e)$ and $P^U(c^j_{1:q_j}, e)$.

We can prove in a similar manner that the ATB upper bound $P^{U_1}(x'|e)$ is as good or better than the upper bound obtained via bounded conditioning. Applying the brute-force algorthm BF, defined above, to bound $P(x', c^j_{1:q_j}, e)$, and $P(c^i_{1:q_i}, e)$, we get from Eq.(29):

$$P_{BF}^{U_1}(x'|e) = \frac{\sum_{i=1}^{h} P(x', c^i, e) + + \sum_{j=1}^{M'} P(c^j_{1:q_j})}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j})} = \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=h+1}^{M} P(c^j)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M} P(c^j)} \tag{36}$$

The expression for $P_{BF}^{U_1}$ gives us the worst-case upper bound that can be obtained by ATB from Eq.(29). In the next theorem, we prove that the upper bound $P_{BF}^{U_1}(x'|e)$ is as good or better than the bounded conditioning upper bound. Namely, $P_A^{U_1}$ dominates the bounded conditioning as long as $P_A^U(x', c^j_{1:q_j}, e) \leq P^U(c^j_{1:q_j})$:

THEOREM **4.3** *Given an algorithm $\mathcal{A}$ that computes lower and upper bounds $P_A^L(x, c^j_{1:q_j}, e)$ and $P_A^U(x, c^j_{1:q_j}, e)$ such that $\forall j$, $P_A^U(x, c^j_{1:q_j}, e) \leq P(c^j_{1:q_j})$ then $P_{\mathcal{A}}^{U_1}(x|e) \leq P^U(x|e)$ where $P_{\mathcal{A}}^{U_1}(x|e)$ is given in Eq.(29) and $P^U(x|e)$ is the bounded conditioning expression given in Eq.(11).*

The proof is provided in Appendix C.

So far, we have assumed that we have an algorithm $\mathcal{A}$ for computing bounds on $P(c_{1:q}, e)$ and $P(x, c_{1:q}, e)$. In the next section, we derive the expressions for ATB bounds for a special case when the algorithm $\mathcal{A}$ can only compute upper bounds on $P(c_{1:q}, e)$. Our motivation is that even if bounding $P(x, c_{1:q}, e)$ via algorithm $\mathcal{A}$ is possible, it maybe computationally infeasible.

4.5 Weak ATB Bounds Framework (ATB^w)

In this section we derive bounds expressions for a weak form of ATB framework, denoted ATB^w. We asumme here that we plug in a bounding scheme $\mathcal{A}$ that only computes upper bound $P^U(c_{1:q}, e)$ on $P(c_{1:q}, e)$. We assume that algorithm $\mathcal{A}$ either cannot compute bounds on $P(x, c_{1:q}, e)$ or incurs a lot of computational overhead doing so. In practice, we may want to avoid this overhead and use the time to generate more cutset tuples (increase h).

Since $\mathcal{A}$ cannot produce non-trivial bounds for $P(x, c_{1:q}, e)$, it instantiates $P_{\mathcal{A}}^L(x, c_{1:q}, e) = 0$ and, since $P(x, c_{1:q}, e) \leq P(c_{1:q}, e)$, it instantiates $P_{\mathcal{A}}^U(x, c_{1:q}, e) = P_{\mathcal{A}}^U(c_{1:q}, e)$. Plugging in lower bound 0 and upprer bound $P(c_{1:q}, e)$ on $P(x, c_{1:q}, e)$

into Eq.(23), we get:

$$P^{L_3}_{\mathcal{A}}(x'|e) \triangleq \frac{\sum_{i=1}^{h} P(x', c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=h+1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)} \tag{37}$$

Similarly, replacing lower and upper bounds on $P(x, c_{1:q}, e)$ with 0 and $P^U_{\mathcal{A}}(c_{1:q}, e)$ in the upper bound expression in Eq. (29), we obtain:

$$P^{U_3}_{\mathcal{A}}(x'|e) \triangleq \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^j_{1:q_j}, e)} \tag{38}$$

Note that the expressions for lower and upper bounds in Eq.(37) and Eq.(38) above depend only on the values $P(c^i, e)$ and $P(x, c^i, e)$ obtained via exact computation over the first h cutset tuples and the upper bound values $P^U_{\mathcal{A}}(c_{1:q}, e)$ for the partially instantiated cutset tuples. Despite simplifications, the framework ATB^w is guaranteed to produce as good or better bounds as those obtained by bounded conditioning. We obtain the proof by plugging into ATB^w the brute-force bounding scheme BF described previously.

The following theorem characterizes the convergence rate of ATB^w bounds interval.

THEOREM **4.4** *Given an algorithm $\mathcal{A}$ that can compute an upper bound on $P(c_{1:q}, e)$, where $c_{1:q}$ is a partial cutset instantiation, given h fully-instantiated cutset tuples c^i, $1 \leq i \leq h$, then:*

$$P^{U_3}_{\mathcal{A}} - P^{L_3}_{\mathcal{A}} \geq \frac{\sum_{i=1}^{h} P(c^i, e)}{P(e)}$$

where $P^{L_3}_{\mathcal{A}}$ and $P^{U_3}_{\mathcal{A}}$ are expressed in Eq.(37) and Eq.(38) respectively.

It is clear that for the same h and for the same plugin algorithm $\mathcal{A}$, the bounds obtained by ATB framework will be as good or better than those computed by ATB^w. The only advantage ATB^w has over ATB is faster computation time. We investigate in the empirical section the trade-offs between plugging in tighter bounds into ATB framework and computing more cutset tuples using ATB^w framework.

5 Incorporating Bounds Propagation

In Section 4 we defined the Any-Time Bounds framework. Founded on the conditioning principles, the framework computes exactly $P(x, c, e)$ and $P(c, e)$ for h cutset tuples and then uses precomputed bounds on the probabilities $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$ for the remaining partially instantiated cutset tuples which will be plugged into the corresponding expressions. The framework does not specify how to bound the joint probabilities for the partially instantiated cutset tuples. In this section, in subsection 5.1, we present a method for bounding probabilities $P(x, c_{1:q}, e)$ and $P(c_{1:q}, e)$ that is based on bound propagation.

As we mentioned earlier, the performance of bound propagation depends on the size of the Markov blanket since the size of the corresponding linear optimization problems grows exponentially with the number of variables in the Markov blanket. In Section 5.2, we describe a simple optimization which allows to reduce node's Markov blanket size and, consequently, improve the performance of BdP by restricting Markov blanket to its relevant subnetwork. We will denote the improved version of BdP via BdP+.

We can reduce the BdP (and BdP+) computation time further by using solving linear optimization problem approximately, trading accuracy for speed. We describe our approximation scheme in Section 5.3.

Although other off-the-shelf bounding schemes exist, such as bounding scheme proposed in [15], we choose bound propagation because it is simpler and because we expect that it can compute tighter bounds in the context of ATB framework than standalone. Since instantiating some nodes results in the reduction of the Markov blanket size of all the neighboring nodes, bound propagation maybe able to compute tighter bounds within ATB framework when network is additinally conditioned on $c_{1:q}$.

5.1 Bounding $P(c_{1:q}, e)$ and $P(x, c_{1:q}, e)$ using BdP

In this section, we show how the ATB framework can be integrated with a specific off-the-shelf algorithm, bound propagation, that computes bounds on posterior marginals using bounds propagation algorithm which only computes bounds on conditional probabilities $P(x|e)$ for any variable X for any evidence e.

We cannot plugin bound propagation directly because it only bounds conditional probabilities. In order to use BdP for computing and cannot be applied directly to bound $P(c_{1:q}, e)$ and $P(x, c_{1:q}, e)$. In order to use BdP for computing bounds on joint probabilities, we factorize the joint probability $P(c_{1:q}, e)$ as follows:

$$P(c_{1:q}, e) = \prod_{e_j \in E} P(e_j|e_1, ..., e_{j-1}, c_{1:q})P(c_{1:q})$$

We know how to compute $P(c)$ if c is an assignment to all loop-cutset variables. We can also compute joint probability $P(c_{1:q})$ where $c_{1:q} = \{c_1, ..., c_q\}$, $q < |C|$, is a partial assignment to loop-cutset variables if $c_{1:q}$ contains the first q cutset nodes (in topological order) utilizing the notion of irrelevant subnetwork.

THEOREM **5.1** *If C is a topologically ordered loop-cutset of a Bayesian network and $C_{1:q} = \{C_1, ..., C_q\}$ is a subset of C, $q < |C|$, then the relevant subnetwork of $C_{1:q}$ consisting of loop-cutset nodes in subset $C_{1:q}$ and their ancestors is singly-connected.*

Since the relevant subnetwork over $c_{1:q}$ is singly-connected, we can compute joint $P(c_1, ..., c_q)$ in linear time using belief updating algorithm.

We can now apply alogorithm BdP to the network $(\mathcal{B}, c_{1:q}, e)$ and obtain bounds on $P(e_1|c_{1:q})$, then on $P(e_2|e_1, c_{1:q})$, and in sequence get bounds $P^L_{BdP}(e_j|e_1, ..., e_{j-1}, c_{1:q})$ and $P^U_{BdP}(e_j|e_1, ..., e_{j-1}, c_{1:q})$ that include all the evidence. Thus:

$$P(c_{1:q}, e) \geq \prod_{e_j \in E} P^L_{BdP}(e_j|e_1, ..., e_{j-1}, c_{1:q})P(c_{1:q}) \triangleq P^L_{BdP}(c_{1:q}, e) \quad (39)$$

$$P(c_{1:q}, e) \leq \prod_{e_j \in E} P^U_{BdP}(e_j|e_1, ..., e_{j-1}, c_{1:q})P(c_{1:q}) \triangleq P^U_{BdP}(c_{1:q}, e) \quad (40)$$

The joint probability $P(x, c_{1:q}, e)$ can be factorized as:

$$P(x, c_{1:q}, e) = P(x|c_{1:q}, e)P(c_{1:q}, e)$$

Using the decomposition above, we can obtain lower and upper bounds on $P(x, c_{1:q}, e)$ as follows:

$$P^L_{BdP}(x, c_{1:q}, e) = P^L_{BdP}(x|c_{1:q}, e)P^L_{BdP}(c_{1:q}, e) \quad (41)$$

$$P^U_{BdP}(x, c_{1:q}, e) = P^U_{BdP}(x|c_{1:q}, e)P^U_{BdP}(c_{1:q}, e) \quad (42)$$

where $P^L_{BdP}(x|c_{1:q}, e)$ and $P^U_{BdP}(x|c_{1:q}, e)$ are obtained by algorithm BdP directly and values $P^L_{BdP}(c_{1:q}, e)$ and $P^U_{BdP}(c_{1:q}, e)$ are obtained from Eq.(39) and Eq.(40) .

The cost of computing $P(c_{1:q})$ is linear in problem size N. Let $O(m)$ bound the complexity of algorithm BdP. Then, the total cost of bounding $P(c_{1:q}, e)$ is $O(N + m \cdot (1 + |E|))$. Therefore, the complexity of ATB when it uses BdP to precompute input bounds is given next:

THEOREM **5.2** *Given a Bayesian network $\mathcal{B}$ over N variables with maximum domain size d, the complexity of ATB with bounds precomputed by algorithm BdP is $O((N + m \cdot (1 + |E|)) \cdot h \cdot (d - 1) \cdot |C|)$ when complexity of BdP is $O(m)$.*

Next, we describe how we can improve the performance of BdP reducing the bounds interval and the computation time by taking into account the network structure.

5.2 Optimizing BdP

As we have already mentioned, the time and memory requirements of BdP algorithm are dependent on the maximum size of the Markov blanket which, in turn, determines the maximum linear optimization problem size. In order to limit the BdP demands for memory and time, we can impose a bound on the maximum size of the Markov blanket and, thus, the maximum size of the linear optimization problem that we solve. When the Markov blanket size of a node X exceeds the maximum, the node's lower and upper bound values will remain equal to their input values (usually, 0 and 1). Of course, it will affect the quality of the bounds for neighboring nodes as well. As we described earlier, each node Y_i in the Markov blanket of X induces a linear constraint of the form:

$$P^L(y_j|e) \leq \sum_{y \setminus y_j, Y_j = y_j} P(y_1, ..., y_k|e) \leq P^U(y_j|e) \tag{43}$$

Clearly, the quality of the bounds on X depends on the tightness of the bounds of the nodes in its Markov blanket. When the lower and upper bounds $P^L(y_j|e)$ and $P^U(y_j|e)$ are 0 and 1 respectively, then $\forall y_j \in Y_i$, the constraints expressed in Eq.(43) become redundant with respect to the sum-to-1 constraint $\sum_y P(y_1, ..., y_k|e) = 1$. The sum of a subset of variables is always non-negative because all variables are non-negative. It is never greater than 1 because the sum of all variables equals 1.

The performance of the bound propagation algorithm can be improved by identifying the irrelevant child nodes and restricting the Markov blanket of X to its relevant subnetwork.

DEFINITION **5.1 (Irrelevant Node and Relevant Subnetwork)** *An* irrelevant node *of a node X is a child node Y that is not observed and does not have observed descendants. The* relevant subnetwork *of X is a subnetwork obtained by removing all irrelevant nodes in the network.*

Removing irrelevant nodes (and their parents) from Markov blanket whenever possible results in a smaller Markov blanket size, shorter computation time, and more accurate bounds which we demonstrate in the empirical section. If the relevant subnetwork of node X is singly-connected then its posteriors should be computed exactly and used to initialize the bound propagation bounds for node X. We denote as BdP+ the bound propagation algorithm that takes advantage of the network structure as described above.

Of course, BdP+ optimizations are obvious and do not resolve the problem of computing bounds for variables whose Markov blanket remains very large even

when restricted to its relevant subnetwork. We propose that we can reduce the number of nodes with 0/1 output bounds by pluggin BdP+ into ATB framework. When we assign values to nodes, the size of the Markov blanket of the neighboring nodes decreases. Thus, in a network that is conditioned on a subset of cutset nodes, in addition to the evidence nodes E, the size of the Markov blanket of all nodes neighboring the cutset nodes will become smaller. Alternatively, we can directly condition on the nodes with large Markov blankets by including them into the cutset C. For example, the primary heuristics in selecting a minimal loop-cutset is the degree of a node [3]. We can cut the primary heuristics ties by selecting the node with larger Markov blanket. More aggressively, we can use Markov blanket size as the primary heuristics.

5.3 Approximating the LP in Bound Propagation

We have defined a scheme that allows us to bound $P(c_{1:q}, e)$ and $P(x, c_{1:q}, e)$, using a combination of bound propagation, used to bound conditional probabilities $P(e_j|e_1, ..., e_{j-1}, c_{1:q})$ and $P(x|c_{1:q}, e)$, and exact inference, used to compute $P(c_{1:q})$. In this section, we propose an algorithm for solving the linear optimization problem approximately, instead of using a simplex solver, to reduce the computation time of the plugin.

When BdP+ scheme is plugged into the ATB framework, we need to bound a large number of tuples $P(x, c^i_{1:q_i}, e)$. We need to invoke bound propagation algorithm $1+|E|$ times to bound one tuple. For every tuple, we need to solve $O(|X\backslash E|)$ linear optimization problems. Thus, within ATB framework, we need to solve thousands if not hundreds of thousands of linear optimisation problems via simplex method which can be impractical timewise. Furthemore, the linear optimize problems formulated in bound propagation algorithm fall into a class of linear packing and covering problems which are known to be especially challening for simplex method [7]. We observed some of this behavior in two of our benchmarks, cpcs360b and cpcs422b. The bound propagation computation time increased by nearly an order of magnitude when input lower and upper bounds were tight (using ATB output as input to BdP+) compared to computing with 0/1 input bounds.

The standard fractional packing and covering problem can be defined as follows:

$$\begin{aligned} \min c^T \overline{x} & & & (44)\\ s.t. & & & (45)\\ A\overline{x} & \geq & l & (46)\\ B\overline{x} & \leq & m & (47)\\ x \geq 0 & & & (48) \end{aligned}$$

The problem without Eq. (47) is a **fractional covering** problem. The problem without Eq. (46) is a **fractional packing** problem.

Obviously, the linear optimization problems in BdP+ include both covering and packing components and have an additional constraint that the sum of all variables must equal 1. Although a large number of approximation algorithms exist [7], we only found the methods that solve either packing or covering problem, but not both and not with the additional sum-to-1 constraint. Therefore, we resort to solving a relaxed problem.

There are many possibilities for relaxing the constraints of the original problem

and we could not exhaustively experiment with all of them. Therefore, we sought to find the simplest approach that would yield reasonable bounds. We considered relaxing the problem to an instance of fractional knapsack packing which can be solved exactly in a greedy fashion. In the relaxed problem, we maintain the sum-to-1 constraint:

$$\sum_{y_1,...,y_k} P(y_1, ..., y_k|e) = 1 \tag{49}$$

but drop the lower bound constraints completely. The upper bound on a sum of variables is replaced with the upper bound on individual variables that is the minimum of all the upper bounds on the constraints in the original problem in which it participates:

$$P(y_1, ..., y_k|e) \leq UB_{y_1,...,y_k} = \min_y P^U(y_i|e) \tag{50}$$

The fractional knapsack packing is solved exactly by ordering nodes by their cost from maximum to minimum for maximization and from minimum to maximum for minimization and then assigning each node the maximum value until the sum of all node values equals 1. The complexity of computation is $O(n \log n)$, where n is the number of variables, due to the complexity of the sorting algorithm.

The second option is to additionally leave constraints on sums of variables pertaining to one node in the Markov blanket of X. For example, if we select node $Y_j \in ma(X)$, then, in addition to constraints in Eq.(49) and Eq.(50), we keep constraints:

$$P^L(y_j|e) \leq \sum_{Y \setminus Y_j, Y_j = y_j} P(y_1, ..., y_k|e) \leq P^U(y_j|e) \tag{51}$$

for each value y_j in the domain of Y_j.

The domains of the constraints expressed by Eq.(51) pertaining to just one node Y_j are disjoint. Hence, the problem can be treated as an instance of the fractional packing with multiple knapsacks, each having an individual capacity bound. If it was not for the sum-to-1 constraint in Eq.(49), we could solve each knapsack packing problem independently. Still, the greedy approach works.

Figure 4 defines a bound propagation scheme which solves relaxed LP using a greedy algorithm. We denote this scheme ABdP+ for approximate BdP+. To conserve space, we only show in detail the solution to the maximization problem, steps 4.1-4.4 in Figure 4. First, we order nodes by their objective function coefficient value from the largest to smallest (step 4.1 in Figure 4) and initialize $L(y_j) = P^L(y_j)$ and $U(y_j) = P^U(y_j)$ (step 4.2 in Figure 4). Then, we assign each node in order the largest value that is $\leq L(y_j)$ and satisfies variable's upper bound (step 4.3 in Figure 4). Whenever a node is assigned some value z_i, the values of the lower bound $L(y_j)$, upper bound $U(y_j)$, and sum total s are decremented. We make a second pass incrementing each node by a maximum value that satisfies all constraints until the sum of all variables equals 1 (step 4.2 in Figure 4). Again, every time we increment a node's value, we decrement the value of the upper bound $U(y_j)$ and sum total s. The sum of all variables equals 1 when $s = 0$. Since we cannot predict which node $Y_j \in Y$ will yield the LP problem with the smallest maximum of the objective function, we repeat computation, steps 4.1-4.2 in Figure 4, for each $Y_j \in Y$, and pick the smallest value, step 5.

The solution to the minimization problem is the same except nodes are ordered by their objective function coefficient value, step 4.1, from smallest to largest. The total complexity is $O(|Y| \cdot n \log n)$, $n = |\mathcal{D}(Y)|$.

Approximate BdP+
Input: A belief network ($\mathcal{B}$), $ma(X)$=U=$\{U_1, ..., U_m\}$, $\forall u$, $P(x|u)$, selected U_j.
Input/Output: lower bounds array LB, upper bounds array UB.
Repeat:
$\forall X_i \in X \backslash E$, $\forall x_i \in \mathcal{D}(X_i)$ do:
1. Compute condtional probabilities over Markov blanket $\mathbf{Y} = \{Y_1, ..., Y_k\}$ of X_i:
   $\forall \{y_1, ..., y_k\} \in \mathcal{D}(\mathbf{Y})$ do:
   $P(x_i|y_1, ..., y_k) \leftarrow \alpha P(x_i|pa_i) \prod_j P(ch_j|pa_j)$
2. Initialize: $\forall y^i \in \mathcal{D}(Y)$, let $z_i = P(y_1^i, ..., y_k^i|e) = P(y^i, e)$, $k_i \leftarrow P(x|z_i)$.

3. Define a linear optimization problem:

$$\max / \min f = \sum_i k_i z_i$$

s.t.

$$\sum_i z_i = 1$$

$$\forall y_j \in \mathcal{D}(Y_j),\ P^L(y_j) \leq \sum_{z_i, Y_j = y_j} z_i \leq P^U(y_j)$$

4. Maximize: $\forall Y_j \in Y$, $f_j \leftarrow \max \sum_i k_i z_i$
4.1 Sort z_i by k_i from largest to smallest.
4.2. Initialize: $\forall z_i \in \mathcal{D}(\mathbf{Y})$, $z_i \leftarrow 0$, $b_i \leftarrow \min_{y_j \in y} P(y_j|e)$, $f_j \leftarrow 0$, $s \leftarrow 1.0$
$\forall y_j \in \mathcal{D}(Y_j)$, $L(y_j) \leftarrow P^L(y_j)$, $U(y_j) \leftarrow P^U(y_j)$
4.3. For $i \leftarrow 1$ to $|\mathcal{D}(\mathbf{Y})|$ do (*satisfy lower bounds*):
   $z_i \leftarrow \min\{b_j, L(y_j)\}$
   $b_i \leftarrow b_i - z_i$, $s \leftarrow s - z_i$, $L(y_j) \leftarrow L(y_j) - z_i$, $U(y_j) \leftarrow U(y_j) - z_i$
   $f_j \leftarrow f_j + k_i \cdot z_i$
   End For
4.4. For $i = 1$ to $|\mathcal{D}(\mathbf{Y})|$ do (*satisfy upper bounds*):
   if (s = 0) break
   $\delta = \min\{b_j, U(y_j), s\}$
   $z_i \leftarrow z_i + \delta$
   $b_i \leftarrow b_i - \delta$, $s \leftarrow s - \delta$, $U(y_j) \leftarrow U(y_j) - \delta$
   $f_j \leftarrow f_j + k_i \cdot z_i$
End For
5. $P^U(x|e) \leftarrow \min_{Y_j \in Y} f_j$
6. Minimize: $\forall Y_j \in Y$, $f_j \leftarrow \min \sum_i k_i z_i$
7. $P^L(x|e) \leftarrow \max_{Y_j \in Y} f_j$
Until Converged

Figure 4: Greedy algorithm for fractional multiple knapsack problem.

6 Searching for High-Probability Tuples

The problem of generating a subset of high-probability cutset has been investigated previously in the context of bounded conditioning and, more generally, in the context of solving MPE problem. The search technique include local greedy search [11] and branch-and-bound search with different types of heuristics (e.g., see [10]). Another option is to use a stochastic simulation such as Gibbs sampling. This is the approach we take in the current work.

Given a problem with a set of random variables $C = \{C_1, ..., C_m\}$ and observations E, we usually apply Gibbs sampling to generate a set of samples $\{c^i\}$ such that the frequency of a tuple c^i reflects its probability mass $P(c^i|e)$ in the posterior distribution $P(C|e)$. Gibbs sampling can also be viewed as a search algorithm looking for high-probability tuples. It performs a guided random walk in the multi-dimentional space of all cutset instances.

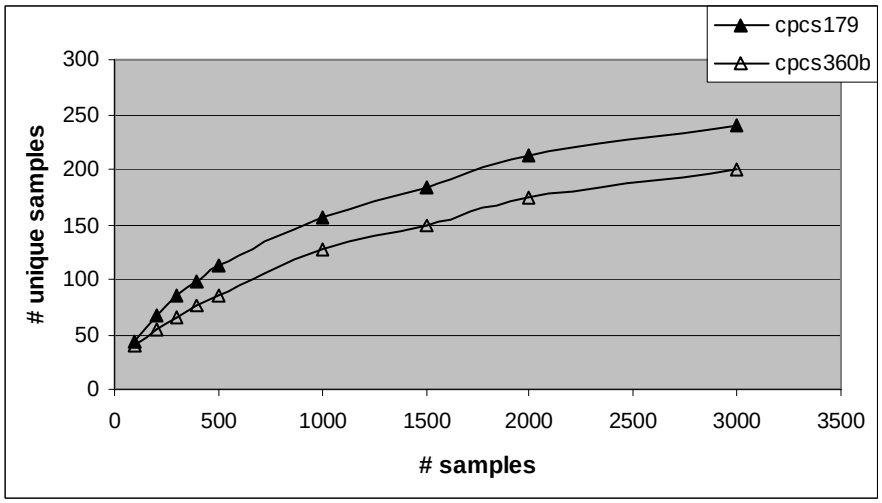

Figure 5: The number of unique samples generated by a Gibbs sampler sampling over a loop-cutset is plotted against the total number of samples for two benchmarks, cpcs179 and cpcs360b. The results are averaged over 20 instances of each benchmark with different evidence values.

Therefore, we propose to generate cutset tuples using w-cutset sampling [5, 6] that implements Gibbs sampling over a subset of variables. We tried first an ordered Gibbs sampler on a cutset selecting all unique cutset instances among samples generated. It was effective for two of our benchmarks, cpcs179 and cps360b, where a small number of cutset tuples contained over 99% of the probability mass of $P(e)$. Figure 6 shows the number of unique samples as a function of h in case of cpcs179 and cpcs360b networks. The results are averaged over 20 instances of each benchmark. As we can see, the curves a logarithmic. The algorithm found most of the high probability tuples quickly and then mostly revisited previously observed tuples.

However, we found that in other benchmarks, w-cutset sampling often required too long to find enough "heavy" tuples. Indeed, this has been observed previously in [11] where the effectiveness of Gibbs sampler and local greedy search in finding the MPE solution were compared over 100 samples. The main difference between Gibbs

sampler and greedy local search is that Gibbs sampling picks the new value for a variable at random from $P(X_i|x_{-i})$ while greedy local search tends to pick the most likely value of X_i more often, at least at the beginning of the search. Furthermore, in [11] it was shown that a combination of Gibbs sampling and greedy local search is more effective than either method alone. We did not incorporate the local greedy search technique although it is a promising direction for future improvements to ATB implementation. We did, however, modified our Gibbs sampler to maximize the value of the output it produced. We elaborate on the options we investigated in the next two paragraphs.

In order to obtain a probability $P(C_i|c_{-i})$ from which we sample a new value for cutset variable C_i in w-cutse sampling, we compute a joint probability $P(c_i, c_{-i})$ for each value in domain of C_i and then normalize. Then, the new value of C_i is chosen and the joint probabilities are discarded. Often, the selected value is not the most probable one. Hence, some tuples with high probability $P(c_i, c_{-i})$ may be computed but not used. Since we generated overall a small number of cutset tuples (a few thousand at the most), it was reasonable to cache all tuples visited. Namely, every time we computed the joint probability $P(c_i, c_{-i})$, we saved the corresponding tuple in the search tree (even if value c_i was not sampled). When sampling was completed, we sorted all tuples by their probability value and marked only the h tuples with the highest probability mass. The rest of the tuples were truncated.

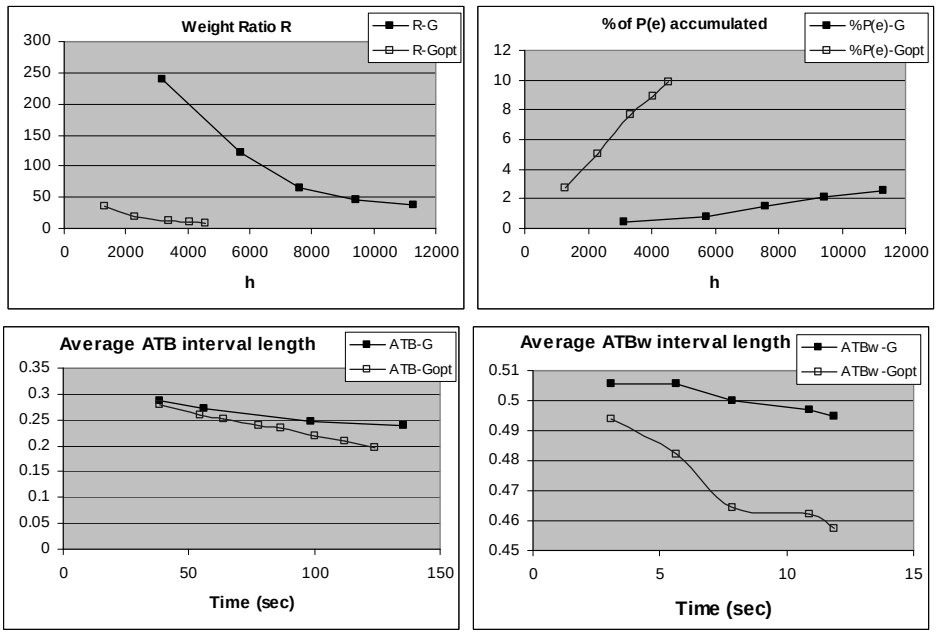

Figure 6: Performance of plain cutset sampling (G) and optimized for search (Gopt) in an instance of Barley network measured by the convergence speed of the ratio R of the probability mass of truncated tuples to the probability mass of h generated tuples (top left), the percent of the probability mass $P(e)$ covered by the h generated tuples (top right) as a function of h. We also show the convergence of the average bounds interval as a function of time for ATB (bottom left) and ATB^w (bottom right) using the two search algorithms.

Since the time necessary for generating one sample was a lot less than the time needed to bound one partially-instantiated tuple, the above scheme was very time-

effective incurring negligible amount of overhead. Within the same time interval, we were able to "pack" more probability mass into h tuples than simply selecting h unique tuples among samples. We demonstrate this in Figure 6, demonstrating the performance of the two schemes on an instance of Barley network with 7 evidence nodes and $P(e) = 3E-06$ that shows their typical behavior.

We denote as G (for plain Gibbs) the algorithm for selecting the h unique cutset tuples from samples generated by ordered Gibbs sampler. We denote the scheme that is optimized for search as G_{opt}. The charts contains two metrics which highlight the efficiency of the search process from different points of view. One parameter is the ratio R between the upper bound on the probability mass of truncated tuples and the mass of the generatd cutset tuples:

$$R = \frac{\sum_{j=1}^{M'} P(c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e)}$$

The top left chart in Figure 6 shows that R decreases faster when G_{opt} algorithm is used. The second parameter is the percent of the probability mass $P(e)$ covered by the generated tuples, namely:

$$\frac{\sum_{i=1}^{h} P(c^i, e)}{P(e)} 100\%$$

Again, the percent of $P(e)$ covered grows much faster when h is selected from a wider range of tuples. The charts on the bottom of Figure 6 show how the method of selecting h cutset tuples affects the speed of convergence of ATB (bottom left) and ATB^w (bottom right) bounds interval. In both cases, the curve corresponding to Gibbs sampling with optimized selection of cutset tuples is lower.

We do not directly compare our strategy to the approach in [9] where h tuples are selected based on their prior weight either via enumeration of all tuples (without evidence) or some greedy search. Obviously, the effectiveness of the prior distribution as selection heuristics depends on how close the distributions $P(C|e)$ and $P(C)$ are. This issue has been well-studied in the context of importance sampling. The convergence speed and quality of approximation depends largely on how long the algorithm takes to find regions of importance of the target distribution. The sampling distribution of likelihood weighting, an instance of importance sampling, is close to prior and it usually performs poorly in benchmarks where $P(e)$ is small compared to other importance sampling algorithms whose sampling distribution is closer to the target distribution.

7 Other improvements to ATB Bounds

In this section, we define additional lower and upper bounds which apply in special cases and can improve over the default ATB bounds. We obtain those bounds utilizing the properties of the posterior estimate based on cutset conditioning formula Eq.(1) but using only h cutset tuples:

$$\hat{P}_h(x|e) = \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} \tag{52}$$

Note that the computed estimates are normalized: $\sum_x \hat{P}(x|e) = 1$. Ordinarily, we do not use estimator $\hat{P}_h(x|e)$ because we can provide no guarantees on the quality of the estimate. Yet, we can establish conditions in the context of ATB framework

when $\hat{P}_h(x|e)$ is a lower or upper bound on $P(x|e)$ if the lower and upper bounds on $P(x, c_{1:q}, e)$ are obtained via:

$$P^L(x, c_{1:q}, e) = P^L(x|c_{1:q}, e)P^L(c_{1:q}, e) \tag{53}$$

$$P^U(x, c_{1:q}, e) = P^U(x|c_{1:q}, e)P^U(c_{1:q}, e) \tag{54}$$

where $P^L(c_{1:q}, e)$ and $P^U(c_{1:q}, e)$ are any bounds on $P(c_{1:q}, e)$ and $P^L(x|c_{1:q}, e)$ and $P^U(x|c_{1:q}, e)$ are any bounds on $P(x|c_{1:q}, e)$. Furthermore, we introduce a second estimator $P'(x|e)$ which we obtain from Eq.(22) replacing both $P(c^j_{1:q_j}, e)$ and $P(x, c^j_{1:q_j}, e)$ with their corresponding upper bounds:

$$P'(x|e) = \frac{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U(x, c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U(c^j_{1:q_j}, e)}$$

and define an upper bound based on the value of $P'(x|e)$ using $\hat{P}_h(x|e)$ value to defined conditions when $P'(x|e)$ is an upper bound on $P(x|e)$. Empirically, adjusting upper bound P^{U_1} for P_h or P' when applicable allows to reduce the average upper bound value by 10-20%.

First, we will show that estimator $\hat{P}$ always falls inside the ATB bounds interval defined by Eq.(37) and Eq.(38).

THEOREM **7.1 (Estimator is Bounded)** *Given a Bayesian network $\mathcal{B}$ with a cutset C and evidence E and some variable X, then*

$$P^{L_3}(x|e) \leq \hat{P}_h(x|e) \leq P^{U_3}(x|e)$$

where $\hat{P}_h(x|e)$ is obtained from Eq. (52) and $P^{L_3}(x|e)$ and $P^{U_3}(x|e)$ are obtained resepectively from Eq.(23) and Eq.(29) by setting $\forall j$, $P^L(x, c^j_{1:q_j}, e) = 0$ and $P^U(x, c^j_{1:q_j}, e) = P^U(c^j_{1:q_j}, e)$.

The proof is provided in Appendix C.

Obviously, the estimator $\hat{P}_h$ converges to $P(x|e)$ as h approaches M. In the limit, when $h = M$, $\hat{P}(x|e) = P(x|e)$. Although the distance between $\hat{P}$ and $P(x|e)$ may not decrease monotonously, we can show the convergence in the norm defined in the next theorem.

THEOREM **7.2 (Convergence Rate)** *Given is a Bayesian network $\mathcal{B}$ with a cutset C and evidence E and variable X. Assume M is the number of cutset state-space and we have generated h out of M cutset tuples yielding estimator $\hat{P}_h$ defined in Eq.(52). Define:*

$$L_h = \frac{\sum_{i=h+1}^{M} P(c^i, e)}{\sum_{i=1}^{M} P(c^i, e)}$$

Then L_h monotonously converges to 0, $L_h \to 0$, as $h \to M$ and the distance between $P(x|e)$ and $\hat{P}_h(x|e)$ converges to 0 as fast as L_h:

$$|P(x|e) - \hat{P}_h(x|e)| \leq L_h$$

The proof is provided in Appendix C. Note that the value of the norm L_h can be bounded by replacing the $\sum_{i=h+1}^{M} P(c^i, e)$ in the numerator and denominator with

its upper bound $\sum_{i=h+1}^{M'} P(c^i_{1:q_i}, e)$ and applying Lemma 4.2, yielding:

$$L_h \leq \frac{\sum_{i=h+1}^{M'} P(c^i_{1:q_i}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M'} P(c^i_{1:q_i}, e)}$$

The next theorem defines two conditions under which we can predict the position of $\hat{P}(x|e)$ relative to the posterior marginal $P(x|e)$ (namely, specify whether it is less than or greater than $P(x|e)$), and thus, use $\hat{P}(x|e)$ as a lower or an upper bound.

THEOREM **7.3 (Using Estimator as a Bound)** *Given a Bayesian network $\mathcal{B}$ with a cutset C and evidence E and variable X, let M be the number of tuples in state-space of cutset C, h be the number of fully-instantiated tuples in C, and $c^j_{1:q_j}$, $j \in [1, M']$, denote a partially-instantiated cutset tuple. Define: $a = \sum_{i=1}^{h} P(x, c^i, e)$, $b = \sum_{i=1}^{h} P(c^i, e)$. Then, $\forall h \in (1, M)$, $\hat{P}_h$ defined in Eq. (52) has following properties:*

1. *(a) If $\forall j \in [1, M']$, $P^U(x|c^j_{1:q_j}, e) \leq \frac{a}{b}$, then $P(x|e) \leq \hat{P}$.*
2. *(b) If $\forall j \in [1, M']$, $P^L(x|c^j_{1:q_j}, e) \geq \frac{a}{b}$, then $P(x|e) \geq \hat{P}$.*

The proof is provided in Appendix C.

Since the value $\hat{P}(x|e)$ is readily available as soon as the first h tuples are generated, we can monitor conditions (a) and (b) while computing bounds on $P(x|c^i_{1:q_i}, e)$ (used in Eq.(53) and Eq.(54)). If the conditions are satisified, we can adjust the upper and lower bounds accordingly. Due to Theorem 7.1, we can guaranteed that $\hat{P}_h$ is as good or a better bound than the bounds obtained in Eq.(23) and Eq.(29) when $\forall j$, $P^L(x, c^j_{1:q_j}, e) = 0$ and $P^U(x, c^j_{1:q_j}, e) = P^U(c^j_{1:q_j}, e)$. Next, we define an alternative upper bound that holds when $\hat{P}(x|e) \leq P^U(x|c^i_{1:q_i}, e)$ for all $i > h$.

THEOREM **7.4 (Upper Bound)** *Given is a Bayesian network $\mathcal{B}$ with a cutset C and evidence E and variable X. Define:*

$$a = \sum_{i=1}^{h} P(x, c^i, e), \quad b = \sum_{i=1}^{h} P(c^i, e),$$

$$\delta = \sum_{i=h+1}^{M'} P^U(x|c^i_{1:q_i}, e) P^U(c^i_{1:q_i}, e), \quad \Delta = \sum_{i=h+1}^{M'} P^U(c^i_{1:q_i}, e)$$

If $\forall i > h$, $\frac{a}{b} < P^U(x|c^i_{1:q_i}, e)$, then:

$$P(x|e) \leq \frac{a+\delta}{b+\Delta} + \frac{\delta}{b+\delta}\frac{\Delta}{b+\Delta}$$

The proof is provided in Appendix C.

The condition of Theorem 7.4 can also be monitored while computing the bounds $P^U(x|c^i_{1:q_i}, e)$. We can apply results of Theorem 7.3 and Theorem 7.4 simultaneously selecting the tightest bound available.

An alternative application of the result of Theorem 7.4 is to adjust the bound $P^U(x|c^i_{1:q_i}, e)$ so that the theorem condition is satisfied. Namely, if we observe that $\frac{a}{b} > P^U(x|c^i_{1:q_i}, e)$ when $P^U(x|c^i_{1:q_i}, e)$ is computed, then we increase the bound

$P^U(x|c^i_{1:q_i}, e) = \frac{a}{b}$. Using upper bound $P^U(x|c^i_{1:q_i}, e)$ that is "adjusted" for $\hat{P}(x|e)$, we can compute alternative bound from Theorem 7.4.

We have observed empirically that the joint effect of the adjustments from three theorems above results in average $\approx 10 - 20\%$ reduction of the bounds interval which we demonstrate for several benchmarks in the empirical section.

8 Experiments

In this section, we compare empirically performance of several bounding schemes. First, we compare the performance of $BdP+$ and BdP algorithms. Next, we compare the performance of ATB framework using bound propagation plugin with different approximation algrotithms for solving linear optimization problems. As $BdP+$ is consistently superior to BdP, we then compare performance of $BdP+$ with conditioning-based schemes, namely, ATB and ATB^w algorithms using bound propagation plugin with using the best of the two approximation algorithms for LP optimization and boosted bound propagation $BBdP+$.

8.1 Algorithms

We evaluate empirically performance of our ATB framework implementation described in Section 5 using approximation propagation algorithm ABdP+, described in Section 5.3, as a plugin. Since we always plugin ABdP+ algorithm, we refer to the resulting algorithm as ATB after the name of the framework. We compare ATB bounds to the bounds obtained by bound propagation scheme $BdP+$ defined in Section 5 which implements the original bound propagation algorithm but restricts the Markov blanket of the node to its relevant subnetwork. For reference, we also report the time and quality of bounds for original bound propagation scheme BdP. We also show that ATB can be used to "boost" bound propagation using ATB lower and upper bounds as input bounds to $BdP+$ algorithm yielding Boosted $BdP+$ scheme (BBdP+).

The empirical results show that ATB bounds usually are well-centered around the posterior marginals.

We also compare ATB bounds and the bounds obtained by pluggin bound propagation ABdP+ into ATB^w framework to bound $P(c_{1:q}, e)$ while setting $P^L(x, c^i, e) = 0$ and $P^U(x, c^i, e) = P^U_{ABdP+}(x, c^i, e)$. We know that for the same h, ATB bounds are always as good or better compared to ATB^w. However, ATB^w can process more tuples in the same amount of time. Hence, we investigate emprically the performance of two schemes as a function of time. We denote the resulting algorithm ATB^w after the name of the framework.

We also computed the bounds generated by ATB framework when we plug in the brute force bounding algorithm $P^L(x, c^i, e) = 0$ and $P^U(x, c^i, e) = P(c^i)$ as is done by bounded conditioning. Recall that the brute force lower bound, expressed in Eq. 34, is equivalent to that of bounded conditioning and the upper bound, expressed in Eq. 36, is at least as good or better (Theorem 4.3). We denote this instance of ATB as ATB-BF. Thus, by evaluating bounds interval for ATB-BF, we obtain the minimum bounded condtioning bounds interval length.

8.2 Methodology

We measure the quality of the bounds via the average length of the interval between lower and upper bound:

$$\overline{I} \quad = \quad \frac{\sum_i \sum_{D(x_i)} (P^U(x_i|e) - P^L(x_i|e))}{\sum_i |D(x_i)|}$$

We compute approximate posterior marginal as the midpoint between lower and upper bound in order to show whether the bounds are well-centered around the posterior marginal $P(x|e)$. Namely:

$$\hat{P}(x|e) = \frac{P^U(x|e) + P^L(x|e)}{2} \tag{55}$$

and then measure average absolute error Δ with respect to that approximation:

$$\Delta = \frac{\sum_i \sum_{D(x_i)} |P(x_i|e) - \hat{P}(x_i|e)|}{\sum_i |D(x_i)|}$$

We control the time and memory of bound propagation by restricting the maximum length of the conditional probability tables over the Markov blanket of a node (Markov CPT). Whenever a Markov CPT table length of a variable exceeded a bound k, its lower and upper bounds were appropriately fixed at 0 and 1. In Section 8.4.2, we show how performance of $BdP+$ varies with k on various benchmarks with and without evidence.

We report $BdP+$ results for a range of values of k. The maximum Markov CPT length tested was at $k = 2^{19}$ (the size of the CPT with 19 bi-valued variables) when the computation demands exceeded available memory. For ATB the maximum Markov CPT state-space was fixed at minimum value of 1025 in most experiments.

We report all results for BdP, $BdP+$, and $BBdP+$ schemes "upon convergence" or after 20 interations, whichever occurs first. Usually algorithms converged in less than 20 iterations. When we compare $BdP+$ vs. ATB, we allocate ATB the same amount of time as $BdP+$ required to converge.

8.3 Benchmarks

Table 1: Complexity characteristics of the benchamrks from UAI repository: N-number of nodes, w^*-induced width, $|LC|$-number of nodes in a loop-cutset, $|\mathcal{D}(LC)|$-loop-cutset state space size, Time(BE)-exact computation time via bucket elimination, Time(LC)-exact computation time via loop-cutset conditioning.

network	**N**	**w***	**|LC|**	**|D(LC)|**	**Time(BE)**	**Time(LC)**
Alarm	37	4	5	108	0.01 sec	0.05 sec
Barley	48	7	12	>2E+6	50 sec	>22 hrs
cpcs54	54	15	6	32768	1 sec	22 sec
cpcs179	179	8	8	32768	2 sec	37 sec
cpcs360b	360	21	26	2^{26}	20 min	> 8 hrs
cpcs422b	422	22	47	2^{47}	50 min	> 2E+9 hrs
Munin3	1044	7	30	$> 2^{30}$	8 sec	> 1700 hrs
Munin4	1041	8	49	$> 2^{49}$	70 sec	> 1E+8 hrs

Our benchmarks are Alarm, cpcs networks (cpcs54, cpcs179, cpcs360b, and cpcs422b), Barley network, and Munin3 and Munin4 networks from UAI repository. The summary of benchmarks and their characteristics is shown in Table 8.3. In all cases, our conditioning cutset C is the loop-cutset of the network. We find loop-cutset of the network using mga algorithm of [3]. Evidence nodes and their values are selected at random. All exact posterior marginals were obtained by bucket elimination [8] using the min-fill heuristics for ordering variables. The posteriors for the Alarm network for monitoring patients in intensive care [4], which has only 37 variables and a loop-cutset of size 5, are easy to compute exactly by cutset conditioning or variable elimination. We include the Alarm network for comparison with previously proposed bounded conditioning and bound propagation schemes. The Barley network is a part of the decision-support system for growing malting barley developed in [14]. Barley network is also relatively small. It has only 48 variables and its induced width is only $w^* = 7$. The exact inference in Barley network takes about 30 seconds by bucket elimination. However, the network has variables with large domain sizes (up to 67 values) and, as a result, although its loop-cutset is 12, the loop-cutset tuple space is over 2 million. Enumerating and computing all cutset tuples, at a rate of about 1000 tuples per second, would take over 22 hours. We have performed experiments with four CPCS networks: cpcs54, cpcs179, cpcs360b, and cpcs422b. CPCS networks are derived from the Computer-Based Patient Care Simulation system and based on INTERNIST-1 and Quick Medical Reference Expert systems [21]. Finally, we have experiemented with Munin3 and Munin4, the subsets of the Munin network which is a part of the expert system for computer-aided electromyography [2].

We implemented bounds propagation algorithm using simplex solver from COIN-OR libraries [1]. The experiments were conducted on 1.8Ghz CPU with 512 MB RAM.

8.4 Results

8.4.1 Cutset sampling as a Search Algorithm

In cpcs179 and cpcs360b, we were able to accumulate 95%-99% of the probability mass of $P(e)$ in a small number of tuples generated:

$$\sum_{i=1}^{h} P(c, e) > 0.95 P(e)$$

In other networks, including cpcs54 and cpcs422b, the distribution $P(C|e)$ was nearly uniform and, thus, the generated cutset tuples accounted for $< 10\%$ of $P(e)$.

8.4.2 Result for BdP vs. $BdP+$

In Table 2 and Table 3 we report the average error, average bounds interval length, and computation times for BdP and $BdP+$ as a function of maximum Markov blanket tuple count k. Each row corresponds to a set of experiments with a single benchmark with a fixed k. Columns 3-5 specify the accuracy and computation time for BdP, while columns 6-7 specify the accurcay and computation time for $BdP+$. We explore the range of values of $k = 2^m$, $m \in [10, 20]$. We report results for all k until increasing k does not result in any changes in the results or the available memory is exceeded.

BdP+ always computes tighter bounds and requires less computation time than BdP. The performance gap is wider in the networks without evidence where the

Table 2: Average error Δ, length of the bounds interval $\overline{I}$, and computation time for BdP and $BdP+$ as a function of the maximum size of the Markov blanket state-space T in networkds without evidence.

		BdP(k)			BdP+(k)		
	k	$\overline{I}$	Δ	**time**	$\overline{I}$	Δ	**time**
Alarm	16384	0.6369	0.1677	14	0.0753	0.0076	0.1
cpcs54	16384	0.4247	0.0229	24	0.0907	0.0049	0.1
	32768	0.4173	0.0224	72	0.0907	0.0049	0.1
	65536	0.4173	0.0224	72	0.0907	0.0049	0.1
	131072	0.4173	0.0224	72	0.0907	0.0049	0.1
	262145	0.4154	0.0221	265	0.0907	0.0049	0.1
cpcs179	16384	0.5759	0.2213	30	0.0006	0.00002	0.3
	32768	0.5759	0.2213	30	0.0006	0.00002	0.3
	65536	0.5759	0.2213	30	0.0006	0.00002	0.3
cpcs360b	16384	0.1505	0.0649	64	0.0006	0.0002	1.2
	32768	0.1485	0.0641	80	0.0006	0.0002	1.2
cpcs422b	16384	0.2339	0.0756	79	0.0082	0.0008	8
	32768	0.2329	0.0751	88	0.0082	0.0008	8

Markov blanket of each node, restricted to its relevant subnetwork, contains node's parents only and $BdP+$ converges after one iteration when processing nodes in topological order. The results are reported in Table 2. For the largest benchamrk, cpcs422b, with 422 nodes and $w^* = 21$, the average bounds interval length is 0.23 for BdP and 0.008 for $BdP+$. At the same time, BdP computations take 190 sec while $BdP+$ only takes 16 sec.

The results over benchmarks with evidence are reported in Table 3. Both BdP and $BdP+$ bounds interval becomes larger and computation takes longer when some nodes are assigned. Still, $BdP+$ remains superior to BdP. Consider the results for cpcs360b network with 360 nodes, averaged over 20 instances of the network with number of assigned nodes $|E|$ ranging from 11 to 23. For $h = 16384$, BdP computes the average lower and upper bound interval of length 0.0637 and requires 15 seconds. $BdP+$ computes an average bound interval of 0.3375 and requires only 68 seconds. We observe similar results for other benchmarks. Note that, as k increases, the computation time of both BdP and $BdP+$ increases fast, while the bounds interval decreases only a little.

8.4.3 Approximating LP using two variants of Fractional Packing

In Figure 8 we compare the performance of ATB framework when plugging in the bound propagation scheme $BdP+$ using the two algorithms for approximately solving the linear optimization problems of $BdP+$. We defined the linear optimization problem and described the two approximation schemes in Section 5.3. One algorithm, $FP1$, is a simple fractional knapsack packing. The second algorithm, FPM, solves a more sophisticated fractional packing and covering problem with multiple knapsacks.

We compare the performance of plain $BdP+$ algorithm and $BdP+$ with two different approximation schemes, denoted $BdP+$-$FP1$ and $BdP+$-FPM, in Figure 7 over 20 instances of cpcs360b network. The chart on the left shows that for the same maximum Markov blanket size, average bounds interval of $BdP+$ is smaller

Table 3: Average error Δ, length of the bounds interval $\overline{I}$, and computation time for BdP and $BdP+$ as a function of the maximum size of the Markov blanket state-space T in networks with evidence.

		BdP(k)			BdP+(k)		
	k	$\overline{I}$	Δ	**time**	$\overline{I}$	Δ	**time**
Alarm	16384	0.8276	0.2661	13	0.6376	0.2084	5.3
$\|E\|$=3-6	65536	0.8276	0.2661	13	0.4401	0.1299	3.0
cpcs54	4097	0.6213	0.0469	12	0.2661	0.0141	3.4
$\|E\|$=2-6	8193	0.6093	0.0454	24	0.2643	0.0138	5.4
	16384	0.6021	0.0448	46	0.2638	0.0138	6.6
	32768	0.5986	0.0445	64	0.2637	0.0138	7.4
	65536	0.5957	0.0440	98	0.2637	0.0138	10
	131072	0.5954	0.0439	116	0.2635	0.0137	16
cpcs179	1024	0.6036	0.2228	30	0.2216	0.0689	7.2
	2048	0.6036	0.2228	30	0.2193	0.0684	7.6
	4096	0.6034	0.2227	31	0.1914	0.0589	10
	8192	0.6034	0.2227	31	0.1766	0.0550	14
	16384	0.6034	0.2227	30	0.1525	0.0456	20
$\|E\|$=12-24	32768	0.6034	0.2227	30	0.1502	0.0450	24
	65536	0.5983	0.2214	90	0.1237	0.0365	130
cpcs360b	1025	0.3614	0.1532	15	0.1239	0.0516	4.2
$\|E\|$=11-23	2049	0.3571	0.1528	18	0.1132	0.0468	4.7
	4096	0.3558	0.1522	19	0.0998	0.0408	6
	8192	0.3522	0.1504	23	0.0825	0.0330	8
	16384	0.3375	0.1423	68	0.0637	0.0247	15
	32768	0.3370	0.1419	85	0.0554	0.0215	24
	65536	0.1430	0.3367	120	0.0500	0.0192	36
	131072	0.1430	0.3366	128	0.0429	0.0160	80
	262144	0.1428	0.3364	190	0.0377	0.0137	130
cpcs422b	16384	0.3326	0.1175	90	0.2043	0.0637	32
$\|E\|$=6-11	32768	0.3310	0.1167	102	0.1999	0.0617	44
	65536	0.3160	0.1092	317	0.1667	0.0467	465

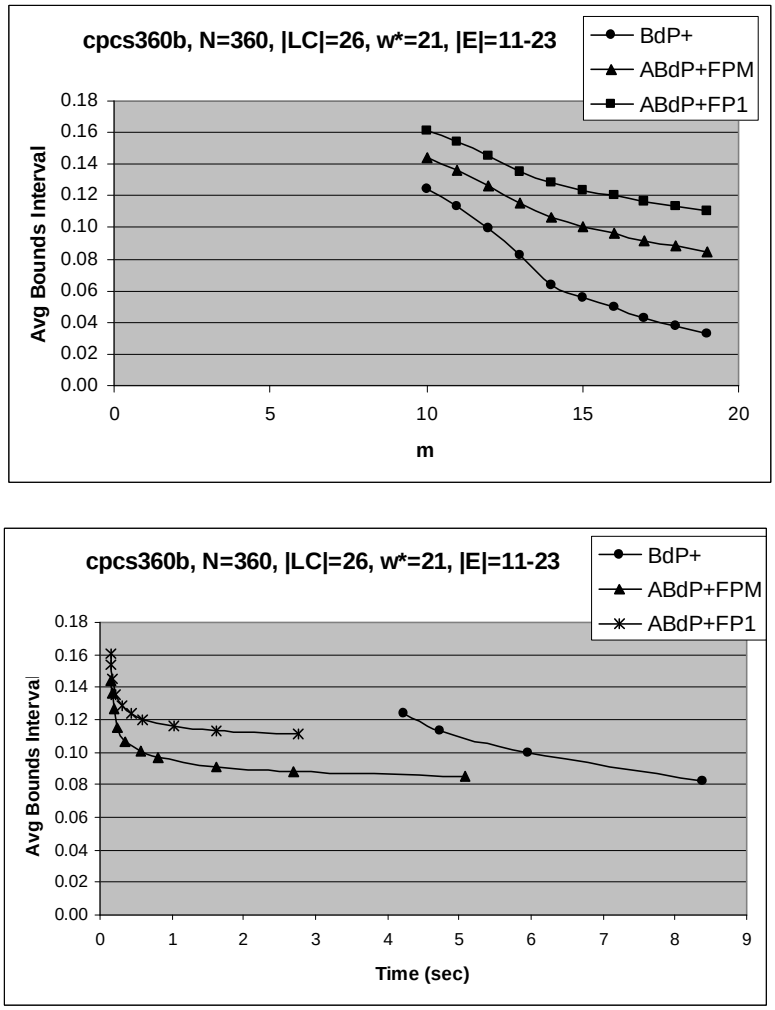

Figure 7: Bounds interval length for BdP+ algorithm optimizing LP by fractional packing with 1 (BdP+-FP1) and many (BdP+-FPM) knapsacks, averaged over 20 instances of cpcs360b network, as a function of m, where maximum Markov table size is bounded by 2^m, and time.

than the bounds interval of BdP+-FPM by about 0.2 which, in turn, is smaller than BdP+-FP1 by 0.2. As the maximum Markov blanket size bound of 2^m increases with m, the distance between BdP+-FP1 and BdP+-FPM remains the same while the distance between BdP+-FP1 and BdP+ increases The chart in Figure 7, right, shows the performance of the tree schemes as a function of time. We see that BdP+ can outperform the FP1 and FPM when given enough time to process large Markov tables. However, in short term, we can obtain more accurate bounds using BdP+-FPM which outperforms both BdP+-FP1 and BdP+.

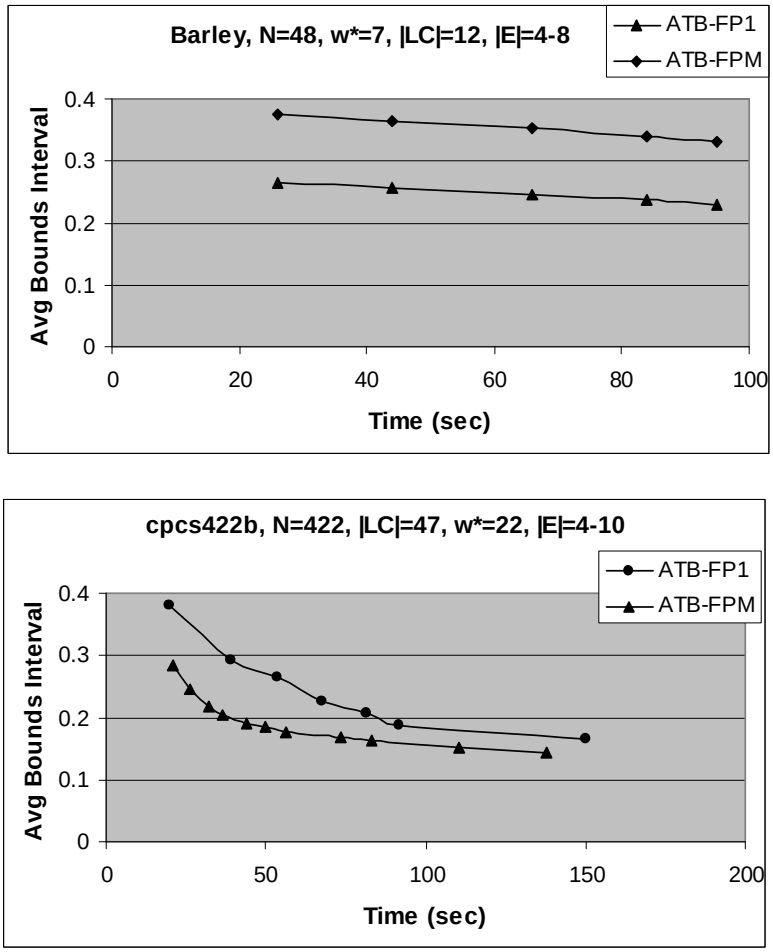

Figure 8: Average bounds length for ATB framework with a bound propagation plugin optimizing LP by fractional packing with 1 (ATB-FP) and many knapsacks (ATB-FPM).

We demonstrate the performance of ATB scheme with BdP+-FP1 and BdP+-FPM plugins, denoting the resulting algorithms ATB-FP1 and ATB-FPM, on two representative benchmarks, Barley and cpcs422b. The results on other benchmarks are similar. We report the bounds interval length, obtained using ATB framework with two variants of BdP+, as a function of time. We control the computation time by the number of generated cutset tuples h. The comptuation time

increases with h. We see in Figure 8 that the ATB-FPM line is consistently lower than ATB-FP1. We also observe that the difference between the results of the two schemes becomes smaller with time because the percentage of $P(e)$ covered by h generated tuples, whose probabilities are computed exactly, increases. This is more noticable in case of cpcs422b, Figure 4, bottom, because the percentage of $P(e)$ covered by generated cutset tuples in cpcs422b grows faster with h (and with time) than in Barley.

8.4.4 Performance of ATB^w, ATB, $BdP+$ and $BBdP+$

In this section, we compare the results of the various bounding schemes, using the superior variants discussed in earlier sections. Namely, we report the results for $BdP+$ variant of bound propagation that incorporates the relevant subnetwork properties into computation. We report bounds obtained using ATB and ATB^w frameworks using bound propagation plugin with FPM approximation algorithm. $BBdP+$ algorithm is equivalent to $BdP+$ except it uses the ATB bounds as input. As we mentioned earlier, we also compute bounds obtain using ATB framework and Brute Force bounding algorithms (ATB-BF). In all of our experiments, the ATB-BF bounds interval length, and, subsequently, bounded conditioning bounds interval, remained > 0.99 for all values h within the time interval tested. Thus, we do not report ATB-BF results in the tables or charts that follow.

We summarize results for each benchmarks in a tabular format and charts. The tables report the results for ATB^w, ATB, and $BBdP+$ algorithms as a function of h. Since $BdP+$ results do not depend on h, we do not include $BdP+$ in those tables. The results for $BdP+$ were reported in Table 3. We use charts to show the convergence of the bounds interval length of ATB^w, ATB, and $BBdP+$ algorithms as a function of h. Separately, we show the convergence of the bounds interval length of ATB^w, ATB, $BBdP+$, and $BdP+$ as a function of time.

Results for Alarm network. As noted before, the Alarm network is easy to for exact computation. We present the results in order to relate to bounds obtained previously by bounded conditioning [9] and bound propagation [16]. Alarm network has $N = 37$ nodes and a loop-cutset of size $|LC| = 5$ with the cutset state-space of size $k = 108$. The exact posterior marginals in Alarm network can be obtained using bucket elimination or exact cutset conditioning in less than a second. Table 8.4.4 reports the average bounds length $\overline{I}$, average error Δ, and average time for ATB^w, ATB, and $BBdP+$ as a function of the number of explored cutset tuples h.

Table 4: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ over 20 instances of Alarm network. Parameter h indicates the number of cutset tuples computed exactly in ATB.

Alarm, N=37, w^*=5, $\|LC\|$=8, $\|D_{LC}\|$=108, $\|E\|$=1-4									
	ATB^w			ATB			$BBdP+$		
h	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time
25	0.51	0.16	*0.021*	0.41	0.12	*0.038*	0.35	0.10	*3.4*
34	0.38	0.12	*0.022*	0.31	0.09	*0.039*	0.27	0.08	*2.3*
40	0.31	0.10	*0.025*	0.25	0.07	*0.044*	0.22	0.06	*2.1*
48	0.20	0.09	*0.035*	0.24	0.05	*0.051*	0.15	0.04	*1.5*
50	0.16	0.06	*0.036*	0.16	0.04	*0.052*	0.12	0.03	*1.2*
54	0.12	0.05	*0.044*	0.13	0.03	*0.059*	0.09	0.02	*0.86*

Recall that for $k \geq 65536$ $BdP+$ obtained an average interval length of 0.44 within 3 seconds. Both ATB^w and ATB compute more accurate bounds starting with $h = 34$

in the second row of Table 8.4.4. For $h = 34$, the average lengths of ATB^w and ATB bounds are $\overline{I}_{ATB^w} = 0.38$ and $\overline{I}_{ATB} = 0.31$ respectively. The computation times are 0.022 and 0.039 seconds, an order of magnitude less than $BdP+$. The corresponding $BBdP+$ bounds interval of $\overline{I}_{BBdP+} = 0.27$ is also smaller than that of $BdP+$ and it is obtained within 2.3 seconds. As a function of h, $BBdP+$ is superior to ATB which is superior to ATB^w. However, it is not cost-effective to inverst additional time in bounding $P(x|c_{1:q}, e)$. For example, while ATB^w computes bounds interval of $\overline{I}_{ATB^w} = 0.12$ within 0.044 seconds, ATB only computes $\overline{I}_{ATB} = 0.25$ within the same time ($h = 40$). This is not suprising because network is very small and the loop-cutset state-space is very small (108 states). As expected, the bounds generated by ATB^w, ATB, and $BBdP+$ decrease as h increases. Oddly, $BBdP+$ computation time decreases as h increases which indicates that simplex algorithm computes faster with tighter input bounds.

Table 5: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ averaged over 20 instances of Barley network. Parameter h indicates the number of cutset tuples computed exactly.

Barley, N=48, w^*=7, $\|LC\|$=12, $\|D_{LC}\|$ >2E+6, $\|E\|$=4-8									
	ATB^w			ATB			BBdP+		
h	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time
3036	0.499	0.200	4.7	0.270	0.093	26.3	0.160	0.044	28.4
6201	0.498	0.199	9.4	0.257	0.088	44.9	0.154	0.042	47.1
9012	0.497	0.199	14.0	0.244	0.083	66.3	0.151	0.041	68.5
12047	0.496	0.198	18.5	0.237	0.080	83.2	0.150	0.041	85.4
15210	0.495	0.198	23.2	0.230	0.077	95.3	0.148	0.041	92.2
19206	0.489	0.195	26.3	0.228	0.075	97.6	0.146	0.040	98.5

Results for Barley network. We applied the algorithms over 20 instances of Barley network with different evidence, picked at random among the input nodes as defined in [14]. The maximum computation time interval is 100 seconds. Within 100 seconds, we were able to compute ATB^w and ATB bounds over $h = 2000$ cutset tuples, that is less than 0.0002% of over 2 million tuples, which accounted for $\approx 1\%$ of the weight of $P(e)$. The results are reported in Figure 9. Both ATB^w and ATB bounds interval length decreases as h increases as we show in Figure 9, top. But the convergence is slow both with time and with h. For example, the ATB^w bounds interval remains close to 0.5 while ATB bounds interval decreases from 0.27, obtained in 25 seconds, to 0.23. The $BBdP+$ improves over ATB, but shows a similar slow anytime convergence. It computes 0.17 bounds interval in 25 seconds which decreases to 0.14 after 90 seconds. The length of bounds interval of ATB^w, ATB, $BdP+$, and $BBdP+$ is plotted as a function of time in Figure 9, bottom. BdP+ converges quickly yielding an average bounds length of 0.23 in less than 2 seconds but does not improve any more with time. It takes ATB about 70 seconds to achieve the same accuracy. The best anytime method is $BBdP+$ since its overhead, compared to ATB computation time, is small. Note, that exact bucket elimination (BE) is superior (takes 50 seconds), but we should compare with loop-cutset that uses the same amount of space and takes over 22 hours to process all cutset tuples. The average errors of ATB and ATB^w in Table 5 are close to a half of the interval length and hence, the true posterior marginals tend to be skewed to one end of the bounds interval. The $BBdP+$ bounds are better centered since the average error is about quarter of the bounds interval length.

Results for CPCS networks. Results for cpcs54 are given in Table 6 and Figure 10, for cpcs179 in Table 7 and Figure 11, for cpcs360b in Table 8 and Figure 12,

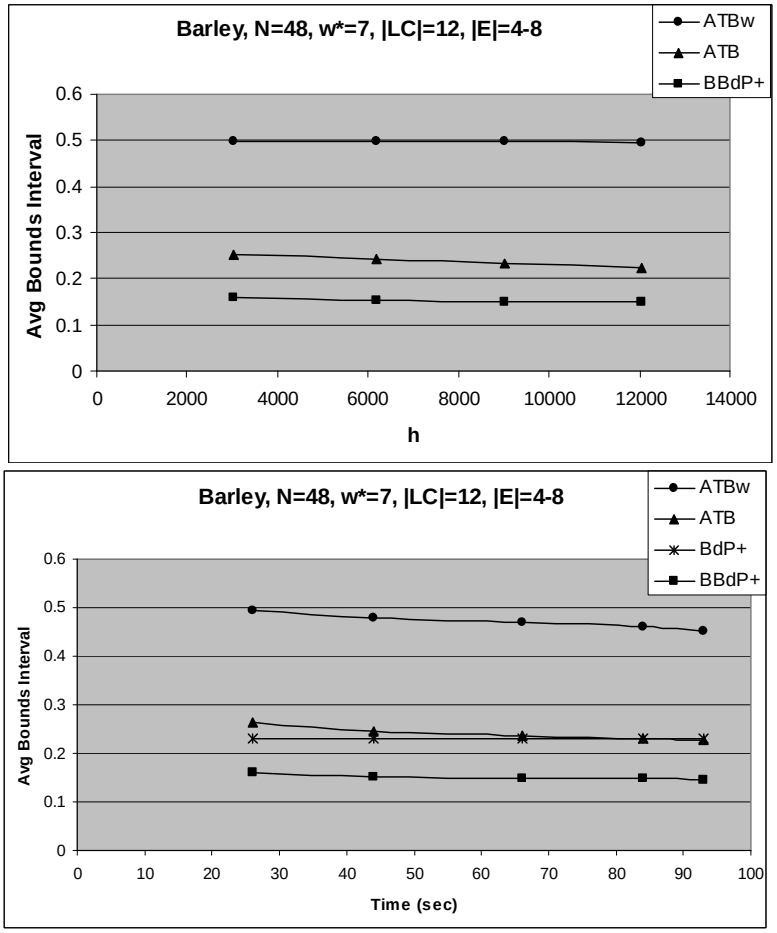

Figure 9: Results for Barley network as a function of h (top) and a function of time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 30 seconds. Exact inference using cutset conditioning is > 22 hours.

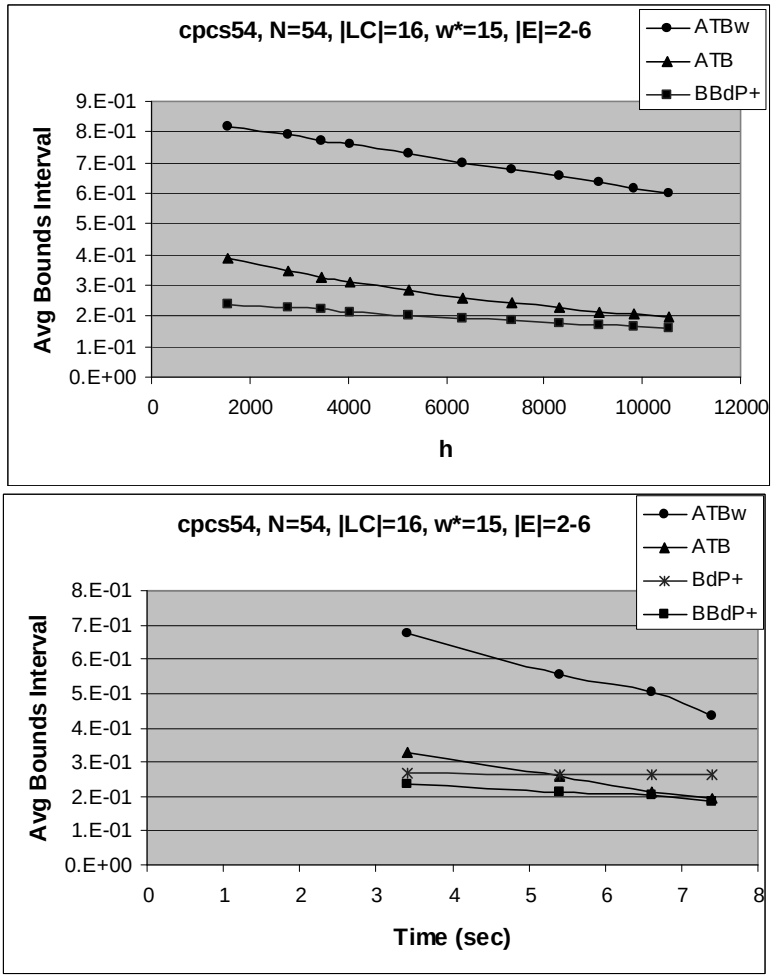

Figure 10: Results for cpcs54 network as a function of h (top) and a function of time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 1 second. Exact inference using cutset conditioning is 15 seconds.

Table 6: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ over 20 instances of cpcs54 network. Parameter h indicates the number of cutset tuples computed exactly in ATB.

	cpcs54, N=54, $\|LC\|$=15, w^*=15, $\|D_{LC}\|$=32678, $\|E\|$=2-8								
	ATB^w			ATB			BBdP+		
h	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time
513	0.8366	0.0618	0.4	0.5101	2.7E-2	0.9	0.3405	1.1E-2	3.1
1114	0.7765	0.0555	0.7	0.4510	2.3E-2	1.5	0.3245	1.0E-2	3.1
1343	0.7560	0.0535	0.9	0.4363	2.3E-2	1.7	0.3196	9.9E-3	3.3
1581	0.7354	0.0516	1.0	0.4192	2.1E-2	1.9	0.3132	9.4E-3	3.4
1933	0.7060	0.0489	1.3	0.3972	2.0E-2	2.2	0.3037	8.5E-3	3.6
2290	0.6800	0.0466	1.5	0.3820	1.9E-2	2.4	0.2966	7.8E-3	3.9
2609	0.6574	0.0447	1.8	0.3665	1.8E-2	2.7	0.2882	7.3E-3	4.0
3219	0.6175	0.0415	2.1	0.3385	1.6E-2	3.2	0.2715	6.7E-3	4.5
3926	0.5731	0.0380	2.7	0.3054	1.4E-2	3.8	0.2515	6.3E-3	5.2
6199	0.4564	0.0292	4.5	0.2299	9.7E-3	5.9	0.1967	5.9E-3	6.6
7274	0.4086	0.0256	5.4	0.1986	8.0E-3	6.9	0.1725	5.6E-3	7.3

for cpcs422b in Table 9 and Figure 13.

We focus on cpcs54 first, the smallest of CPCS networks, with 54 nodes and induced width $w^* = 15$. Its loop cutset size is 16 and yields 65536 (2^{16}) cutset tuples. Exact inference in cpcs54 by bucket elimination takes less than 1 second. while cutset conditioning requires 15 seconds.

The average bounds interval length obtained by ATB^w is the worst both as a function of h, Figure 10, top, and as a function of time, Figure 10, bottom. $BBdP+$ is the best in both cases and offers substantial improvement over ATB in the first 5 seconds of computation. In 6 seconds, ATB and $BBdP+$ curves become very close. The $BdP+$ scheme obtains quickly the bounds interval of 0.26 with the maximum table size of 4096 and does not improve much as maximum Markov table size is increased up to the memory limit (see Figure 3). ATB outperforms $BdP+$ after 5 seconds. The bounds of ATB, ATB^w, and $BBdP+$ appear to be well centered around the $P(x|e)$ as Table 6 shows.

Results for cpcs179. The results for cpcs179 network, averaged over 20 instances with difference evidence, are shown in Table 7 and Figure 11. The results for ATB, ATB^w, and $BBdP+$ as a function of time are similar and speak for themselves. As a function of time, $BdP+$ is the worst algorithm. Within the first 20 seconds, the best algorithm is ATB. The performance of $BBdP+$ as a function of time at first is lagging behind both ATB^w and ATB. The average $BBdP+$ computation time per network instance is 8 seconds. Hence, its first data point on the chart is at 10 seconds (with ATB time added). However, as ATB bounds improve, the $BBdP+$ bounds interval decreases very fast. After 17 seconds, it outperforms ATB^w. Extrapolating the results, we can predict that $BBdP+$ will outperform ATB after about 23 seconds.

cpcs360b. The results for cpcs360b are summarized in Table 8 and Figure 12. The network has loop-cutset of size 26. Since all nodes have domains of size 2, the loop-cutset state space size is 2^{26}, prohibitevely large for complete enumeration. The exact computation time for cpcs360b by bucket elimination is about 20 minutes since the induced width is 21. We experimented with 20 instances of the network number of evidence nodes ranging from 11 to 23.

Table 7: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ over 20 instances of cpcs179 network. Parameter h indicates the number of cutset tuples computed exactly in ATB.

	cpcs179, N=179, w^*=8, $\|LC\|$=8, $\|D_{LC}\|$=32768, $\|E\|$=12-24								
	ATB^w			ATB			BBdP+		
h	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time
417	0.4638	0.1696	1.7	0.1538	0.0347	3.4	0.0740	0.0186	11.0
590	0.3529	0.1283	2.0	0.1039	0.0221	4.2	0.0555	0.0137	12.0
827	0.2592	0.0937	2.5	0.0718	0.0151	4.9	0.0419	0.0105	13.0
989	0.2056	0.0741	2.9	0.0533	0.0109	5.5	0.0325	0.0080	14.0
1128	0.1721	0.0619	3.4	0.0420	0.0086	6.2	0.0272	0.0067	15.0
1250	0.1473	0.0529	3.4	0.0347	0.0072	6.6	0.0232	0.0057	14.0
1634	0.0935	0.0334	4.5	0.0210	0.0045	7.6	0.0146	0.0035	16.4
1878	0.0682	0.0243	5.4	0.0141	0.0031	8.7	0.0105	0.0026	16.5
2107	0.0539	0.0192	6.2	0.0111	0.0025	9.6	0.0084	0.0021	17.3
2282	0.0446	0.0158	7.3	0.0089	0.0020	10.9	0.0068	0.0017	19.2
2416	0.0388	0.0138	7.8	0.0076	0.0017	11.3	0.0060	0.0015	18.9

Table 8: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ as a function of the number of generated cutset tuples h averaged over 20 instances of cpcs360b network with random evidence on leaves.

	cpcs360b, N=360, $w^* = 21$, $\|LC\| = 26$, $\|E\|$=11-23								
	ATB^w			ATB			BBdP+		
h	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time
56	0.1256	0.0598	2.7	0.0051	9.5E-4	5.3	0.0031	8.4E-4	29.9
94	0.0788	0.0372	4.4	0.0027	4.5E-4	8.6	0.0017	4.1E-4	44.8
142	0.0532	0.0250	5.5	0.0019	3.3E-4	10.2	0.0013	3.1E-4	55.1
205	0.0361	0.0168	7.6	0.0010	1.7E-4	13.5	0.0007	1.6E-4	46.3
263	0.0275	0.0128	9.2	0.0007	1.2E-4	15.1	0.0008	1.3E-4	54.0
306	0.0234	0.0109	10.5	0.0006	9.5E-5	17.6	0.0004	8.9E-5	43.5
381	0.0179	0.0083	13.0	0.0004	5.2E-5	21.6	0.0003	4.9E-5	57.3
485	0.0133	0.0061	16.6	0.0003	4.1E-5	26.5	0.0002	3.8E-5	55.2
586	0.0102	0.0047	20.9	0.0002	3.1E-5	32.5	0.0002	2.8E-5	65.3
686	0.0084	0.0038	24.1	0.0002	2.6E-5	36.2	0.0001	2.4E-5	62.6

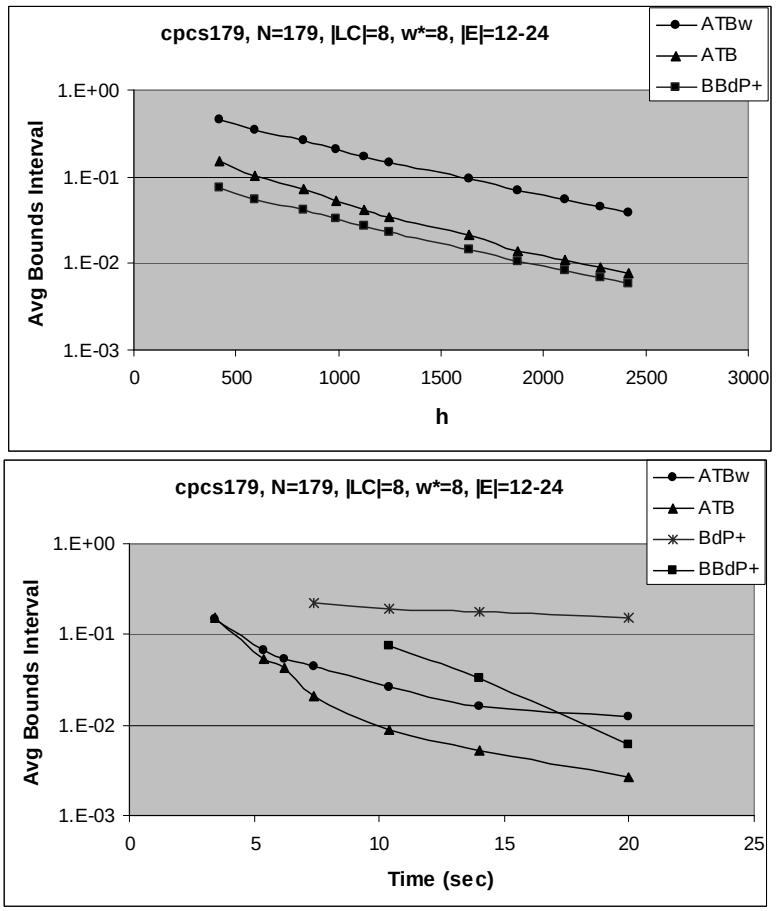

Figure 11: Results for cpcs179 network as a function of h (top) and a function of time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 2 seconds. Exact inference using cutset conditioning is 37 seconds.

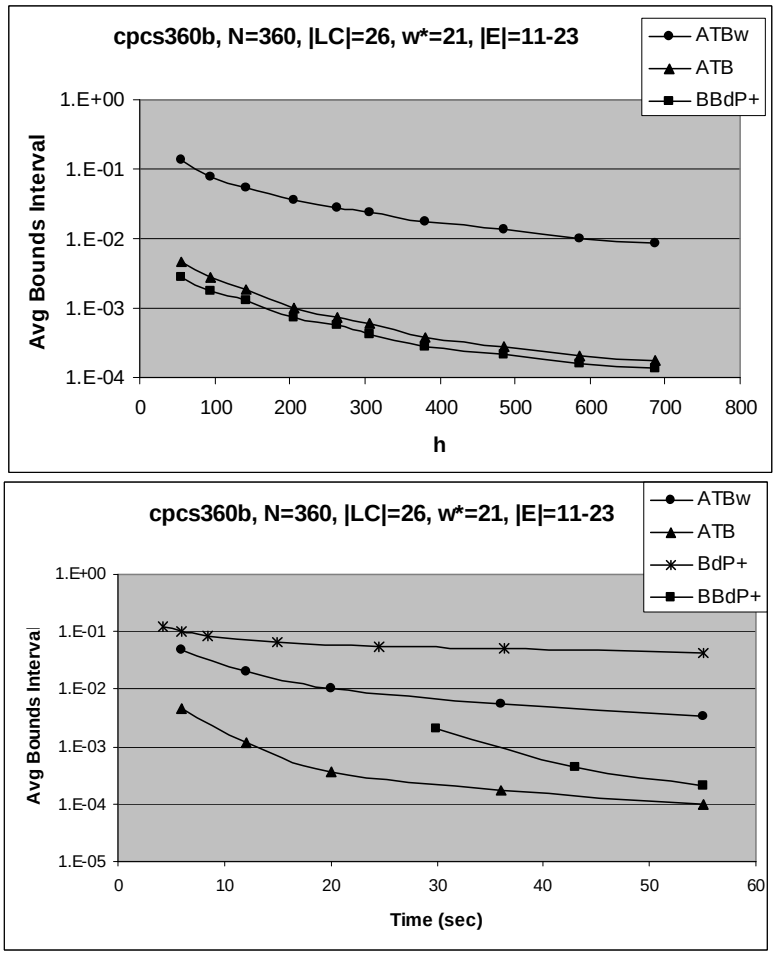

Figure 12: Results for cpcs360b as a function of h (top) averaged over 20 instances. Evidence is chosen randomly among leaf nodes only. Exact inference using bucket elimination is 20 minutes. Exact inference using cutset conditioning is > 8 hours.

We have compared the performance of all four bounding algorithms within 5 to 60 seconds that is only a small fraction of the time necessary to compute posterior marginals exactly. The convergence of all conditioning-based schemes was very fast in cpcs360b. At $h \approx 700$, the h cutset tuples contained on average about 98% of the probability mass of $P(e)$. The results are overall similar. ATB^w algorithm converges fast decreasing from $\overline{I} = 0.06$ for $h = 56$ to $\overline{I} = 0.008$ for $h = 686$. However, ATB and, consequently $BBdP+$, converged considerably faster. All bounds except ATB^w appear to be well centered around $P(x|e)$.

$BBdP+$ computation time varied sporadically with h as the input bounds to the linear optimization problem changed. On average, $BBdP+$ required considerably longer to compute 1 instance of the network, minimum of 25 seconds, compared to plain $BdP+$ which computed the bounds for the same network instances with the same maximum Markov table bound in 6 seconds. As we see from cpcs360b results, we encountered the rare instances in which the linear packing/covering problems requiring a long time to computing using simplex solver. Indeed, this is the class of problems that inspire the development of approximate methods for linear packing/covering problems. As we see, ATB^w, ATB, and $BBdP+$ schemes outperformed $BdP+$. The bounds computation in [15], when applied to cpcs360b benchmark,. achieved average bounds interval length of 0.03 in 10 seconds. Within the same time, ATB computes an average bounds interval of ≈ 0.001. However, the comparison may not be on the same instances since the evidence nodes are not the same.

Table 9: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ over 20 instances of cpcs422b network. Parameter h indicates the number of cutset tuples computed exactly in ATB.

	cpcs422b, $\|E\|$=6-11								
	ATB^w			ATB			BBdP+		
h	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time
256	0.7611	0.3243	12.2	0.2464	9.0E-2	26.1	0.1563	5.2E-2	109.2
379	0.7506	0.3194	15.1	0.2166	7.8E-2	32.0	0.1427	4.8E-2	41.0
561	0.7382	0.3137	16.7	0.2045	7.3E-2	36.6	0.1361	4.5E-2	46.0
862	0.7320	0.3109	19.8	0.1903	6.8E-2	44.0	0.1285	4.3E-2	54.1
1182	0.7215	0.3061	22.1	0.1835	6.5E-2	50.1	0.1253	4.1E-2	60.0
1502	0.7148	0.3031	24.7	0.1775	6.3E-2	56.4	0.1223	4.1E-2	65.7
2427	0.6898	0.2918	31.9	0.1678	5.9E-2	73.2	0.1179	3.9E-2	82.2
3062	0.6791	0.2871	35.9	0.1639	5.7E-2	83.3	0.1156	3.8E-2	92.3
4598	0.6554	0.2765	46.4	0.1530	5.4E-2	110.5	0.1095	3.6E-2	119.3

cpcs422b. The result for the fourth and the largest cpcs network, cpcs422b, are shown in Table 9 and Figure 13. Cpcs422b is challenging for any inference scheme. as it has large induced width of $w^* = 22$. It has a loop-cutset size $|LC| = 47$ and the loop-cutset tuple count is 2^{47}. We estimated that enumerating all those tuples would require over $2E + 9$ hours.

ATB consistently outperforms ATB^w as a function of h and time and outperforms $BdP+$ after 35 seconds. $BBdP+$ improves considerably over ATB as a function of h in Figure 13, top. The $BBdP+$ result for h=1182, $\overline{I} = 0.1253$, is the best of all algorithms for the 60 second time interval. However, the computation time of the $BBdP+$ was unpredictable again, similar to cpcs360b, and much longer than $BdP+$ algorithm. Hence, we did not plot the bounds interval for $BBdP+$ as a function of time in Figure 13, bottom, and refer the reader to the Table 9.

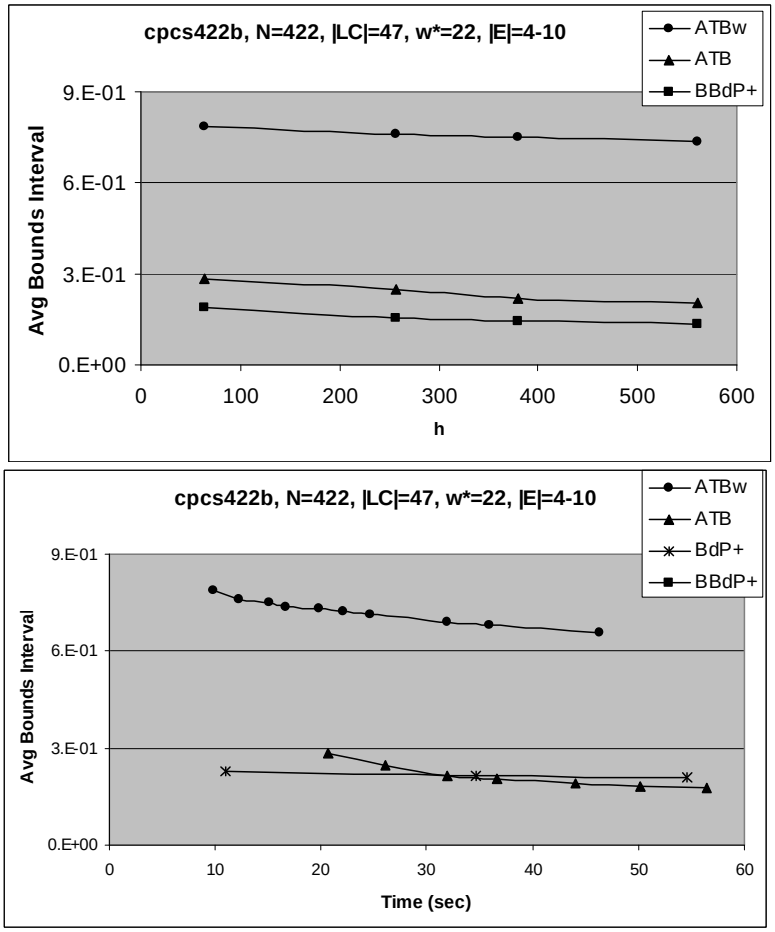

Figure 13: cpcs422b average bounds length as a function of h (top) and time (bottom), averaged over 20 instances. Exact inference using bucket elimination is 50 minutes. Exact inference using cutset conditioning is $> 2E + 9$ hours.

The bounds interval obtained by ATB and $BdP+$ are comparable to the average interval length of 0.15, obtained within 30 seconds, as reported in [15]. After 30 seconds, both ATB (see Table 9, $h = 379$) and $BdP+$ (see Table 3) compute average bounds interval of length ≈ 0.21. ATB's' bounds interval is reduced to 0.15 after 100 seconds (see Table 9, $h = 4598$). Note also that $BBdP+$ computes a smaller, $\overline{I} = 0.12$, bound than [15] in 40 seconds.

Table 10: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ over 20 instances of Munin3 network. Parameter h indicates the number of cutset tuples computed exactly in ATB.

	Munin3, N=1044, w^*=7, $\|LC\|$=30, $\|E\|$=257								
	ATB^w			ATB			BBdP+		
h	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time
392	0.2931	0.1270	10.0	0.0473	1.8E-2	14.3	0.0458	1.9E-2	23.5
882	0.2277	0.0960	14.9	0.0287	1.1E-2	21.9	0.0279	1.1E-2	32.5
1176	0.2077	0.0865	15.5	0.0272	9.9E-3	21.6	0.0265	1.0E-2	31.7
1568	0.1857	0.0762	18.7	0.0243	8.7E-3	26.6	0.0237	8.8E-3	37.6
1764	0.1831	0.0749	20.1	0.0241	8.6E-3	28.6	0.0235	8.7E-3	42.1
3332	0.1714	0.0696	34.4	0.0187	6.6E-3	47.5	0.0182	6.6E-3	56.5
5096	0.1589	0.0639	50.7	0.0172	6.0E-3	69.4	0.0167	6.0E-3	78.8
5586	0.1543	0.0618	56.5	0.0160	5.5E-3	76.3	0.0156	5.6E-3	85.0

Table 11: Average error Δ, bounds interval $\overline{I}$, and computation time t for $BdP+$, ATB, and $BBdP+$ over 20 instances of Munin4 network. Parameter h indicates the number of cutset tuples computed exactly in ATB.

	Munin4, N=1041, w^*=8, $\|LC\|$=49, $\|E\|$=235								
	ATB^w			ATB			BBdP+		
h	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time	$\overline{I}$	Δ	time
1372	0.7962	0.3637	11.2	0.3621	1.6E-1	20.1	0.2202	9.4E-2	31.4
3087	0.7482	0.3409	17.5	0.2991	1.3E-1	29.7	0.2065	8.8E-2	38.8
4802	0.7321	0.3333	22.8	0.2848	1.2E-1	39.5	0.2020	8.6E-2	57.0
6517	0.7087	0.3221	29.0	0.2651	1.1E-1	51.8	0.1951	8.3E-2	60.7
8232	0.6932	0.3147	33.9	0.2536	1.1E-1	59.7	0.1903	8.1E-2	72.9
16807	0.6440	0.2914	63.6	0.2151	8.8E-2	108.5	0.1726	7.3E-2	121.3
25382	0.6249	0.2823	93.3	0.2025	8.3E-2	160.5	0.1659	7.0E-2	174.1
33957	0.6058	0.2732	122.4	0.1909	7.8E-2	207.8	0.1594	6.7E-2	221.0
42532	0.5917	0.2665	151.6	0.1832	7.4E-2	255.6	0.1548	6.5E-2	265.4

Munin's benchmarks. Our last two benchmarks are Munin3 and Munin4. The evidence in each network instance has been pre-defined. Both networks are large, with 1044 and 1041 nodes. However, their induced widths are relatively small. The induced width of Munin3 is $w^* = 7$ and induced width of Munin4 is $w^* = 8$. Subsequently, exact inference by bucket elimination is fairly easy. As we report in Table 8.3, the exact computation time of Munin3 is 8 seconds and Munin4 is 70 seconds. The empirical results for each network are summarized in Tables 10 and 11 in Figures 14 and 15.

The behavior of the algorithms in Munin3 and Munin4 is similar and is self-explanatory. First, we take a look at the charts demonstrating the convergence of the ATB^w, ATB, and $BBdP+$ bounds interval with h in Figure 14, top, and

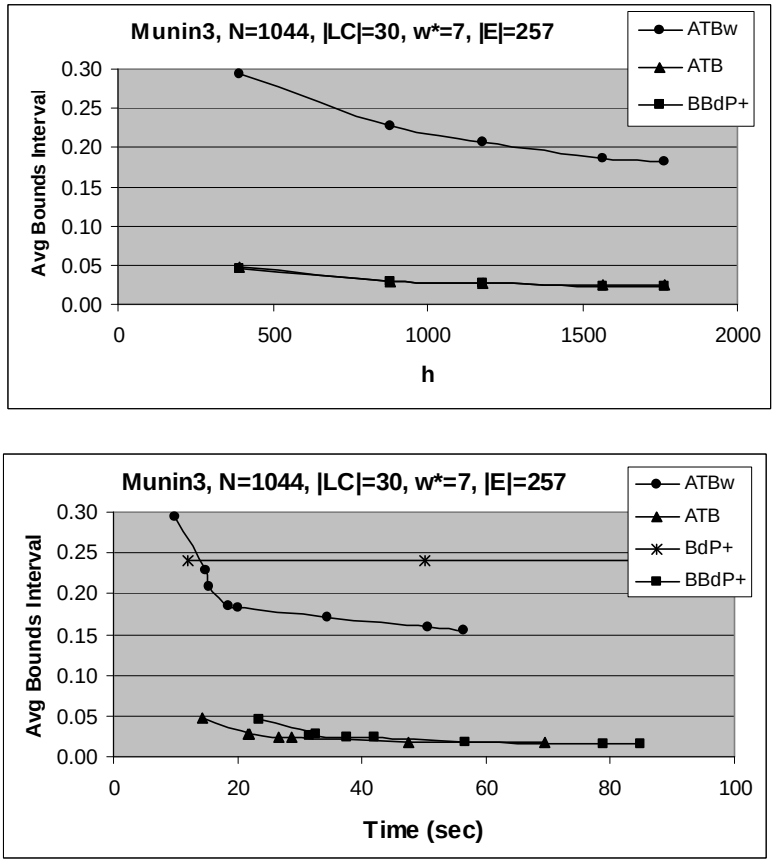

Figure 14: Munin3 average bounds length as a function of h (top) and time (bottom). Exact inference using bucket elimination is 8 seconds. Exact inference using cutset conditioning is $>$ 1700 hours.

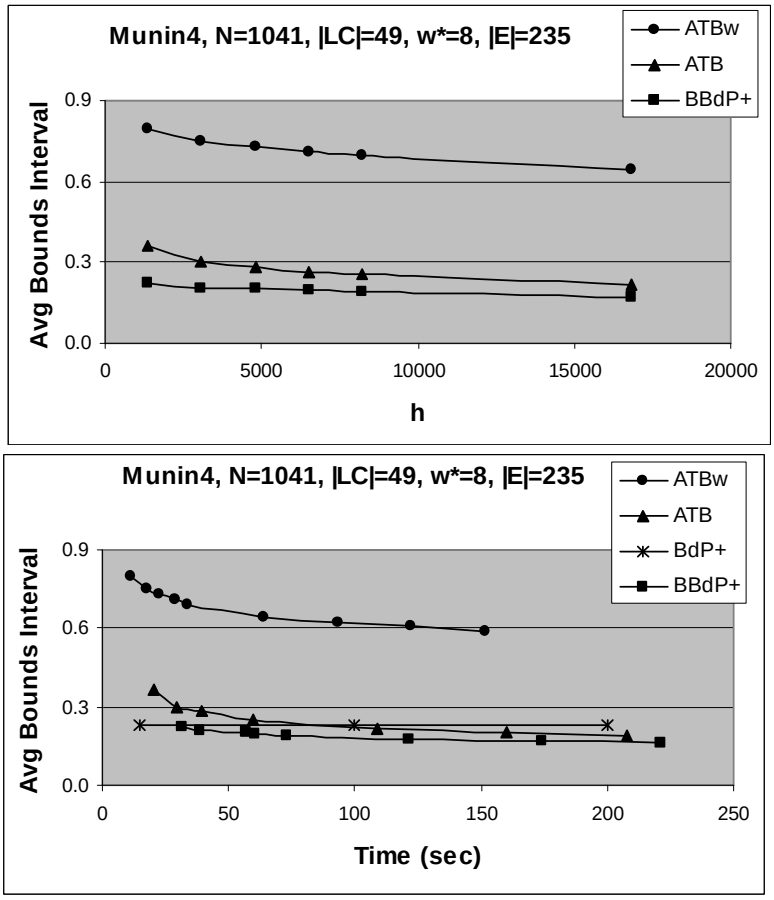

Figure 15: Munin4 average bounds length as a function of h (top) and time (bottom). Exact inference using bucket elimination is 70 seconds. Exact inference using cutset conditioning is $> 1E + 8$ hours.

Figure 15, top. The ATB^w algorithm is the worst. Its performance is especially poor in case of Munin4. After $h \approx 1000$, the ATB^w bounds interval length remains close to 0.8. Its performance improves with h very slowly. The ATB bounds interval is an order of magnitude smaller than ATB^w for Munin3 but improvement is more noticeable in Munin4. The bounds in this case appear to be skewed to the end of the bounds interval for all three schemes since average error is very close to half of the bounds interval length.

Now, we look at the performance of ATB^w, ATB, $BBdP+$, and $BdP+$ as a function of time. $BdP+$ algorithm computes bounds quickly, in a few seconds, and does not improve with time. It computes bounds interval of 0.24 for Munin3 and 0.23 for Munin4 within 12 and 15 seconds respectively. In Munin3, ATB outperforms $BdP+$ by a wide margin yielding a bounds interval of 0.05 in 12 seconds. $BBdP+$ performance becomes similar to ATB after 30 seconds. In Munin4, the loop-cutset size is larger and, subsequently, covergence of ATB is slower. Intially, $BdP+$ outperforms ATB. In 15 seconds, ATB computes bounds interval length of 0.37. As h increases, ATB outperforms $BdP+$ after 100 seconds. Although $BBdP+$ incurs initial comptuation overhead compared to ATB, it consistently outperforms ATB timewise. $BBdP+$ outperforms $BdP+$ after about 25 seconds.

8.5 Summary

Comparison of BdP and BdP+ over all benchmarks in Tables 2 and 3 indicates that taking advantage of available information about network structure can yield significant improvements in performance of bound propagation scheme.

Using cutset sampling as a search method proved very effective. We were able to accumulate over 90% of the weight of $P(e)$ in a few thousand tuples in cpcs179, cpcs360b, and Munin3. In cpcs422b and Munin4, we accumulated up to 20-30% of probability mass of $P(e)$ after generating just a few hundred of cutset tuples.

The empirical results confirm our expectations that by combining conditioning and bound propagation in the ATB framework, we can obtain a bounding scheme that is superior to the stand-alone bound propagation. Note that bound propagation plugin solved the linear optimization problem approximately, giving $BdP+$ an advantage over ATB at the start. Despite that, in several benchmarks, where the distribution $P(C|e)$ indeed peaked over just a few cutset tuples, the ATB algorithm outperformed $BdP+$ by a wide margin. In those benchmarks where the distribution $P(C|e)$ was closer to uniform, the convergence of ATB bounds was slower and required 30-100 seconds to achieve the same accuracy as $BdP+$.

In larger networks, cpcs360b and cpcs422b, ATB obtained a small bounds interval in a fraction of time needed to compute exact posterior marginals by bucket elimination or cutset conditioning. It computed average bounds interval of 0.0002 for cpcs360b and 0.15 for cpcs422b within 36 and 110 seconds respectively. In networks with smaller induced width, both ATB nor $BdP+$ were quickly outperformed by bucket elimination. However, for fair comparison, we have to contrast ATB computation time to that of cutset conditioning algorithm using linear amount of memory. ATB produced non-trivial lower and upper bounds in a fruction of time that cutset conditining requires to enumerate all tuples.

We also demonstrated that in all benchmarks except Alarm network, ATB converges faster with time than ATB^w. Namely, in most instances, generating extra cutset tuples using the time saved by eliminating computation of bounds on $P^U_{ABdP+}(x|c^j_{1:q_j}, e)$ is not enough to compensate for the slower bounds convergence rate. Thus, it is usually cost-effective to invest time into bounding $P(x, c_{1:q}, e)$.

Comparison of BdP and BdP+ over all benchmarks in Tables 2 and 3 indicates that taking advantage of available information about network structure can yield significant improvements in performance of bound propagation scheme.

The results show that boosting the bound propagation algorithm by using output ATB bounds as input to $BdP+$ is cost-effective when bound propagation computation time in $BBdP+$ is small compared to the ATB computation time. The performance of $BBdP+$ in several benchmarks affected by the sporadic changes in the computation time of the simplex solver, a problem that has been previously observed when optimizing fractional packing and covering problem and motivated the development of fast approximation algorithms.

Overall, we have demonstrated that ATB framework can yield meaningful bounds interval after processing a few thousand tuples out of the millions of tuples in cutset domain.

9 Conclusions and Future Work

In this paper, we evaluated empirically the bound propagation algorithm [16] and defined enhancements that take advantage of network connectivity. We demonstrated empirically that their implementation improves the bounds and reduces computation time. We also defined approximation algrithms for solving exactly the linear optimization subproblems in the bound propagation scheme trading accuracy for speed. Subsequently, we defined an any-time bounding scheme (ATB) that builds on two previously proposed methods: bounded conditioning [9] and bound propagation. It generates a subset of cutset tuples, computing exactly their probabilities, and then uses bound propagation scheme (using an approximate algorithm for solving linear optimization problem) to bound the tails of the distribution $P(c, e)$ over the un-explored cutset tuples. We showed that the bounds computed by any-time bounding framework can be used to boost the performance of bound propagation algorithm by using ATB bounds as input.

Other related work includes the search algorithm for estimating posterior probabilities proposed in [19]. The similarity with our work is that a search tree is partially-explored while the sum of the remaining probabilities is bounded. In fact, the bounds posterior marginals obtained in [19] are similar to BCE bounds derived in this paper. However, in [19] the search tree corresponds to the state space of the whole network and hence, it is exponential in the network size. Instead of bounding the sum of probabilities on partially instantiated tuples directly, they "update" the bounding function when a conflict is discovered (tuple with probability 0). The ATB framework subsumes both bounded conditioning and the search algorith in [19] offering a unifying approach to bounding that combines search and exact inference over part of the search space (corresponding to the enumeration of the first h cutset tuples) and bounds in some way the sum of probabilities over the rest of the search space. In bounded conditioning, the rest of the probability mass is bounded via prior and in [19] it is refined via counting of the conflicts.

One of the alternative approaches for computing bounds was proposed in [20] where "context-specific" bounds were obtained from simplifying the conditional probability tables. The method performs a variant of bucket elimination where intermediate tables are collapsed by grouping some probability values together. However, since the method was validated only on a small car diagnosis network with 10 variables, it is hard to draw conclusions about its effectiveness. In [15], the bounds are also obtained by simplifying intermediate probability tables in the variable elimination order but, instead of grouping probabilities, the author solves an optimization prob-

lem to find a table decomposition that minimizes the error. A specialized *large deviation bounds* approach for layered networks is proposed in [13, 12] and an elaborate bounding scheme with non-linear objective function was proposed in [17].

It is hard to compare all of the above methods side by side as they exploit different Bayesian network properties in order to compute bounds. However, only bounded condition and ATB offer any-time properties where we can improve our bounds by investing more time and exploring more cutset instances. The main improvement in ATB over bounded conditioning is that 1) we compute much tighter bounds on $P(c, e)$ than prior $P(c)$ used by bounded conditioning; 2) we compute upper bounds using $P(x, c, e)$ that is often a lot smaller than $P(c, e)$. It is also worth noting that our approach offers a complete framework for computing bounds where any bounding algorithm can be used to bound P(c,e) and P(x,c,e) for partially-instantiated tuples.

9.1 Future Work

There are many options for improving the performance of ATB with bound propagation plugin. Performance of bound propagation depends on the efficiency of the linear optimization algorithm. We have looked at only two approximation schemes which computed bounds fast but at the cost of loosing a lot of accuracy. Other approximation algorithms can be tried offering different time/accuracy trade-offs. Bound propagation algorithm can be also improved by exploiting further the underlying network structure. Tighter constraints could be defined for linear optimization problems by recognizing the nodes in Markov blanket that are independent from the others. Then, in the extreme case where all nodes are independent, we would have the number of constraints equal to the number of variables yielding tighter bounds that are easy to compute. Performance of ATB framework with other plugin schemes also should be investigated.

Appendix A Analysis of Bounded Conditioning

In [9], the lower and upper bounds are computed first for case of evidence e where all tuples c^i are explored (bounding from complete state) and then for case of adding new evidence f where only a subset of tuples is explored (bounding from incomplete state). We will disregard here evidence e and bounding from complete state because our objective is to avoid ever exploring all tuples, with or without evidence. Also, to maintain the same notation used throughout this paper, we will denote new evidence with e, not f as in [9]. Thus, we consider a simple case where we are given a Bayesian network, a cutset C, evidence e, and some means of selecting h cutset tuples out of total M. Following the rules of bounding from incomplete state in [9] while disregarding evidence e in [9], we have following lower and upper bounds:

$$P^L(x|e) = \sum_{i=1}^{h} P(x|c^i,e)w_i^L \tag{56}$$

$$P^U(x|e) = \sum_{i=1}^{h} P(x|c^i,e)w_i^U + \sum_{i=h+1}^{j} w_i^U + \sum_{i=j+1}^{M} w_i^{U'} \tag{57}$$

Since the sum $\sum_{i=h+1}^{j} w_i^U$ in $P^U(x|e)$ corresponds in [9] to summing over tuples where we compute $P(c^i,e)$ but not $P(x,c^i,e)$ and we do not allow this situation to occur (if we took the trouble of computing $P(c^i,e)$, it makes sense to compute $P(x|c^i,e)$ and obtain the $P(x,c^i,e) = P(x|c^i,e)P(c^i,e)$), then we set h=j and simplify:

$$P^L(x|e) = \sum_{i=1}^{h} P(x|c^i,e)w_i^L \tag{58}$$

$$P^U(x|e) = \sum_{i=1}^{h} P(x|c^i,e)w_i^U + \sum_{i=h+1}^{M} w_i^{U'} \tag{59}$$

The weights in the above expressions are defined as follows:

$$w_i^L = \frac{P(c^i|e)}{\sum_{k=1}^{h} P(c^k|e) + \sum_{k=h+1}^{M} P(c^k)} = \frac{P(c^i,e)}{\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k)} \tag{60}$$

$$w_i^U = \frac{P(c^i|e)}{\sum_{k=1}^{h} P(c^k|e)} = \frac{P(c^i,e)}{\sum_{k=1}^{h} P(c^k,e)} \tag{61}$$

$$w_i^{U'} = \frac{P(c^i)}{\sum_{k=1}^{h} w_k^L + (1 - \sum_{k=h+1}^{M} w_k^U)} \tag{62}$$

We can simplify computation of $w_i^{U'}$ observing that actually:

$$\sum_{k=h+1}^{M} w_k^U = \frac{\sum_{k=1}^{h} P(c^k,e)}{\sum_{k=1}^{h} P(c^k,e)} = 1$$

and then substituting w_i^L with its expanded form, we obtain:

$$w_i^{U'} = \frac{P(c^i)}{\sum_{k=1}^{h} w_k^L + 1 - 1} = \frac{P(c^i)}{\sum_{k=1}^{h} w_k^L} = \frac{P(c^i)(\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k))}{\sum_{k=1}^{h} P(c^k,e)}$$

Substituting weight formulas in the bounds expressions, we obtain:

$$P^L(x|e) = \sum_{i=1}^{h} P(x|c^i,e) \frac{P(c^i,e)}{\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k)}$$

$$
\begin{aligned}
&= \frac{\sum_{i=1}^{h} P(x|c^i,e)P(c^i,e)}{\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k)} \\
&= \frac{\sum_{i=1}^{h} P(x,c^i,e)}{\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k)} \\
P^U(x|e) &= \sum_{i=1}^{h} P(x|c^i,e)w_i^U + \sum_{i=h+1}^{M} w_i^{U'} \\
&= \sum_{i=1}^{h} P(x|c^i,e)\frac{P(c^i,e)}{\sum_{k=1}^{h} P(c^k,e)} + \sum_{i=h+1}^{M} \frac{P(c^i)(\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k))}{\sum_{k=1}^{h} P(c^k,e)} \\
&= \frac{\sum_{i=1}^{h} P(x|c^i,e)P(c^i,e)}{\sum_{k=1}^{h} P(c^k,e)} + \frac{\sum_{i=h+1}^{M} P(c^i)(\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k))}{\sum_{k=1}^{h} P(c^k,e)} \\
&= \frac{\sum_{i=1}^{h} P(x,c^i,e)}{\sum_{k=1}^{h} P(c^k,e)} + \frac{\sum_{i=h+1}^{M} P(c^i)(\sum_{k=1}^{h} P(c^k,e) + \sum_{k=h+1}^{M} P(c^k))}{\sum_{k=1}^{h} P(c^k,e)} \\
&= \frac{\sum_{i=1}^{h} P(x,c^i,e)}{\sum_{k=1}^{h} P(c^k,e)} + \sum_{i=h+1}^{M} P(c^i) + \frac{\sum_{k=h+1}^{M} P(c^k)}{\sum_{k=1}^{h} P(c^k,e)}
\end{aligned}
$$

Appendix B Bounding posteriors of cutset nodes

So far, we only considered computation of posterior marginals for variable $X \in \overline{X} \backslash C, E$. Now we focus on computing bounds for a cutset node $C_k \in C$. Let $c'_k \in \mathcal{D}(C)$ be some value in domain of C_k. Then, we can compute exact posterior marginal $P(c_k|e)$ using Bayes formula:

$$P(c'_k|e) = \frac{P(c'_k, e)}{P(e)} = \frac{\sum_{i=1}^{M} \delta(c'_k, c^i) P(c^i, e)}{\sum_{i=1}^{M} P(c^i, e)} \tag{63}$$

where $\delta(c'_k, c^i)$ is a Dirac delta-function so that $\delta(c'_k, c^i) = 1$ iff $c^i_k = c'_k$ and $\delta(c'_k, c^i) = 0$ otherwise. To simplify notation, let $Z = C \backslash Z$. Let M_k denote the number of tuples in state-space of Z. Then we can re-write the numerator as:

$$\sum_{i=1}^{M} \delta(c'_k, c^i) P(c^i, e) = \sum_{i=1}^{M_k} P(c'_k, z^i, e)$$

and the denominator can be decomposed as:

$$\sum_{i=1}^{M} P(c^i, e) = \sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=1}^{M_k} P(c'_k, z^i, e)$$

Then, we can re-write the expression for $P(c'_k|e)$ as follows:

$$P(c'_k|e) = \frac{\sum_{i=1}^{M_k} P(c'_k, z^i, e)}{\sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=1}^{M_k} P(c_k, z^i, e)} \tag{64}$$

Let h_{c_k} be the number of full cutset tuples where $c^i_k = c_k$. Then, we can decompose the numerator in Eq.(64) as follows:

$$\sum_{i=1}^{M_k} P(c'_k, z^i, e) = \sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{i=h_{c'_k}+1}^{M_k} P(c'_k, z^i, e)$$

Similary, we can decompose the sums in the denominator:

$$\sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=1}^{M_k} P(c_k, z^i, e) = \sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=1}^{h_{c_k}} P(c_k, z^i, e) + \sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=h_{c_k}+1}^{M_k} P(c_k, z^i, e)$$

After decomposition, the Eq.(64) takes on the form:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{i=h_{c'_k}+1}^{M_k} P(c'_k, z^i, e)}{\sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=1}^{h_{c_k}} P(c_k, z^i, e) + \sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=h_{c_k}+1}^{M_k} P(c_k, z^i, e)} \tag{65}$$

Now, for conciseness, we can group together all fully instantiated tuples in the denominator:

$$\sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=1}^{h_{c_k}} P(c_k, z^i, e) = \sum_{i=1}^{h} P(c^i, e)$$

Then, Eq.(65) transforms into:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{i=h_{c'_k}+1}^{M_k} P(c'_k, z^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=h_{c_k}+1}^{M_k} P(c_k, z^i, e)} \tag{66}$$

Now, we can replace each sum $\sum_{i=h_{c'_k}+1}^{M_k}$ over unexplored cutset tuples with a sum over the partially-instantiated cutset tuples. Denoting as $M'_{c_k} = M_k - h_{c_k} + 1$ the number of partially instantiated cutset tuples for $C_k = c_k$, we obtain:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{c_k \in \mathcal{D}(C_k)} \sum_{j=1}^{M'_{c_k}} P(c_k, z^j_{1:q_j}, e)} \tag{67}$$

In order to obtain lower and upper bounds formulation, we will separate the sum of joint probabilities $P(c'_k, z^j_{1:q_j}, e)$ where $C_k = c'_k$ from the rest:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P(c_k, z^j_{1:q_j}, e)} \tag{68}$$

In the expression above, probabilities $P(c_k, z^i, e)$ and $P(c^i, e)$ are computed exactly since they correspond to full cutset instantiations. Probabilities $P(c_k, z^i_{1:q_i}, e)$, however, will be bounded since only partial cutset is observed. Observing that both numerator and denominator have component $P(c'_k, z^i_{1:q_i}, e)$ and replacing it with an upper bound $P^U(c'_k, z^i_{1:q_i}, e)$ in both numerator and denominator, we will obtain an upper bound on $P(c'_k|e)$ due to Lemma 4.2:

$$P(c'_k|e) \leq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^U(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P^U(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P(c_k, z^j_{1:q_j}, e)} \tag{69}$$

Finally, replacing $P(c_k, z^j_{1:q_j}, e)$, $c_k \neq c'_k$, with a lower bound (also increasing fraction value), we obtain:

$$P(c'_k|e) \leq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c_k}} P^U(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c_k}} P^U(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P^L(c_k, z^j_{1:q_j}, e)} = P_c^{U_1} \tag{70}$$

The lower bound derivation is similar. Taking Eq.(66) as the basis, we first group together all partially-instantiated tuples:

$$\sum_{c_k \in \mathcal{D}(C_k)} \sum_{i=h_{c_k}+1}^{M_k} P(c_k, z^i, e) = \sum_{i=h+1}^{M} P(c^i, e)$$

transforming Eq.(66) into:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{i=h_{c'_k}+1}^{M_{c_k}} P(c'_k, z^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i, e)} \tag{71}$$

Now, replacing the summation of unexplored fully-instantiated tuples in Eq(69) with summation over corresponding partially-instantiated tuples, we obtain:

$$P(c'_k|e) = \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j}, e)} \tag{72}$$

We obtain lower bound by replacing $P(c^j_{1:q_j}, e)$ in the denominator with an upper bound and $P(c'_k, z^j_{1:q_j}, e)$ in the numerator with a lower bound yielding:

$$P(c'_k|e) \geq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U(c^j_{1:q_j}, e)} = P_c^{L_1} \tag{73}$$

We can obtain a different lower bound if we start with Eq.(68) and replace $P(c'_k, z^i_{1:q_i}, e)$ in numerator and denominator with a lower bound. Lemma 4.2 guarantees that the resulting fraction will be a lower bound on $P(c'_k|e)$:

$$P(c'_k|e) \geq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P(c_k, z^j_{1:q_j}, e)} \tag{74}$$

Finally, replacing $P(c_k, z^j_{1:q_j}, e)$ in Eq.(74) with a corresponding upper bound, we obtain the second lower bound $P_c^{L_2}$:

$$P(c'_k|e) \geq \frac{\sum_{i=1}^{h_{c'_k}} P(c'_k, z^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'_{c'_k}} P^L(c'_k, z^j_{1:q_j}, e) + \sum_{c_k \neq c'_k} \sum_{j=1}^{M'_{c_k}} P^U(c_k, z^j_{1:q_j}, e)} = P_c^{L_2} \tag{75}$$

The lower bounds $P_c^{L_1}$ and $P_c^{L_2}$ are respective cutset equivalents of the lower bounds P^{L_1} and P^{L_2} obtained in Eq.(23) and Eq.(27). Hence, the result of Theorem 4.2 and Corollary 4.2 apply.

With respect to computing bounds on $P(c'_k, z_{1:q}, e)$ in Eq.(70) and Eq.(75) in practice, we distinguish two cases. We demonstrate them on the example of upper bound.

In the first case, each partially instantiated tuple $c_{1:q}$ that includes node C_k, namely $k \leq q$, can be decomposed as $c_{1:q} = z_{1:q} \bigcup c'_k$ so that:

$$P^U(c'_k, z_{1:q}, e) = P^U(c_{1:q}, e)$$

The second case concerns the partially instantiated tuples $c_{1:q}$ that do not include node C_k, namely $k > q$. In that case, we compute upper bound by decomposing:

$$P^U(c'_k, z_{1:q}, e) = P^U(c_k|c_{1:q})P^U(c_{1:q}, e)$$

Appendix C Proofs

THEOREM **4.2 (Lower Bounds)** Given a Bayesian network $\mathcal{B}$ with a cutset C and evidence E, let X be some variable in $\mathcal{B}$ and x' be a value in the domain of X. Let $P^L(x|c^i_{1:q_i}, e)$ and $P^U(x|c^i_{1:q_i}, e)$ denote some bounds on $P(x|c^i_{1:q_i}, e)$. Let $P^L(c^i_{1:q_i}, e)$ and $P^U(c^i_{1:q_i}, e)$ denote some bounds on $P(c^i_{1:q_i}, e)$. Assume $P^{L_1}(x'|e)$ and $P^{L_2}(x'|e)$ are obtained from Eq.(23) and Eq.(27) where $P^L(x, c^i_{1:q_i}, e)$ and $P^U(x, c^i_{1:q_i}, e)$ are defined as follows:

$$\forall x \in \mathcal{D}(X), P^L(x, c^i_{1:q_i}, e) = P^L(x|c^i_{1:q_i}, e)P^L(c^i_{1:q_i}, e)$$
$$\forall x \in \mathcal{D}(X), P^U(x, c^i_{1:q_i}, e) = P^U(x|c^i_{1:q_i}, e)P^U(c^i_{1:q_i}, e)$$

If $P^L(x'|c^i_{1:q_i}, e) = 1 - \sum_{x \neq x'} P^U(x|c^i_{1:q_i}, e)$ then $P^{L_1}(x'|e) \geq P^{L_2}(x'|e)$. **Proof.** The numerators are the same. Hence, we only need to compare denominators. Let D_1 denote denominator in P^L_1 and D_2 denote denominator in P^L_2. Each denominator contains a $\sum_{i=1}^{h} P(c^i, e)$ component which will cancel out in $D_1 - D_2$. Therefore, the difference is:

$$\begin{aligned}
D_1 - D_2 &= \sum_{i=h+1}^{M'} P^L(x', c^i_{1:q_i}, e) + \sum_{x \neq x'} \sum_{i=h+1}^{M'} P^U(x, c^i_{1:q_i}, e) - \sum_{i=h+1}^{M'} P^U(c^i_{1:q_i}, e) \\
&= \sum_{i=h+1}^{M'} [P^L(x', c^i, e) + \sum_{x \neq x'} P^U(x, c^i_{1:q_i}, e) - P^U(c^i_{1:q_i}, e)] \\
&= \sum_{i=h+1}^{M'} [P^L(x'|c^i, e)P^L(c^i_{1:q_i}, e) + \sum_{x \neq x'} P^U(x|c^i_{1:q_i}, e)P^U(c^i, e) - P^U(c^i_{1:q_i}, e)] \\
&= \sum_{i=h+1} [P^L(x'|c^i_{1:q_i}, e)P^L(c^i, e) - P^U(c^i_{1:q_i}, e)(1 - \sum_{x \neq x'} P^U(x|c^i_{1:q_i}, e))]
\end{aligned}$$

By theorem condition, $P^L(x'|c^i_{1:q_i}, e) = 1 - \sum_{x \neq x'} P^U(x|c^i_{1:q_i}, e)$. Therefore:

$$\begin{aligned}
D_1 - D_2 &= \sum_{i=h+1}^{M'} [P^L(x'|c^i_{1:q_i}, e)P^L(c^i_{1:q_i}, e) - P^U(c^i_{1:q_i}, e)P^L(x'|c^i_{1:q_i}, e)] \\
&= \sum_{i=h+1}^{M'} P^L(x'|c^i_{1:q_i}, e)(P^L(c^i_{1:q_i}, e) - P^U(c^i_{1:q_i}, e)) \leq 0
\end{aligned}$$

Thus, $D_1 < D_2$. Then, $P^L_1 \geq P^L_2$. $\square$

THEOREM **4.3** If $\forall j,\ P^U(x, c^j_{1:q_j}, e) \leq P(c^j_{1:q_j})$ then $P^{U_1}(x|e) \leq P^U(x|e)$ where $P^{U_1}(x|e)$ is given in Eq.(29) and $P^U(x|e)$ is the bounded conditioning upper bound given in Eq.(11). **Proof.** Setting $P^L(x, c^j_{1:q_j}, e) = 0$, $x \neq x'$, in Eq.(29) and, hence, reducing the denominotaor, we obtain:

$$P^{U_1}(x'|e) \leq \frac{\sum_{i=1}^{h} P(x', c^i, e) + \sum_{j=1}^{M'} P^U(x', c^j_{1:q_j}, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U(x', c^j_{1:q_j}, e)}$$

By assumption, $P^U(x', c^j_{1:q_j}, e) \leq P(c^j_{1:q_j})$. Setting the upper bound on $P(x', c^j_{1:q_j}, e)$ to its maximum value $P(^j_{1:q_j})$ in equation above yields:

$$P^{U_1}(x'|e) \leq \frac{\sum_{i=1}^{h} P(x, c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j})}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^j_{1:q_j})} \tag{76}$$

$$= \frac{\sum_{i=1}^{h} P(x, c^i, e) + \sum_{i=1}^{M} P(c^j)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=1}^{M} P(c^j)} \triangleq P^{U_3}(x'|e) \quad (77)$$

The bound $P^{U_3}(x'|e)$ in Eq.(77) represents the maximum value of $P^{U_1}(x'|e)$ under the assumption that $P^U(x', c^j_{1:q_j}, e) \leq P(c^j_{1:q_j})$. We will show next that the maximum value of $P^{U_1}(x'|e)$ is always less or equal to the bounded conditioning upper bound.

Rewrite $P^{U_3}(x|e)$ as a sum of fractions:

$$P^{U_3}(x|e) = \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i)} + \frac{\sum_{i=h+1}^{M} P(c^i)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i)}$$

The first summant in $P^{U_3}(x|e)$ is smaller then the first summant in eq.(11):

$$\frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i)} \leq \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e)}$$

The second summant in $\hat{P}^U(x|e)$ is smaller then the second summant in eq.(11):

$$\frac{\sum_{i=h+1}^{M} P(c^i)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i)} \leq \sum_{i=h+1}^{M} P(c^i)$$

The theorem follows. □

THEOREM **5.1** If C is a topologically ordered loop-cutset of a Bayesian network and $C_{1:q} = \{C_1, ..., C_q\}$ is a subset of C, $q < |C|$, then the relevant subnetwork of $C_{1:q}$ consisting of loop-cutset nodes in subset $C_{1:q}$ and their ancestors is singly-connected. **Proof.** First, we prove that the relevant subnetwork of any loop-cutset C_q is singly-connected when all loop-cutset preceeding C_q in the ordering are assigned. Proof by contradiction. Assume C_q is observed. If the relevant subnetwork of node C_q is not singly-connected, then there is a loop L with a sink S s.t. either S is observed or S has an observed descendant among $C_1, ..., C_{q-1}$ or C_q is a descendant of S (otherwise S would be irrelevant). Let C_i, $1 \leq i \leq q$ denote the node for which S is the ancestor (or $S = C_i$). By definition of loop-cutset, $\exists C_m \in L$ s.t. $C_m \neq S$ and $C_m \in C$. Then, C_m is ancestor of C_i. Since cutset is topologically ordered and all cutset nodes preceeding C_i are observed, then C_m must be observed, thus, breaking the loop. Contradiction. Applying the above result recursively, we have: relevant subnetwork of C_1 is singly-connected, relevant subnetwork of C_2, conditioned on on C_1, is singly-connected, and so on. The theorem follows. □

THEOREM **4.4** Given an algorithm $\mathcal{A}$ that can compute an upper bound on $P(c_{1:q}, e)$, where $c_{1:q}$ is a partial cutset instantiation, given h fully-instantiated cutset tuples c^i, $1 \leq i \leq h$, then:

$$P^{U_3}_{\mathcal{A}} - P^{L_3}_{\mathcal{A}} \geq \frac{\sum_{i=1}^{h} P(c^i, e)}{P(e)}$$

where $P^{L_3}_{\mathcal{A}}$ and $P^{U_3}_{\mathcal{A}}$ are expressed in Eq.(37) and Eq.(38) respectively. **Proof.** Let q denote the fraction of the probability mass covered by the explored cutset tuples:

$$q = \frac{\sum_{i=1}^{h} P(c^i, e)}{P(e)}$$

Then, the bounds interval $P^U_{\mathcal{A}} - P^L_{\mathcal{A}}$ is always lower bounded by $1-q$. We begin by computing the bounds interval:

$$P^U_{\mathcal{A}} - P^L_{\mathcal{A}} = \frac{\sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^j, e)}$$

We replace $\sum_{j=1}^{M'} P^U_{\mathcal{A}}(c^i, e)$ in both numerator and denominator with exact probability sum $\sum_{j=1}^{M'} P(c^j, e)$, yielding a lower bound on the bounds interval length:

$$P^U_{\mathcal{A}} - P^L_{\mathcal{A}} \geq \frac{\sum_{j=1}^{M'} P(c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{j=1}^{M'} P(c^i, e)} \tag{78}$$

Since, the $\sum_{j=1}^{M'} P(c^j, e) = \sum_{i=h+1}^{M} P(c^i, e)$, then the Eq. (78) transforms into:

$$P^U_{\mathcal{A}} - P^L_{\mathcal{A}} \geq \frac{\sum_{i=h+1}^{M} P(c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P(c^i, e)} \tag{79}$$

Replacing the sums in denominator in Eq. (78) with $P(e)$ and replacing $\sum_{i=h+1}^{M} P(c^i, e)$ in the numerator with $P(e) - \sum_{i=1}^{h} P(c^i, e)$, we get:

$$P^U_{\mathcal{A}} - P^L_{\mathcal{A}} \geq \frac{P(e) - \sum_{i=1}^{h} P(c^i, e)}{P(e)} = \frac{P(e) - qP(e)}{P(e)} = 1 - q \tag{80}$$

□

THEOREM **7.1 (Estimator is Bounded)** Given Bayesian network $\mathcal{B}$ with a cutset C and evidence E and some variable X, then

$$P^{L_3}(x|e) \leq \hat{P}_h(x|e) \leq P^{U_3}(x|e)$$

where $\hat{P}_h(x|e)$ is obtained from Eq. (52) and $P^{L_3}(x|e)$ and $P^{U_3}(x|e)$ are obtained resepectively from Eq.(23) and Eq.(29) by setting $\forall j$, $P^L(x, c^j_{1:q_j}, e) = 0$ and $P^U(x, c^j_{1:q_j}, e) = P^U(c^j_{1:q_j}, e)$. **Proof.** The relationship between $\hat{P}(x|e)$ and $P^{L_3}(x|e)$ is obvious:

$$\hat{P}(x|e) = \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e)} \geq \frac{\sum_{i=1}^{h} P(x, c^i, e)}{\sum_{i=1}^{h} P(c^i, e) + \sum_{i=h+1}^{M} P^U(c^i, e)} = P^L(x|e)$$

To prove the relationship between $\hat{P}(x|e)$ and $P^{U_3}(x|e)$, we first simplify notation. Let $a = \sum_{i=1}^{h} P(x_i, c^i, e)$, $b = \sum_{i=1}^{h} P(c^i, e)$, $\delta = \sum_{i=h+1}^{M} P^U(x, c^i, e)$. Using this notation, we have $\hat{P}(x|e) = \frac{a}{b}$ and $P^U(x|e) = \frac{a+\delta}{b+\delta}$ by definition. Then:

$$\hat{P}(x|e) = \frac{a}{b} \leq \frac{a+\delta}{b+\delta} = P^U(x|e)$$

due to Lemma 4.2. □

THEOREM **7.2 (Convergence Rate)** Given is a Bayesian network $\mathcal{B}$ with a cutset C and evidence E and variable X. Assume M is the number of cutset state-space

and we have generated h out of M cutset tuples yielding estimator $\hat{P}_h$ defined in Eq.(52), subindexed with h to indicate that it is a function of h. Define:

$$L_h = \frac{\sum_{i=h+1}^{M} P(c^i, e)}{\sum_{i=1}^{M} P(c^i, e)}$$

Then L_h monotonously converges to 0, $L_h \rightarrow 0$, as $h \rightarrow M$ and the distance converges in the norm L_h:

$$|P(x|e) - \hat{P}_h(x|e)| \leq L_h$$

Proof. Define additionally:

$$a = \sum_{i=1}^{h} P(x, c^i, e), \quad \delta = \sum_{i=h+1}^{M} P^U(x, c^i, e), \quad s_x = \sum_{i=h+1}^{M} P(x, c^i, e), \quad s = \sum_{i=h+1}^{M} P^U(c^i, e)$$

where $P^U(x, c^i, e)$ and $P^U(c^i, e)$ are any upper bounds on $P(x, c^i, e)$ and $P(c^i, e)$. Then, by definition:

$$P(x|e) = \frac{a + s_x}{b + s}, \quad P_h(x|e) = \frac{a}{b}$$

Then:

$$P(x|e) - P_h(x|e) = \frac{a + s_x}{b + s} - \frac{a}{b} = \frac{ab + as - bs_x - ab - as}{b(b + s)} = \frac{bs_x - as}{b(b + s)}$$

Factoring out b and s in the numerator, we obtain:

$$P(x|e) - P_h(x|e) = (\frac{s_x}{s} - \frac{a}{b})\frac{bs}{b(b + s)} = (\frac{s_x}{s} - \frac{a}{b})\frac{s}{b + s}$$

Taking the absolute value of the left and righ sides of the equality, we obtain:

$$|P(x|e) - P_h(x|e)| = |\frac{s_x}{s} - \frac{a}{b}|\frac{s}{b + s}$$

Since both $\frac{s_x}{s} \leq 1$, and $\frac{a}{b} \leq 1$, then $|\frac{s_x}{s} - \frac{a}{b}| \leq 1$. Applying this result to the equation above we get:

$$|P(x|e) - P_h(x|e)| \leq \frac{s}{b + s} = L_h$$

Thus, we have proved that $|P(x|e) - P_h(x|e)| \leq L_h$. Next, we show that L_h monotonously converges to 0 as $h \rightarrow M$.

Clearly, in the limit, $L_M = 0$. It is also obvious that L_h monotonously decreases as h increases since the numerator decreases as h increaes while the denominator remains unchanged. $\square$

THEOREM **7.3 (Using Estimator as a Bound)** Given a Bayesian network $\mathcal{B}$ with a cutset C and evidence E and variable X, let M be the number of tuples in state-space of cutset C, h be the number of fully-instantiated tuples in C, and $c^j_{1:q_j}$, $j \in [1, M']$, denote a partially-instantiated cutset tuple. Define: $a = \sum_{i=1}^{h} P(x, c^i, e)$, $b = \sum_{i=1}^{h} P(c^i, e)$. Then, $\forall h \in (1, M)$, $\hat{P}_h$ defined in Eq. (52) has following properties:

1. (a) If $\forall j \in [1, M']$, $P^U(x|c^j_{1:q_j}, e) \leq \frac{a}{b}$, then $P(x|e) \leq \hat{P}$.

2. (b) If $\forall j \in [1, M']$, $P^L(x|c^j_{1:q_j}, e) \geq \frac{a}{b}$, then $P(x|e) \geq \hat{P}$.

Proof. Let $s_x = \sum_{i=h+1}^{M} P(x, c^i, e)$, $s = \sum_{i=h+1}^{M} P(c^i, e)$ Using this notation, ATP estimator is given by:

$$\hat{P}(x|e) = \frac{a}{b}, \quad (x|e) = \frac{a + s_x}{b + s}$$

The estimator $\hat{P}(x|e)$ can fall in either side of $P(x|e)$. Estimate distance:

$$P(x|e) - \hat{P}(x|e) = \frac{a + s_x}{b + s} - \frac{a}{b} = \frac{ab + s_x b - ab - as}{b(b + s)} = \frac{s_x b - as}{b(b + s)} = \frac{s_x - \frac{a}{b}s}{b + s} \quad (81)$$

It is clear that the sign in the $P(x|e) - \hat{P}(x|e)$ depends on the sign of numerator $D = s_x - \frac{a}{b}s$. Let us expand the sums:

$$D = s_x - \frac{a}{b}s = \sum_{i=h+1}^{M} P(x, c^i, e) - \frac{a}{b} \sum_{i=h+1}^{M} P(c^i, e) \quad (82)$$

$$= \sum_{i=h+1}^{M} P(x|c^i, e)P(c^i, e) - \frac{a}{b} \sum_{i=h+1}^{M} P(c^i, e) \quad (83)$$

$$= \sum_{i=h+1}^{M} [P(x|c^i, e) - \frac{a}{b}]P(c^i, e) \quad (84)$$

$$= \sum_{j=1}^{M'} [P(x|c^j_{1:q_j}, e) - \frac{a}{b}]P(c^j_{1:q_j}, e) \quad (85)$$

$$(86)$$

Thus, if $P^U(x|c^j_{1:q_j}, e) < \frac{a}{b}$, $1 \leq j \leq M'$, then $P(x|c^j_{1:q_j}, e) < \frac{a}{b}$ and subsequently $D \leq 0$ and $P(x|e) \leq \hat{P}(x|e)$. Namely, $\hat{P}(x|e)$ is an upper bound on $P(x|e)$.

If $P^L(x|c^j_{1:q_j}, e) \geq \frac{a}{b}$, $1 \leq j \leq M'$, then $P(x|c^j_{1:q_j}, e) \geq \frac{a}{b}$ and subsequently $D \geq 0$ and $P(x|e) \geq \hat{P}(x|e)$. Namely, $\hat{P}(x|e)$ is a lower bound on $P(x|e)$. □

THEOREM **7.4 (Adjusting Upper Bound)** Assume given is a Bayesian network $\mathcal{B}$ with a cutset C and evidence E and X is some variable in $\mathcal{B}$. Let $a = \sum_{i=1}^{h} P(x, c^i, e)$, $b = \sum_{i=1}^{h} P(c^i, e)$. Let $\delta = \sum_{i=1}^{M'} P^U(x|c^i_{1:q_i}, e)P^U(c^i_{1:q_i}, e)$. Let $\Delta = \sum_{i=1}^{M'} P^U(c^i_{1:q_i}, e)$. If $\forall i \in [1, M']$, $\frac{a}{b} < P^U(x|c^i_{1:q_i}, e)$, then:

$$P(x|e) \leq \frac{a + \delta}{b + \Delta} + \frac{\delta}{b + \delta} \frac{\Delta}{b + \Delta}$$

Proof. Let $s_x = \sum_{i=h+1}^{M} P(x, c^i, e)$, $s = \sum_{i=h+1}^{M} P(c^i, e)$. Then:

$$P(x|e) \leq \frac{a + s_x}{b + s}$$

Define:

$$P_h(x|e) = \frac{a + \delta}{b + \Delta}$$

Compute the difference:

$$P(x|e) - P_h(x|e) = \frac{a + s_x}{b + s} - \frac{a + \delta}{b + \Delta} \quad (87)$$

$$
\begin{aligned}
&= \frac{ab + a\Delta + s_x b + s_x\Delta - ab - as - \delta b - \delta s}{(b+s)(b+\Delta)} && (88)\\
&= \frac{a\Delta + s_x b + s_x\Delta - as - \delta b - \delta s}{(b+s)(b+\Delta)} && (89)\\
&= \frac{s_x\Delta}{(b+s)(b+\Delta)} + \frac{a\Delta + s_x b - as - \delta b}{(b+s)(b+\Delta)} - \frac{\delta s}{(b+s)(b+\Delta)} && (90)\\
&= \frac{s_x\Delta}{(b+s)(b+\Delta)} - \frac{as + \delta b - a\Delta - s_x b}{(b+s)(b+\Delta)} - \frac{\delta s}{(b+s)(b+\Delta)} && (91)
\end{aligned}
$$

Next, we prove that:

$$\frac{as + \delta b - a\Delta - s_x b}{(b+s)(b+\Delta)} > 0 \tag{92}$$

The denominator is clearly positive. We show that numerator is positive too. Consider:

$$
\begin{aligned}
\gamma &= as + \delta b - a\Delta - s_x b = [\delta b - a\Delta] - [s_x b - as]\\
&= [b\sum_{j=h+1}^{M} P^U(x|c^j,e)P^U(c^j,e) - a\sum_{j=h+1}^{M} P^U(c^j,e)]\\
&- [b\sum_{j=h+1}^{M} P(x|c^j,e)P(c^j,e) - a\sum_{j=h+1}^{M} P(c^j,e)]\\
&= \sum_{j=h+1}^{M} [bP^U(x|c^j,e)P^U(c^j,e) - aP^U(c^j,e)] - \sum_{j=h+1}^{M} [bP(x|c^j,e)P(c^j,e) - aP(c^j,e)]\\
&= \sum_{j=h+1}^{M} [bP^U(x|c^j,e) - a]P^U(c^j,e) - \sum_{j=h+1}^{M} [bP(x|c^j,e) - a]P(c^j,e)\\
&= \sum_{j=h+1}^{M} b[P^U(x|c^j,e) - \frac{a}{b}]P^U(c^j,e) - b[P(x|c^j,e) - \frac{a}{b}]P(c^j,e)\\
&= b\sum_{j=h+1}^{M} [P^U(x|c^j,e) - \frac{a}{b}]P^U(c^j,e) - [P(x|c^j,e) - \frac{a}{b}]P(c^j,e)\\
&= b\sum_{j=1}^{M'} [P^U(x|c^j_{1:q_j},e) - \frac{a}{b}]P^U(c^j_{1:q_j},e) - [P(x|c^j_{1:q_j},e) - \frac{a}{b}]P(c^j_{1:q_j},e)
\end{aligned}
$$

It is given that $\frac{a}{b} \leq P^U(x|c^j_{1:q_j},e)$ and, thus, $P^U(x|c^j_{1:q_j},e) - \frac{a}{b} \geq 0$.

Assume $\frac{a}{b} \leq P(x|c^j_{1:q_j},e)$. Then, $P(x|c^j_{1:q_j},e) - \frac{a}{b} \geq 0$. Since $P^U(x|c^j_{1:q_j},e) \geq P(x|c^j_{1:q_j},e)$ and $P^U(c^j_{1:q_j},e) \geq P(c^j_{1:q_j},e)$, then:

$$
\begin{aligned}
P^U(x|c^j_{1:q_j},e) &\geq P(x|c^j_{1:q_j},e)\\
P^U(x|c^j_{1:q_j},e) - \frac{a}{b} &\geq P(x|c^j_{1:q_j},e) - \frac{a}{b}\\
(P^U(x|c^j_{1:q_j},e) - \frac{a}{b})P^U(c^j_{1:q_j},e) &\geq (P(x|c^j_{1:q_j},e) - \frac{a}{b})P(c^j_{1:q_j},e)
\end{aligned}
$$

Thus, if $\frac{a}{b} < P(x|c^j_{1:q_j},e), P^U(x|c^j_{1:q_j},e)$, then every summant is positive and $\gamma \geq 0$.

Assume $P(x|c^j_{1:q_j}, e) \leq \frac{a}{b} \leq P^U(x|c^j_{1:q_j}, e)$. Then, $\frac{a}{b} - P(x|c^j_{1:q_j}, e) \geq 0$. Rewriting γ as:

$$\gamma = b \sum_{j=h+1}^{M} [P^U(x|c^j_{1:q_j}, e) - \frac{a}{b}]P^U(c^j_{1:q_j}, e) + [\frac{a}{b} - P(x|c^j_{1:q_j}, e)]P(c^j_{1:q_j}, e)$$

we see that every summant is positive and, thus, the γ is positive.

Thus, if $\frac{a}{b} \leq P^U(x|c^j_{1:q_j}, e)$, then inequality Eq.(92) holds. Then, dropping the negative addands from the Eq.(91), we obtain:

$$P(x|e) - P_h(x|e) \leq \frac{\delta}{b+\delta}\frac{\Delta}{b+\Delta} = \frac{a+\delta}{b+\Delta} + \frac{\delta}{b+\delta}\frac{\Delta}{b+\Delta}$$

We obtain the result of the theorem moving $P_h(x|e)$ to the right hand side of the expression above. $\square$

References

[1] Computational infrastructure for operations research. *www.coin-or.org*.

[2] Munin - an expert emg assistant. In John E. Desmedt, editor, *Computer-Aided Electromyography and Expert Systems, ch. 21*. Elsevier Science Publishers, Amsterdam, 1990.

[3] A. Becker and D. Geiger. A sufficiently fast algorithm for finding close to optimal junction trees. In *Uncertainty in AI*, pages 81–89, 1996.

[4] I. Beinlich, G. Suermondt, R. Chavez, and G. Cooper. The alarm monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Second European Conferece on AI and Medicine, Berlin, 1989*. Springer–Verlag, 1989.

[5] B. Bidyuk and R. Dechter. Cycle-cutset sampling for bayesian networks. In *Sixteenth Canadian Conf. on AI*, pages 297–312, 2003.

[6] B. Bidyuk and R. Dechter. Empirical study of w-cutset sampling for bayesian networks. In *Uncertainty in AI*, pages 37–46, 2003.

[7] D. Bienstock. *Potential function methods for approximately solving linear programming problmes: theory and practice*. Kluwer Academic Publishers, 2002.

[8] R. Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113:41–85, 1999.

[9] E.J. Horvitz, H.J. Suermondt, and G.F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Workshop on Unertainty in Artificial Intelligence*, pages 181–193, 1989.

[10] K. Kask and R. Dechter. Branch and bound with mini-bucket heuristics. In *International Joint Conference on Artficial Intelligence (IJCAI'99)*, pages 426–433, 1999.

[11] K. Kask and R. Dechter. Stochastic local search for bayesian networks. In D. Heckerman and J. Whittaker, editors, *Workshop on AI and Statistics '99*, pages 113–122. Morgan Kaufmann Publishers, 1999.

[12] M. Kearns and L. Saul. Large deviation methods for approximate probabilistic inference, with rates of convergence. In *In Uncertainty in AI*, pages 311–319. Morgan Kaufmann, 1998.

[13] M. Kearns and L. Saul. Inference in multilayer networks via large deviation bounds. *Advances in Neural Information Processing Systems*, 11:260–266, 1999.

[14] K. Kristensen and I.A. Rasmussen. The use of a bayesian network in the design of a decision support system for growing malting barley without use of pesticides. *Computers and Electronics in Agriculture*, 33:197–217, 2002.

[15] D. Larkin. Approximate decomposition: A method for bounding and estimating probabilistic and deterministic queries. In *UAI'2003*, 2003.

[16] M. A. R. Leisink and H. J. Kappen. Bound propagation. *Journal of Artificial Intelligence Research*, 19:139–154, 2003.

[17] Michael V. Mannino and Vijay S. Mookerjee. Probability bounds for goal directed queries in bayesian networks. *IEEE Transactions on Knowledge and Data Engineering*, 14(5):1196–1200, September/October 2002.

[18] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, 1988.

[19] D. Poole. Probabilistic conflicts in a search algorithm for estimating posterior probabilities in bayesian networks. *Artificial Intelligence*, 88(1–2):69–100, 1996.

[20] David Poole. Context-specific approximation in probabilistic inference. In *Proceedings of Uncertainty in Artificial Intelligence (UAI-98)*, pages 447–454, 1998.

[21] M. Pradhan, G. Provan, B. Middleton, and M. Henrion. Knowledge engineering for large belief networks. In *Proc. Tenth Conf. on Uncertainty in Artificial Intelligence*, 1994.